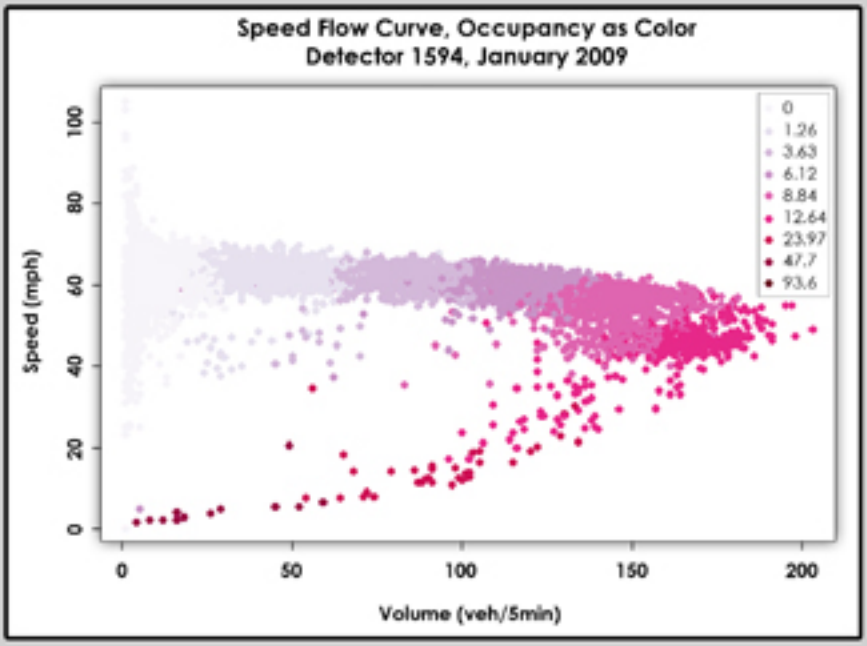
First Edition

Understanding and Communicating Multimodal Transportation Data



by Chris Monsere

Activity-Based Learning

Understanding and Communicating Multimodal Transportation Data

First Edition

Development, Deployment, and Assessment of a New Educational Paradigm for Transportation Professionals and University Students (A Collaboration of the Region X Transportation Consortium)

by Chris Monsere

published by

Pacific Crest
Plainfield, IL

Understanding and Communicating Multimodal Transportation Data

First Edition

by Chris Monsere

Layout and Production by Denna Hintze

Published by

Pacific Crest
13250 S. Route 59, Unit 104
Plainfield, IL 60585
815-676-3470
www.pcrest.com

ISBN: 978-1-60263-440-4

Companion website:

http://www.transportation-data.com/

Table of Contents

List of Figures

Student Notes

Preface

This course will introduce students to appropriate research methods for using transportation data sets and communicating the results of their work to a broad audience. The course content includes:

(a) selections of the appropriate graphical method (making knowledge-based decisions on selections for best perceptions)

(b) managing, extracting, and filtering large-scale data

(c) understanding types and dimensions of data (time resolution, discrete, continuous, and aggregations)

(d) techniques for visualizing data and exploratory analysis

(e) basic statistical analysis applied to transportation problems (public transportation, traffic, safety, freight, bicycle performance) using open-source script-based statistical tools (R) and databases (PostgresSQL)

(f) selection of appropriate analysis technique

(g) presentation of material in a technical summary

This is a gateway course; the knowledge gained in this course will be applied throughout the remaining graduate curriculum.

Students taking this course will have had an introductory transportation course, an undergraduate course in statistics and probability, and an engineering problem solving course with an exposure to programming logic. Three audiences are envisioned for this course:

(a) Graduate-level civil engineering students with an emphasis in transportation, in their first quarter.

(b) Transportation professionals with a desire to expand their knowledge of data analysis

(c) Advanced senior undergraduate civil engineering students with necessary skills and permission of the instructor.

The course uses the open source language R. Use of the PostgreSQL database will require comfort with various computing platforms (Unix, Windows) including the installation of software, downloading and installing web-based technologies.

The long-term behaviors, roles, and way of being will be supported by this course:

(a) Problem solver

(b) Researcher

(c) Communicator

(d) Collaborator

(e) Open-source software

i. R

ii. PostgreSQL

Reference books (required)

i. Keen, Kevin. *Graphics for Statistics and Data Analysis with R*

ii. Dalgaard, Peter. *Introductory statistics with R*. 2nd ed.

iii. *Scientific Approaches to Transportation Research* Volumes 1 and 2, web book

In this activity textbook, each activity includes an overview, a description of the task, a description of the deliverable, and the assessment method. Activities are also shown as in class or out of class. The following structure will be used to assess activities:

1. **Participation Activities**
    a. These activities require quick assessment and feedback. You will receive credit for completing these activities.

2. **Annotated Code Activities**
    a. In these activities you be asked to only submit a script or code file that contains comments and demonstrates active exploration of the objectives within the activity.

3. **Peer Assessment Activities**
    a. Some activities will require you to assess the work of your fellow students. In these activities, your performance will be based on your work assessed by the instructor, your feedback to peers, and your peers' assessment of your work.

4. **Short Response Activities**
    a. Many activities are structured such that you respond to a set of questions. You will receive credit both for completing these activities and for the depth and detail of your responses. We will attempt electronic submittal and feedback for these activities.

5. **Discovery Activities**
    a. These activities require you to build on knowledge and skills introduced to you in previous activities. These activities will be open ended and you will receive credit for completing these activities and for the creativity of your exploration.

There will also be a final project which is an independent structured analysis which you will select from a set of open-ended questions devised by the instructor presented in Chapter 6. This project will serve as the final assessment that the student has made progress in developing knowledge and skills in this class. The project is due during the final exam period, where students will make a brief presentation on their results to the class. You are encouraged to make an early selection of the project topic to begin your work in advance.

A number of people have contributed to make this version of the course document. The early version of this course and activities were developed by the primary author, Chris Monsere. Contributions include those from Ashley Haire, Chengxin Dai, and Joel Barnett (who did a lot of work doing the final editing of the course design document. This workbook benefited from the collaboration and input of Michael Kyte and Steve Berylein, University of Idaho, Kelly Pitera, University of Washington, Shane Brown, Washington State University, and Ming Lee, University of Alaska. The work was funded by FHWA TDEDP program. All errors and omissions are the responsibility of the primary author. Robert Bertini initiated Portland State's collaboration on this project .

CHAPTER 1

Principles of Scientific Graphical Display

"The only thing worse than one pie chart is two pie charts."

Edward Tufte, Statistician

Transportation professionals need to be able to create effective graphics for communicating technical results in reports and presentations. Like good writing, making good graphics takes effort. However, it is rare to receive any formal instruction at the undergraduate level; instead, the expectation is that you learn how to make good graphs by absorbing examples. Unfortunately, there are many poor examples that violate even the most basic guidelines for producing effective graphics. I know that this has been my experience and I expect that it is of many others.

The purpose of this chapter is to get you thinking about the purpose of graphics and to introduce you to the science behind the effective presentation of data. Though we'll only really be scratching the surface of these ideas, effective graphical representation is an issue that has a recognizable impact on nearly everyone who lives and operates in a data-rich environment. This definitely includes you, so let's get started!

Activity List

Activity Type	Number and Title	Assessment Type
Out-of-class	Activity 1: Principles of Graphics	Short response
Out-of-class	Activity 2: Overview of the Datasets Available	Short quiz
In class	Activity 3: An Experiment in Graphical Perception	Participation
In class	Activity 4: Critiquing a Graphic for Graphicacy	Peer feedback

Student Notes

Principles of Graphics

ACTIVITY 1

The first chapter of Keen's ***Graphics for Statistics and Data Analysis with R*** is an excellent introduction to many of the concepts and principles this course will cover. You may not have thought anything about the graphics you create, but graphics are an excellent way to communicate ideas and concepts and, if not properly executed, can distort or mislead the observer into drawing inaccurate conclusions regarding your message. Even worse, a poorly executed graphic might convey to the reader that the rest of your work is sloppy.

Purpose

This discovery-based activity gives you the opportunity to learn concepts and principles that form the basis of future activities.

Learning Objective

To prepare you for participating in discussions and activities focused on the principles of graphics.

Required Resources

- Chapter 1, Keen pp 3-20

Time Allocated

70 minutes out-of-class

Tasks

Write a short 1-page reflection on something presented in Keen's chapter that you hadn't previously thought much about in terms of preparing graphs or figures. Think about recent figures you have seen or created. Can you identify one or more aspects of what Keen writes about? Don't search for anything just yet; we will get a chance to critique figures made by others in future activities.

Deliverable

Submit your essay to the class dropbox.

Assessment

This is a short response activity. The rubric score is based on quality and thoughtful response.

Activity 1 Grading Rubric

	Excellent (10)	Good (8)	Poor (6)	NONE
Presentation	Good formatting and attention to detail	Decent formatting	Sloppy formatting	Did not submit reflection/discussion
Quality	Language was professional and not complicated	Informal use of language	Difficult to recognize the substance due to poor structure and phrasing	Did not submit reflection/discussion
Thought	Clearly stated reflection with personal formulated commentary or critique of Keen's discussion	Response was a reiteration of the reading assignment points	Discussion was vague and did not specifically address any component of Keen's Discussion	Did not submit 1-page reflection

Student Notes

Overview of the Datasets Available for the Class

ACTIVITY 2

Review the description of the various data sources that we will be using in the class. These descriptions of the data and the table definitions are found on the companion website:

http://www.transportation-data.com/

Purpose

The purpose of this activity is to give you the opportunity to preview the data that we will use later in class.

Learning Objective

Gain familiarity with the datasets what will be used to demonstrate learning objectives in future activities.

Required Resources

- Web browser

Time Allocated

60 minutes out-of-class

Tasks

Read the data summaries and descriptions (meta data) making sure to review the linked documents and explanations.

Deliverable

Complete the short quiz on the class site before coming to the next class session. It is timed; you will have 10 minutes to complete the quiz once you start.

Assessment

Short quiz

Student Notes

An Experiment in Graphical Perception

ACTIVITY 3

Graphical perception is "the visual decoding of information encoded on graphs." As you read in Keen Chapter 1, Cleveland and McGill were leaders in the exploration of graphical perceptions. This activity replicates, to some degree, to one of the original Cleveland and McGill experiments about graphical perception from 1986[1]. A set of basic graphical methods were identified for the perceptual experiment. Presentation of elements such as position scale, direction, area and angle were tested on how accurately quantitative information could be extracted. In this activity, you will be asked to extract quantitative information from graphs and test results will be analyzed in a group setting. In addition to gaining an appreciation for the ability of making graphical judgments, you will be asked to critique the quality of the graphs analyzing the data. Finally, you will see an example of how R can be used to analyze and produce graphs quickly and with surprising depth.

Purpose

The purpose of this activity is to:

- Give you the opportunity to make actual judgments on various graphical elements in a controlled experiment and analyze the results.
- Expose you to the ease with which data can be explored graphically and in depth with R scripts.

Learning Objective

- Add to your conceptual knowledge about graphical perception by making connections between the activity, lecture, and reading.
- Increase your awareness of how your selection of the types of graphs may improve or reduce an audience's ability to make accurate judgments.

Required Resources

- Pencil or pen
- Answer sheet (provided)
- Log sheets (provided)
- Experimental instrument (provided)

Time Allocated

50 minutes in-class

Tasks

A. Introduction

This activity gives you the opportunity to conduct a series of predesigned graphical perception tests and report the statistical analyses of the test results. Each student will answer four sets of graphical perception tests with six plot methods in each set, or 24 plots in total. The six plots are designed to allow you to assess how easy it is to make quantitative judgments about the information presented in graphs. You will be asked to judge graphs that show the following:

- Dot position along a common scale
- Dot position along identical non-aligned scales
- Vertical length
- Horizontal length

[1] Cleveland, William S., and Robert McGill. 1986. "An Experiment in Graphical Perception." *International Journal of Man-Machine Studies* 25 (5) (November): 491–500. doi:10.1016/S0020-7373(86)80019-0.

- Area
- Angle

The original Cleveland and McGill experiment stimuli are shown below:

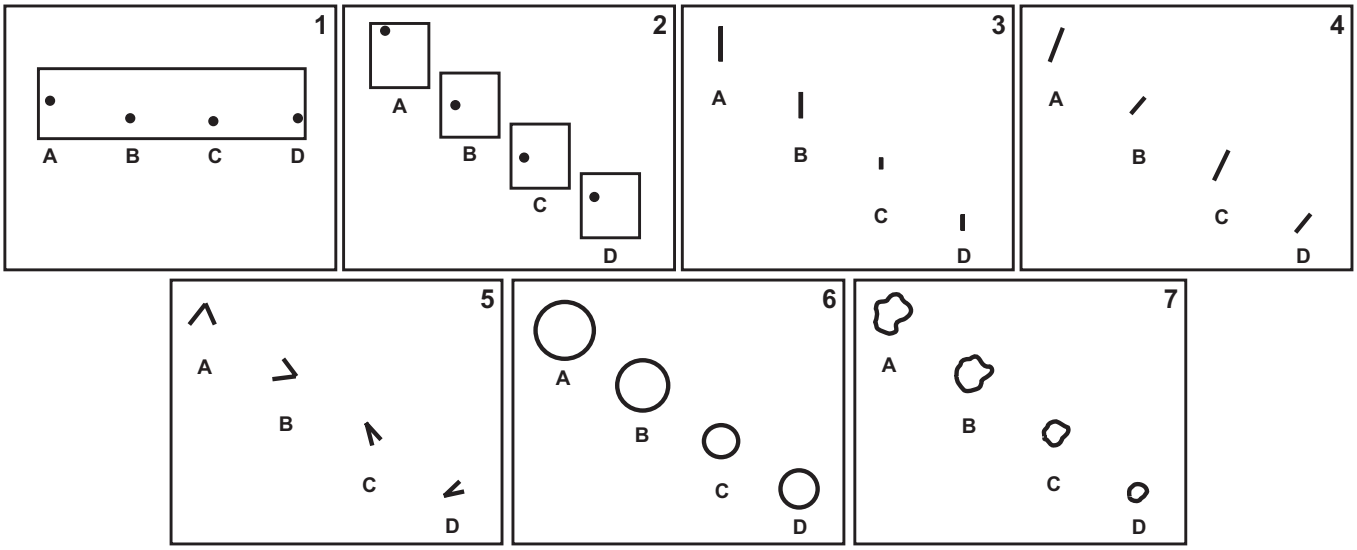


Fig. 2. Stimuli from experiment. An experiment was run to investigate the relative accuracy of basic graphical judgments. The seven types of displays in this figure were judged by subjects. The displays required the following judgments (proceeding from left to right and top to bottom). (1) position along a common scale; (2) postition along identical, non-aligned scales; (3) length, (4) slope; (5) angle; (6) area; (7) area.

Figure 1 Original Cleveland and McGill experiment stimuli (Cleveland and McGill, 1986)

We have created similar instruments and log sheets where you will record your observations. Unlike the original experiment we did not include type 7 for area. The types of plots and the judgments have been randomly generated.

B. Instructions

1. The instructor will assign you a student number. Record this in at the top of the answer sheet for each of the four color-coded logs.

2. The code on the Student Answer Log sheet (11-16, 21-26, 31-36, 41-46) corresponds to the code on the Experiment Instrument, in the lower right-hand corner.

3. For each plot you will make 3 judgments about the relative size of elements B, C, D using A as the benchmark. The scale is not provided to you but the value of A is always 1.0. For each plot code, enter your observations in the column for B, C, and D. So if B is half (½) the size of A in your judgment, you should record "0.5". Alternatively, if B is twice the size of A, you should record "2.0". You can make any marks on the experiment instrument you want but there is no need to spend much time on each set, as the objective is to show how *easily* you can extract quantitative judgments quickly. Enter your answers as shown in Figure 2. Answers for instrument 12 will go in the next column. Answers precise to one decimal place are sufficient.

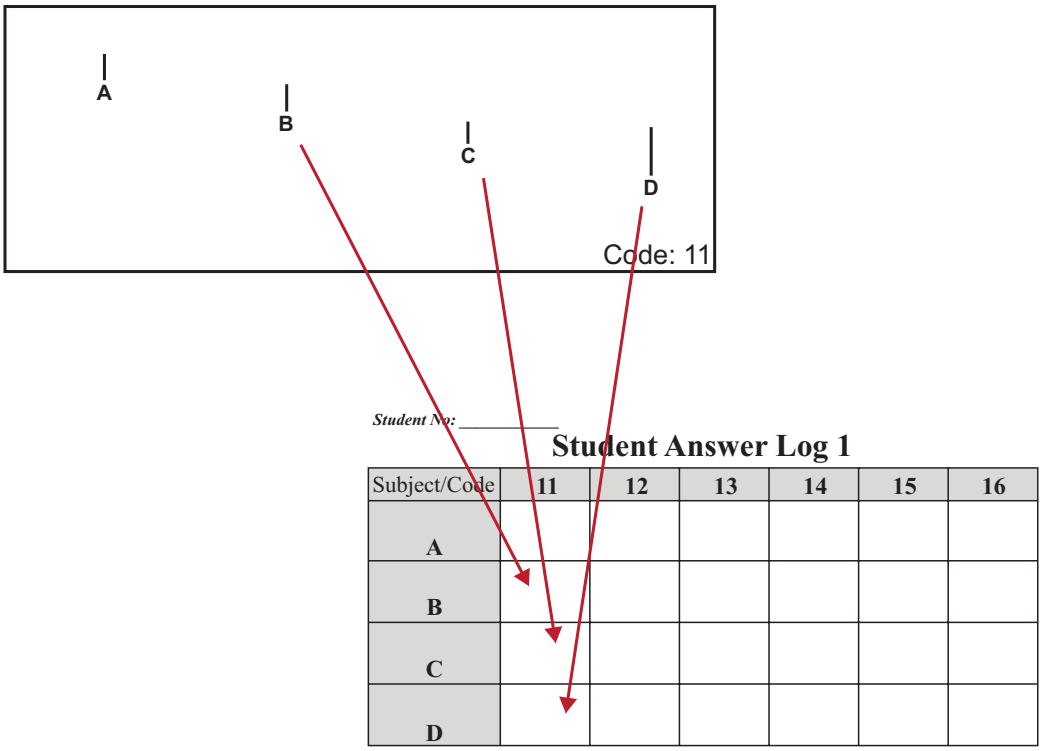


Figure 2 Placement of observation values on the Answer Log

4. This assessment is not concerned with correct answers. Please make reasonably quick judgments. You should be able to complete all four sets of test within 15 minutes in class based on previous experience.
5. Please wait to start until everyone is ready and the instructor announces "start". Enter your answers in each COLUMN for each numbered experimental instrument. When you are finished with the first answer log, place it at the upper right corner of your desk for your instructor to collect. He or she will begin entering these data in the overall class spreadsheet while you continue working.
6. When you are finished and while your instructor is compiling the results, write down a short sentence or two reflecting on the activity on the Activity Worksheet responding to the following questions:
    - Which of the graph types did you find the easiest to make the relative judgments about?
    - On which of the graph types did you think you were most accurate at judging the differences?
7. When you have finished responding to the questions in 6, please read section C below while your instructor is recording all student responses.
8. Please remain quiet until all answer logs are completed at which time you may discuss the activity with fellow students.

C. Data Presentation and Analysis

The observed data will be processed by a previously written R script. The initial exploration of the experimental data will then be displayed to the class. You will be able to see responses by student number but the focus will be on presenting data from the entire class.

In groups of two, you will discuss the plot results and the information that can be gleaned from it.

For each plot, write down your answers to the following questions on the activity sheet (both students should make their own notes of the joint discussion):

1. For which graphical presentation does it appear that most people were able to make the "best" extraction of quantitative information from? The "worst"?
2. Describe how you made this judgment from the plot.
3. Make one additional observation about the data shown in the plot.
4. What information is missing or hard to make inferences about that would be useful to know in making your judgment about the "best" and "worst"?
5. What was/were the good graphical features about the plot(s)?
6. What suggestions could you make regarding improvement of the plot(s)?

Your group will be asked to share your answers with the class. A description of the three data analysis plots are:

- ***Plot 1:*** A set of six plots for each of the graphical judgments. The scatterplots show the observed observation on the y-axis and the true value on the x-axis. An equal line is plotted in dashed dark grey. If all observations are accurate, they will be clustered around this line. Points below the equal line show observations where students overestimated the difference. Two additional pieces of information are annotated on the plots: 1) a least-squares linear regression line and 2) tabulation of the root mean square error (RMSE).
- ***Plot 2:*** A horizontal boxplot of the deviance by each of the graphical judgment types. Boxplots are a plot that you may not have seen yet. In short, the plot is information dense. It shows five descriptive statistics: smallest observation by the left edge of the fence, lower quartile (25^{th} percentile) by the left edge of the box, median (50^{th} percentile) represented by the solid dark black line, upper quartile (75^{th} percentile) the right most edge of the box, and largest observation by the right end of the fence. The width of the box (the spacing between Q1 and Q3) represents dispersion.
- ***Plot 3:*** A set of six plots for each of the graphical judgments that shows boxplots of the deviance values for each student OR a plot of the deviance by true value for each graphical judgment.

Deliverable

Turn in your activity worksheet.

Assessment

This is a participation activity. Your attendance and active participation in the discussion are required to earn all 10 points available for this activity.

Critiquing a Graphic for Graphicacy

ACTIVITY 4

This activity will use an individual task (discussion) and a peer participation task (showing your PowerPoint and graph to the class) to take the past few lessons and readings and apply them in a critiquing exercise. This will help you further develop your understanding of graphicacy in application.

Purpose

The purpose of this activity is to apply the understanding of graphicacy to graphics from an outside source.

Learning Objective

To recognize and discuss the failings as well as the successes in the visual presentation of data from studies in various research communities.

Required Resources

- Web browser
- Microsoft PowerPoint
- Chapter 1, Keen

Time Allocated

30 minutes in-class

Tasks

A. Analysis of Study Generated Graphic

Review the website ***http://flowingdata.com/*** and find one graphic that appeals to you (any). Answer questions 1 c) through g) from the questions at the end of Keen's Chapter 1 for the graph that you have selected.

B. Create a Graphic Rubric

A rubric is a checklist for evaluating the quality of something. It should not be a yes/no checklist, but rather distinguish across a category how well something is. A simple example would be a good peanut butter and jelly sandwich:

Category	Excellent (5 pts)	Good (3 pts)	Poor (1 pts)
Bread	Fresh, chewy, wheat or white bread	Regular white or wheat bread	Stale, or not
Peanut Butter	Crunchy	Smooth	Oily or old
Jelly	Fresh, homemade, strawberry	Natural, other flavors, but still pieces of fruit	Processed goo

Figure 3 Grading rubric for a peanut butter and jelly sandwich

Whether the whole is a graphic, research project, or a sandwich, dissection of the discernible parts is necessary. Each component will have individual aspects that will contribute to the overall acceptability of the total package.

It is important to describe the various levels for each part to assist in understanding the whole within the rubric. If all parts fall within the description of excellent then the sum of the parts is excellent. However, if the peanut butter is processed goo, but the jelly and bread are excellent, the sandwich is neither excellent nor poor, instead residing somewhere between poor and excellent.

Your task is to create a rubric of what makes a good graphic (in this specific case) that you will use to judge the other group's graphics. Do this now.

Deliverable

Prepare a three-slide PPT that summarizes your analysis. Upload your PowerPoint to the class dropbox folder.

Assessment

Tasks A and B: Short Response

Activity 3 Individual Component Rubric

	Excellent (10)	Good (8)	Poor (6)	0
Discussion	Insightful discussion or commentary relating to the question at hand demonstrating student understanding of the task.	Discussion lacked depth for one or two of the six questions, whereas the remaining parts had discussion that was thoughtful.	The discussion as a whole lacked depth demonstrating thoughtful application of the graphicacy.	Did not participate.
Quality	Document is typed, formatted, contains appropriate grammar and language.	Document has minor grammatical errors or inappropriate language.	Document was unorganized, contained inappropriate language and/or grammatical errors.	Did not participate.

Be prepared to share your PPT in a class lecture.

CHAPTER 2 Getting Started with Data

"Why are things as they are and not otherwise?"
Johannes Kepler (1571-1630) German astronomer

Large complex data sets can be challenging to analyze. Enormous amounts of data are being collected, stored, and analyzed in an ever increasing array of fields. In transportation, this evolution is no different. As vehicles increasingly become probes — connecting subsecond vehicle pulses of data containing position, speed, throttle, brake, and other monitoring information — operational data will only expand. On the planning side, the use of smartphones and other devices constantly reporting position and travel patterns will be invaluable sources of data to better plan, operate, and design our cities and rural environments. Even seemingly ubiquitous data streams such as information from traffic signals, freeway sensors, and bus fleet information produce very large datasets in a short amount of time. These large datasets quickly exceed the capacity of traditional analysis tools that most engineering students are very familiar with, such as Excel.

The purpose of this chapter is to introduce the concepts of data, databases, and structured query language (SQL). The chapter begins by using a very familiar data, in Excel, to demonstrate some very basic concepts of data. From this strawman, students are introduced to an open source database: PostgreSQL. Students are introduced to data types, tables, schemas and then create their first database. Finally, students are challenged to attempt the use of SQL language in order to generate simple queries that will assist with future activities.

Activity List

Activity Type	Number and Title	Assessment Type
Out of class	Activity 5: Setting Up Your Accounts	Participation
In class	Activity 6: Creating a Simple Database in Excel	Short response
Out of class	Activity 7: Overview of SQL	Short Response
In class	Activity 8: Creating a Simple Database in PostgreSQL	Short Response
In class	Activity 9: Writing and Testing Some Simple SQL	Participation

Student Notes

Setting up Your Accounts

ACTIVITY 5

In order to complete the remaining activities in this book, you will need to make sure the relevant computer accounts are active and functioning. Which accounts you may need will depend on how your instructor has decided to provide you access to the class data sets. Most of the scripts provided to work. As a registered student, you should already have a PSU ODIN account but you may need to setup your engineering accounts.

Purpose

The purpose of this activity is to set up your required computer accounts so that you can access software and tools.

Learning Objective

Become familiar with needed student accounts.

Required Resources

- Additional instructions from course instructor

Time Allocated

15 minutes, to be completed prior to first class meeting.

Tasks

Account Setup

A. If needed, confirm access to your course management software. In the remaining activities this is referred to as online course management software.

B. If needed, setup your accounts to access the computers in the teaching laboratories. Confirm that you can log in to these computers.

C. If needed, create and setup your PostgreSQL account:

   - If your instructor has created a local PostgreSQL database, you will need an account to do this. After you have created, test logging in to your database by connecting.

Deliverable

Login to course management site and complete the survey for Activity 5.

Assessment

Participation points (10) for completing survey

Student Notes

Creating a Simple Database: An Excel Strawman

ACTIVITY 6

Most students are familiar with an Excel spreadsheet. Data in a spreadsheet – in the form of rows and columns – are a simple form of a database. This activity aims to expose students to database concepts using this as a strawman. Since Excel format is binary and proprietary, data cannot be read without Excel or other software. The comma separated values (CSV) file is a more common way that spreadsheet-like data can be exchanged or provided between source and analysis program. Since it is not formatted for a specific software type and is ASCII text file, it can be read by most software. In this file format, the comma separates values between columns of data. The first row is typically, but not always the names of the columns. The comma is a called a "delimiter". Note that characters other than commas can be used as a delimiter such as ";" or "|" or other text characters.

In this activity you are given two files that are excerpts from TriMet's stop-level bus AVL system data. Don't worry too much about what data these files represent, we will work on that later. They have been edited for demonstration and learning purposes. They are:

- **stop_level.csv** (sample of TriMet's Bus Dispatch System data)
- **stop_names.csv** (from the RLIS regional database this is table information about stops on the TriMet system)

Purpose

The purpose of this activity is to expose the student to a simple model of relational database using Excel as the strawman.

Learning Objective

- Using Excel as a simple model of database, be able to define tables, columns, rows, data types, and do simple joins.
- Using filters and joins, answer a set of simple count-type questions.

Required Resources

- Microsoft Excel
- Files posted on class site: **stop_level.csv** & **stop_names.csv**
- TriMet metadata description posted on the class website

Time Allocated

90 minutes in class

Tasks

A. Working with CSV files

Retrieve the **stop_names.csv** from the class web site and open the csv file in a text editor, such as Notepad or Textpad. A good text editor is going to be very useful. Notepad or Wordpad has no tools or options and has a row limitation (on the number of records it can read in). My suggestion is to use Textpad. You can find it under "General Applications".

1. How many rows of data do there appear to be (Hint: use the status bar in Textpad)?

Excel can also easily read the csv files into a worksheet:

- Open a blank workbook in Excel. Create 2 tabs, stops and stop_names.
- To read the files into Excel there are a couple of options:
  1) go to the ***Data*** tab and ***Get External Data***, select ***From Text*** and follow the prompts.
  2) select the data from the text file, paste in Excel, then use the ***Data → Text to Columns*** option.

  Feel free to try both.

B. A Simple Database

A relational database allows data to be stored in a more compact format and can be a useful in doing analysis. The most common relationship is what is called a *one-to-many* relationship.

An easy way to think of this is represented below. In one table, a column has entries in each cell for data elements. The data elements in the column are often repeated. In the example below, the table SALES is a log of all customer transactions, the field CUSTOMERID can be related to another TABLE (CUSTOMER) that contains all the information about the customer. In this way, there is no need to repetitively store all of the information about each customer. Rather, this information can be extracted at any time by joining (i.e., linking) the two tables on CUSTOMERID column.

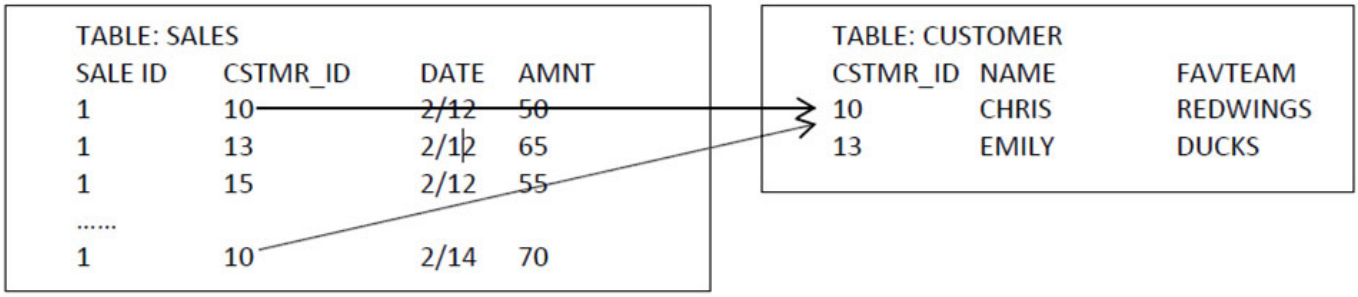


Figure 4 Sample Database

Let's try a simple example of linking the two tables you have in Excel. First, look at the two tabs.

2. Which column is the obvious one to make the join?

Excel really isn't a database, though it has some functionality that is similar to a database. Note: Microsoft's desktop database is Access. One way to make the join, is to use the ***VLOOKUP*** function. TIP: Use the function insert tool in Excel to help you learn about ***VLOOKUP*** (see the arrow pointing to it in Figure 5).

Produce a third table that has the one column from the main stop table and all of the columns from the joined table.

It should be clear what fields do this "join" on

Now, let's take a minute to explore the "stops_name" tab in the spreadsheet. In Figure 4, the table "Customers" is like the "stops" data. It contains

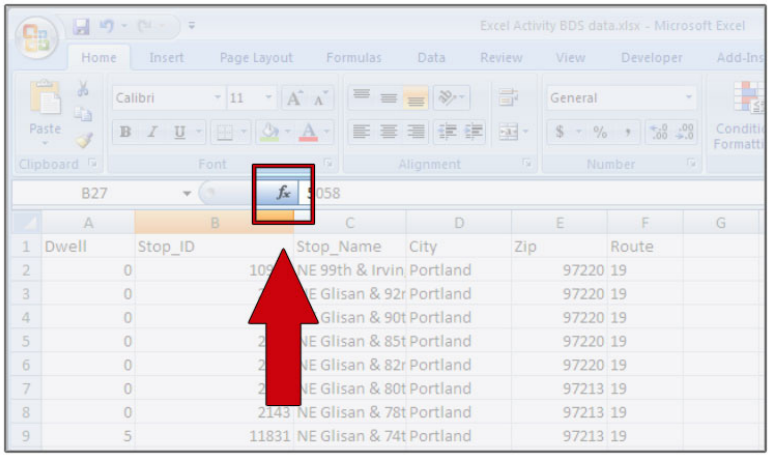


Figure 5 Excel Screen Capture of Stop Table

information about each stop. There is one important difference about the stops table

3. What is it?
4. Now look at the "joined" table you created. What looks wrong about this? Hint: look at the Rte_No in the stop level data. How would you fix this?

Export this tab to a CSV format. Before you do that, save the workbook that you are currently working in. To export, use the ***FILE*** → ***SAVE AS*** and select other format (see Figure 6).

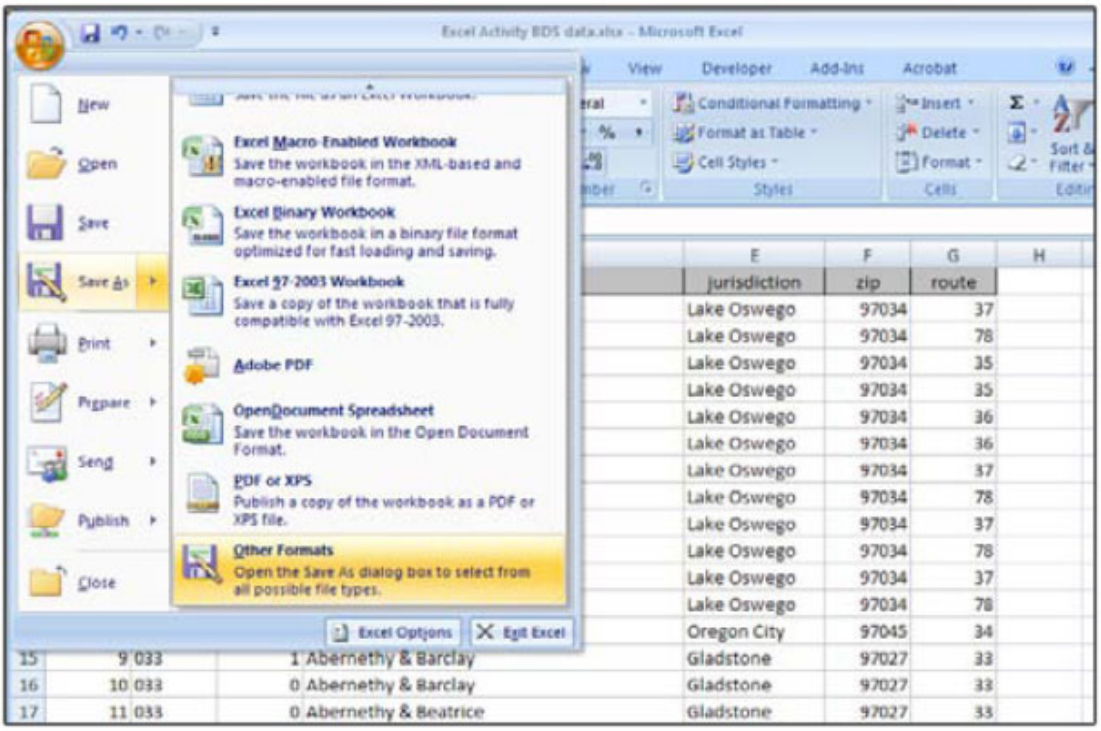


Figure 6 Exporting the Excel file to a CSV format

C. Tabulating Some Answers

Let's explore the Excel "database" in a little more detail. Use the filter option in the SORT & FILTER tools to answer these following questions.

5. How many stops were made at STOP_ID 805?
6. How many stops were made at STOP_ID 2001?
7. How many stops were made where the STOP_ID was greater than 1500 in the table **stop_level**?
8. How many stops were made at time points (use the TriMet data dictionary ***http://www.gcu.pdx.edu/data/dictionary.htm***)?
9. How many routes serve Cornelius?
10. Which routes serve the intersection of W Arlington & Barton?
11. What is the total amount of dwell time that occurred with a stop on Glisan? (*Hint: use text filters for cells that contain Glisan.*)

Deliverable

Provide a short-typed text or Word document with your answers to questions 1-11 due at the end of this class session. Upload a PDF of the document to the class dropbox

Assessment

Participation for this activity is based on the submission of answers and text files.

Student Notes

Overview of SQL

ACTIVITY 7

To prepare for the upcoming activities related to SQL it is helpful to do a little reading about the history and general syntax of the language in advance. We will spend time writing queries so you should read this material with balance between understanding and knowing where to find answers to questions regarding syntax that will arise in the near future.

Purpose

The purpose of this activity is to provide you with an overview of the SQL Language

Learning Objective

To make a first attempt at "pseudo code"

Required Resources

- Access to a web browser

Time Allocated

30 minutes out-of-class

Tasks

A. Read Wikipedia article

The article available at: ***http://en.wikipedia.org/wiki/SQL*** gives a decent background and explanation of what SQL is and how you will be using it to query data.

B. Read (the entire text)

Available at PostgreSQL: ***http://www.postgresql.org/docs/9.1/static/tutorial-sql.html***

C. Skim the section on SQL

Skim the next chapter: ***http://www.postgresql.org/docs/9.1/static/sql.html***. Pay particular attention to section 8.1 on datatypes.

D. Write pseudo query syntax

Before getting stuck in the language syntax it is helpful to write the logic of the query that you might attempt in simple English. For example, for question 5 from Activity 6: "How many stops were made at STOP_ID 805?" you might write:

- Select the rows (records) from the stop table where the STOP_ID field value is equal to 805.
- Count the number of rows selected.

Now write the logic for questions 6-11 from Activity 6.

Deliverable

Submit Logic Task D in the course dropbox.

Assessment

Activity 7 Grading Rubric

	Excellent (10)	Good (8)	Poor (6)	0
Logic	Logic is sound. A step-by-step process is presented as a sequential series of processes resulting in the solution.	Logic is sound for most answers. A step-by-step process is presented in most cases and is a sequential series of processes.	Logic is difficult to understand. The processes were not presented in sequential or comprehensible order.	Did not submit.

Student Notes

Creating a Simple Database: Now with PostgreSQL

ACTIVITY 8

We are going to do the simple exercise of creating a database, then uploading the TriMet files from Activity 6. In the next activity, you will use SQL to answer the same questions you did with Excel about the TriMet data. Expect some frustration today in getting things to work. This is **normal**. There are known errors in the Excel file that will cause upload errors. One of the outcomes of this class is to help you overcome the learning curve and begin solving problems. Pay attention to syntax (i.e., upper/lower case letters, spelling, commas, quotes, spaces, etc.). Issuing commands to a computer by lines is always a little time consuming until you get the hang of a particular program. When something doesn't work, there is always a reason. It takes time, but it will be helpful if you learn to recognize error messages and work backwards to solve problems. Look for inconsistencies in something (e.g., Is that a word in my numeric field?). If I get really stuck, I always find that doing something incrementally helps. I start with something I know works first then start adding or modifying things until I find out what the problem is.

Purpose

The purpose of this activity is to give you the opportunity to learn how to set up a database and upload data.

Learning Objective

- Transfer the Excel model of a simple database in Activity #2 to PostgreSQL. Learn to define tables, columns, rows, data types in the phpPgAdmin.
- Learn to navigate the phpPgAdmin tool.

Required Resources

- Web browser
- Files posted on class web site for Activity 6: **stop_level.csv** & **stop_names.csv**
- TriMet metadata description posted on the class website
- PostgreSQL data definitions: ***http://www.postgresql.org/docs/8.1/static/datatype.html***

Time Allocated

60 minutes in class

Tasks

A. Login to phpPgAdmin

Creating the database and table, managing data, uploading files, and other tasks to the database can be done many ways. We are going to start by using the web browser tool phpPgAdmin.

Login to your database: ***https://cat.pdx.edu/phpPgAdmin/***

You should see something similar to Figure 7:

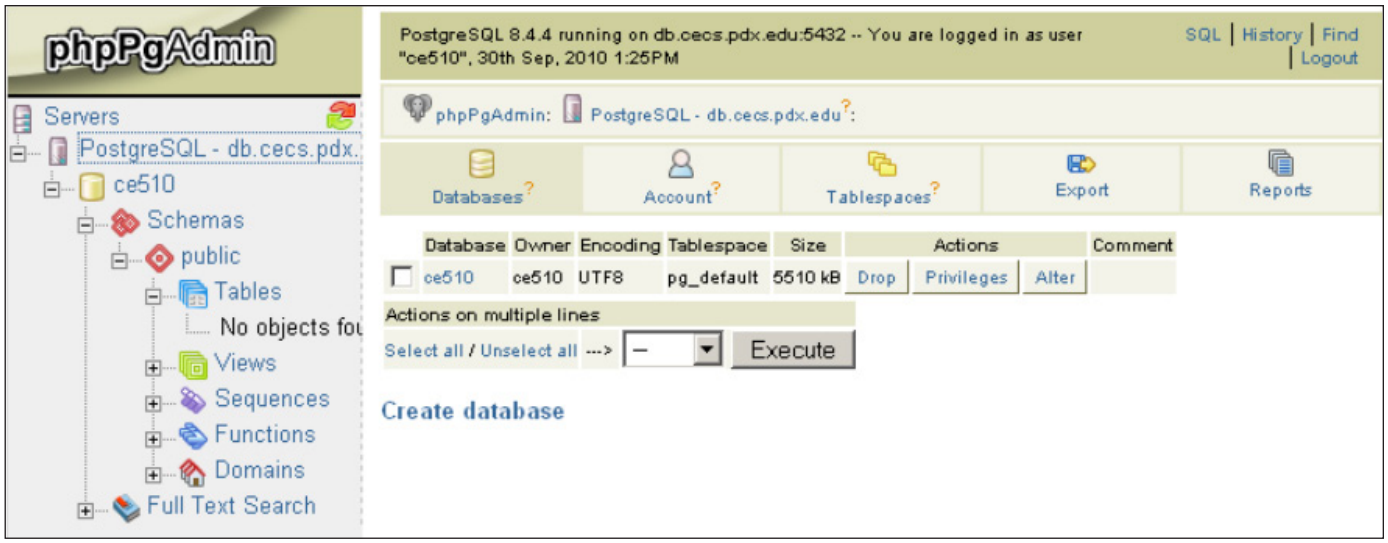


Figure 7 Screen capture of the phpPgAdmin login screen.

This interface allows you to do some point/click navigating of the PostgreSQL database. You can also issue commands directly with the SQL interface. Spend a few minutes navigating this interface. When you are comfortable clicking around the software, let's get started.

B. Select Your Database

Select your username listed under either "Database" in the mainframe, or under PostgreSQL – db.cecs.pdx.edu in the navigation window. You should see something similar to Figure 8 where the username is "**ce510**":

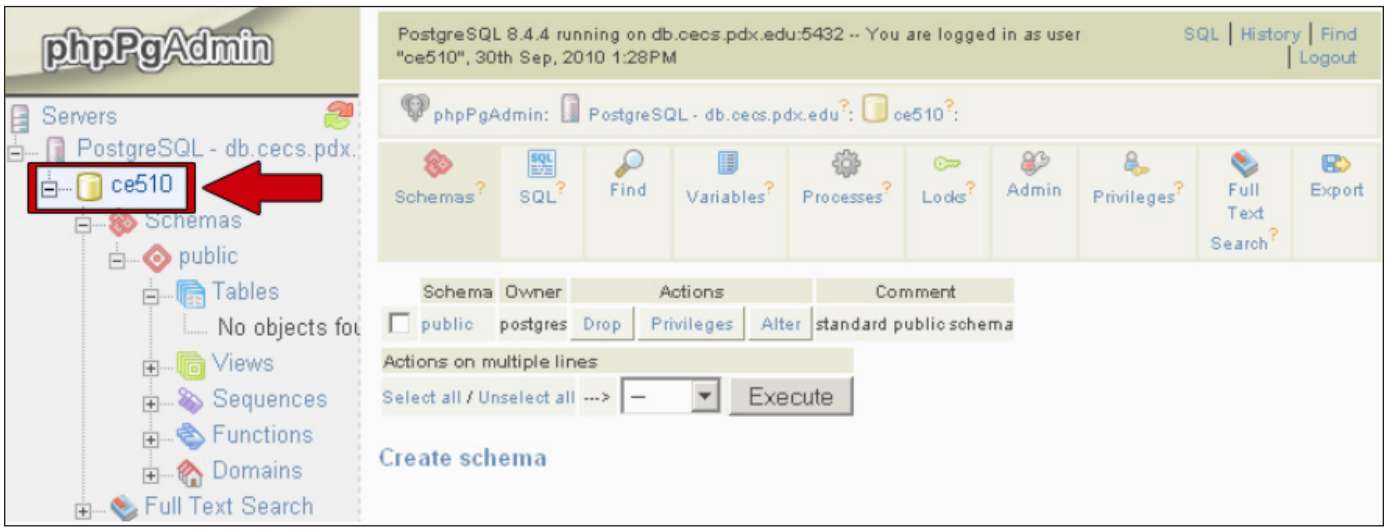


Figure 8 Screen capture of phpPgAdmin database screen

C. Create A Schema

While not necessary or required, it is helpful to organize your database with a "filing" system. This filing system is called a *schema*.

Schemas are like creating folders in your operating system and help you to keep things better organized. In the PostgreSQL manual it says schemas are "are analogous to directories at the operating system level, except that schemas cannot be nested."

PostgreSQL has a default schema called "public". We could use that, but instead let's create a schema transit.

First, click on 

Then select Create Schema, type the name (transit), then ***CREATE***. You should see on the left, under the public Schema that there is a new one called transit.

Click on the transit schema.

The syntax to refer to objects/tables/etc. that are filed under this schema is **`transit.XXXX`** where **`XXXX`** is the name of the object (e.g., table) we are referring to in SQL.

D. Type of Data

An important element in understanding data is to be able to define the types of data that is stored in each column. We will use the rather detailed PostgreSQL data definitions which can be found here: ***http://www.postgresql.org/docs/8.1/static/datatype.html***.

Browse the table of contents, each major heading describes the general types. We will be most interested in:

- 8.1. Numeric Types
- 8.3. Character Types
- 8.5. Date/Time Types
- 8.6. Boolean Type

E. Create Table

Read each of these subheadings to get a feel for which data types can be defined. It can be rather tedious to go ahead and create all of these data types for all the fields by hand. In defining data types, as a general rule of thumb you should try to select the "smallest" data type that will work. For example, if an *int* (4 byte integer) definition will work, use it. Don't select *bigint* (8 byte *int*) since in some data elements, the larger types take more disk space.

The primary components of the database are tables. You might want to think of tables as tabs in Excel spreadsheets. Data are organized in columns (that are named) and rows (called records). Let's go ahead and use the interface to create a table for **stop_names**. Be sure that you are in the schema *transit*.

Click on the ***Create Table*** (lower left of the screen) and follow the prompts.

Be careful not to specify a ***LENGTH*** for data types that don't have this option. Refer to Table 8.1. Only those data types with [ ] in their description can have length or other options specified.

Browse the table. Notice that all column names are forced to lower case.

Notice the icons across the top (Columns, Indexes, Contraints, etc.—see Figure 9). We will come back to these later. Notice the buttons ***BROWSE***, ***ALTER***, ***DROP***.

BROWSE lets you look at the data, ***ALTER*** lets you change properties about the column, and ***DROP*** removes it.

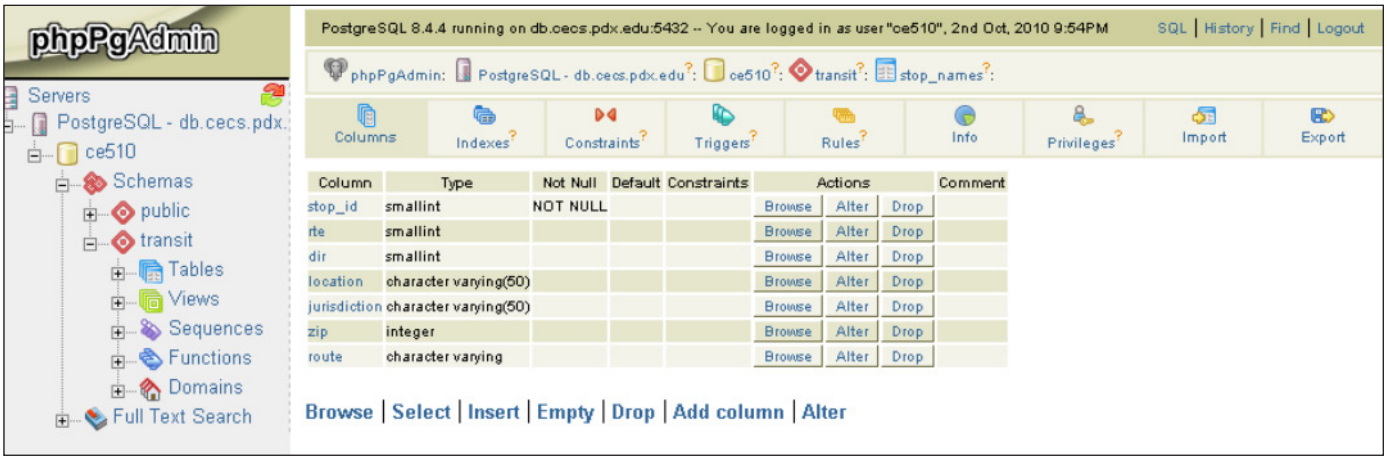


Figure 9 Screen Capture of the phpPgAdmin add table screen

F. Import The Data

First, let's insert the first row of data from the CSV table manually. Click the ***INSERT*** option. Follow the prompts. Now use the ***BROWSE*** link to see the data you inserted.

Before we import data, we need to clear the data you just added to the database. Use the EMPTY link to purge all rows of the database (only the one row you created).

Now we are going to import the CSV file into the database. A good text editor is going to be very useful. My suggestion is to use Textpad. You can find it under "General Applications".

First, use the Textpad editor to verify the first row of columns names in the CSV file match **EXACTLY** the names of the columns your created. Change either the text file or ***ALTER*** the table to make the column names match.

It is good practice to use all lowercase for column names in SQL databases. So, while we are in the CSV, be sure to change the column names to lowercase. Textpad has a handy feature for making this conversion easy in the ***Edit*** menu. Save your edits. You can leave the file open in Textpad; the application doesn't lock the file from being read by other software, which is a nice feature!.

Select the Import icon to upload the CSV file in the database: 

You can leave the Format as ***AUTO*** and leave all other options as default. Navigate to the CSV file and click on ***import***.

G. Debugging

Error messages are going to be a fact life when learning new computer programs, tools or writing algorithms yourself. If you received an error message, in most cases a careful inspection of the error message can give you a clue to what might have happened. The import runs a script that reads each line of the CSV then executes a ***SQL INSERT*** statement to the put the data in the database. phpPgAdmin echoes the failed statement to the browser and tells you in blue which line failed to insert.

Therefore, if you get a statement back you know the program at least read the file. The error message will say something. In this case it gives you the row number where the data upload failed. Look at that row in the CSV that contains the error and ask yourself if something is different. What data mismatch error could there be?

The problem almost always is some data mismatch, either by the type you defined or a rogue data element in the file you are importing. This can be fixed by altering the column data types or by changing the data element.

Also, sometimes it is helpful to look at options for executing a function. Here, some of the upload problems can be solved by changing the import options from the default.

H. Create and Import the Data for table stop_level

Open the SQL code editor by clicking on the icon 

Copy the SQL code into the code editor and choose execute, creating the table **stop_level**. To import the **stop_level** data, import the data the same way as the **stop_names** table.

Deliverable

None; you will need the tables you inserted for the next activity.

Assessment

Participation

Student Notes

Simple SQL

ACTIVITY 9

We are going to "complete the loop" of the Excel-to-PostgreSQL model. In this activity, you will write SQL queries to retrieve the answers to the same questions asked of you in Activity 6. (You will be using the TriMet data you uploaded, also as part of Activity 6.) We will work through examples.

Purpose

The purpose of this activity is to help you learn how to do simple SQL queries in PostgreSQL using the phpPgAdmin interface.

Learning Objective

- Using SQL queries and joins, answer a set of simple count type questions.
- Make the connection between using this approach and the Excel model.

Required Resources

- Class introductory notes about using a database (will be passed out and reviewed by instructor)
- Web browser

Time Allocated

50 minutes in class

Tasks

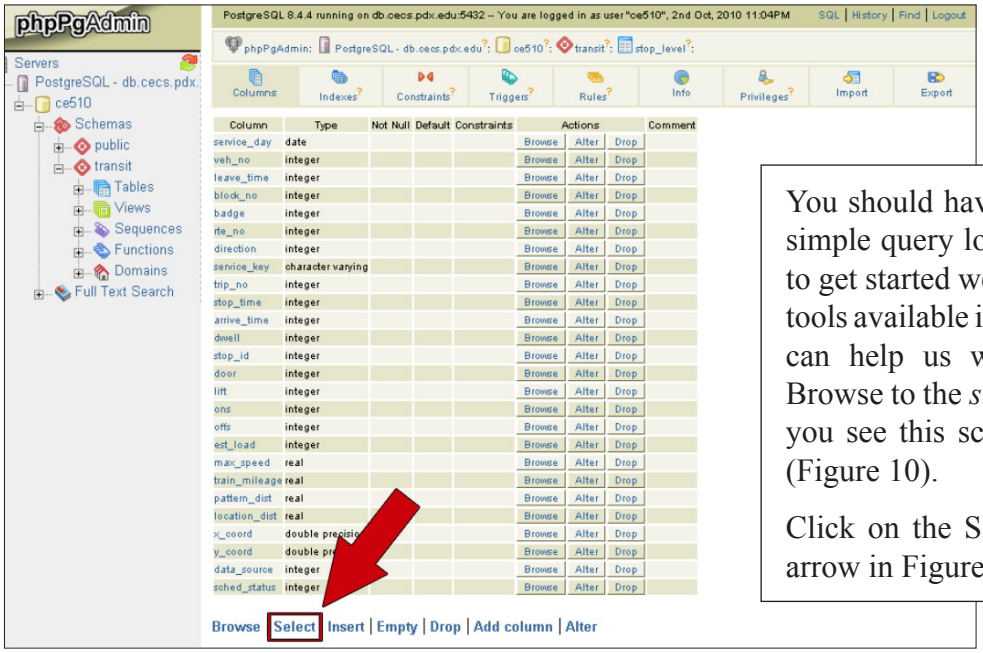
A. Login to phpPgAdmin

Login to your database and select your database.

B. Run Some Sample Queries

Let's try to answer this question from Activity 6:

1. How many stops were made at STOP_ID 805?



You should have already written the simple query logic in Activity 7, but to get started we will use some of the tools available in phpPgAdmin which can help us write simple queries. Browse to the *stop_level* table so that you see this screen in phpPgAdmin (Figure 10).

Click on the SELECT link (see the arrow in Figure 10, at left).

Figure 10 phpPGAdmin table browser screen capture

You should be taken to a "query" form. You can use this form to ask simple queries of the database. Note all of the different operators for the logical selection criteria. Take a minute to see if you can find a description of the logical operators in the PostgreSQL help manual. When you do, you should bookmark the page. Now, enter 805 in the *stop_id* field and put a check box next to the columns you want returned in the query. Note that you can select all fields in the lower left corner.

The number of rows returned should be the same as in Activity 6, when you used Excel.

You can also enter more complicated queries in the SQL window. Click on the Edit SQL link. You should see the following window (Figure 11):

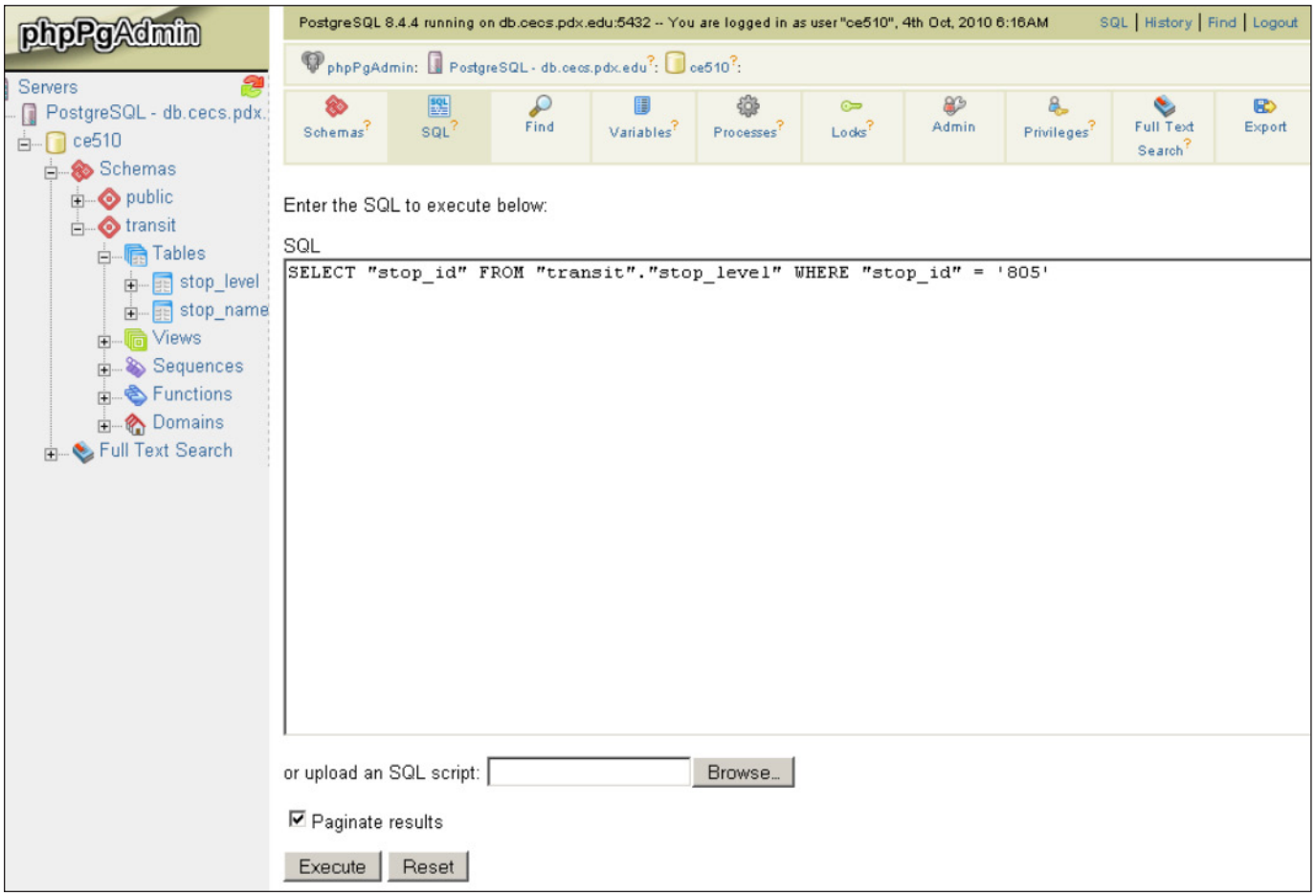


Figure 11 phpPGAdmin SQLwindow screen capture

Inspect the statement so that you understand the logic and structure behind it. Note that the columns you requested are listed after the ***SELECT*** statement, the ***FROM*** includes the schema (*transit.stop_level*), and the ***WHERE*** condition lists the logical filter. Now let's use the SQL window to answer the second question:

2. How many stops were made at STOP_ID 2001?

Edit the SQL statement to modify the necessary criteria, then click the ***EXECUTE*** button. Note that if you click the ***DOWNLOAD*** link, you are given the option to return the query you just executed in CSV format.

C. Using Queries

Now duplicate the remaining "queries" that were done in Excel "database". For each of these questions, include the SQL statement and the result.

3. How many stops were made where the STOP_ID was greater than 1500 in table *stop_level*?
4. How many stops were made at time points in table *stop_level*?
5. How many routes serve Cornelius in table *stop_names*?
6. Which routes serve the intersection of W Arlington & Barton?
7. Do the JOIN between the tables. Output the CSV to a file.
8. What is the total amount of dwell time that occurred with a stop on Glisan *(Hint: use an aggregate operator to return this answer to you directly)*.

Deliverable

Provide a typed answer to all questions. Include discussions and show comparisons to Excel work if needed. Submit to Dropbox.

Assessment

This activity is a short response activity. The score that you will be given is based on the quality and depth of discussion. The response expected differs by question as described in the rubric below:

Activity 9 Grading Rubric

	Excellent (10)	Good (8)	Poor (6)	NONE
Discussion	Insightful discussion or commentary relating to the question at hand demonstrating student understanding of the task.	Discussion was competent regarding the lessons demonstrated by a correct response, but lacked discussion other than the obtained value.	The discussion did not address the lessons or applicability of the activity.	Did not submit
Comparison	If a difference in results is noted then the student successfully demonstrated and discussed the differing results between PostgreSQL and Excel	Comparison lacked specific discussion regarding the possible reasons for the results.	No comparison was included even though a difference in results was present.	Did not submit
Quality	Document is typed, formatted, contains appropriate grammar and language.	Document has minor grammatical errors or inappropriate language.	Document was unorganized, contained inappropriate language and/or grammatical errors.	Did not submit

Student Notes

CHAPTER 3 Introduction to R

"To some people R is just the 18th letter of the alphabet. To others, it's the rating on racy movies, a measure of an attic's insulation or what pirates in movies say."

Ashlee Vance, NY Times "Data Analysts Captivated by R's Power" Jan 6, 2009

R is a diverse open source application that is supported by a strong community of users, including many who create packages to assist other users or simplify tasks in processing and analyzing data. The sheer number of packages can make finding the right functionality a daunting task. However, if the right package is available, the power of R, mainly, the statistical and data visualization, will shine through.

As with many other complex applications, R is known for its steep learning curve, as such time is required to understand the intricacies of the flexible, script-based analysis. The ability of R to generate robust graphics is founded in the user-established arguments within R's plotting function and available packages. This means that rough graphics, not of presentational quality, are easy to develop, but polished graphics take effort.

The purpose of this chapter is to provide an introduction to R and building graphics. This chapter starts with an introduction to R Studio and basic R language discussions to develop the syntax and language knowledge required for further exploration of the class data sets. After developing a familiarity with the language used in R, plotting and the variability of the plot options are explored, including the functions involved with organizing and analyzing data. Finally, the chapter concludes with establishing a connection to the class databases to further explore the statistical analysis abilities of R.

Activity List

Activity Type	Number and Title	Assessment Type
Out-of-Class	Activity 10: An Introduction to R and R Studio	Short quiz
In Class	Activity 11: Setting Up R	Participation
In Class	Activity 12: A Starting Point – Some Simple R	Annotated Script
In Class	Activity 13: Reading in Data Files	Annotated Script
Out-of-Class	Activity 14: R Plots	Short quiz
In Class	Activity 15: Learning Some Simple Plotting Features of R	Annotated Script
In Class	Activity 16: Your First Advanced Plot	Short Response
In Class	Activity 17: Code Sharing	Peer Assessment
In Class	Activity 18: Thinking Like A Computer – Psuedo-Coding	

Activity Type	Number and Title	Assessment Type
I/O Class	Activity 19: Write Your Own Function	
In Class	Activity 20: Connecting to the Class Database via RODBC & PostgreSQL	Participation
In Class	Activity 21: Some Advanced R	
I/O Class	Activity 22: Packages	Short Response
Out-of-class	Activity 23: Working with Time in R	Discovery

Student Notes

An Introduction to R and R Studio

ACTIVITY 10

Before we get started with R, it would be good to read a little about some of the details of R and the program we will use to interface with R: **R Studio**. Don't worry about trying out any code yet; the objective here is to orient yourself in advance of the class.

Purpose

The purpose of this activity is to provide you with an opportunity to preview some details of R and R Studio.

Learning Objective

To begin to learn about R and R Studio.

Required Resources

- Chapters 1 and 2 in *Introductory statistics with R* (Dalgaard)

Time Allocated

50 minutes out-of-class

Tasks

A. Read Chapters 1 and 2 in Dalgaard, pgs 1-53

B. Open a web browser and open the R Studio home page for the integrated development environment (IDE) at ***http://www.rstudio.com/ide/***. Watch the screencast (you will need to be at a computer with speakers or headphones).

After this screencast, browse the documentation ***http://www.rstudio.com/ide/docs/***. Browse and read all of the topics listed under the "Using RStudio" heading.

Deliverable

Complete the short quiz on the class course management site before coming to the next class session. If your instructor is not using the online learning environment, complete the quiz posted on the companion website for the course.

Assessment

Short quiz

Student Notes

Setting up R

ACTIVITY 11

Purpose

The purpose of this activity is give you the opportunity to set up R for use in class.

Learning Objective

To gain familiarity with the operation of R Studio.

Required Resources

- R Studio

Time Allocated

30 minutes in class

Tasks

A. R Studio

Start R Studio on your computer. Depending on your system, the course instructor may have separate instructions needed to configure the program to work. Now, spend a few minutes browsing the menu items and the icons. Note that if you hover the mouse over an icon a tool tip will appear, explaining the function of that icon. You will use many of these shortcuts.

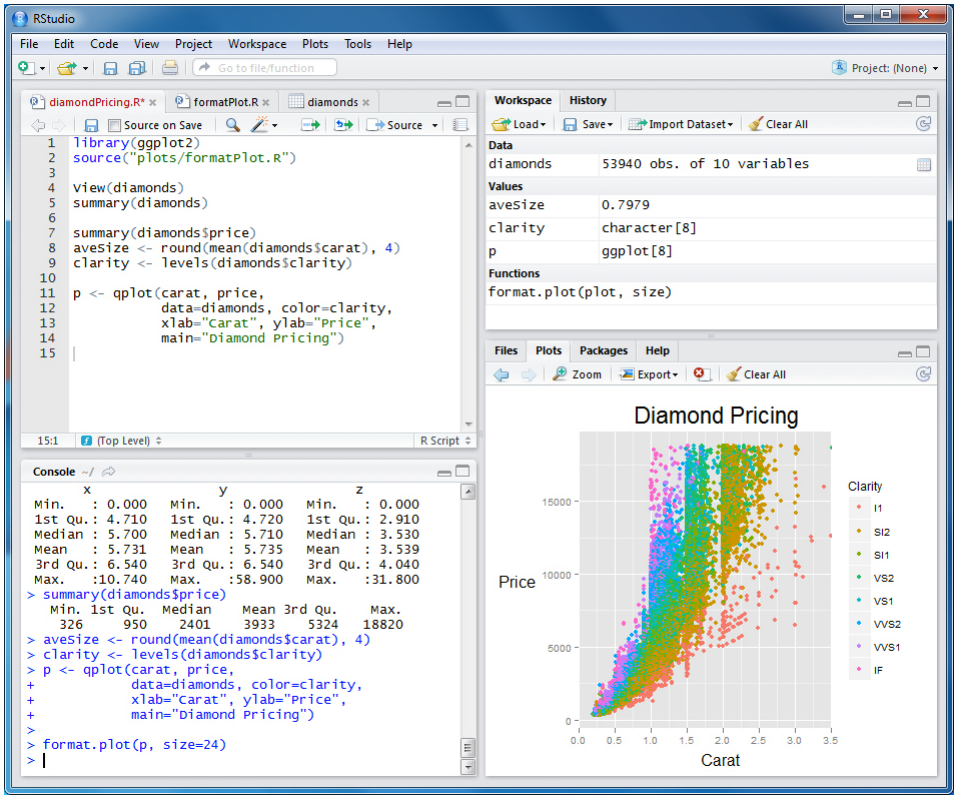


Figure 12 R Studio Interface

Deliverable

Complete the checklist indicating that you have explored the various setup options (Figure 12) in R Studio.

- General
  - R-Version is 2.12.2
  - Initial working directory is set
  - Establish saving preferences
  - Establish CRAN Mirror (Oregon State)
- Editing
  - Highlight selected word
  - Highlight selected line
  - Tab spacing
  - Show margin
- Editing (con't)
  - Auto-insert matching parens/quotes
  - Soft-wrap R source files
  - Show syntax highlighting in console input
- Appearance
  - Font size
  - Editor theme
- Pane Layout

Assessment

Participation points (10)

All complete	MIssing 1 – 2 options	> than 1 – 2 options not explored
10 points	9 points	5 points

A Starting Point: Some Simple R

ACTIVITY 12

We are going to do some simple things to learn about R. This will be a guided exercise – meaning that the instructor will demonstrate the steps on the screen while you follow along on your desktop.

Purpose

The purpose of this activity is to give you the opportunity to become familiar with the R interface and syntax.

Learning Objective

To be able to use R.

Required Resources

- R, R Studio
- Chapters 1 & 2 in *Introductory statistics with R* (Dalgaard)
- R Script for this activity from class companion website

Time Allocated

60 minutes in class

Tasks

This activity begins assuming R Studio is open.
If not, please start R Studio and open the sample script for the activity (if provided)

A. Concept: Storing Values or Objects and Variables

At the R console prompt, type:

```
1+1
```

It returns:

```
[1] 2
```

The [1] identifies the index of the number on that line of the answer. It is not that helpful here, but there will be more on that later.

In R, you can assign values to variables. The assignment operator is <- is the operator that "assigns" the results of the operation to the right to the variable to the left.

You can use any text you want to create variables, though it must not start with a number or a dot followed by a number.

IMPORTANT: R syntax is case-sensitive. `Y` is **not** the same variable as `y`. Type:

```
y <-1+1
```

Notice now that R just returns you to an empty command prompt. But the value of operation 1+1 has been stored as the variable "y" in R's memory. This only lasts until you clear the memory, or end the session. You will see later that there is a lot of flexibility in assigning many objects to variables, including entire data frames.

Type `y` at the command prompt to retrieve the value of y. Notice that you get the same output as before:

```
> y<- 1+1
> y
[1] 2
```

TIP: name your variables in one case (upper or lower) and try to use short but memorable names. If you want to use more than one word combine them with a . or _ character. Names like `my.variable` or `my_variable` are examples. Don't use spaces as it will only cause you trouble. Again, variables cannot begin with a number.

1. Now, you try to assign another variable the value of `y+y`. What should be the value of that variable?

```
> x<- y+y
> x
[1] 4
```

Type `ls()` at the R prompt. This shows you a list of objects in memory. Until you reassign or clear the memory, R will remember the values of these variables.

```
ls()
[1] "x" "y"
```

You can also see these two objects shown in the workspace tab of R Studio. This is a very helpful window since it shows you the current objects in memory and their values.

To clear objects in memory, type

```
rm(list=ls(all=TRUE))
```

2. Does R still know what value of `x` you assigned to the variable you created to store `y+y`?

Now let's go ahead and create a script file that you can log your program in. Choose ***File > New > Script File***. Select the file you downloaded from the class website.

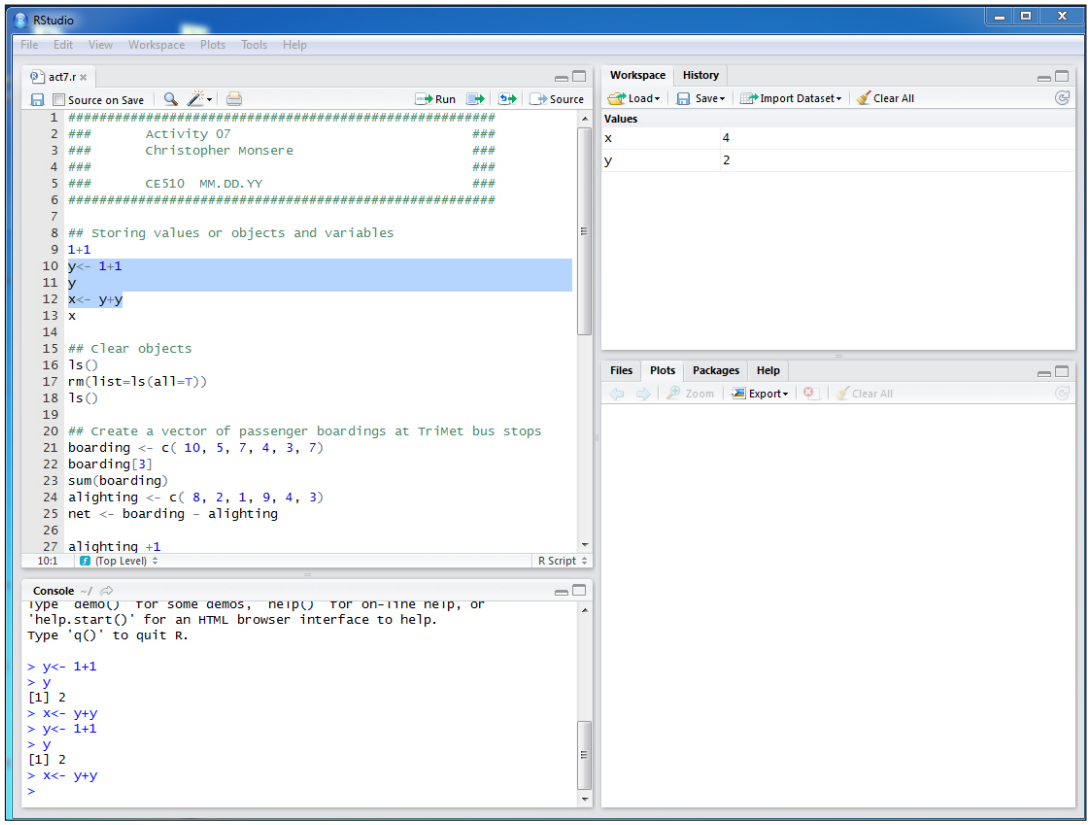


Figure 13 Screen Capture of R Studio (with R Script File open)

B. Concept: Vectors

One of the strengths of R is its use of vectors and the ease of which things can be manipulated and calculated with the use of vectors. These vectors can be numbers or text. The most common way to create a simple vector is to use the `c(,,,)` construct. Each element is separated by a comma (`,`). Let's create a vector of passenger boardings at a sequence of TriMet bus stops:

```
boarding <- c( 10, 5, 7, 4, 3, 7)
```

Notice that there are six elements in the vector. Click on that object in the R Studio workspace window. It returns to you the value of the boarding objects. We can extract the value in any one position by using the index. The index of a one row vector is the position of the element in the assignment. To get the value in the third position (a value of 7), we tell R with this syntax:

```
boarding[3]
```

R will apply many standard operations to the vector. To sum the values type:

```
sum (boarding)
```

3. Can you write the R syntax that would do this the "hard" way by adding each element in the vector?

R will apply "vectorized" arithmetic to operations combining vectors of the same length. Let's create another variable of the number of people getting off the bus at each stop:

```
alighting <- c( 8, 2, 1, 9, 4, 3)
```

Let's subtract these to show the "net" passengers at each stop:

```
net <- boarding - alighting
```

Can you see how this operation is akin to creating the formula in a cell in Excel, then copying and pasting this operation for each row?

If vectors are not the same length, the shorter vector is "recycled" and used again. A special case of this is a vector of single value — try adding 1 to every alighting value.

```
alighting +1
```

See how 1 is added to every element of *alighting*? Now try adding the vector `c(1,10)` to *alighting*

```
alighting + c(1,10)
```

To see how this works, first confirm that the function `c(1,10)` produces a 2-element vector. When added to alighting (which is 6 elements long), the `c(1,10)` is recycled twice. Notice how 1 is added to the first element, 10 to the second, 1 to the third and so on. The bottom line is that you need to be careful if vectors are not of the same length and you are performing operations on them.

4. If there were 40 people on the bus at the first stop, how many people are on the bus at the last stop? Write the R code to calculate this.

Lastly, there are lot of other helpful functions and syntax to create vectors.

One is the shorthand for from:to; R knows you want a vector back. Try typing:

```
1:10
```

Another way to do the same thing:

```
seq (from=1,to=10, by=1) or simply seq(1,10,1)
```

This short cut to creating vector lists can be really helpful.

C. Concept: Data Frames

If it isn't already obvious, working with data is the theme of this class. Vector concepts are the "building blocks" of data frames in R. Let's create another variable called *stop_id*:

```
stop_id <- c( 1200, 1030, 1456, 300, 1251, 1800)
```

Let's "glue" these vectors together. If we use the command `cbind` ('column bind') we can stick these vectors together as columns.

```
cbind ( stop_id, boarding, alighting)
```

Notice what is returned:

```
stop_id boarding alighting
[1,] 1200 10 8
[2,] 1030 5 2
[3,] 1456 7 1
[4,] 300 4 9
[5,] 1251 3 4
[6,] 1800 7 3
```

The syntax `[1,]` is a row, column notation. To refer to any value in this combined data frame, we can use `[r,c]` where **`r`** is the row index and **`c`** is the column index for the element.

Before we do that, notice that we didn't actually assign this `cbind` operation to anything; right now the only output is to the screen. R won't remember this until you tell it where to store it.

Let's assign this `cbind` operation to the variable *trimet*:

```
trimet <- cbind ( stop_id, boarding, alighting)
```

5. Can you extract the number of passengers boarding at stop 1456 from this *trimet* vector array using the row, column syntax?
6. You can also "glue" things together with `rbind`. ('row bind'). Try it and see what you get here. Do you notice the difference?

It is often extremely helpful in R to know what type or class an object is. A quick check of this will help you in a lot of debugging exercises. The command to use is `str()` ('structure').

```
str(trimet)
```

The return here is a little cryptic but this object is a numeric array of 6 rows (1:6) and 3 columns (1:3) and has attributes of NULL row names and character column names. Learning to decode these will help you in a lot of debugging exercises.

Let's turn this numeric array into a `data.frame` (a special class of R objects)

```
trimet <- as.data.frame (trimet)
```

It is not necessarily good practice to overwrite a variable (the original *trimet* has been overwritten by the data frame *trimet*) but as long as you keep track of things it can be okay in development of code. One thing that you will want to do is at the end of your code, to clear things from memory and run it to make sure everything works as planned. Now check the structure:

```
str(trimet)
```

What does it say now? Notice how R was smart enough to take the variable names and assign them as the column names. In a data frame, you have two options to select data and both can be helpful.

The first is by `dataframe$columnname`. Try

```
trimet$boarding
```

The second is by the `[r,c]` syntax. Try

```
trimet[,2]
```

Read this as [all rows, just column #2]. You can also specify it like this,

```
trimet[,c(1,3)]
```

Which reads, [all rows, column #1, #3]. Or, use the negative sign to mean "not this column or row" as in

```
trimet[,-2]
```

Let's add the variable net to the data frame. This is very easy to do using the `dataframe$columnname` syntax.

```
trimet$net <- trimet$boarding - trimet$alighting
```

Subsetting data frames is also easy and like many things in R, there is more than one way to do things, each helpful in different contexts. This flexibility can be part of the reason that new users get frustrated with R! Think of subsetting data frames as very similar to the SQL exercises that you were doing.

Let's say we only want records from column `stop_id` with `stop_id >= 1200`

```
trimet[stop_id >= 1200, ]
```

The `[r,c]` notation and the comma make all the difference. Try

```
trimet$stop_id[stop_id >= 1200]
```

What are you getting here? Just the elements of `stop_id` that are greater than or equal to 1200.

I think by far the easiest syntax to remember is the `subset` function:

```
subset (trimet, stop_id >= 1200)
```

Notice this returns the whole data frame subsetted. You can combine logic in the subsetting criteria such as

```
subset (trimet, stop_id >= 1200 & net >=0)
```

It is helpful to assign the subsetted dataframe to another variable name so that you can keep the original data intact:

```
trimet_stop1200 <- subset (trimet, stop_id >= 1200)
```

Notice now that if you will need to refer the subsetted data frame as:

```
trimet_stop1200$net
```

R Studio is also advantageous because it has a built in viewer that allows you to see the data frame. Click on *trimet* in the workspace window and see the view of the dataframe.

D. R Concept: Packages

The base R install will do much of what you need. However, sometimes you need to add functionality. These packages are all on the CRAN. Downloading and installing them is an easy, menu-driven process. Be sure you set up the **RLibs** directory.

Let's get the package ('ISwR') for the Dalgaard textbook.

In the R Console, choose ***Packages > Install Packages***. Select an appropriate "mirror" site (choose your favorite University or country on the list), then find the package you are interested in. It is advisable to choose a "close" server.

Packages can contain functions and or data and are contributed by open source users.

Deliverable

Annotated code – submit your R script to the Dropbox with the answers to the six questions. In your R code use comments to indicate your answers but make sure the R code that you modify to answer the questions can be run by the instructor to verify that it works. For example:

```
#Question 3 - The "hard" way would be to add each element of the vector
individually using the [] syntax. My code to do this would be

boarding[1]+boarding[2]+boarding[3]+boarding[4]+boarding[5]+boarding[6]
```

Assessment

Activity 12 Grading Rubric

	Excellent (10)	Good (8)	Poor (6)	0
Script	Organized, complete, accurate and executes.	Missing minor parts, but executes and is otherwise organized and accurate.	Missing significant portions of the activity, unorganized, inaccurate, but executes.	Code does not execute
Annotation	Annotations are complete and describe what the code is accomplishing.	Some annotations are incomplete or do not describe what the code is accomplishing.	Annotations were not included.	Code does not execute
Discussion	Insightful discussion or commentary relating to the question at hand demonstrating student understanding of the task.	Discussion or commentary was partially incomplete.	Minimal to no discussion or commentary.	Code does not execute

Reading in Data Files

ACTIVITY 13

One of the first things you need to learn to do with R is bring data in for analysis. This is often the first of lines in our R scripts. R can work with many different data formats (CSV, TXT, XLS, google spreadsheet, SAS, SPSS) through the base R or packages that others have written. In this class, we will work with two different data files. In most of the class, we will interface directly with the class database through an ODBC connection (on Windows boxes). It is common, though, to have to read in data files from a format called CSV, or comma separated values. From the class database, we selected records with this query:

```
SELECT * FROM trimet.route19 WHERE service_day >= '03-04-2007'
AND service_day < '03-10-2007'
```

and exported it to a .csv file. We've placed this .csv file on the companion website for easy reading.

Purpose

The purpose of this activity is to help you become familiar with reading in external data frames for analysis in R.

Learning Objective

Read in data files from CSV, Excel, and direct connection to PostgreSQL.

Required Resources

- R, R Studio
- Sample script file from class web site

Time Allocated

15 minutes in class

Tasks

This activity begins assuming R Studio is open. If not, please start R Studio.
In R Studio, start a new script file.

It is always good practice to use comments throughout your code. One good tip is to create a file header such as the following:

```
#######################################################
###     Activity 13  - Reading in Data Files        ###
###                                                 ###
###     Understanding and Communicating             ###
###     Multimodal Transportation Data              ###
###                                                 ###
###     Script: C. Monsere                          ###
###     Revision: 03.22.2012                        ###
#######################################################

#-------------------------------------------------------
# 1 - Read in the CSV file and explore
#-------------------------------------------------------
```

A. Loading the CSV file

We are going to read this rather large CSV file into R. The function to do this is `read.csv`. But first, at the R prompt, type:

```
? read.csv or help(read.csv)
```

Take some time to browse the options. The following command will read in the CSV file, assuming that you set the working directory properly. Make sure you are familiar with the options.

```
trimet <- read.csv("http://www.transportation-data.com/resources/
route19march07.csv", header=TRUE, sep=",", na.strings="NA", dec=".",
strip.white=TRUE)
```

Use the `str()` function to see what R is interpreting the structure to be of the data frame *trimet*. This can be very valuable in the future. This tells you how R is reading each column. You can also view the data elements in R Studio by clicking on the name *trimet* in the data window. A spreadsheet-like display will appear in the source pane (Figure 14). Notice that within the workspace pane R Studio gives you a helpful summary: 29,928 obs. of 26 variables. This means there are 29,928 rows and 26 columns of data.

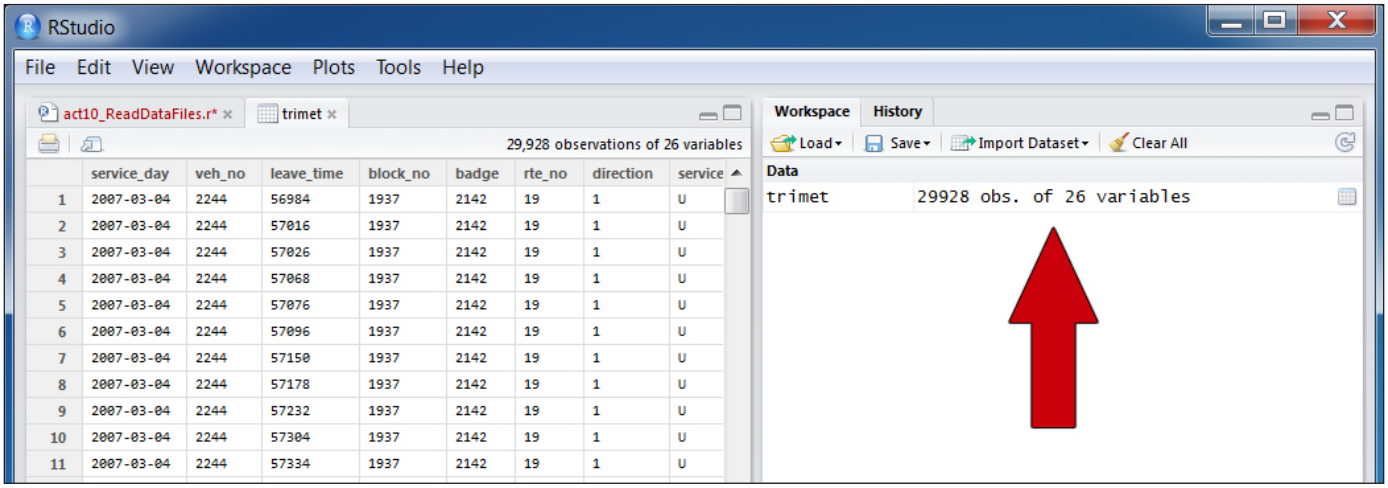


Figure 14 Screen capture of table view of data set in RStudio

There a number of R functions that give you this same information:

How many variables (columns) in the data frame?

```
ncol(dataframe)
```

How many records in the data frame?

```
nrow(dataframe)
```

What data type does R seem to think the *service_day* column is?

```
str(trimet$service_day)
```

What are the levels of service day?

```
levels(trimet$service_day)
```

How many elements in the service day column?

```
length (trimet$service_day)
```

B. Count Functions

These count functions can be helpful in programming the analysis of the data frame. In this data, the rows are structured such that each row is a stop event. Just knowing the number of rows tells you how

many stops were observed. Another useful function is one that returns unique values in a column. To answer the question, how many unique stops are represented in the data frame, you could do this with:

```
numuniqstops <- unique(trimet$stop_id)
length (numuniqstops)
```

1. Can you adapt the above code to answer how many types of schedule status flags appear in the data frame?

Now, can you create a new data frame containing only records that occurred at `stop_id = 2107`?

2. How many records in the subsetted data frame?
3. What is the total amount of dwell time in the subsetted data? Hint, use the function `sum`
4. How many different unique vehicles were used in this data set to provide service?

C. Reading in a File from Your Local Directory

If you have not done so already, create a directory where you will be saving files from this class. Download the CSV file from the web to this directory:

http://www.transportation-data.com/resources/route19march07.csv

One helpful step that I take in R is to set the working directory. Setting the working directory is not required. If you set this, it becomes the default directory. The command is:

```
setwd ("N:/My Documents/CE510")
```

This will be the default path for the R session until you change it. Note that I have set this to be a folder in my N: drive. This is now the root path for all R operations (reading, saving, etc). It is critical that you understand that R requires that the normal convention for windows backslashes \ to be reversed / . If you don't set this, you will need to specify the full path name to the file. In this example, the working directory is set to **N:/My Documents/CE510**. The CSV is copied from the web

```
setwd ("N:/My Documents/CE510")
trimet_local <- read.csv("route19march07.csv", header=TRUE, sep=",",
na.strings="NA", dec=".", strip.white=TRUE)
```

You can also use the R Studio option to ***Import Data Set*** under the Workspace menu. This has a nice GUI but has the disadvantage of requiring you to do the point-click every time you want to load the file, rather than just running the script.

D. Reading in Excel File Using RODBC

Open the CSV and save as XLS (**not** XLSX). Go ahead and change the name of the tab to something other than the file name. In my example, I changed the name of the tab to *data*.

Now install the package RODBC from the R Studio menu options. Once a package is installed, to load it in an active session of R you need to call it. Add the package to your R session using the command:

```
library(RODBC)
```

RODBC is a windows platform that allows you to interact with set up a connection to XLS. Since the file is in our working directory, only the file name is needed (not the entire path).

```
xls_connect <- odbcConnectExcel("route19march07.xls")
```

Now call the `odbcConnect` function that we established previously and tell R you want data from the tab named what you called it in the first step.

```
trimet_xls <- sqlFetch(xls_connect, "data")
```

Check this data using `str()`.

Deliverable

An R script file showing the work in this activity. Be sure to add comments and headers to your file.

Assessment

Annotated code – submit your R script to the Dropbox. To receive all 10 points, the code must contain comments and the answers to the 4 questions.

Activity 12 Grading Rubric

	Excellent (10)	Good (8)	Poor (6)	0
Answers	0 Incorrect	2 or less Incorrect	>2 Incorrect	Code does not execute
Script	Organized, complete, accurate and executes.	Missing minor parts, but executes and is otherwise organized and accurate.	Missing significant portions of the activity, unorganized, inaccurate, but executes.	Code does not execute
Annotation	Annotations are complete and describe what the code is accomplishing.	Some annotations are incomplete or do not describe what the code is accomplishing.	No annotations were provided.	Code does not execute

R Plots

ACTIVITY 14

Purpose

The purpose of this activity is to preview some details of R and R Studio.

Learning Objective

To become more familiar with R and R Studio

Required Resources

- Quick-R section on "Advanced Graphs"

Time Allocated

50 minutes out-of-class

Tasks

Browse the following sections:

http://www.statmethods.net/advgraphs/parameters.html

http://www.statmethods.net/advgraphs/axes.html

http://www.statmethods.net/advgraphs/layout.html

Deliverable

Complete the short quiz on the class site before coming to the next class session. This quiz is timed, you will have 30 minutes to complete the quiz once you start.

Assessment

Short quiz

Student Notes

Learning Some Simple Plotting Features of R

ACTIVITY 15

This independent exercise will help you learn how R plotting functions work. This activity focuses on how you might use graphics to help you interpret large data sets. The sample data we are using – stop level data for TriMet's route 19 from six days in March 2007 (March 4-9, 2007) includes 29,928 observations of 26 variables. In short, a lot of data.

You should think of figure creation in R as "painting". The basic plot functions will make readable, useable graphs for your interpretation and exploration without much effort on your part. Customization will require you to adopt this "painting" logic and add or subtract elements. You will need to become familiar with margins, layouts, and plot regions to become an R graph guru. You will explore More on this in later activities.

Purpose

The purpose of this activity is to give you the opportunity to become familiar with plotting in R.

Learning Objective

Become familiar with some basic plotting options in R.

Required Resources

- R, R Studio
- TriMet Route 19 Map: **http://www.trimet.org/schedules/r019.htm** *(Note that this is the current Route 19 which is slightly different than the route in 2007.)*
- TriMet Data Dictionary

Time Allocated

60 minutes in class

Tasks

A. Read in the data file posted on the web

First, load the data frame with stop-level data for TriMet's route 19 from six days in March 2007 (March 4-9, 2007) from the web using the following syntax:

```
trimet <-
read.csv("http://www.transportation-data.com/resources/route19march07.
csv", header=TRUE, sep=",", na.strings="NA", dec=".", strip.white=TRUE)
```

B. A Simple Plot

As you work through this exercise, enter these R commands in the R Studio file and use it to process R operations by using the send line or group of lines option. Check with a fellow student or instructor if you don't know how to use the R Studio interface yet.

R's base graphics can produce default plots that look pretty decent without specifying any customization. Let's create a scatterplot of the *stop_time* and the *est_load* on the bus. By convention, let's plot the time on x-axis and dwell on the y-axis:

```
plot( trimet$stop_time, trimet$est_load)
```

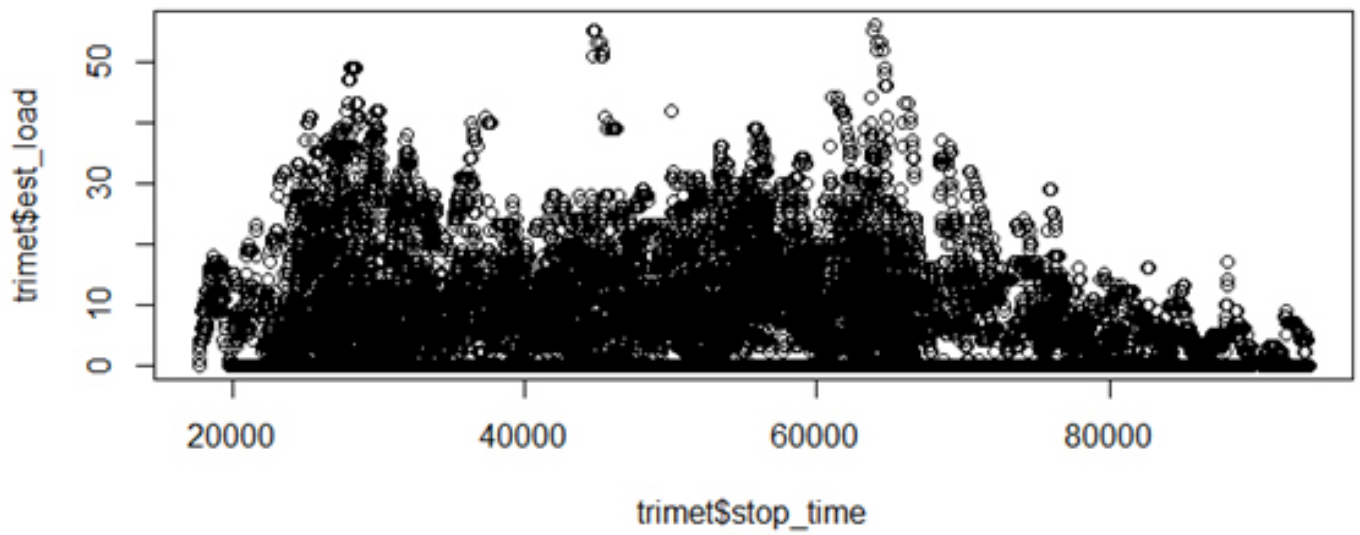


Figure 15 Plot of trimet$stop_time vs trimet$est_load

This plot of 29,928 observations does a few things. First, it is clear from the plot that there are two "peaks" of the load data. Second, there are many records with an estimated load of zero.

IMPORTANT TIP: To save the windows graphics output, size the window to the desired dimensions and use the export option in R Studio. You can also use the File Copy to Clipboard to paste in a Word or PPT document. The windows metafile option produces the best quality for copy/paste options.

Sometimes it is helpful to just ask R to return a plot of the variable. Depending on the type of data, the call to plot with just one variable returns the "best" of R's base plots to show the variables. For example, the call to plot with just estimated load `plot(trimet$est_load)` returns an index plot (Figure 16).

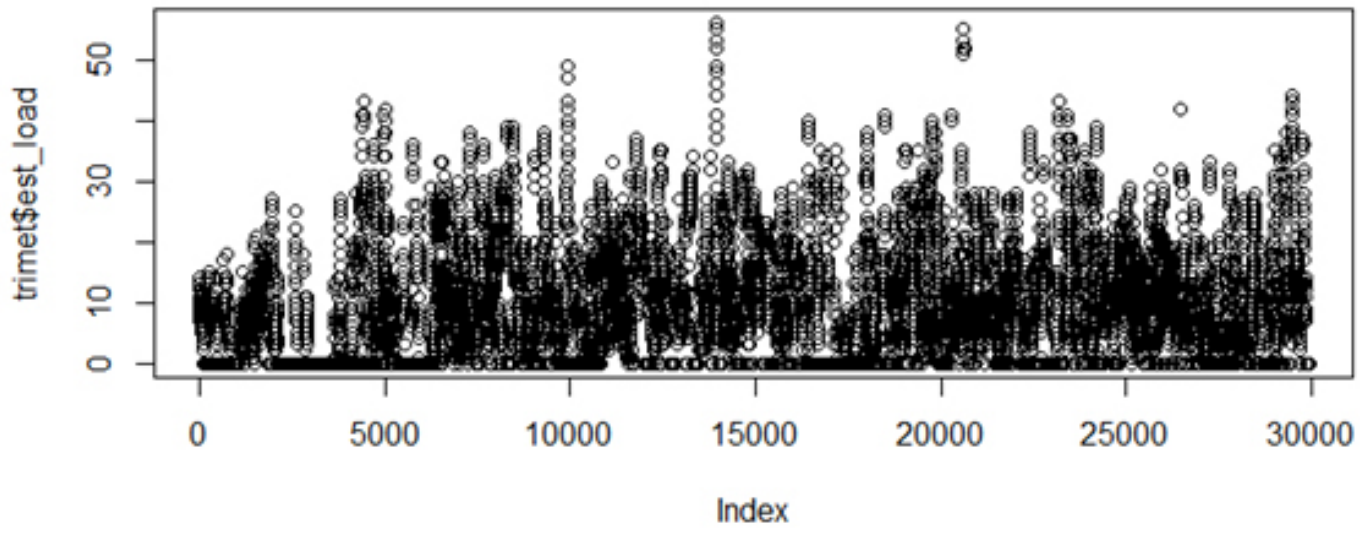


Figure 16 Plot of trimet$est_load

Figure 16 is a scatterplot with the x-axis labeled 'Index' and the y-axis labeled '*trimet$est_load*'. The index is the "row number", so this plot shows the data as it appears in the data frame. Notice that these two plots are not the same (Figures 15-16).

1. What do the differences between Figure 15 and Figure 16 indicate about the order of the data in the TriMet data frame?

The calls to plot for data that are "factors" returns a barplot. R considers these discrete data. The call to `plot(trimet$service_day)` returns the barplot shown in Figure 17.

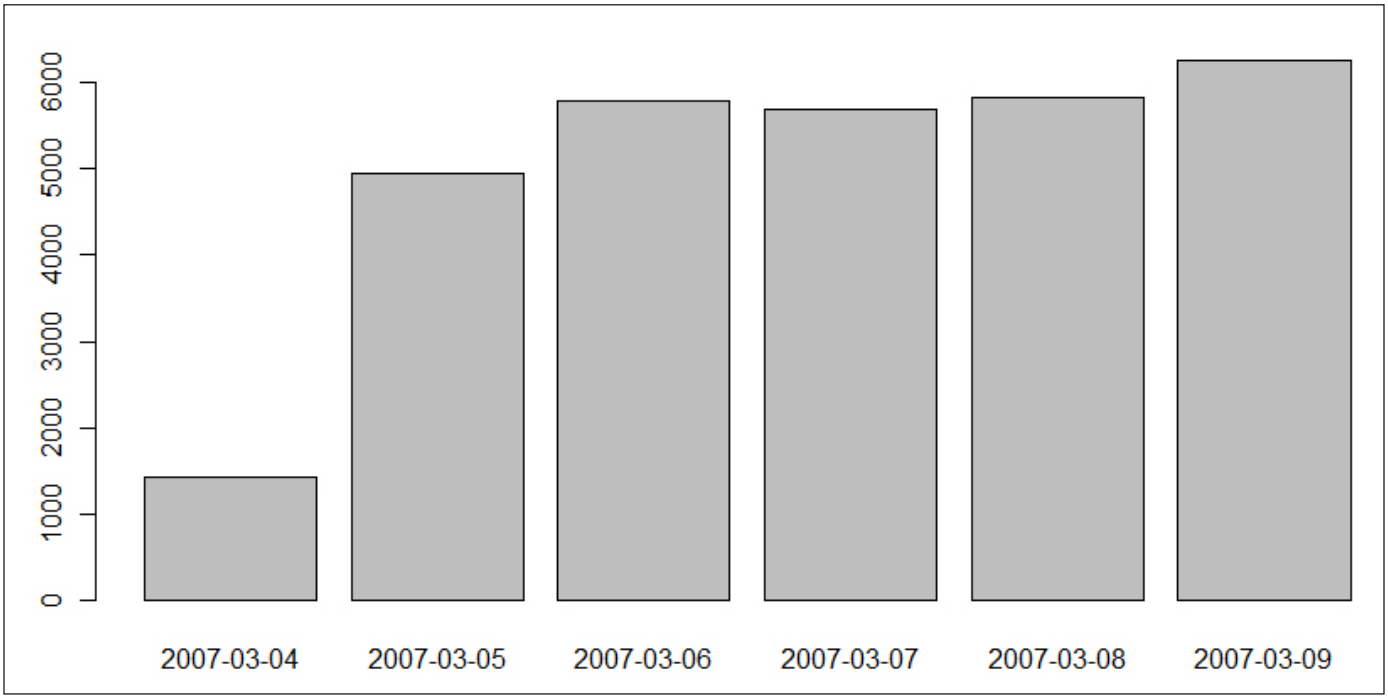


Figure 17 Plot of trimet$service_day

These plots can be helpful because they allow you to quickly see the range of the data. In the plot above, you can clearly see the dates in the data frame and the number of records that appear for each day.

2. Why do you think March 4, 2007 has so few records compared to other days?
3. Can you find another variable that R considers a factor and plot it?

C. Some Simple Customizations

Let's explore the plot function options in more detail by checking in on the help menu. Do this by typing,

```
? plot
```

R's help can be helpful or not, depending on your perspective. The call to plot allows many, many options. The help file lists the details for `type`, `main`, `sub`, `xlab`, `ylab`, and `asp` options. Note that there is a line that says: `Arguments to be passed to methods, such as graphical parameters (see par)`. Many plot options are controlled in the par settings, but more on those later. Let's first see how some of the plot options are specified. Options are specified with the `option=specification`. The points type is the default. Let's try plotting the lines connecting the points.

```
plot( trimet$stop_time, trimet$est_load, type="l")
```

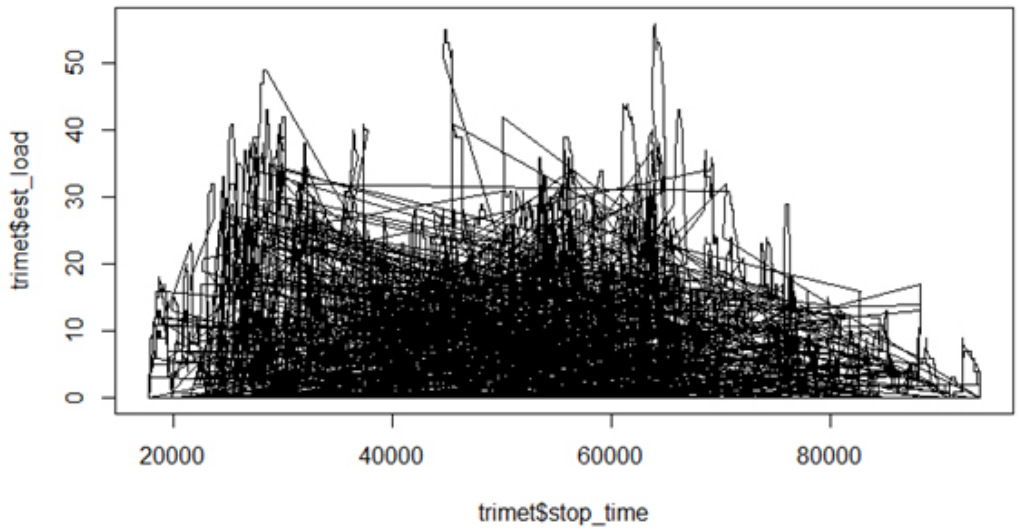


Figure 18 Load by time of day, plot type="l"

One thing to notice is that R connects the points in the order that they appear in the data frame. You already noticed that the records are not ordered in time, and that is the reason for this jagged mess (Figure 18). You will learn how to order the data later, which would solve this. Let's specify a plot type of both points and lines, labels to the x and y axis, and a title (Figure 19).

```
plot(trimet$stop_time, trimet$est_load, type="b", main="Load vs Time
of Day, Route 19, March 4-9, 2007", xlab="Time (sec past midnight)",
ylab="Est Load")
```

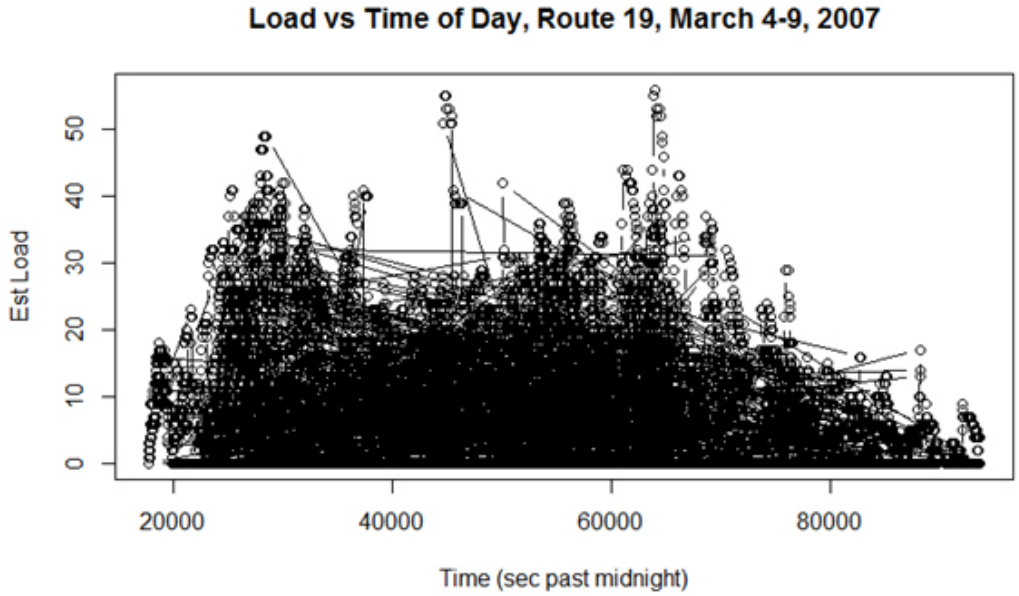


Figure 19 Load by time of day, x and y labels and title added

D. A little debugging

R can get a little frustrating if you are not used to being careful with syntax. A missing comma, quote, or a capital letter can make the difference between a successful execution of a script and a failure. Try executing the following code:

```
plot(trimet$stop_time, trimet$est_load, ylab)
```

Oops! Something is missing from the above statement — what is it? The R console gives you pretty good feedback on what the error is:

```
Error in plot.xy(xy, type, ...) : object 'ylab' not found
```

So it is telling you it doesn't know what to do with '`ylab`'. Sometimes you forget to close a function call and R is still waiting for you to send it something. The consoled line changes from > to +. If you hit the ***ESC*** key, R cancels your command and returns to the > prompt as long as the Console window is the active window.

```
plot(trimet$stop_time, trimet$est_load, ylab=
```

As debug tool, it is sometimes helpful to run pieces of the statement to see if you can detect the error. For example, if you just sent the command to plot with no options

```
plot(trimet$stop_time, trimet$est_load)
```

and it executed, you now know that your error is in the options you have specified. It is easy to do this in R Studio by just selecting the code you want to execute, sending it to the R Console with the 

icon, and typing any needed closing code such as the parentheses. An approach like this could help you find the error in your statement.

The *trimet* data *stop_time* field is seconds past midnight. Maybe seconds past midnight isn't that interpretable for you. To convert seconds past midnight to hour of the day, just divide by 3600 (Note: You can just perform the operation in the plot function itself).

```
plot( trimet$stop_time/3600, trimet$est_load)
```

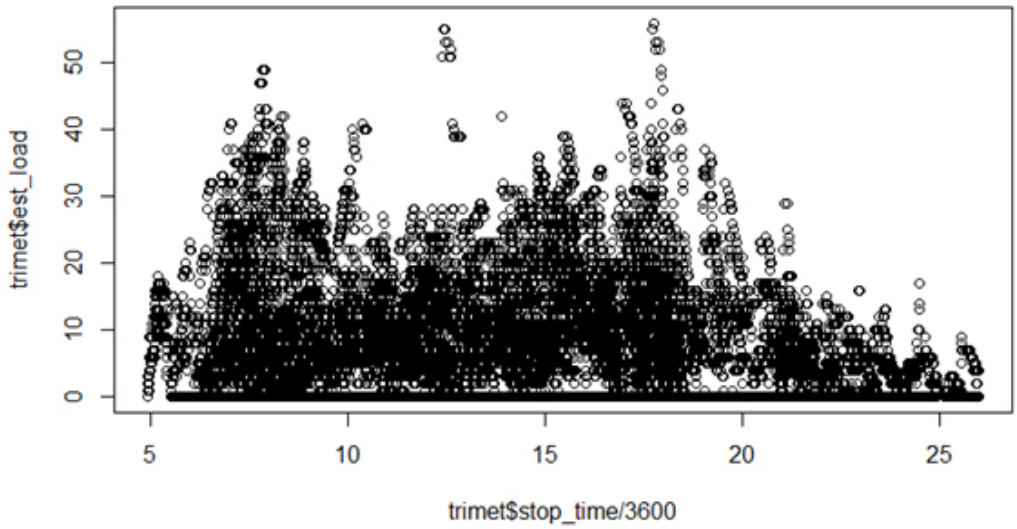


Figure 20 Load by time of day

E. Some Par Settings

In R, you have near infinite control of the plotting options. Anything can be changed; you just have to remember how to do it! Options passed in the `plot()` command are only active for that call. Meaning, if you change the line type, the next plot will have the default line type not the one specified previously. Many parameters can be passed through the plot command and are listed in the `par` options. Open help on `par` settings – Wow! You can see the detail that R will allow over plot options!

Let's try some changes and you can explore other options on your own.

i. X and Y Plotting Limits

One of the nice features of the default plotting option is that it does a pretty good job of selecting the axis limits. The `xlim` and `ylim` options allow you to change the plotting window limit of the respective axes. It requires the limits in a vector format, c(start, end). Try these options reflected in Figures 21 and 22.

```
plot(trimet$stop_time/3600, trimet$est_load, xlim=c(0,25))
plot(trimet$stop_time/3600, trimet$est_load, xlim=c(14,18),
ylim=c(0,30))
```

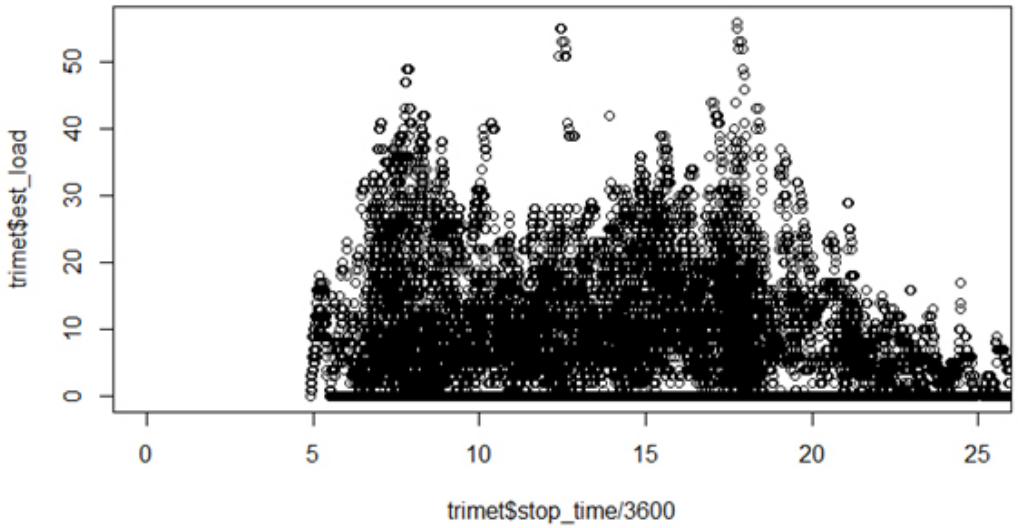


Figure 21 Plot of trimet$stop_time vs trimet$est_load with modified x-axis (0,25)

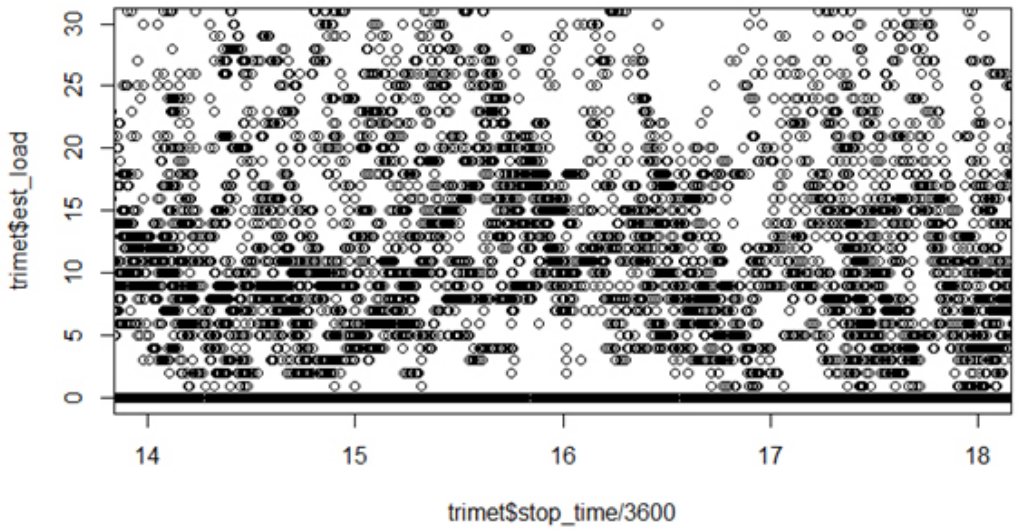


Figure 22 Plot of trimet$stop_time vs trimet$est_load with modified x-axis (14, 18) and y-axis limits (0,30)

ii. Symbols, Line Type, Line Weight

The plotting symbol, line type and weight can be controlled with `pch`, `lty`, and `lwd` options

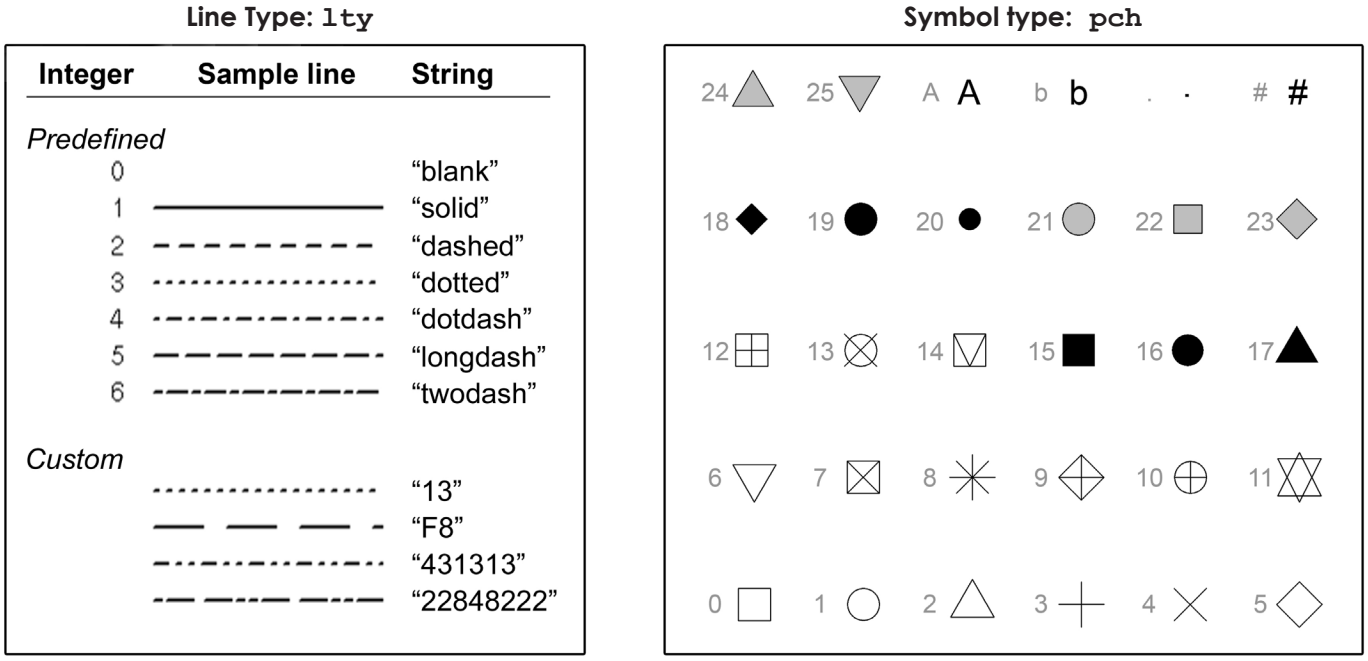


Figure 23 Line Type and Symbol Type (from Murrell, R Graphics)

iii. Colors

Color is one important element in building accurate and effective representation of data. Misuse of color can distort the meaning of underlying data, leading you to incorrect interpretations and conclusions. This next task will familiarize you with basic color functions in R. Colors are sent to R in hexadecimal format. If you are not familiar with this color coding, see this article: ***http://en.wikipedia.org/wiki/Web_colors***

Color by names: At the simplest level, R has a set of built-in color names. If you type colors() at the R console it will return a complete list of possible color names. Figure 24 on the following page (from Maindonald & Braun, *Data Analysis and Graphics Using R*, Cambridge University Press, 2003) shows these names color ranges. This same figure, "Selection of Color Palettes" is also posted on the course website. It is not a bad idea to print a color copy of this key reference. Note that col="blue", col= "blue1", col= "blue2", col= "blue3", and col= "blue4" are predefined colors.

Colors by integer: You can also refer to colors by an integer such as col=1. The 1 refers to the first color in the default R palette. To see what those currently are you can type `palette()` at the R console. It will return a list of colors:

```
> palette()

[1] "black"  "red"  "green3"  "blue"  "cyan"  "magenta"  "yellow"  "gray"
```

So when you specify col=1, R will use "black". Likewise, col=2 will use "red". You can change the default palette if you desire.

Colors by hexadecimal code: The named colors are in essence aliases for referring to the colors by hexadecimal values. No one will expect you to remember these but as an example these are equivalent: col= "#FF0000" and col= "red".

Figure 24 Selection of Color Palettes (from Maindonald & Braun, *Data Analysis and Graphics Using R*, Cambridge University Press, 2003)

Colors by RGB code: Finally, you can also set colors by referring to the red, green, values for each color. The `rgb()` function returns the hex format of a color given the red, green, and blue values. One nice aspect is that you can give this function a transparency level. By using the transparency value, the problem caused by over plotting can be mitigated. In the following plot below (Figure 25), as more data points appear on top of each other, the color becomes more saturated.

```
plot (trimet$stop_time/3600, trimet$est_load,
col=rgb(r=255,b=0,g=0,a=10, maxColorValue=255), pch=16)
```

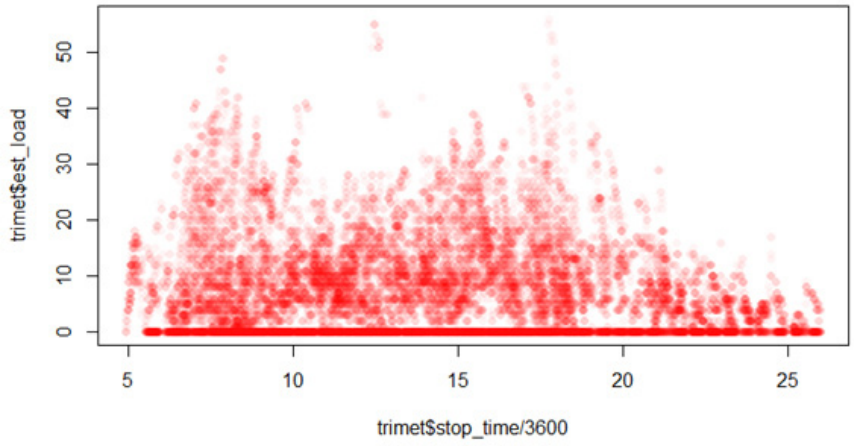


Figure 25 Plot of trimet$stop_time vs trimet$est_load pch=16 and col="red" with a 10% tranparency value

The function `col2rgb` returns the rgb values for any of R's named colors. This can be helpful if you want to apply transparency values to a named color.

Lastly, R has many options to prepare color ramps. These are a vector list of n colors that can be used for plotting. The default palettes are shown in Figure 26. The package R Color Brewer has an excellent set of additional ramps which we will explore later. You can use these ramps to show a third dimension on a plot.

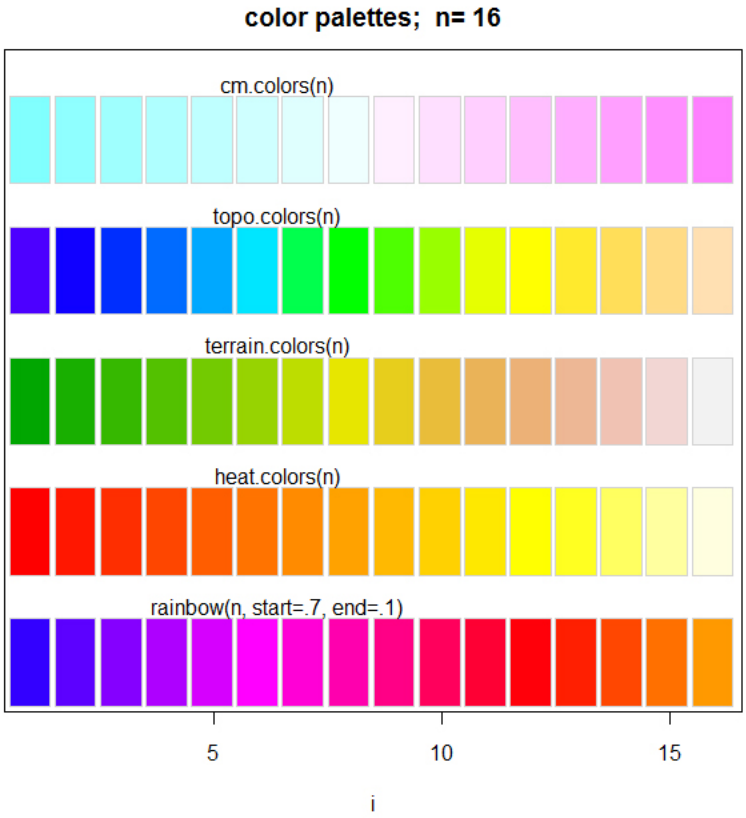


Figure 26 Default color palettes in R, here shown with n=16 elements

iv. Margins

Many `par` options can be set globally for one session of a graphics window. Calls to `par` will apply to the open graphics device (usually the screen) until you close it. A couple of helpful options are margins and layout. Margins are set by the number of lines (or inches) for both outer and figure region margins.

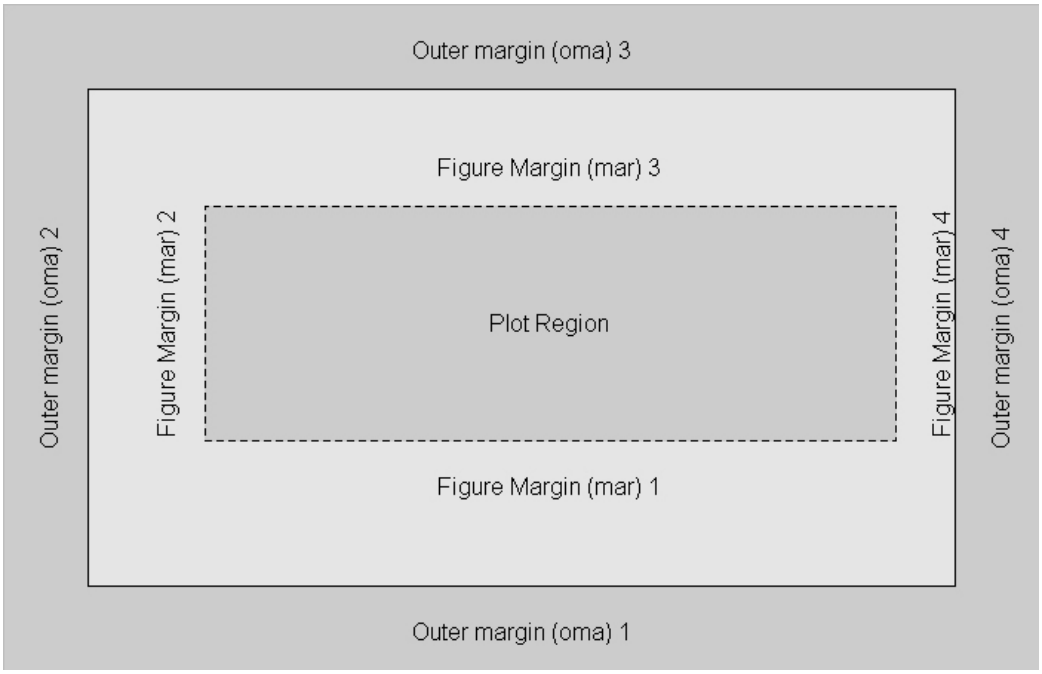


Figure 27 Description of margin layout surrounding the plot region

The `par` setting for margins is given in a vector such as

```
par (mar=c(3,4,3,4), oma=c(3,4,3,4))
```

`mar` is figure margins in lines, `oma` is outside margin in lines. Play around with this option.

v. Plot Layouts

Another helpful `par` option is to format the plot window to allow for multiple plots arranged in sequence. The following R code shows how to use the `par(mfrow=c(r,c))` syntax to create a simple layout of more than one graph. Note that the layout will stay in place until you change the `par` setting or close the graphics window. Three examples of changing the `par` settings are included below and identified as Figures 28, 29, and 30.

```
par ( mfrow= c(2,2) )  #layout is 2 rows, 2 cols, R plots across ROWS
plot(trimet$stop_time/3600, trimet$dwell, col="red", main="1st plot")
plot(trimet$stop_time/3600, trimet$dwell, col="blue", main="2nd
plot")
plot(trimet$stop_time/3600, trimet$dwell, col="seagreen", main="3rd
plot")
plot(trimet$stop_time/3600, trimet$dwell, col="orchid", main="4th
plot")
```

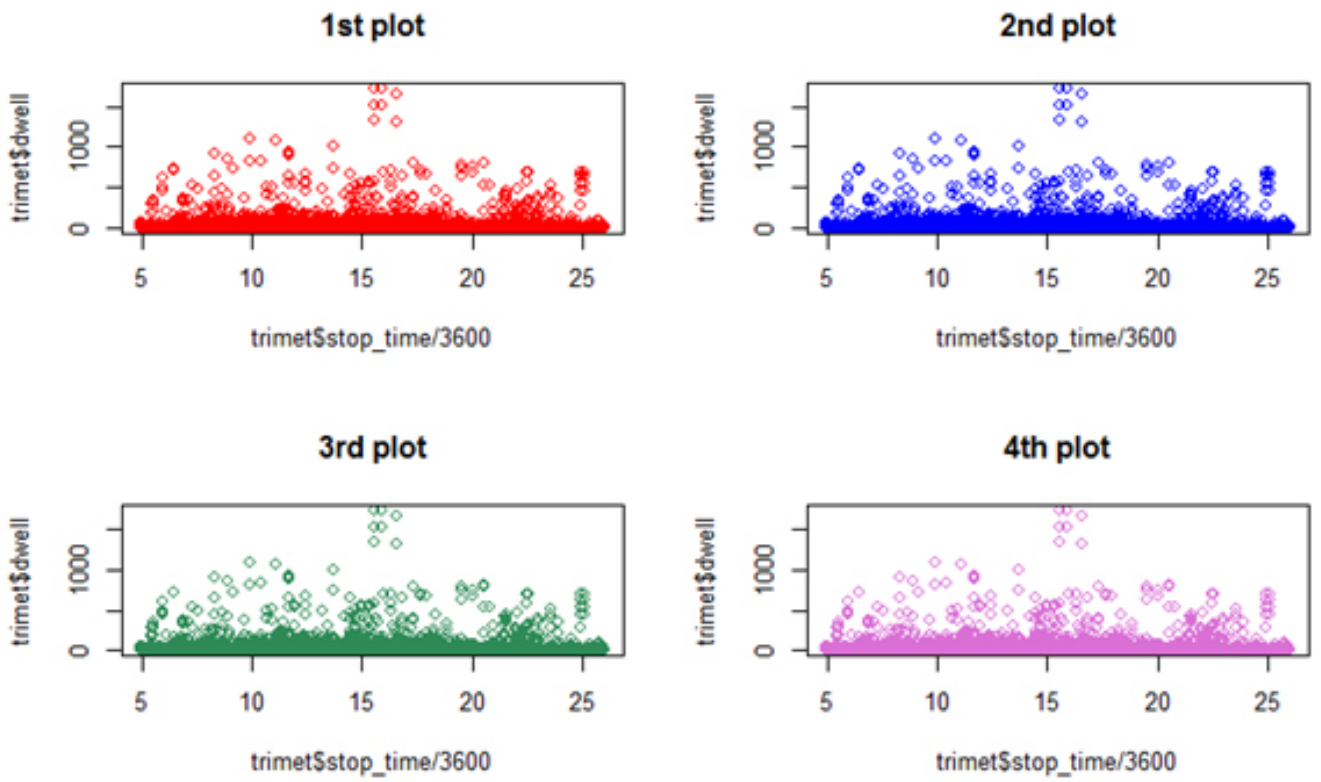


Figure 28 Plot of trimet$stop_time versus trimet$dwell with 2 row and 2 column layout.

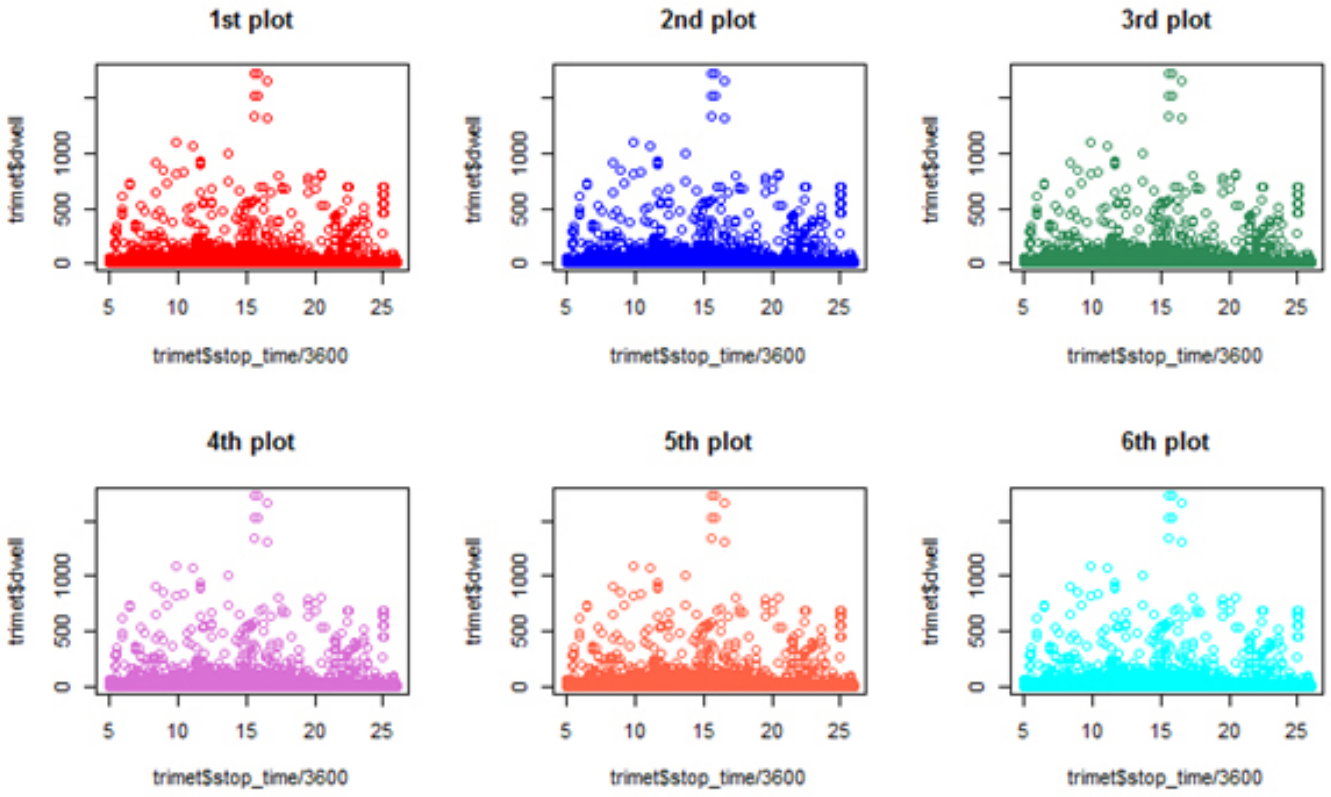


Figure 29 Plot of trimet$stop_time versus trimet$dwell with 2 row and 3 column layout.

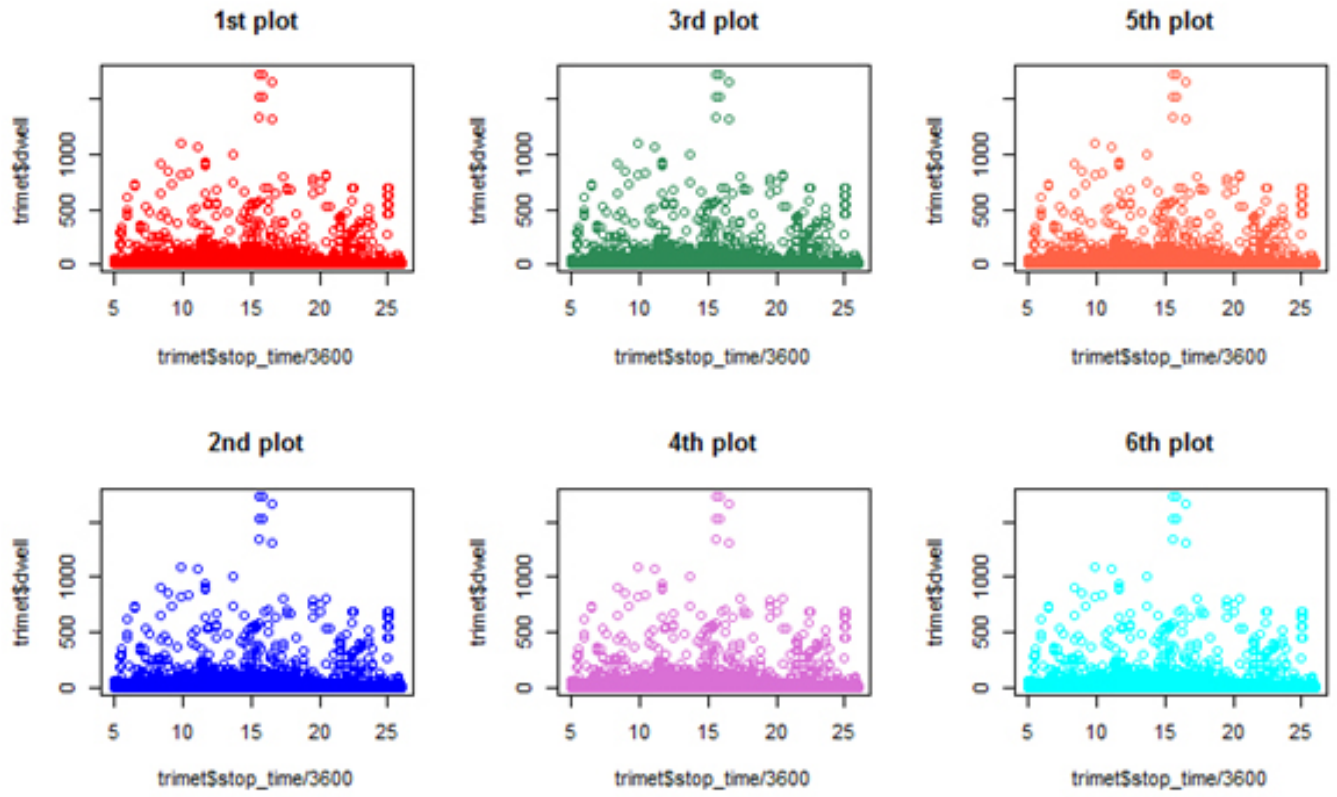


Figure 30 Plot of trimet$stop_time versus trimet$dwell with 2 row and 3 column layout with ordering completed down columns

No doubt, all of the `par` options can be daunting but once you figure out how this works, you will be able to really control the appearance of graphs. The options `bg`, `lwd`, `pch`, `col`, `cex`, and `las` are some of the more common ones and can be passed to the graphics options in the plot command.

4. In your R script, include code that shows changing at least three of these options (and include multiple examples of the option changes, perhaps even in a nice layout option). Try at least four other `par` options that look interesting to you not listed previously. Prepare an R code that you can share with the class showing your experiment.

F. Adding Other Data to Graph

This is the first example of how R "paints" graphs. Note that the first call to plot sets the x and y limits so if you might need to adjust the `xlim` and `ylim` so that both data sets fit, or change the order.

```
plot(trimet$stop_time/3600, trimet$dwell, col="red")
points( trimet$stop_time/3600, trimet$est_load, col="blue")
```

`points` is a function that adds points to an existing plot window.

5. Reverse the order of how *dwell* and *est_load* are plotted. Do you see how the first call sets the plot window?

Try adding a special line with this function (see Figure 31):

```
abline (v=10, lty=2, col="grey")
```

```
abline (h=500, lty=2, col="black")
text (15, 1000, "TriMet Data Rules")
```

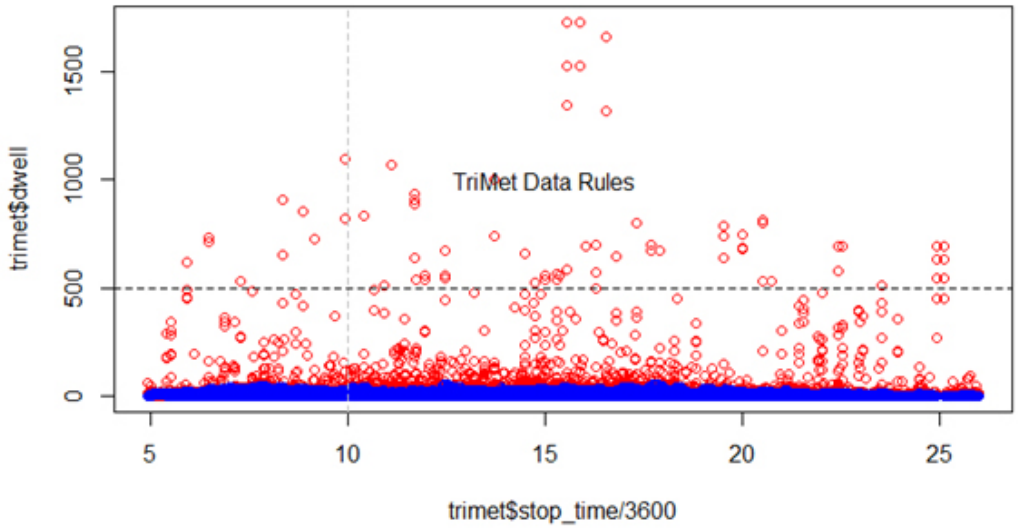


Figure 31 Plot of trimet$stop_time vs trimet$dwell and trimet$load

In R, as long as the plot window is open you can keep annotating the plot with various functions. We will see later how to add text, legends, etc.

i. Plotting a third dimension as a color or symbol

If we check the schedule status field, with a call to plot we note that it has just six types of schedule status in data frame *trimet*.

```
plot(trimet$sched_status)
```

Note that many `par` options such as `col` or `pch` can be passed as a vector (in this case just another data column). You should try to understand this as we will use this feature often. Let's look at the values for `stop time`, `dwell`, and `schedule status` for the rows 20-30 of the *trimet* data frame. We can do this with this syntax:

```
trimet[20:30, c("stop_time", "dwell", "sched_status")]
```

which returns

```
    stop_time dwell sched_status
20      40402    10            2
21      40484    12            2
22      40581     0            2
23      40581     0            1
24      40809     1            1
25      40809    14            3
26      40889     0            2
27      40980     0            4
28      41017    13            2
29      41041     0            2
30      41066    15            2
```

Let's say we want to color code the points by the value of schedule status. The stop time is the x value (1st dimension), the dwell time is the y-axis (2nd dimension), and we can color code the symbol to represent the schedule status (3rd dimension). To help you see how R can make things easier, let's plot these eleven elements for rows 20-30 (which we have assigned to *s_plot*):

```
plot(s_plot$stop_time, s_plot$dwell, pch=16, col=s_plot$sched_
status)
text (s_plot$stop_time, s_plot$dwell, s_plot$sched_status, pos=3)
```

As shown in the plot, each point is plotted at the x and y values above. The color is specified as the value of the *sched_status* field. For convenience, we've plotted the value of the schedule status as text above the data point. Remember that `col=1` is black, `col=2` is red, `col=3` is green and `col=4` is blue. So for the data point in row 27, the x-value is 40980, the y-value is 0, and the `color=sched_status=4`, which is blue!

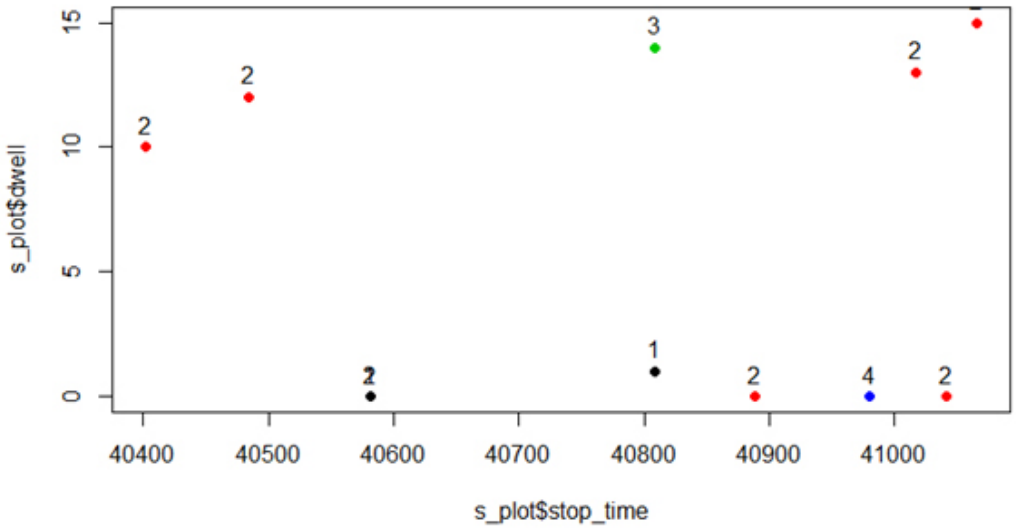


Figure 32 Plot of s_plot$stop_time vs. s_plot$dwell with third dimension of sched_status differentiated by color

Now, let's apply this same strategy to the entire data frame.

```
plot(trimet$stop_time/3600, trimet$dwell, col=trimet$sched_status,
pch= trimet$sched_status)
```

You can add a legend to the plot with the `legend()` function

```
legend ("topright", col= sort(unique(trimet$sched_status)), pch=
sort(unique(trimet$sched_status)), c("1 = unscheduled stop",
    "2 = regular stop",
    "3 = pseudo stop (unofficial time point)",
    "4 = time point (official time point)",
    "5 = first stop in trip",
    "6 = last stop in trip"
))
```

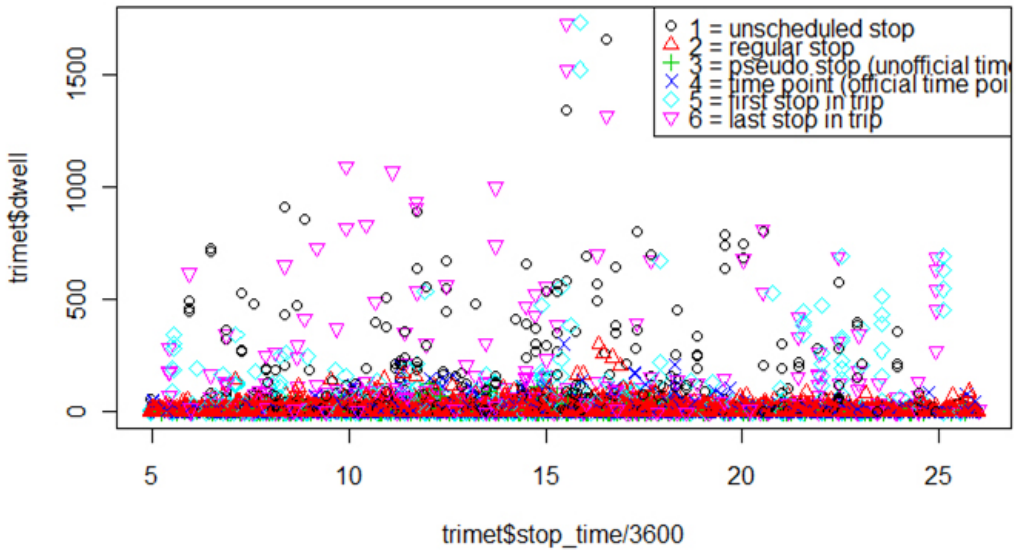


Figure 33 Plot of trimet$stop_time vs trimet$dwell with schedule_status as 3rd dimension

6. Look at the legend code. Can you see why that we applied the `sort(unique())` function to the column of schedule stops? One helpful tip is just run `unique(trimet$sched_status)` to see what it returns. You might need to refer to the legend help function.

We could be a little more specific in applying the color to the plot, rather than letting the colors be set by the integers 1, 2, 3. So , what if I make a vector of the colors I want and name it *schedule*:

```
schedule <- c("green", "blue", "yellow", "orange", "red", "black")
```

Determine for yourself that `schedule[1]` or `schedule [2]` returns a plot color. Now, if we stick this in the plot call, R will plot the color of the symbol by looking up the color from schedule based on the value in *trimet$sched_status* column. Play with the R code until you can see what is happening.

```
plot(trimet$stop_time/3600, trimet$dwell, col=schedule[trimet$sched_
status], pch= 16)
```

You can add a legend to the plot with `legend()` function with a slight variation

```
legend ("topright", fill=schedule, c("1 = unscheduled stop",
    "2 = regular stop",
    "3 = pseudo stop (unofficial time point)",
    "4 = time point (official time point)",
    "5 = first stop in trip",
    "6 = last stop in trip"
  ))
```

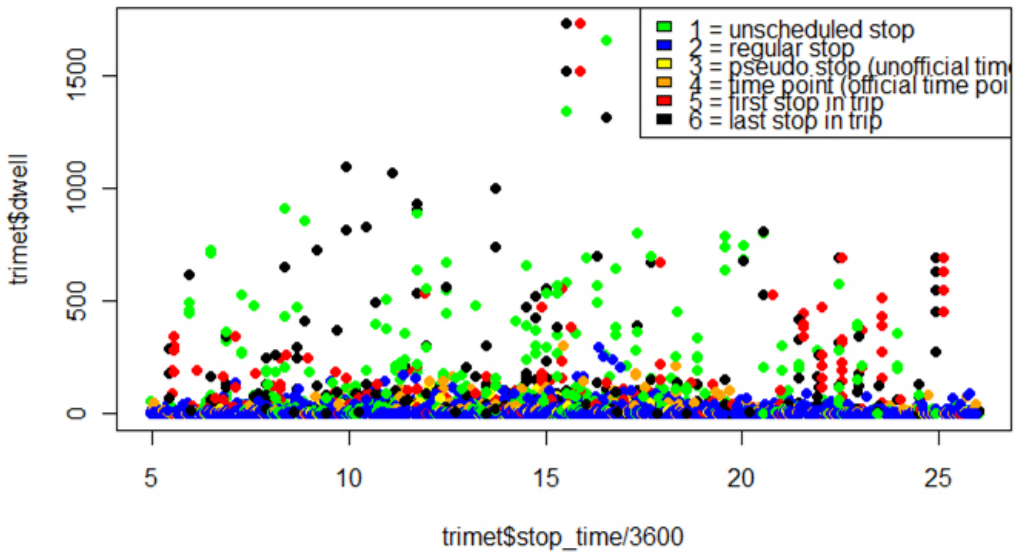


Figure 34 Plot of trimet$stop_time vs trimet$dwell with schedule_status as 3rd dimension

7. What does this plot reveal about the high dwell times?

This really is one of the strengths of plotting large amounts of data. If you had blindly calculated average dwell time, your result would be biased by including the first, last and some unscheduled stops. Perhaps these are relevant data points in the analysis that you will conduct, but this simple exploration helps you detect that.

Deliverable

Save your R script file. Clear the list of objects in R memory, then rerun your script to make sure all works. Revise your R code so that it includes just the answers to the seven questions in this activity. Include comments in your R script. Submit in the course dropbox.

Assessment

The following points are assigned and will be used to a score to your assignment.

All code executes and has good, clear comments =	4
Interesting par options =	2
Answers to questions =	4
Total points =	10

Your First Advanced Plot

ACTIVITY 16

Purpose

The purpose of this activity is to set the stage for the following activity where you will see some code samples from fellow students, ask questions, and provide some feedback.

Learning Objective

Become more aware of ways to solve problems in R

Required Resources

- R, R Studio
- TriMet Data Dictionary

Time Allocated

70 minutes out-of-class

Tasks

Some things to think about: direction, time of day (peak and off peak), good color choices

Now, let's just plot the x-y coordinates of the data frame.

```
plot(trimet$x_coord, trimet$y_coord)
```

It should look similar to a section of TriMet's Route 19 (See the resources for this activity).

1. Can you write code to add the estimated load to this plot as a 3rd dimension? Based on the sample code in Activity 15, one easy way would be to subset the data into load ranges, then plot those as colors. We will share code snippets in the next class!

Deliverable

You will share the output and logic behind this code with the class in the following activity, so submit your completed and fully functioning R code to the dropbox.

Assessment

This is a discovery exercise. Prepare a short write up to accompany your plot. Submit in pdf format to the class site. You will be assessed based on (see following page):

Activity 16 Grading Rubric

	Excellent (10)	Good (8)	Poor (6)	NONE
Discussion	Insightful discussion or commentary relating to the question at hand demonstrating student understanding of the task.	Discussion was competent regarding the lessons, but lacked in insightful discussion.	The discussion did not address the lessons or applicability of the activity.	Did not submit
Graphic	Intent of the graphics easily understood. Graphic includes legend, axes labels, title, etc. Information contained within the data was presented using creative formatting.	Graphic was missing 1 or 2 labels, legends, or other components. Graphic was relatively easy to understand with direct communication of information contained within the data.	Graphic is difficult to understand due to lack of labeling and visual complexity.	Did not submit
Quality	Document is typed, formatted, contains appropriate grammar and language.	Document has minor grammatical errors or inappropriate language.	Document was unorganized, contained inappropriate language and/or grammatical errors.	Did not submit

Code Sharing

ACTIVITY 17

We will share some code samples of adding the 3rd dimension to the trimet route.

Purpose

The purpose of this activity is to give you the opportunity to view code samples from fellow students, ask questions, and provide some feedback.

Learning Objective

Appreciate the variety of approaches to problem solving in R.

Required Resources

- R Studio

Time Allocated

30 minutes in class

Tasks

When asked, provide clarification on the code that is presented to the class.

Deliverable

Completed peer review form

Assessment

Peer assessment (see following page).

Reviewer's Name:

Activity 17 Grading Rubric

	Excellent (10)	Good (8)	Poor (6)
Graphic	Intent of the graphics easily understood. Graphic includes legend, axes labels, title, etc. Information contained within the data was presented using creative formatting.	Graphic was missing 1 or 2 labels, legends, or other components. Graphic was relatively easy to understand with direct communication of information contained within the data.	Graphic is difficult to understand due to lack of labeling and visual complexity.

	Student	Score
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		

Thinking Like a Computer: Pseudo-coding & Functions

ACTIVITY 18

Sometimes it is helpful prior to tackling writing code to spend just a minute and think about the problem and then jot down the general steps that you need to accomplish in order to complete the task. This pseudo-code need not be written in syntax but can provide a map that helps you to accomplish the code. The key to successfully turning your map into code will be to link these steps with what you know about R so far. The other key is to recognize that a computer script will only do what you tell it! We will also write our own functions in this activity.

Purpose

The purpose of this activity is to help facilitate thinking like a computer (i.e., in logical step sequence). You will also reinforce the looping logic and vector applications by writing the code for yourself.

Learning Objective

- Step-by-step thinking and analysis
- Pseudo-coding
- Writing functions

Required Resources

- R, R Studio
- TriMet dataframe

Time Allocated

50 minutes in class

Tasks

This activity begins assuming R Studio is open.
If not, please start R Studio and open the sample script for the activity (if provided).

A. Control Structures

Control structures such as loops and if-then statements are easy ways to control the execution of a set of commands. R's indexing of vectors make looping through code to generate plots very easy. Dalgaard has a quick summary on page 336.

Loops are the first syntax:

```
-----------------------------------------------------
# 5 - Control Structure
#-----------------------------------------------------
# Loops
#++++++++++++++++++++++++++++++
for ( j in 1:4){
print (j)
}
```

This loop will execute the code between the `{}` four times. Each time `j` gets incremented when step (e.g., `j=1, j=2, j=3, and j=4`).

Note you do not need to initialize the loops with a `j=1` or add a `j+1` at the last step nor does the code after the `in` statement need to be sequential. Let's look at this `for` statement:

```
x <- 1:10
```

```
y <- 1:10
col.list <- c("red", "blue", "green")
par (mfrow=c(1,3))

for ( k in col.list ) {
       plot (x,y, col=k, main=k, pch=16)
     }
```

The table shows the number of steps in the loop and the value of `k` in each step. See how the value of `k` changes based on the loop step? The resulting plot is shown below (Figure 35).

	Start of Loop	Step 1	Step 2	Step 3
Value of k	null	red	blue	green
col.list		col.list[1]	col.list[2]	col.list[3]

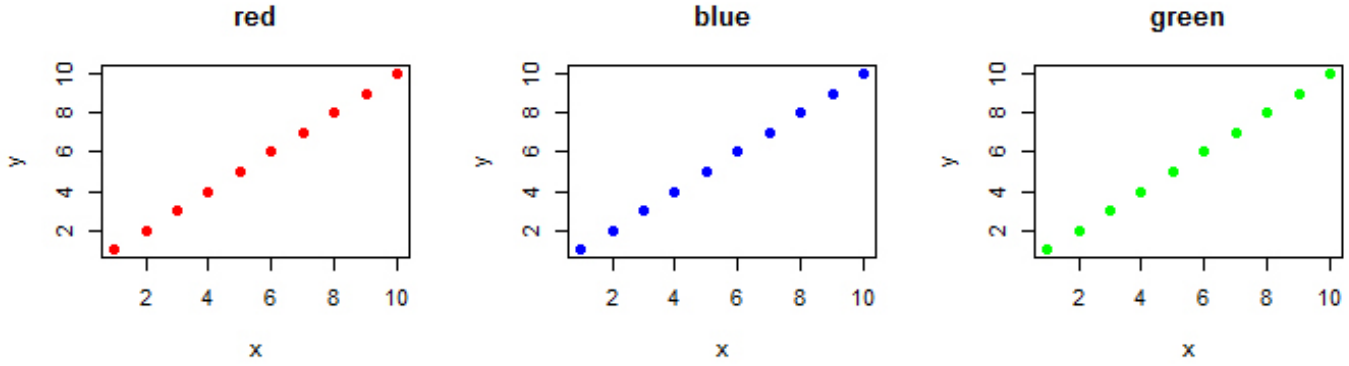


Figure 35 Plot outputs from sample loop code

Conditional statements such as `if-then` are the next most common control structure. R's syntax for defining your own function in code looks like this;

```
#if then conditions
#++++++++++++++++++++++++++++++
#if (condition is TRUE){
# do
# } else {
#do this
# }

mileposts<- c(22.1, 25.2, 43, 29.1,29.2,26.3)

for (j in 1:length(mileposts)) {
    if (mileposts[j]<27) {
      print("too slow")
    }   else {
      print("too fast")
```

```
    }
  }
```

In this code, R loops three times and evaluates `mileposts[j]`. If it is less than 27, it prints the logical evaluation of `mileposts[j]`. If this condition is not met, it executes the code in the `else {}` braces. A common error is not including the correct matching braces. R Studio highlights the matching pairs.

B. Functions

You have already seen built-in functions in R such as:

```
plot(x,option1=1, option2=2,option3=3)
```

The plot function takes the input `x` and returns a plot. Any options are also specified and passed to the function.

It is easy to write your own function in R. R's syntax for defining your own function in code looks like this:

```
testfunc <- function(x,y){
                    z <- x + y
                    return (z)
}
#testfunc is created (see object appear in R studio objects)
testfunc( 3600,86400 )
```

Here *testfunc* is created as a function. It takes inputs `x` and `y`, executes the code enclosed by `{ }` and returns the output of `z`. The return is optional, but is required if you have something specific you want the function to return. This is because the value of `z` is not stored by R (it will not show in your R Studio workspace, it is internal to the function operation).

A function will be helpful if you are planning to repeat a set of code again and again. It is helpful to first write working code, identify the variables you might want the function to take as input, then create the function.

C. Good Practice: Write Psuedo Code

I often find it is helpful to spend just a minute and think about the code I am trying to write. If you remember to think like a computer this will easily translate to code. Obviously the program will only do what you tell it.

Let's return to the code you wrote for the last activity using the data frame *trimet*.

1. Create a new R script file for this activity, copy the commands to load the *trimet* data frame and load the data from the course website.

You've been asked to create a function that creates plots for an entire day that show (1) pattern distance vs. load, (2) pattern distance vs. dwell, (3) pattern distance vs. max speed. You need to create these 3 plots for AM peak (6AM-9AM), off-peak (9AM-4PM), and PM peak (4PM-7PM). The user of your function will also want to specify direction.

If I wanted to write the pseudo-code it might look something like this :

1. Read in the *trimet* data table
2. Subset data frame
    a. To user specified date (*service_day*) and to user specified (*direction*).
        i. To AM Peak (*leave_time >= 6*3600 and < 9*3600*)
        ii. To Mid Day (*leave_time >= 9*3600 and <= 16*3600*)
        iii. To PM peak (*leave_time >= 16*3600 and <= 19*3600*)
3. Arrange a 3 column by 3 column plot window
4. *Check the y and x ranges of each so the plot will be on the same scale.*
5. Plot
    a. From data set (i), pattern_dist vs est_load, pattern_dist vs dwell, pattern_dist vs max_speed for AM peak.
    b. From data set (ii), Plot pattern_dist vs est_load, pattern_dist vs dwell, pattern_dist vs max_speed for mid-day
    c. From data set (iii), pattern_dist vs est_load, pattern_dist vs dwell, pattern_dist vs max_speed for PM peak.
6. Label the plots with *service_day* and *direction*

When I first translate my code and look at the plots, I notice that I missed step 4. If I don't check the range for entire data set, I won't be able to easily compare the plots for each day. I have a decision to make: Do I want the ranges to vary by day or should i hard code them so that each day looks the same? Right now, I opt for each day to be different.

The base code looks like this:

```
day <- subset(trimet, service_day==”2007-03-08” & direction==0)
am <- subset(day, leave_time >= 6*3600 & leave_time< 9*3600)
mid <- subset(day, leave_time >= 9*3600 & leave_time < 16*3600)
pm <- subset(day, leave_time >= 16*3600 & leave_time < 19*3600)

#Find the range for the plots, hard code some after inspection
x.dist <- range (day$pattern_dist)
y.load <- range (day$est_load)
y.dwell <- c(0,200)
y.speed <- c(0,60)

par (mfrow=c(3,3), oma=c(1,1,3,1)) #set up 3x3 plot window and create
an outer margin to label all the plots with oma
plot (am$pattern_dist, am$est_load, xlim=x.dist, ylim=y.load)
plot (am$pattern_dist, am$dwell, main=”AM Peak”, xlim=x.dist, ylim=y.
dwell)
plot (am$pattern_dist, am$max_speed, xlim=x.dist, ylim=y.speed)

plot (mid$pattern_dist, mid$est_load, xlim=x.dist, ylim=y.load)
plot (mid$pattern_dist, mid$dwell, main=”Mid-day”, xlim=x.dist, ylim=y.
dwell)
plot (mid$pattern_dist, mid$max_speed, xlim=x.dist, ylim=y.speed)

plot (pm$pattern_dist, pm$est_load, xlim=x.dist, ylim=y.load)
plot (pm$pattern_dist, pm$dwell, main=”PM Peak”, xlim=x.dist, ylim=y.
dwell)
plot (pm$pattern_dist, pm$max_speed, xlim=x.dist, ylim=y.speed)

#label the plot sets
mtext (line=0, “service_day==2007-03-08 & direction==0” , outer=TRUE,
font=2)
```

You could make a few more tweaks but seems to be useful and working for one day and direction.

To turn this into a function, what are the things that will need to change for each set of plots?

Circle those items in the code above before going on to the page.

What needs to be changed is highlighted:

```
day <- subset(trimet, service_day==”2007-03-08” & direction==0)
am <- subset(day, leave_time >= 6*3600 & leave_time< 9*3600)
mid <- subset(day, leave_time >= 9*3600 & leave_time < 16*3600)
pm <- subset(day, leave_time >= 16*3600 & leave_time < 19*3600)

#Find the range for the plots, hard code some after inspection
x.dist <- range (day$pattern_dist)
y.load <- range (day$est_load)
y.dwell <- c(0,200)
y.speed <- c(0,60)

par (mfrow=c(3,3), oma=c(1,1,3,1)) #set up 3x3 plot window and create
an outer margin to label all the plots with oma
plot (am$pattern_dist, am$est_load, xlim=x.dist, ylim=y.load)
plot (am$pattern_dist, am$dwell, main=”AM Peak”, xlim=x.dist, ylim=y.
dwell)
plot (am$pattern_dist, am$max_speed, xlim=x.dist, ylim=y.speed)

plot (mid$pattern_dist, mid$est_load, xlim=x.dist, ylim=y.load)
plot (mid$pattern_dist, mid$dwell, main=”Mid-day”, xlim=x.dist, ylim=y.
dwell)
plot (mid$pattern_dist, mid$max_speed, xlim=x.dist, ylim=y.speed)

plot (pm$pattern_dist, pm$est_load, xlim=x.dist, ylim=y.load)
plot (pm$pattern_dist, pm$dwell, main=”PM Peak”, xlim=x.dist, ylim=y.
dwell)
plot (pm$pattern_dist, pm$max_speed, xlim=x.dist, ylim=y.speed)

#label the plot sets
mtext (line=0, “service_day==2007-03-08 & direction==0” , outer=TRUE,
font=2)
```

So, we just have to figure out how to get R to the subset by accepting a variable as the logic for the subset. Fortunately, this is easy:

```
day.plt <- ‘2007-03-08’
dir.plt <- 0
test <- subset(trimet, service_day==day.plt & direction==dir.plt)
```

Now let's rewrite our code as a FUNCTION

```
trimet.plot <- function (df, day.plt, dir.plt) {
  day <- subset(df, service_day==day.plt & direction==dir.plt)
  am <- subset(day, leave_time >= 6*3600 & leave_time< 9*3600)
  mid <- subset(day, leave_time >= 9*3600 & leave_time < 16*3600)
  pm <- subset(day, leave_time >= 16*3600 & leave_time < 19*3600)

  x.dist <- range (day$pattern_dist)
  y.load <- range (day$est_load)
  y.dwell <- c(0,200)
  y.speed <- c(0,60)

  par (mfrow=c(3,3), oma=c(1,1,3,1))
  plot (am$pattern_dist, am$est_load, xlim=x.dist, ylim=y.load)
  plot (am$pattern_dist, am$dwell, main="AM Peak", xlim=x.dist, ylim=y.
dwell)
  plot (am$pattern_dist, am$max_speed, xlim=x.dist, ylim=y.speed)

  plot (mid$pattern_dist, mid$est_load, xlim=x.dist, ylim=y.load)
    plot (mid$pattern_dist, mid$dwell, main="Mid-day", xlim=x.dist,
ylim=y.dwell)
  plot (mid$pattern_dist, mid$max_speed, xlim=x.dist, ylim=y.speed)

  plot (pm$pattern_dist, pm$est_load, xlim=x.dist, ylim=y.load)
  plot (pm$pattern_dist, pm$dwell, main="PM Peak", xlim=x.dist, ylim=y.
dwell)
  plot (pm$pattern_dist, pm$max_speed, xlim=x.dist, ylim=y.speed)

  #label the plot sets
   mtext (line=0, paste("service_day==", day.plt, " & direction==", dir.
plt) , outer=TRUE, font=2)
}
```

This function takes three inputs (**df**, **day.plt**, **dir.plt**) and assigns them to these respective values in the function code. You can follow this in the code with color coding. Now to create our six plots, all that is required is:

```
trimet.plot (df=trimet, day.plt='2007-03-05', dir.plt=0)
```

Experiment with changing the date and direction. We will work as a class to turn this into a loop!

Deliverable

None, but save your R script file.

Assessment

None

Write Your Own Function

ACTIVITY 19

The collection of raw data provides researchers, engineers, and planners the opportunity to extrapolate conventional and understood metrics from a series of times, speeds, and other raw values. R is an excellent platform for such tasks as looping algorithms can be generated to calculate the required metrics. This activity explores the use of looping using the *trimet* data frame to generate headway values between bus stops.

Purpose

The purpose of this activity is to allow you to further investigate the looping function and the applicability of R in generating algorithms to process raw data.

Learning Objective

Use the loop with your own function to generate a working algorithm.

Required Resources

- R
- R Studio

Time Allocated

60 minutes in class
20 minutes out-of-class

Tasks

Imagine that you have been asked to create a scatterplot of the time between buses (headway) by time of day at each *stop_id* in the dataframe *trimet*. To keep this simple, let's do this for just one day: March 8, 2007. We can ignore *direction* since stops are numbered individually.

Your first task is to write the pseudo-code and share it with your instructor. You can type this, but you can also just write it on a blank sheet of paper. Write the steps out such as:

7. Read in the *trimet* data table
8. Determine the possible set of stop ids
9. …..

A sample of the plot for headways at one stop (2107) is shown in Figure 36.

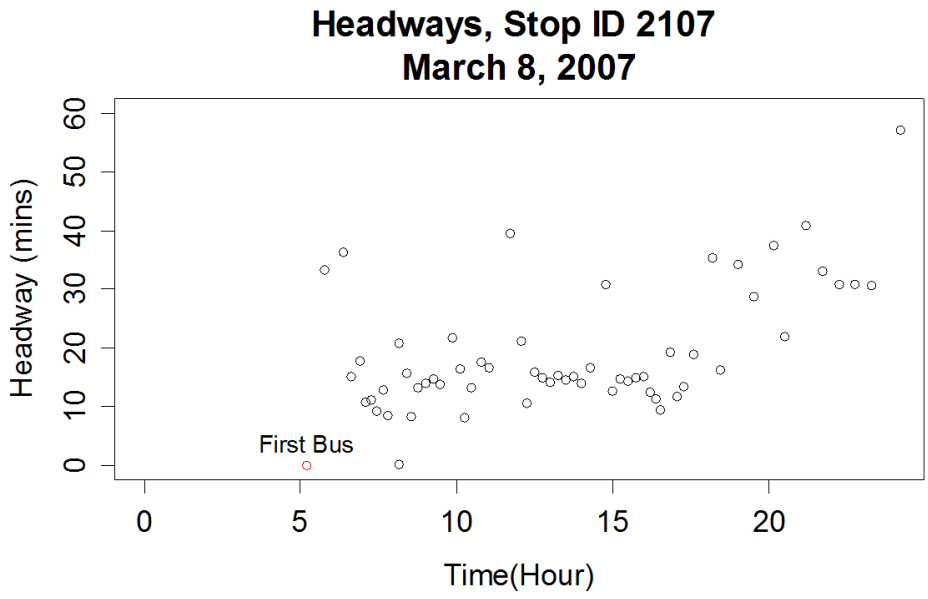


Figure 36 Headways at Stop ID 2107, March 8, 2007

Some tips for you to think about:

- You are only required to plot the data (not store it).
- There are 3 columns for time at each stop; use the *arrive_time* to calculate headway
- Take a look at the time order of the data (plot the *arrive_time* field), as it may need to be sorted in order to determine headways. This can be done with order function. The syntax is rather clumsy: ***http://www.statmethods.net/management/sorting.html***. Note in this example the dataframe is "attached" which allows the sorting column (`mpg`) to be referred to without its full name `mtcars$mpg`.

Now, write the code to do this. Remember to incrementally build your code. Be sure to add the **stop id** to the main titles of the graphs. Arrange the plots in a 4×4 window. Save your R script file. Clear the list of objects in R memory, then rerun your script to make sure all works.

Deliverable

Submit your script file to the class dropbox submission. Include comments on constructing this function + loop.

Assessment

Activity 19 Grading Rubric

	Excellent (10)	Good (8)	Poor (6)	NONE
Script	Organized, complete, accurate and executes.	Missing minor parts, but executes and is otherwise organized and accurate.	Missing significant portions of the activity, unorganized, inaccurate, but executes.	Code does not execute
Graphic/ Annotation	Graphic and/or annotations are complete and describe what the code is accomplishing.	The graphic(s) or some annotations are incomplete or do not describe what the code is accomplishing.	No graphics or annotations were provided.	Code does not execute
Discussion/ Commentary	Insightful discussion or commentary relating to the question at hand demonstrating student understanding of the task.	Discussion or commentary was incomplete.	Minimal to no discussion or commentary.	Code does not execute

Connecting to the Class Database via RODBC and PostgreSQL Drivers

ACTIVITY 20

This is an independent exercise to connect you to the class database if your instructor has setup a course database for you to use. If there has not been a database set up for you, you will not need to do this activity. You only have read access to the database. *Note that if you are trying to connect via RODBC on your personal PC, you will need to download the PostgreSQL ODBC driver from the web.*

Purpose

The purpose of this activity is to help familiarize you with reading in external data frames for analysis in R.

Learning Objective

Create a direct connection to PostgreSQL with R

Required Resources

- R Studio
- PostgreSQL ODBC driver

Time Allocated

20 minutes in class

Tasks

We will connect directly to the class PostgreSQL database using the RODBC package.

A. Connecting with PhpPgAdmin

If your instructor has set up a read-only access to the database you will need to write down the password:

```
User name= XXXXXXX [get from instructor]
Password = XXXXXXX [get from instructor]
```

B. Reading from PostgreSQL into R Using RODBC

We can issue SQL statements and return data frames. Or we can process the entire data frame in R using its subset tools. This will depend on how large the file is; R can sometimes be a "memory hog." You can execute any of the SQL queries we have reviewed, including the `GROUP BY` options.

To install the PostgreSQL RODBC driver, go to ***Control Panel***, ***Administrative tools*** and open the ***ODBC Data Sources***. The ODBC Data Source Administrator window will open. Select the ***User DSN*** tab and click on ***Add***… (See Figure 37.)

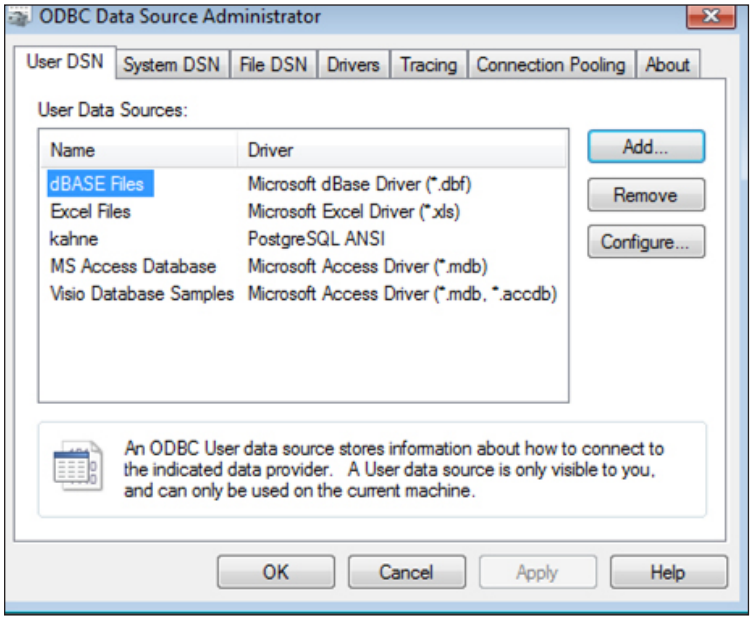


Figure 37 ODBC Data Source Administrator pop-up window

Scroll to the PostgreSQL ANSI(x64) driver, highlight the driver and click ***Finish***.

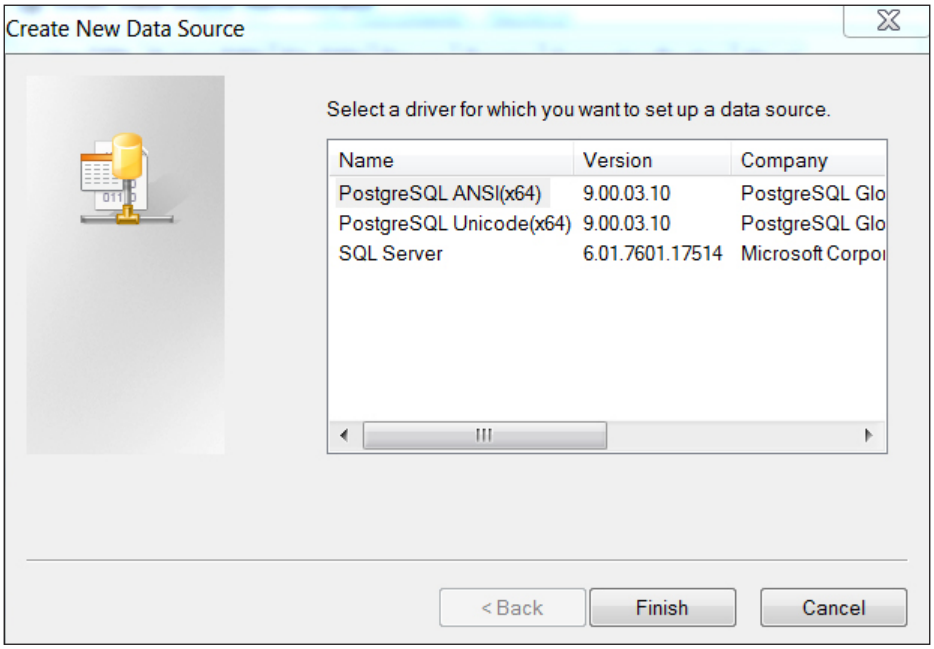


Figure 38 PostgreSQL ANSI driver pop-up window

The PostgreSQL ANSI(x64) ODBC Driver (psqlODBC) Setup window will appear (see Figure 38) and require you to input information to complete the installation.

The following are the options:

```
Datasource= A name you choose for the driver. Keep it simple, no spaces.
Database= the name of the database (i.e., ce510)
Server= ip of server with database (i.e., db.cecs.pdx.edu)
User name= xxxxxx
Description = optional
SSL mode = allow
Port= 5432
Password = xxxxxx
```

In R Studio, download and install the package **RODBC**

Deliverable

None – you will need this for all future work in the class.

Assessment

Participation points for this activity

Using R with PostgreSQL

ACTIVITY 21

Now that the connection between PostgreSQL and R has been established, access to the remaining data sets is available. More significantly, the data from PostgreSQL can be queried and aggregated as needed rather than requiring R to process through all the data. This saves programming and processing time, an important consideration for projects that, even when narrowed, contain substantial amounts of data.

Purpose

The purpose of this activity is to introduce some applications of sqlQuery() and allow you to explore the connection between R and PostgreSQL with emphasis in querying and aggregating information.

Learning Objective

Use SQL queries to pull data from SQL based resources into R.

Required Resources

- R, R Studio
- ODBC connection to PostgreSQL

Time Allocated

60 minutes in class

Tasks

A. Install and Load the RODBC Package

To make a call to RODBC we use this set of commands:

```
channel <- odbcConnect("ce510", uid="ce510")
qry <- " SELECT * FROM wim.wimdata WHERE timestamp >= '08-08-2009' AND
timestamp < '08-09-2009'"
wim <-sqlQuery(channel, qry)
```

`Channel` opens the connection to the ODBC driver you set up in Activity 20. The first "`ce510`" in the `odbconnect()` call is the name of the driver in your windows ODBC library. This is the "Datasource" you named. The uid="`ce510`" is the name of the user that is authorized to access the PostgreSQl database. This is the user name you set in Activity 20. The variable `qry <-` is just a holder of the text string to pass to the database. The line wim `<-sqlQuery(channel, qry)` calls the `sqlQuery ()` function, passes the text in `qry` to the `channel`, then stores the return in a dataframe that will be named `wim`. For the most part, the datatypes will be handled much better in this rather than reading. For the most part, the datatypes will be handled much better using RODBC than when reading the files in from CSV or other text file.

B. R Operations or SQL Operations

You can be ambidextrous when deciding whether to use R or SQL. In this activity, we will show you some simple comparisons of R and SQL operations. Before we start, it is helpful to narrow the date range in the WIM data frame since we may not want to query in all records (there are 1.4 million!)

```
qry <- " SELECT min(timestamp), max (timestamp) FROM wim.wimdata "
range <-sqlQuery(channel, qry)
```

So, for the purposes which follow, lets read in the data from the wim schema for the WIM records for just one day

```
qry <- " SELECT * FROM wim.wimdata WHERE timestamp >= '08-08-2009' AND
timestamp < '08-09-2009'"
wim <-sqlQuery(channel, qry)


plot (wim$timestamp, wim$gvw)
```

We can easily add another criteria to the SELECT statement to be only type 11 trucks (5-axle semis)

```
qry <- " SELECT * FROM wim.wimdata WHERE timestamp >= '08-08-2009' AND
timestamp < '08-09-2009' AND type='11'"
wim.11 <-sqlQuery(channel, qry)
```

You could also do this in R from the original wim dataframe

```
wim.11R <- subset(wim, type==11)
```

In R Studio workspace, using the number of records, confirm that the subsets are the same. Alternatively, use plots:

```
par (mfrow=c(1,2))
plot (wim.11$timestamp, wim.11$gvw, main=paste("Num Recs", nrow(wim.11),
"\nAvg GVW", mean(wim.11$gvw)))
plot (wim.11R$timestamp, wim.11R$gvw, main=paste("Num Recs",
nrow(wim.11R), "\nAvg GVW", mean(wim.11R$gvw)))
```

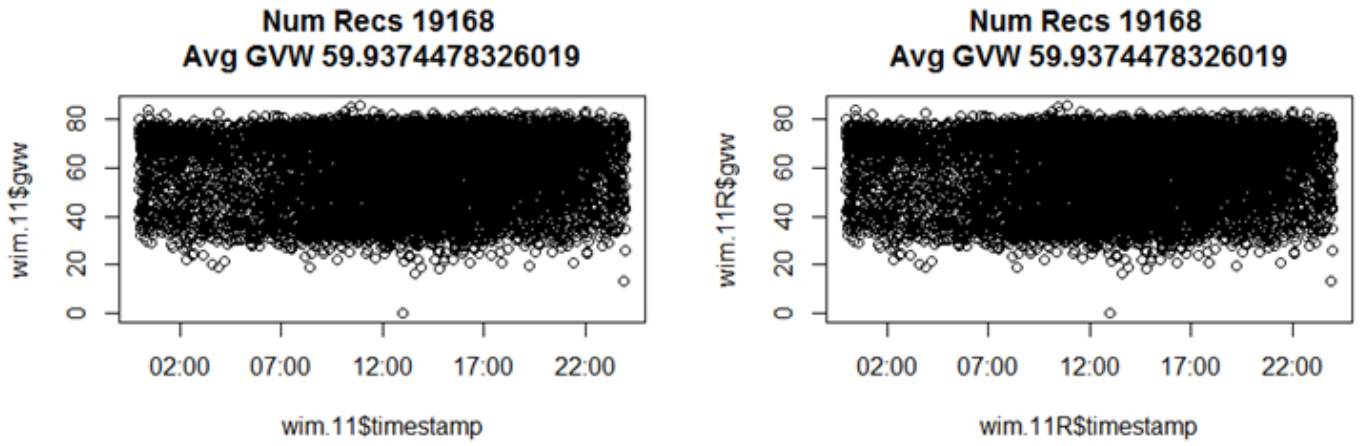


Figure 39

Note that since each record is a truck, we can count up records to get flow

```
qry <- " SELECT hour, COUNT(hour) FROM wim.wimdata WHERE timestamp >=
'08-08-2009' AND timestamp < '08-09-2009' GROUP BY hour"
wim5 <-sqlQuery(channel, qry)
barplot (wim5$count, names.arg=wim5$hour) # hours not in order!
```

```
qry <- " SELECT hour, COUNT(hour) FROM wim.wimdata WHERE timestamp >=
'08-08-2009' AND timestamp < '08-09-2009' GROUP BY hour ORDER BY hour"
wim5 <-sqlQuery(channel, qry)
```

In R, we can use the table function to count

```
table (wim$hour)
```

If you want to turn the table return into a data frame, use the following syntax

```
wim5R <- as.data.frame( table (wim$hour))
names(wim5R) <- c("hour","count")
str(wim5R)
```

Show plots

```
par (mfrow=c(1,3))
barplot (wim5$count, names.arg=wim5$hour)
barplot (wim5R$count, col="dodgerblue") #use the dataframe
barplot (table (wim$hour), col="green") #just use the table function
return
```

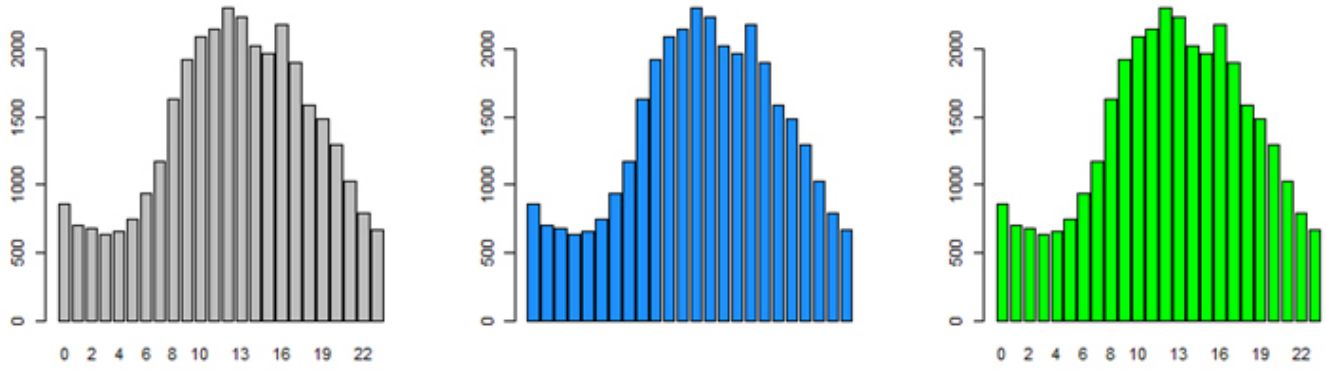


Figure 40

Use SQL to calculate average GVW by station

```
qry <- " SELECT stationnum, avg(gvw) AS avggvw FROM wim.wimdata WHERE
timestamp >= '08-08-2009' AND timestamp < '08-09-2009'
GROUP BY stationnum
ORDER BY stationnum"
gvwbysta <-sqlQuery(channel, qry)
barplot (gvwbysta$avggvw, names.arg=gvwbysta$stationnum)
```

Doing aggregate operations in R is also easy (see Dalgaard, page 75 for more tips). A built-in function called `tapply` does the grouping operation over any function that you can apply.

```
tapply (wim$gvw, wim$stationnum, mean)
```

It takes the variable gvw, groups it by *stationnum*, then applies whatever function you specify (here, the mean). There is no limit to the types of operations that can be applied using `tapply` hence it is more flexible than the SQL aggregate options.

```
par (mfrow=c(1,2))
barplot (gvwbysta$avggvw, names.arg=gvwbysta$stationnum)
barplot(tapply (wim$gvw, wim$stationnum, mean), col="dodgerblue")
```

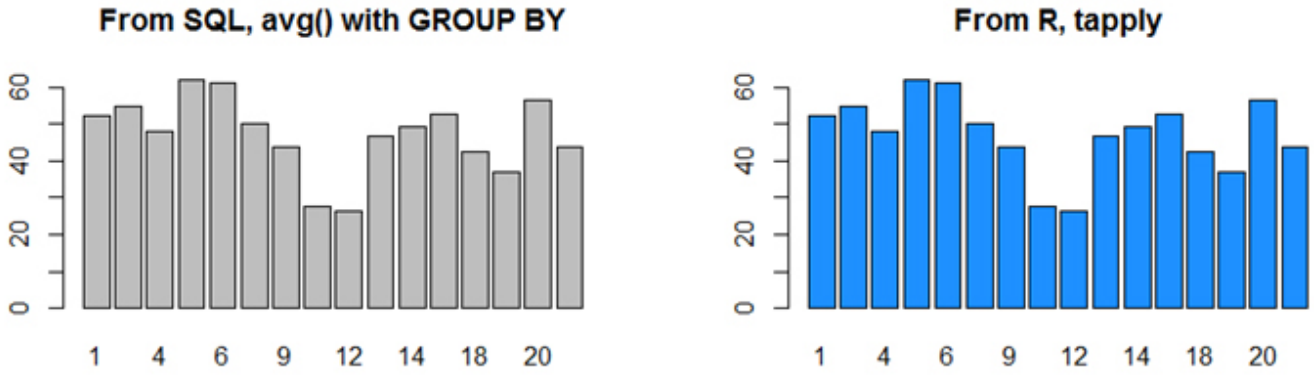


Figure 41

C. Group Collaborations

Now it is your turn to try to show your skills working with SQL and R selection and grouping methods. Working with a partner, you will assigned by the instructor one of the data sets to explore further. We will post these codes to a Google doc so that the instructor can share your code with the class.

First, after being assigned the data frame select the variable that appears most interesting to you. Write a SQL statement to select "ALL" of the records. You will then show how these records could be subset in SQL and R, how the GROUP BY and table functions return the same values, and how the GROUP BY aggregate functions (`avg`, `stddev`, `sum`, `variance`, `min`, `max`) and match with the tapply options in R. To "prove" they match you are required to make the comparison plots in R as shown in the script examples.

The instructor will give you the data set (*incident*, *trimet*, *loopdetector*, *weather*, *bicycle*). Your team should browse the tables and select the primary variable to work with and what a logical subset and grouping variable should be. In the example, we picked gvw and subset by type (after limiting to trucks for one day). Your script needs to produce, in a 2 column by 3 row arrangement, the following comparisons (based on the sample R script):

SQL STATEMENT	R OPERATIONS
PLOT 1 : SELECT A SUBSET OF RECORDS	PLOT 2 :subset (df, criteria)
PLOT 3 :SELECT COUNT() GROUP BY	PLOT 4 :table (df$col)
PLOT 5: SELECT GROUP BY AGGEGATE FUNCTION OPTIONS = avg, stddev, sum, variance, min, max	PLOT 6: tapply (df$variable, df$groupby, fun) OPTIONS FOR fun = mean, sd, sum, var, min, max

The corresponding R Script for the activity includes sample code of what would this look like for the following options:

1. Data set: wim
2. Selection of "ALL" records= all trucks from August 8, 2009

3. Subsetting variable = trucks where type=11
4. Grouping variable = hour

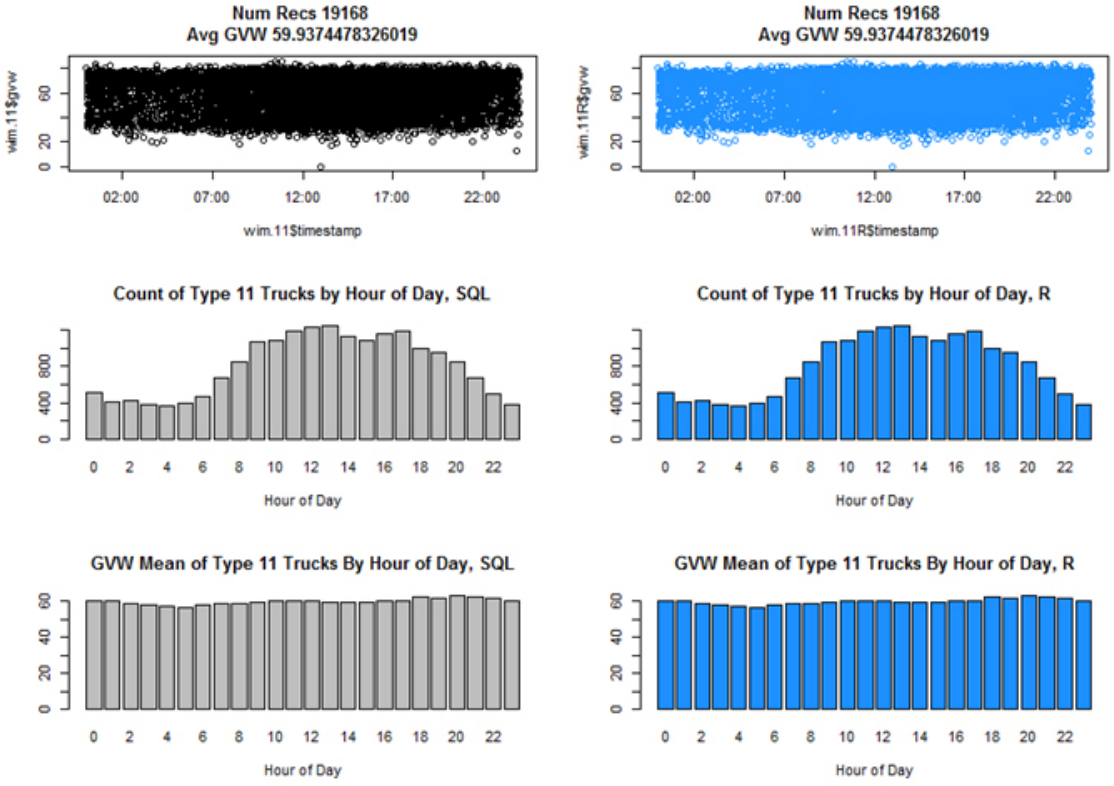


Figure 42

Deliverable

A completed R code posted on the Google doc and your explanation shared with the class.

Assessment

Participation!

Student Notes

Packages

ACTIVITY 22

Certainly one of the strengths of R is the open-community that presents so many packages to do so many things. All R functions are stored in packages. Because only the default packages are loaded when R starts, a user who wishes to add additional functions must download, unpack, and call the package into each R session. You will find that there are some nice packages that are very helpful with analysis or plotting. Finding the packages you need and getting them to work can be a little overwhelming but the websites listed in resources should make things a bit easier!

Purpose

The purpose of this activity is to help you discover the vast library of R packages and to help you sort some of the wheat from the chaff in the R packages.

Learning Objective

Communication to peers and investigation of R packages.

Required Resources

- *http://unknownr.r-forge.r-project.org/toppkgs.html*
- *http://crantastic.org/*

Time Allocated

35 minutes out-of-class
15 minutes in-class

Tasks

Review the list to see what packages you think might be most interesting for you to present. Rather than present an obscure package, try to select one of the higher rated packages from one of the Required Resources links. Download and install the package, then load it in an R session with the call:

```
library (packagename)
```

Your task is to prepare a two-slide PowerPoint that gives an overview of the package. Briefly describe what the package does and, if possible, share an example by exploring the package using the data involved in this class (i.e., from the class database). To avoid duplication, once you decide on what package you will present to the class, claim it in the "Package" discussion in the class course management site. First come, first served.

Deliverable

PowerPoint and brief presentation to the class in the next class period.

Assessment

This is a short response activity. Your score is based on the quality of your communication. A peer assessment form will be provided.

Activity 22 Grading Rubric

	Excellent (10)	Good (8)	Poor (6)
Communication	Intent of the presentation easily understood. The general intent of the package was clearly communicated.	Communication of the package was satisfactory, but confused some of the general ideas of the package.	Communication of the package was ineffective.

Student Notes

Working with Time in R

ACTIVITY 23

This activity orients you with time class operations, time string and time plotting functions in R. This is a somewhat tricky topic, mostly because of the way that the basic packages within R handle time operations. This activity exposes some of the intricacies of time class operations and introduces the *lubridate* package which takes the complexity of working with time and simplifies the scripting process by generating more intuitive functions for working with time.

Purpose

The purpose of this activity is to familiarize you with POSIXlt and "POSIXct" time classes operations in R, and ways to format and convert time objects, and plot with time series axes by using R.

Learning Objective

To develop an understanding of how R stores and displays time-based data and some techniques for manipulating, displaying and modifying.

Required Resources

- R, R Studio
- R code on working with time
- Packages: RODBC and lubridate

Time Allocated

60 minutes out-of-class

Tasks

"POSIXlt" and "POSIXct" are two main classes that represent calendar dates and time to the nearest second. Class "POSIXct" is a numeric vector that represents the (signed) number of seconds since the beginning of 1970. Think of "ct" as continuous time. An object that is POSIXct is stored in R as a decimal, in seconds since 1-1-1970.

When you plot time or look at the data at the R prompt, R knows this is a time/date and displays it in a more human readable format: POSIXlt. A POSIXlt is a vector of characters that represent time (e.g., month, day, hour, second, timezone, etc). With the POSIXlt class you can control the format or order of how the characters are displayed (e.g., MM-DD-YY or YYYY-Month). Think of this like changing the "format" of date/time cells in Excel. You can convert from one to the other pretty easily (and R is usually fairly smart about it), or you can use the functions to explicitly make conversions. For example, as.POSIXlt() converts POSIXct class values to POSIXlt values.

As mentioned in the introduction to this activity, there is a package available that greatly reduces the complexity of working with time by making the process more intuitive. A prime example of this is the function now() which returns the system time the same as Sys.time() and today() which returns the same value as Sys.date(). The following sections describe both the R way of working with time and the *lubridate* simplification.

Let's start with a sample working with time.

```
Sys.time()
```

This function returns the system's idea of the current date with time.

```
[1] "2012-07-04 11:58:47 PM"
```

Similarly `Sys.Date()_` returns the date without the time.

```
[1] "2012-07-04"
```

Logical comparisons and limited arithmetic are available for both `POSIXlt` and `POSIXct`. One can add or subtract a number of seconds from a date-time object, but not add two date-time objects. Subtraction of two date-time objects is equivalent to using the function difftime. Be aware that `POSIXlt` objects will be interpreted as being in the current time zone for these operations, unless a time zone has been specified.

A. Paste Function

You have not yet seen this, but the paste function is very helpful for "adding" strings and variables together to form a new string. Try this code in R

```
string <-   paste ("hello", "world", sep=" ")
print (string)
text1 <- "hello"
text2 <- "world"
print ( paste (text1, text2, sep=" ")
```

You can use this function so many ways; it will be a common tool in your toolbox!!

B. Convert and Format

Numeric input is first converted to class `POSIXct`. Similarly, character input is first converted to class `POSIXlt` by `strptime` function. Try the following two lines of code:

```
str(as.POSIXct(strptime(paste("02/28/92","07:03:20"),"%m/%d/%y
%H:%M:%S")))
str(as.POSIXlt(as.POSIXct(699260600,origin=ISOdateti
me(1970,01,01,0,0,0))))
```

Either format or `as.character` converts both `POSIXlt` and `POSIXct` to character vectors. Conversion to and from character strings require definition of which values are days, months, AM/PM indicator and separators in formats such as %x. A full list of the multitude of character options can be viewed using `?strptime`. Let's explore some of the options!

First, let's see what time it is:

```
Sys.time()
```

Notice how the format is "Year-Month-Day Hour-Minute-Second"? This is the standard form and is coded like this:

```
format(Sys.time(), format="%Y-%m-%d %H:%M:%S")
```

To rearrange the values or change what time format is displayed, change the character to the wanted outcomes within the formal argument `format=" "` demonstrated by the following examples;

24-Hour Clock:

```
format(Sys.time(), format="%m-%d-%Y %H:%M:%S")
```

12-Hour Clock with AM/PM:

```
format(Sys.time(), format="%m-%d-%Y %I:%M:%S %p")
```

Numerical Month:

```
format(Sys.time(), format="%m")
```

Abbreviated Month Name:

```
format(Sys.time(), format="%b")
```

Full Month Name:

```
format(Sys.time(), format="%B")
```

Redundant:

```
format(Sys.time(), format="%Y-%y %B")
```

Lubridate simplifies the conversion of dates with the use of a function in the form of

```
ymd_hms()
```

Use of this function format simplifies the formatting of time and dates while still allowing for diverse options regarding the formatting of time found within the basic R packages. *Lubridate* refers to these functions as parsing and recognizes *y* as year, *m* as month, and *d* as day. Similar to the basic R functions, *lubridate* uses *h* as hour, *m* as minute, and *s* as second. Although possibly confusing the double use of *m*, *lubridate* uses a function structure as opposed to the argument used by the `strptime()` function within the basic R packages. A list of the parsing functions can be found using the help call, `?lubridate`. In some situations, text character strings need to be defined as `POSIXlt` or `POSIXct` time classes. To do this, use the `strptime()` function to define how the text string is structured. For example, if the text string is "July 4, 2012" then the command in R would be:

```
strptime("July 4, 2012", "%B %d, %Y")
```

The important thing to remember when converting text strings to `POSIXct` or `POSIXlt` classes is that the spaces and delimiters like commas are necessary in the object argument of the function. In the above example, the comma is after the day of the month, just as it is in the original text string.

The result of converting text strings without class specification is `POSIXlt`. However, as already demonstrated, a simple conversion can be completed to define the class as `POSIXct`. Now change the code in the script provided, getting comfortable with converting text strings and format changes.

If a numeric representation is given, the `ISOdatetime()` function can generate a `POSIXct` class. The function is relatively simple, requiring only integer entries for the object arguments. Or,

```
ISOdatetime(2012,07,04,14,0,0,tz = "")
```

Which returns,

```
[1] "2012-07-04 14:00:00 PDT
```

Use the provided script to explore the use of the vector generating functions to create a series of `POSIXct` time classes.

C. Time Zone

In the previous task we changed the format of the date and time without consideration of the time zone. R assumes the current time zone unless otherwise specified (`tz=""`).

With `POSIXlt` the time zone is an item in the list or,

```
POSIXlt(x,"EST")
```

With `POSIXct` the time zone is an attribute that you can define or,

```
POSIXct(x,tz=" ")
```

Beware that some operations will cause vectors to lose their attributes. This means that if you have defined a time zone (i.e., `tz="GMT"`), you could lose it and R will revert to the current time zone. Always double-check your plots to ensure that the hours are correct (this includes daylight savings time effects). For example,

```
as.POSIXct(1472562988, origin="1965-01-01", tz="GMT")
[1] "2011-08-31 13:16:28 GMT"
as.POSIXct(1472562988, origin="1965-01-01", tz="")
```

[1] "2011-08-31 14:16:28 PDT"Since the UTC does not undergo changes like daylight savings, an unwanted change in the time could occur as shown above.

As is demonstrated by the use of as.`POSIXct` function, the assignment of an origin is required to establish a time datum. In *lubridate*, the origin date (1970-01-01 00:00:00 UTC) is assigned as origin for convenience. All numeric representations of time and dates are based on this datum without the necessity of forcing an origin. If a different origin is wanted or warranted, the use of the `as.POSIXct` function can provide the appropriate change. Another difference between the basic R time package and *lubridate* is with respect to the establishment of a time zone. The majority of functions within the *lubridate* package generate values based on UTC, but can be changed by the `with_tz()` function within *lubridate*. If the only change that is wanted is to change the time zone element the function `force_tz()` can be used. However, it should be noted that the time difference between time zones will change as it does with the use of the `with_tz()` function. Explore the differences between these two functions within the R script provided for this discovery exercise.

D. Plot in Time Series Axes

Now to make `plot()` do what you want, you have to learn make "custom" axes. It is good to think of figure creation in R as "painting." Each command you issue adds that element to the plot. When R creates a plot, the first thing it does is draw the plot region. This is scaled to the data (though you can control with `xlim` and `ylim` options). If you suppress the x-axis, you can "paint" on your own, but you have to tell R where to put the tick marks and what to label the ticks with. The axis will be at the same scale as the figure region you created. This might make sense when you run the script.

All three plots represent data in time series. With the axis function, you can get more control of labels and their positions. (See Figure 43.)

E. Generate Data with Time Parameters

In the previous sections of this activity we explored how to convert and work with time. Therefore, in this final section we will apply what we have learned to the loop dataset. Let's start with a plot of the subset data from the most recent section.

```
plot(ll$starttime, ll$volume)
```

We can see from the daily data that volume decreases after midnight and increases throughout the morning and into the late afternoon where the volume begins to decrease steadily. Is this indicative behavior of OR-217 SB for a weekday? To answer that, plot a week's worth of data from February.

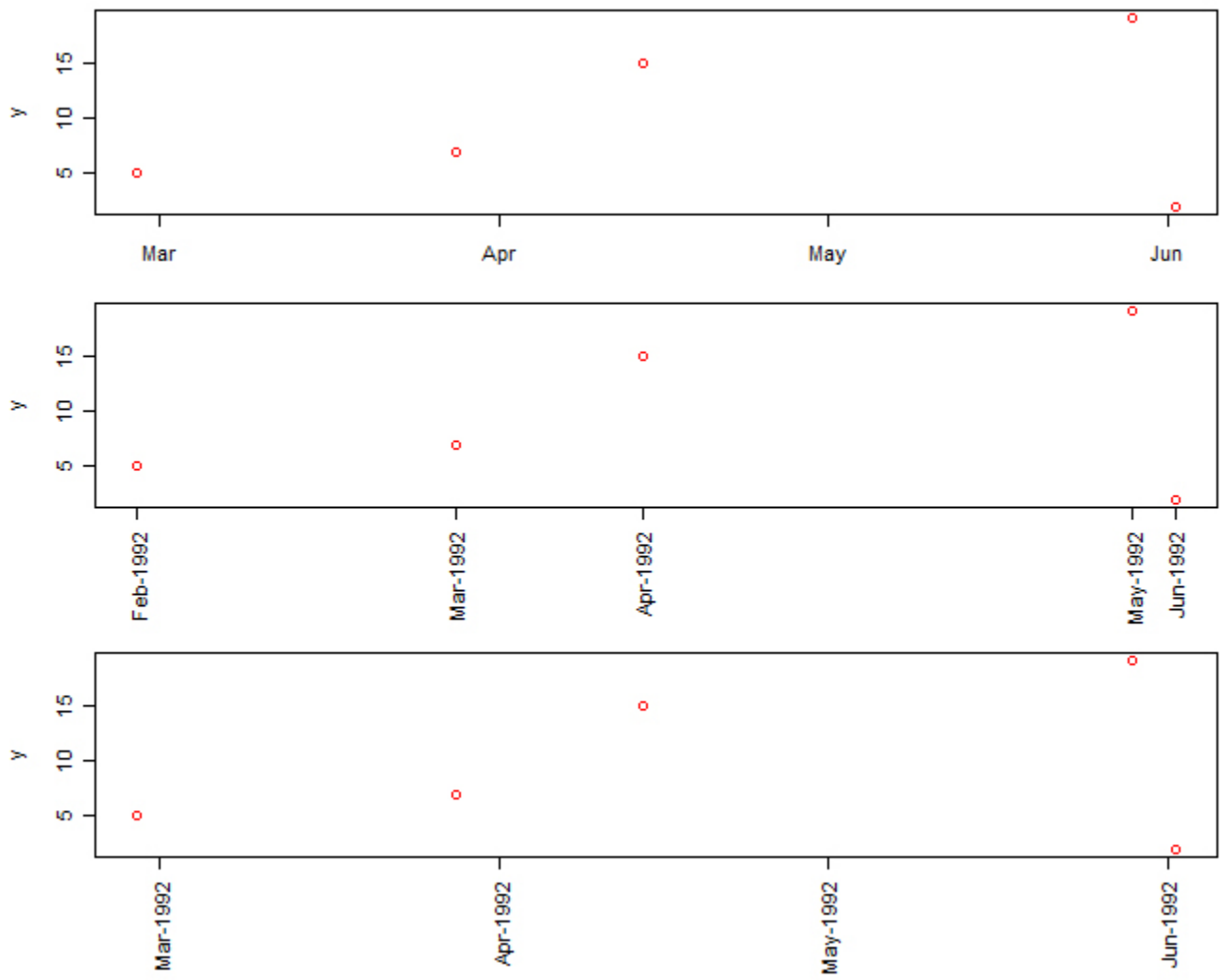


Figure 43 Plots showing R time

```
loopfeb<-subset(loop, starttime < force_tz(ymd(20090210), "America/Los_
Angeles")
                    & starttime >= force_tz(ymd(20090203),
                "America/Los_Angeles"))
plot(loopfeb$starttime, loopfeb$volume)
```

Describe the differences between the previous two plots and provide suggestions for similarities.

To take this one step further we can use *lubridate* to further narrow down the data without having to the generate a looping function. Instead we can use the `hour()` function to generate a look into the afternoon hours of the week of 2/3/2009- 2/10/2009. This is accomplished with the following script:

```
lll<-subset(loop, hour(starttime) >=14 & hour(starttime) < 18
          & starttime < force_tz(ymd(20090210),
                "America/Los_Angeles")
          & starttime >= force_tz(ymd(20090203),
                "America/Los_Angeles")  )
```

After plotting the data, we see the volumes between the hours of 2:00PM and 6:00PM for every day within the week of data (Figure 44).

F. Create a Plot

Your final task is to generate a plot of the speed data for the first two weeks of February speed data using the loop dataset. Only include weekdays. Briefly discuss the basic descriptive statistics (i.e., mean, standard deviation) detailed analysis is not required. In order to detect differences, try to plot both time series on the same graph or arrange them so that you can see differences

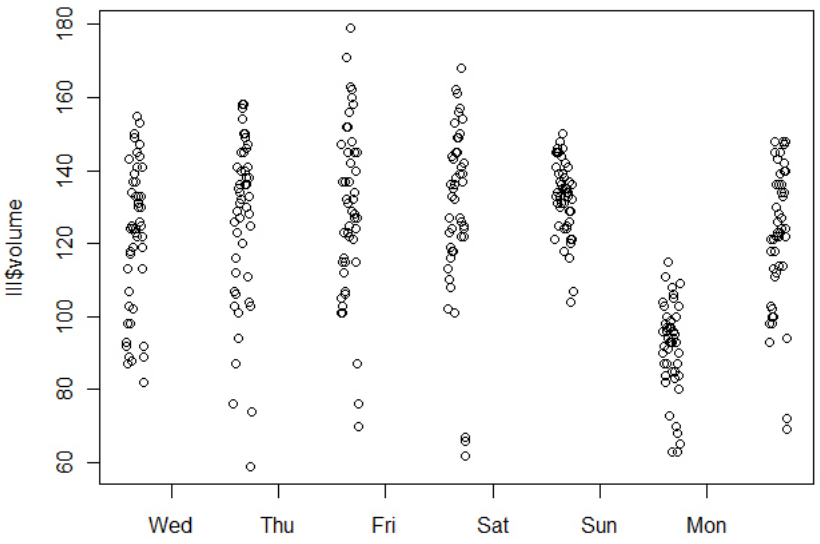


Figure 44 Volume on SB OR-217at OR-10 2:00PM-6:00PM in five minute intervals for a seven day period

Deliverable

Submit the cleaned R code generating the work week subsets and basic statistics to the class website dropbox.

Assessment

This is a discovery activity where you put together a variety of things you have learned to this point. Don't worry about going into detail about why it worked (assuming it did), but do upload your R code, cleaned up and commented upon, to the class dropbox.

Activity 23 Grading Rubric

	Excellent (10)	Good (8)	Poor (6)	NONE
Script	Organized, complete, accurate and executes.	Missing minor parts, but executes and is otherwise organized and accurate.	Missing significant portions of the activity, unorganized, inaccurate, but executes.	Code does not execute
Annotation	Annotations are complete and describe what the code is accomplishing.	Some annotations are incomplete or do not describe what the code is accomplishing.	No annotations were provided.	Code does not execute
Commentary	Demonstrated active engagement with exploring the various options of the functions within the activity.	Demonstrated some minor changes to the instructors code.	No changes and thus no commentary that demonstrates active engagement with exploring the options within the activity.	Code does not execute

CHAPTER 4 Using Graphics for Exploratory Data Analysis

"Use a picture. It's worth a thousand words."
Arthur Brisbane (1864-1936) Newspaper Editor"

There are a lot of good reasons to use a graphical-based approach to data mining. The first is that you are able to process much more information graphically rather than when reading data from tables. Prior to any complex statistical analysis of data sets, an exploratory glance of the data is required to understand the distribution characteristics of the data, in an effort to apply the best available method of statistical testing. This chapter further expands on the development of more complex graphics and statistical analysis within R.

There are two types of variables: discrete and continuous. This chapter focuses on the exploratory graphics available for both types of variables and recounts methods of descriptive statistics and distribution diagnostics. It also presents an introduction to working with time variables in R because time classes in R can be a daunting unless you have a base understanding of how R handles the two time classes.

The majority of the chapter is concerned with the exploration of data sets, both graphically and statistically. A basic review of descriptive statistics and an introduction to the statistical functions within R are presented. Following this, confidence intervals and hypothesis testing are presented through an in depth look into the Z and t distributions for single continuous variables. Multivariate analysis is also presented in the form of the ANOVA test and appropriate post-hoc tests. The chapter concludes with activities introducing and expanding upon multivariate graphics using previously presented functions and introduces several packages that generate refined graphics for two continuous variables.

Activity List

Activity Type	Number and Title	Assessment Type
In Class	Activity 24: Interactive Review of Basic Statistics Using R	
Out-of-class	Activity 25: Basic Charts for Single Discrete Variable	Short quiz
In Class	Activity 26: Exploring Single Discrete Variable Plots	Participation
Out-of-class	Activity 27: Exploratory and Diagnostic Plots for the Distribution of a Single Discrete Variable	Short quiz
Out-of-class	Activity 28: Probability Distributions	Short quiz
In Class	Activity 29: Kernel Density Estimates and Histograms	Short Response
In Class	Activity 30: Diagnosing a Distribution	Short Response

Activity Type	Number and Title	Assessment Type
Out-of-class	Activity 31: Depicting the Distribution Involving Discrete Variables	Short quiz
Out-of-class	Activity 32: Depicting the Distribution of Two Continuous Variables	Short quiz
In Class	Activity 33: Introduction to Final Project Topics	Short Response
Out-of-class	Activity 34: One and Two Sample Tests	Short quiz
I/O Class	Activity 35: Exploring Confidence Intervals & Simple Hypothesis Testing	Short Response
Out-of-class	Activity 36: An Application of Hypothesis Testing	Discovery

Student Notes

Interactive Review of Basic Statistics Using R

ACTIVITY 24

As has been discussed briefly, R is statistical software that can provide statistical analysis, whether in basic descriptive statistic computations or in advanced methods involving both non-parametric and parametric analytical methods. This activity revisits basic statistical analysis using R. It also re-introduces normal distribution and components of analysis regarding the basic assumptions of Gaussian distributions.

Purpose

The purpose of this activity is to introduce the basic statistical analysis package of R

Learning Objective

Review descriptive statistics, Gaussian distributions, and the coding methods and packages for R.

Required Resources

- R, R Studio
- Chapter 4, Dalgaard
- R Packages: RODBC, moments
- ODBC connection
- R script from class website

Time Allocated

40 minutes in class

Tasks

A. Data setup

The first order of business is to obtain the weigh-in-motion data from the class database for Type 11 Trucks (5-axle) recorded on the 8th and 9th of August, 2009.

```
qry <- "SELECT * FROM wim.wimdata WHERE timestamp >= '08-08-2009' AND
timestamp < '08-09-2009' AND type='11'"

wim <- sqlQuery(channel, qry)
```

As you may noticed, the script provided has variable of *datatoexplore*. This generic term is defined as the variable given to you at the start of this activity. The script reads"

```
datatoexplore <- wim$spc2

label <- "Space 2"
```

Where `wim$spc2` is the spacing between the second and third axle. To facilitate exploration of the assigned variable, please define *datatoexplore* as your assigned variable.

Now we can start our exploration of the basic statistical functions that R provides. Let's plot the data.

```
plot(datatoexplore)
```

1. What would be your best guess of what an observation would be for your assigned variable (just by looking at the plot)?

B. Measures of Central Tendency

Unsurprisingly, R has the ability to generate histograms which can provide quick insight into the shape of the data. The function `hist()` analyzes the data and generates a histogram to provide a measure of central tendency. The primary formal argument within `hist()` is the argument `breaks=`,

which determines the breakpoints between the histogram cells. We will explore more on the histogram in subsequent activities.

R can also calculate the `mean()` and `median()`, but there is not a function to determine the mode, or the value that occurs the most often in the data. However, the mode is relatively easy to determine with a few lines of code in R.

```
table(datatoexplore)
max(table(datatoexplore))
```

The `table()` function counts the number of occurrences for values within the dataset. The `max()` function returns the maximum number of occurrences counted by the `table()` function. From this, the mode can be extrapolated by visual inspection of the table in R or determined by plotting a histogram using the `plot()` function.

```
plot(table(datatoexplore), type="h")
```

The `stripchart()` function can be used for small sample sizes. In the case of large sample sizes the detail is lost, especially if the data is highly centered around a small range of values. To see two comparative examples run the following R code.

```
par(mfrow=c(1,2))
stripchart(wim$spc3, pch=21, method="stack")
stripchart(wim$spc3, pch=21, method="jitter")
```

Another quick method to visually inspect the central tendency of a data set is to use the `stem()` function which generates a stem and leaf plot within the console of R Studio, but does not print to the plots tab in the respective pane. As with other functions in this section, a large sample size and non-descriptive breakpoints does not provide the necessary detail that can be obtained from using a histogram or other method of distribution description.

C. Measures of Relative Standing

The idea behind descriptive statistics is to describe the distribution of the data in an effort to assist in describing the relativity between data points within the same set. Establishing the quantiles of a data set assists in determining the percentiles of data that fall within a certain range. An easy way of determining the range and quantiles of a data set is to request the information using the `summary()` function.

2. Generate the summary statistics for your assigned data set and annotate your code with the results.

There exist other ways to generate percentile information to be used to described keys points of distribution within a data set, such as `quantile()` and for the 50 percent percentile or Q2, `median()`. The `quantile()` function can calculate the 50th percentile as demonstrated;

```
quantile(datatoexplore,probs=(0.5))
```

As would be expected the formal argument, `probs=` can be calculated for any probability value between 0 and 1.

3. Calculate the 25, 50, and 75 percentiles using the `quantile()` function and compare Q2 with the `median()` function.

Another way of discussing the relationship of data is by plotting empirical cumulative distribution

function, or `plot.ecdf()`. Similarly, the call can be submitted as the following code:

```
plot(ecdf(datatoexplore), col="dodgerblue", xlab=label, ylab="Cum %",
xlim=range(datatoexplore))
```

4. Plot an empirical cumulative distribution function overlaid by a normal cumulative distribution function.

Boxplots are a common method to quickly provide the median, interquartile range, and any outliers in a data set. The function for a boxplot is `boxplot()`. The great thing about `boxplot()` is the acceptance of a formula to quickly separate categories of data. Let's see this in action (see Figure 45 for the graphic).

```
boxplot (wim$spc1, wim$spc2)
```

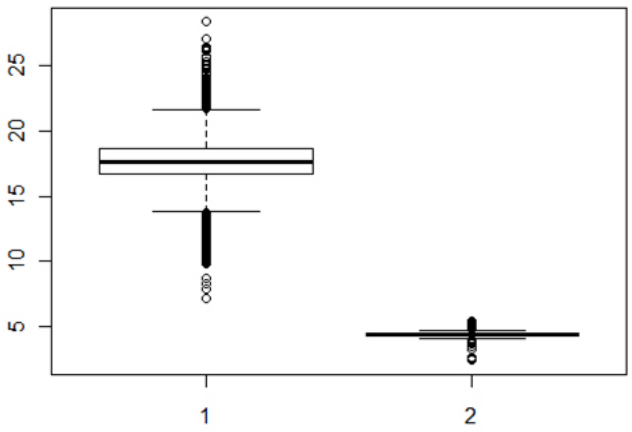


Figure 45 Side by Side box plots of axle space 1(left) and 2(right)

By calling *wim$spc1* and *wim$spc2* both box plots for the corresponding data will be plotted in the same graphic. Multiple values `boxplot(wim$spc1, wim$spc2, wim$spc3, wim$spc4,...)` can be coded to generate a comparison of many variables. A faster way to generate a series of box plots is described below (see Figure 45 for the graphical representation).

```
boxplot (wim$spc2 ~ wim$station)
```

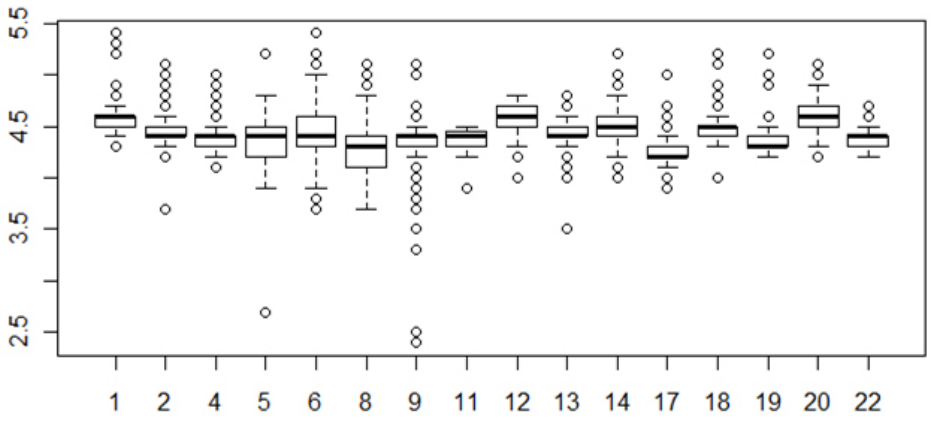


Figure 46 Box plots of axle space 2 by station

This method allows for multiple box plots to be generated based on a categorical variable like the stations for the weigh-in motion data as described by Figure 46.

D. Measures of Variability

Common variability measures are standard deviation or `sd()`, variance or `var()`, and the interquartile range which can be generated using the function `IRQ()` using `library(stats)` if not already active. Or since the interquartile range is the range between Q3 and Q1, the interquartile range can be calculated using the `quantile()` function.

5. Calculate the interquartile range by calculating the value from the `quantile()` function and check the value against `IQR()` function for your assigned variable.
6. Find the range(minimum value, maximum value) using the `range()` function and verify the result.
7. What is the coefficient of variation? Is there a function for coefficient of variation (CV)? If so, what package? Generate a quick function that can calculate the coefficient of variation.

E. Skewness and Kurtosis

As was discussed in the lecture portion of this activity, *skewness* is a numerical value demonstrating the asymmetrical behavior of a variable in a given data set. Upon visual inspection of a plot of counts like a histogram, skewness measures if predominance exists to the left or right of an assumed asymmetrical center.

Kurtosis is the measure of the peaked nature of the data, or the measure of the predominance of data points located near the mean.

R has the capability of calculating the both measures of shape with the addition of the moments package.

8. Using the `skewness()` and `kurtosis()` functions determine the skewness and kurtosis of your assigned data and briefly discuss the results.

Deliverable

Submit a PDF with your output, interpretations, and answers to the questions. Include your R code as an appendix in the PDF document. Submit in the course dropbox.

Assessment

Activity 24 Grading Rubric

	Excellent (10)	Good (8)	Poor (6)	NONE
Script	Organized, complete, accurate and executes.	Missing minor parts, but executes and is otherwise organized and accurate.	Missing significant portions of the activity, unorganized, inaccurate, but executes.	Code does not execute
Annotation	Annotations are complete and describe what the code is accomplishing.	Some annotations are incomplete or do not describe what the code is accomplishing.	No annotations were provided.	Code does not execute
Discussion/ Commentary	Insightful discussion or commentary relating to the question at hand demonstrating student understanding of the task.	Discussion or commentary was incomplete.	Minimal to no discussion or commentary.	Code does not execute

Basic Charts for Single Discrete Variables

ACTIVITY 25

One way to demonstrate key points regarding discrete variables is to present the information in a graphical display. This activity introduces the process of developing dot, bar, and pie charts using R, and expands on the variations available within the individual functions.

Purpose

The purpose of this activity is to help you explore the graphical display options in R for a single discrete variable.

Learning Objective

Develop an understanding of the available techniques to graphically display discrete variables.

Required Resources

- Chapters 2 and 3 in *Graphics for Statistics and Data Analysis with R* in *Introductory statistics with R* (Keen)

Time Allocated

50 minutes out-of-class

Tasks

A. Read Chapters 1 and 2 in Keen

Deliverable

Complete the short quiz on the class site before coming to the next class session. It is timed; you will have 10 minutes to complete the quiz once you start.

Assessment

Short quiz

Student Notes

Exploring Single Discrete Variable Plots

ACTIVITY 26

We will use the incidents_217_2009 data set to introduce the topics discussed in Keen's Chapter 2 on plotting techniques for a single discrete variable. You should think broadly when creating these plots: what do these discrete variables tell us?

Purpose

This activity helps you to explore R's ability to review large data sets quickly and make assessments about data types.

Learning Objective

- Recognize the difference between discrete and continuous variables in plots and data types.
- Recognize that counting records are equivalent to counting events (because many databases are event-level)

Required Resources

- R, R Studio, PostgreSQL ODBC driver, Direct connection to PostgreSQL with R
- Brief overview of discrete and continuous variables

Time Allocated

65 minutes in class

Tasks

A. Orienting Yourself to the Incident Data Set

Let's browse one of the class databases available for exploration: the *incidents* data set. The instructor will give a brief overview of how the *incidents* data set. Browse the metadata for incidents tables. Use phpPgadmin to browse through the *incidents.incidents_217_2009* table. For all of the *incidents* questions, exclude all of the Boolean error checking variables.

B. Connect to the Class Database

Connect to the database and read the *incidents* data into R. The SQL query is (don't forget the R syntax to connect and query).

```
library(RODBC)
channel<-odbcConnect("DATASOURCE","ce510")
qry <- "SELECT * FROM incidents.incidents_217_2009 ORDER BY incidentid"
incidents <-sqlQuery(channel, qry)
```

Let's select one column and issue a simple plot command for each the variables in question 1. R is pretty smart about plotting. Let's just pick the impact type column (*impacttypeid*), which certainly appears discrete.

```
plot (incidents$incidenttypeid)
```

In R, we can use a simple plot to see all of the data plotted

```
#-----------------------------------------------------
par(mfrow=c(4,4), mar=c(2,2,3,1))
```

```
for (i in 1:58) {
plot (incidents[,i], main=names(incidents[i]))
}
```

Note the use of a helpful function to extract the names of a column to use on the plot is:

```
names(incidents)[i]
```

Remember: when *i* is replaced with an integer, the call returns the index or column name corresponding to the number. The first column of the incidents dataframe is *incidentid*. The first plot frames look like:

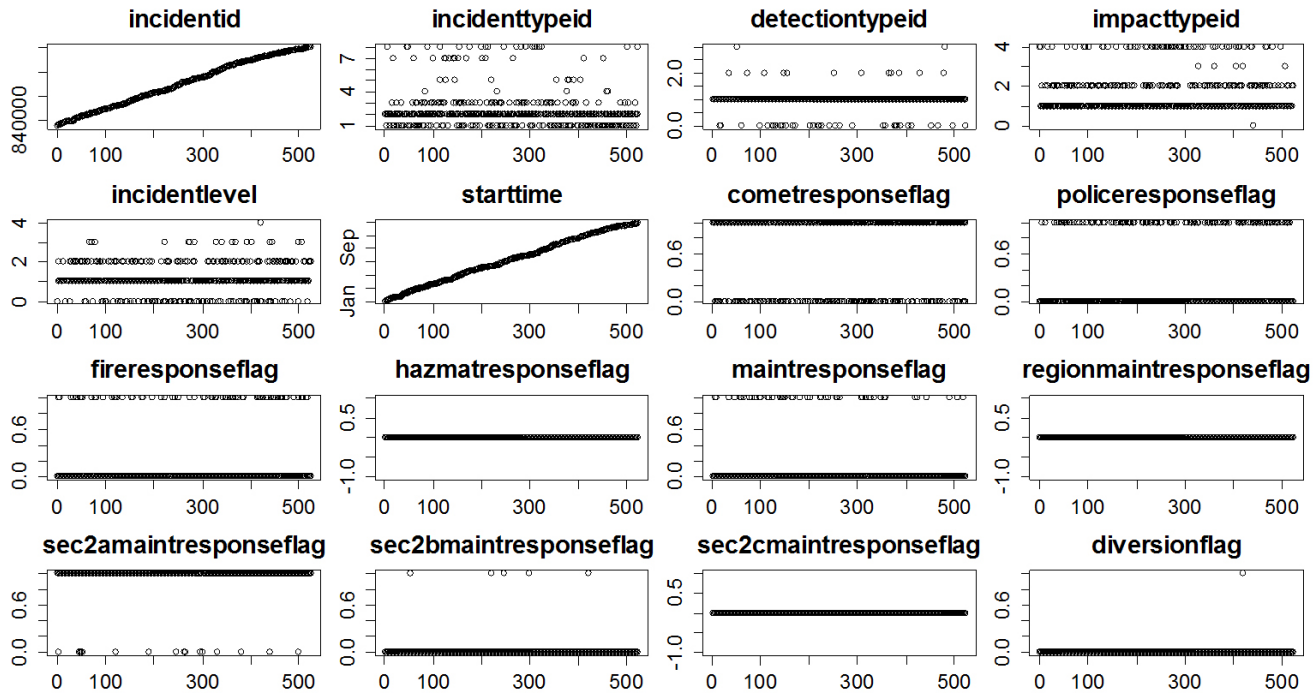


Figure 47

We will use the column *incidentypeid* in the rest of the activity. Incident type has the following values:

- 0 Unknown;
- 1 Accident;
- 2 Stall;
- 3 Debris;
- 4 Tow;
- 5 Construction;
- 6 Congestion;
- 7 Other Closure;
- 8 Other Incident;
- 9 Tag

You should select from the plots 4 variables which look interesting and are discrete variables to make your plots. Do not use any of the flag variables. Figure 48 shows a jittered stripplot of the incident type

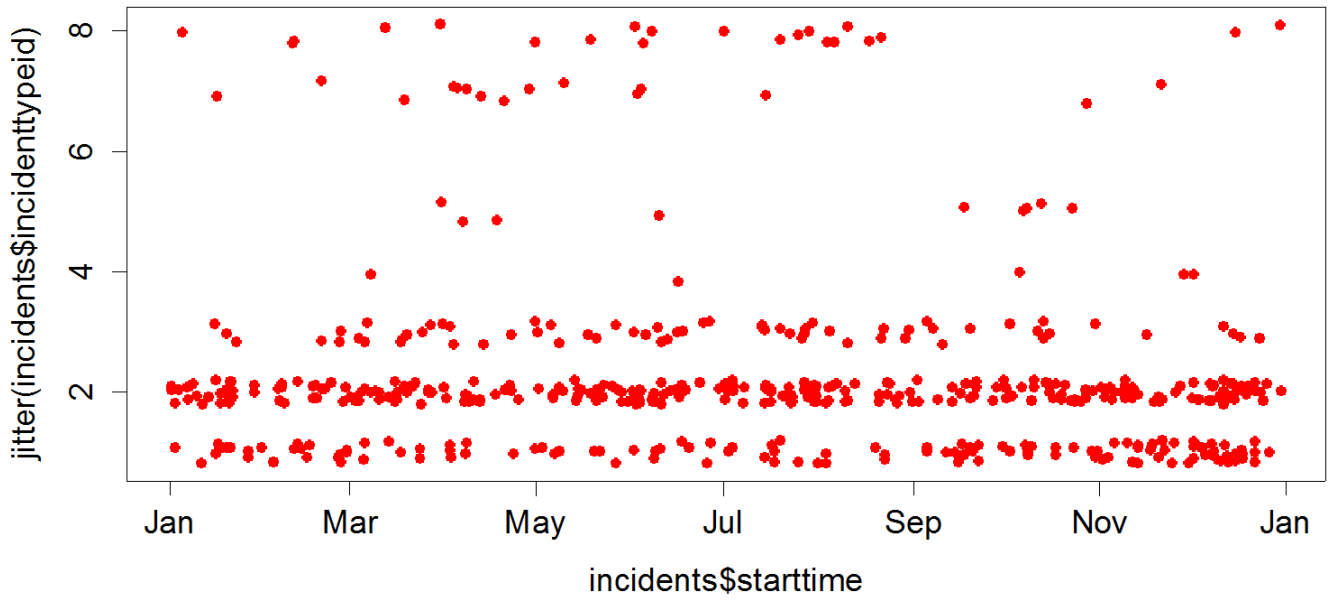


Figure 48

C. Counting Row Events

Before we move to the types of discrete plots, let's start by having R count the number of rows that are in the data frame *incidents* for each discrete element of `incidents[,4]`

```
table (incidents$incidenttypeid)
```

which returns

```
  1   2   3   4   5   7   8
126 283  65   5   9  15  21
```

What do these counts represent? Think about this, since so many of the data we will encounter are event-level data where each row is "something." Counting how many rows have each element of a discrete variable is akin to counting the number of "somethings."

It is also easy to do a 2-way table. Let's extract, extracting month from startime with the lubridate package

```
table ( incidents$incidenttypeid, month (incidents$starttime))
```

#Calculating the col percentages using margins. Change the 2 to 1 to calculate row %

#Columns sum to 1

```
prop.table(            table(incidents$incidenttypeid,            month
(incidents$starttime)),2)
```

#Rows sum to 1

```
prop.table(            table(incidents$incidenttypeid,            month
(incidents$starttime)),1)
```

D. Type of Plots

Run the R scripts to see how to create

- Stripchart
- Bar Chart
- Pie Chart
- Dot Plot

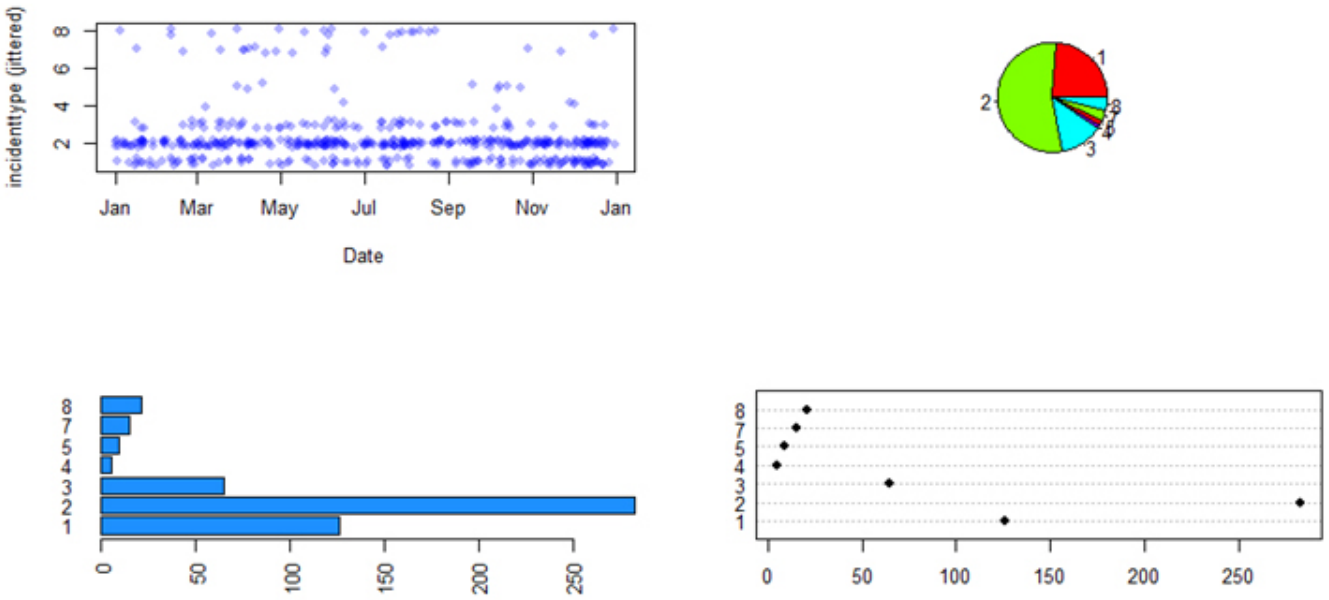


Figure 49 Sample Plots of Incident Type

E. On Your Own

Using the existing code, make plots of the four variables you identified. Try making a two-way table by month or day of the week to see possible trends in the data.

Deliverable

Submit a pdf document with a short write up to the course management dropbox.

Assessment

This activity only requires you to complete and discuss the plots, you will not be assessed on the graphical presentation quality (though you are free to make them as "pretty" as you like).

All plots complete	1 – 2 mlssing	3 – 4 mlssing	> than 4 missing
10 points	9 points	8 points	5 points

Exploratory and Diagnostic Plots for the Distribution of a Single Continuous Variable

ACTIVITY 27

This activity presents a brief introduction to the graphics available for presenting a continuous variable. Continuous variables such as weight or speed often appear to be discrete in table, but are really continuous and can be presented using histograms, stem-leaf plots, and other graphical displays of the distribution characteristics of the dataset.

Purpose

The purpose of this activity is to introduce the graphical display options within R that can assist in demonstrating key concepts regarding a continuous variable.

Learning Objective

Develop an introductory understanding of the graphical presentation of continuous variables

Required Resources

- Chapter 4 and 6 of *Graphics for Statistics and Data Analysis with R* in *Introductory statistics with R* (Keen)

Time Allocated

50 minutes out-of-class

Tasks

Read chapters 4 and 6. Respond to the short quiz on the class course management site.

Deliverable

Complete the short quiz on the class site before coming to the next class session. It is timed; you will have 10 minutes to complete the quiz once you start.

Assessment

Short quiz

Student Notes

Probability Distributions

ACTIVITY 28

The next step in the process is to address distributions of data sets. Many different distributions have been developed to assist in defining the information contained within individual data sets. This activity focuses on two chapters. The first is from Dalgaard and discusses random sampling, probability calculations, and distributions. The second is from Keen and further discusses boxplots, revisiting the inner-quartile range and the calculation of the outer fences.

Purpose

The purpose of this activity is to help you become familiar with the built-in functions for distributions and to revisit probability and the aids to diagnosing distribution-type.

Learning Objective

Recognize some of the more common distributions and the available methods in R to assist in diagnosing the distribution of data sets.

Required Resources

- Chapter 3, in *Introductory Statistics with R* (2nd Ed.) (Dalgaard)
- Chapter 5, in *Graphics for Statistics and Data Analysis with R* in *Introductory statistics with R* (Keen)

Time Allocated

50 minutes out-of-class

Tasks

A. Read Chapter 3 from the Dalgaard text.

B. Read Chapter 5 from the Keen text.

Deliverable

Read the assign chapters and complete the short quiz on the class course management site.

Assessment

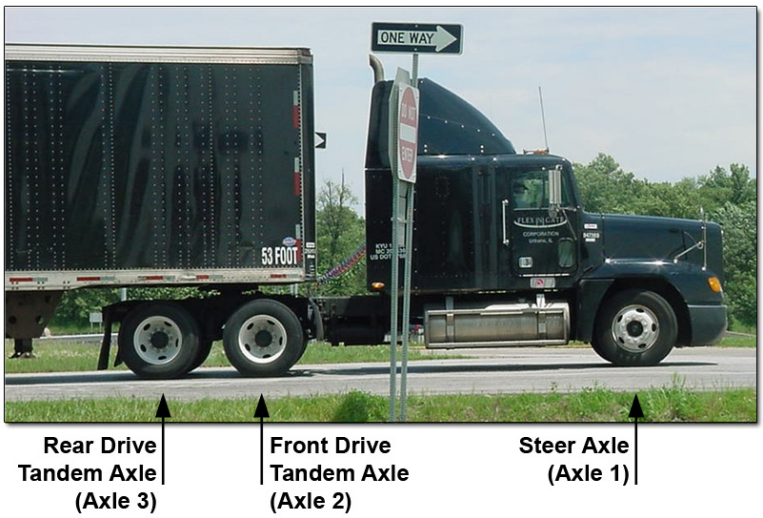
Short quiz

Student Notes

Kernel Density Estimates and Histograms

ACTIVITY 29

Figure 50 Axles 1, 2 and 3 (Picture: Andrew Nichols)



The histogram and the kernel density estimate are two great tools that allow you to quickly see the distribution of a continuous variable. You are probably familiar with the histogram, but probably not with its nuances. We'll explore those. The kernel density estimate (KDE) might be new to you but you will grow to love it. We'll start by looking at the WIM data and then you will get a chance to explore further. We will focus on the axle weights of 5-axle semi-trucks in this exploration.

Purpose

To combine lessons from descriptive statistics with an exploration of data.

Learning Objective

Use graphical analysis and scripting to identify possible data quality issues.

Required Resources

- R, R Studio
- PostgreSQL ODBC driver
- Direct connection to PostgreSQL with R
- Sample R script

Time Allocated

90 minutes in class

Tasks

Open the R script for Activity 29 from the class website. We will start our exploration with the just the class 11 trucks from all stations. Note that we have sped up the read from the class database if you only select the columns you need, rather than using the all `(*)` SQL operator.

```
qry <- " SELECT stationnum, axl1, axl2,axl3, axl4, axl5 FROM wim.wimdata
WHERE type='11'"
```

Later, you will be analyzing some 750,258 rows of data. For parts A and B of this activity, we will subset the data to just station 8.

A. Introduction to the Histogram

The main display decision for the histogram is how many bins to count the frequency of the data. The number of bins can be set with the `breaks=` option in the call to histogram. In Keen, the bin width is the class width. The class width is evenly divided over the range of the data (rounded to an integer). Keen lists and describes ways to determine the bin width and number of classes:

- Rule of Twelve
- Robust Rule of Twelve

- Sturges Rule
- Doanes Rule
- Scott's Rule
- Freedman-Diaconis Rule

In R, it can be hard-coded with an integer, or specified in a call to a function. Options are Sturges, Scott and Freedman-Diaconis. Run the code that creates the histograms shown in Figure 51. Note that the y-axis, which is the number in each bin, changes for each.

1. Create the same set of histograms for axl2, axl3, axl4, axl5. Can you see the bimodal aspect of any of the distributions? Is it easier to see with one of the break settings?

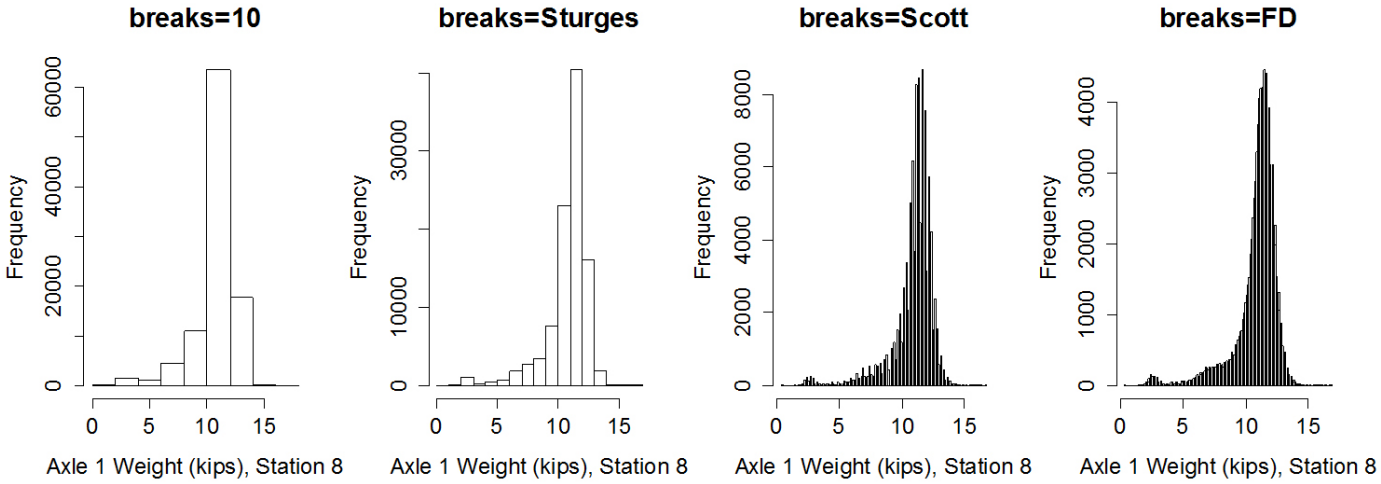


Figure 51 Histograms of Station 8, Axle 1 Weight in Kips, August 2009

Another useful option is that you can store the results of the histogram without plotting them by setting the call to `plot=FALSE`. Then you extract the names count, density, or breaks vectors.

```
y <- hist (wim8$axl1, breaks=10, plot=FALSE)
str(y) # see the values of the histogram class
y$breaks # pull out the vector of breaks
y$counts # set the counts
sum(y$density)  #what should this sum to ?
```

You can also plot the histogram with the density (#bin/total obs) instead of the frequency on the y-axis. With this call:

```
hist (wim8$axl1, breaks="Sturges", freq=FALSE)
```

B. Introduction to Kernel Density Estimate (KDE)

The kernel density estimate plot is essentially a smooth histogram. The resulting plot is non-parametric estimate of the variable under consideration's density function. It is akin to the pdf of random variable. You have a choice of the bandwidth and kernel. The bandwith is the effective width of the sliding window used to generate the density. As the sample gets larger, the bandwidth can be made larger. The kernel is function over which the density is estimated. It can be one of the following in R `"gaussian"`, `"epanechnikov"`, `"rectangular"`,`"triangular"`, `"biweight"`,`"cosine"`, `"optcosine"`).

The density plot is generated with a call to density function first, which can then be plotted:

```
denmass <- density(wim8$axl1, kernel="gaussian")
plot(denmass, main="gaussian")
```

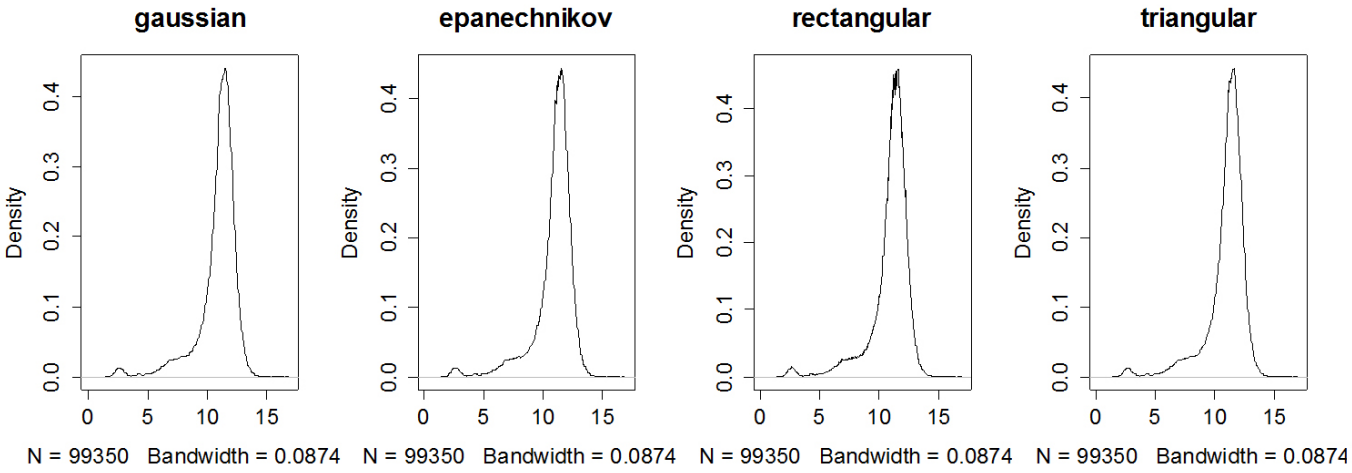


Figure 52 KDE plots of Station 8, Axle 1 Weight in Kips, August 2009

Modify the KDE to explore the effect of the `bw=setting`. Check the function `bw.nrd0()` which returns the default bandwidth. You should see how large bandwidths smooth out the KDE.

C. On Your Own

The steering axle weight of the 5-axle semi truck (in the *wim* data type 11 trucks, Class 9 in the FHWA scheme) ranges between 9,000-11,000 lbs. This is the steering axle and mainly handles the weight of the engine and tractor. If there is a lot variability in these data or the observations of one station are lower than the others, it could be an indication that a station is not performing correctly and that calibration may be required.

Using the KDE plots, explore the station to station variability of axle 1 weight to show the distributions of the axle1 weight for all stations. In these plots, look for different shapes of the distribution and for differences in the in the central location of the distribution. First, set up the plots so that they are generated in their own frame and arrange the plots. Then, try to put KDEs of the plots in all one plot on the same scale (using the lines function). You should also use other plots that we have explored (such as the boxplot) to help you diagnose variability and median means.

Repeat this same exercise for the other axle weights. With these weights, it is harder to see differences, since we have the effect of loaded and unloaded trucks.

Deliverable

Prepare a short write up of your discovery about the underperforming stations. In your write-up, clearly identify the stations that appear to be in need of calibration or error checking. Elaborate on your reasons. Submit a PDF of this write up to the drop box. Include R code as an appendix.

Assessment

This activity is a short response activity. The score that you receive will be based on the quality and depth of discussion. The response expected differs by question as described in rubric below:

Activity 29 Grading Rubric

	Excellent (10)	Good (8)	Poor (6)	NONE
Discussion	Insightful discussion or commentary relating to the question at hand demonstrating student understanding of the task.	Discussion was competent regarding the lessons, but lacked in insightful discussion.	The discussion did not address the lessons or applicability of the activity.	Did not submit
Graphic(s)	Graphic correctly demonstrated the distribution of the data set.	Graphic did not accurately demonstrated the distribution of the data.	No graphics were present in the discussion.	Did not submit
Quality	Document is typed, formatted, contains appropriate grammar and language.	Document has minor grammatical errors or inappropriate language.	Document was unorganized, contained inappropriate language and/or grammatical errors.	Did not submit

Diagnosing a Distribution

ACTIVITY 30

We often want to know what theoretical distribution best represents some observed empirical data. You may need to do this to select the appropriate parametric statistical test, analysis method, or validate some assumptions. In this activity, we will explore a new plot: the *quantile-quantile plot* (or q-q plot). This is an excellent diagnostic tool to identify whether a sample of data fits a particular distribution. While you may be interested in any number of distributions (e.g., normal or Gaussian distribution, exponential, Poisson, t distribution, binominal distribution, or Chi-Squared distribution), this activity will focus on the most common q-q plot for comparing data to the normal (Gaussian) distribution. The Keen text shows you how to construct a q-q plot for other distributions. You already have a good sense of how to visualize the distributions based on the diagnostic graphics you have studied to date (histogram, KDE, boxplot, empirical cumulative distribution function) and calculations of the descriptive statistics.

Today's activity will also introduce you to R's ability to synthesize random distributions easily. We will "fake" some empirical data using these procedures then apply the graphical diagnostics to them. Since we know the underlying distribution, we can easily see what the diagnostic graphs should look like if we have "normally distributed" data. In this way, when we apply this method to "real" data it should be clear what we are trying to diagnose. This is a way to explore some statistical concepts that cannot be overstated. In fact, many of my flashes of insight (those "ah-ha" moments) have come from comparing graphically and quantitatively "real" random distributions to ones that have been observed. The activity includes branches for you to explore for other distributions once you understand the basics.

Purpose

This activity will give you the opportunity to construct q-q plots for the purpose of graphically diagnosing distributions

Learning Objective

Become familiar with R code technique and principles of distributions

Required Resources

- R, R Studio
- PostgreSQL ODBC driver, direct connection to PostgreSQL with R,
- Sample R script for activity
- Excel file showing the construction of Normal Q-Q plot

Time Allocated

90 minutes in class

Tasks

First, the instructor will present a short overview PPT that will explain some of the key concepts and show you some sample code. Follow along with the R script for Activity 30 provided on the class website and lecture PowerPoints.

A. In the R script, section #2, there is example code for showing 2 plots. The first shows a set of theoretical normal distributions with 4 means and the same standard deviations. The second shows the maximum mean in first with 4 different standard deviations. Inspect the code line by line. The plots are created by using the `dnorm` function which returns the familiar "bell curve." Change `mu` and `stdev` in the code until you are satisfied that you understand the differences in the shape, spread, and the mean of the distributions. Select one of the combinations to explore further (it doesn't matter which one).

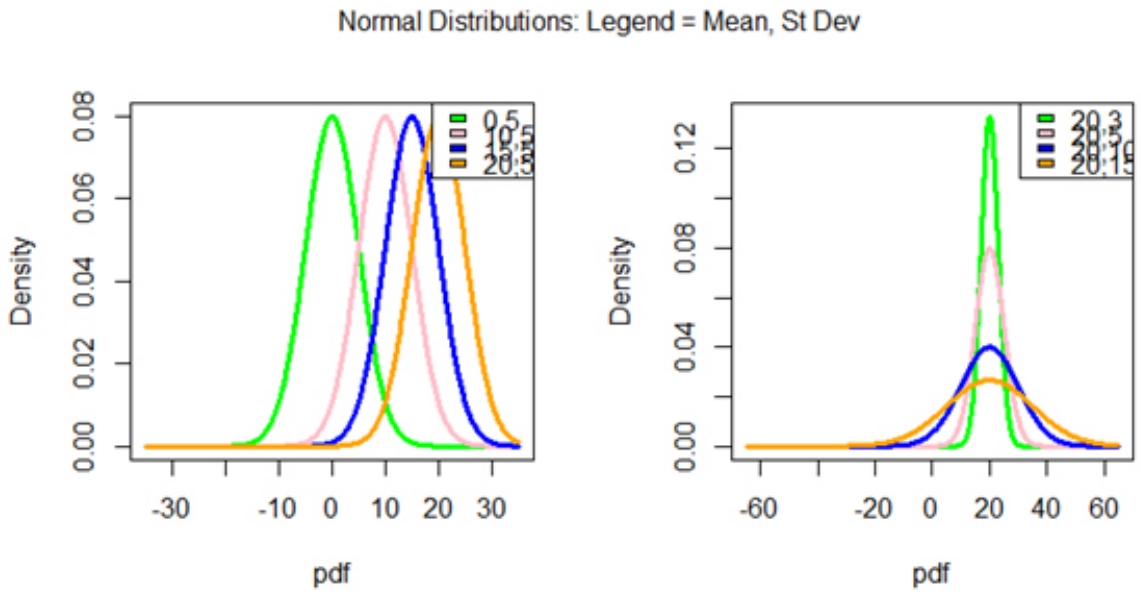


Figure 53 Normal distributions with varying means and equal standard deviations (left), and varying standard deviations and equal means(right)

a. Modify the code to produce the cumulative distribution curves. But before you do this, sketch out what you think the two plots will look like. For example, in left plot, how will the cumulative density function that you plot be arranged? Will they have the same slope/shape? What about in the right plot?

b. Now execute the code you modified (hint: just change the `dnorm` to the appropriate function).

c. If the plots don't look the way you expected, see if you can figure out why. Include both sets of plots in your write up and discussion. Write a short observation about what you know or don't know.

B. In the R script, section #3, there is code that is very similar to the code in section #2 but we will explore in more detail the density (`dnorm`) and cumulative density (`pnorm`) and quantile (`qnorm`) functions. Review the code line by line. Pay special attention to the `abline` functions that plot the dashed lines. Be sure that you can understand what is being plotted.

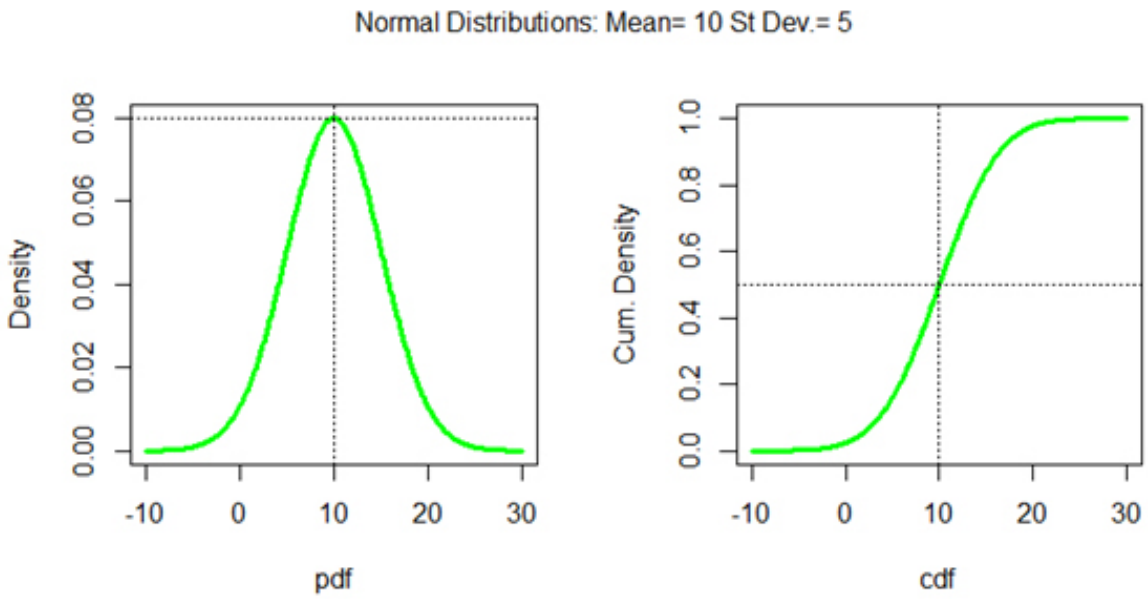


Figure 54 Normal distribution density function plot (left) and cumulative density function plot (right)

a. As mentioned previously, the special case of the normal distribution is the standard normal where the mean is 0 and standard deviation is 1. This is also called the *Z distribution*. Can you create your own set of plots, with the mean equal to zero and the standard deviation equal to one? Add lines for the 0.025, 0.05, 0.50, 0.95, and 0.975 percentiles and the corresponding quantile, probability density, and cumulative density. Have you seen these values before? Find a standard normal Z-table in a statistics textbook or online and think about how they are related.

C. In the R script, section #4 just produces all of the diagnostic plots that you are seen so far for a synthetic distribution that you generated with the following code:

```
mu_r <- 5 ; stdev_r <- 2; n <-200
set.seed (10)
s_data <- rnorm(n, mean=mu_r, sd=stdev_r)
```

The call to `rnorm` generates a vector of n elements that will have a mean and standard deviation as specified. Note that every call to `rnorm` will produce a different vector unless we set the seed for the random number generator. The call to `set.seed` sets the seed for the random number generator. The `rnorm` will return the same sequence when called in this session. You know these plots represent data that are normally distributed (since `s_data` is randomly generated to be normally distributed). The bottom plots include the KDE plotted over with the theoretical distribution that our sample data was drawn from. Inspect all of these plots so that you get a sense of what a "real" normal distribution looks like.

a. First, try a few different sizes of sample draws, small to large such as n (10, 100, 1000, 10000) without changing the mean and standard deviation. Before running the code, how do you expect the plots to change?

b. Comment out the `set.seed` command and rerun the code for the same *n*, mean, and standard deviation. After creating 5 or so plots, play them back with the arrow button in R Studio and watch the data change. Now, do the same thing but include the execution of the `set.seed` command. Describe what you see.

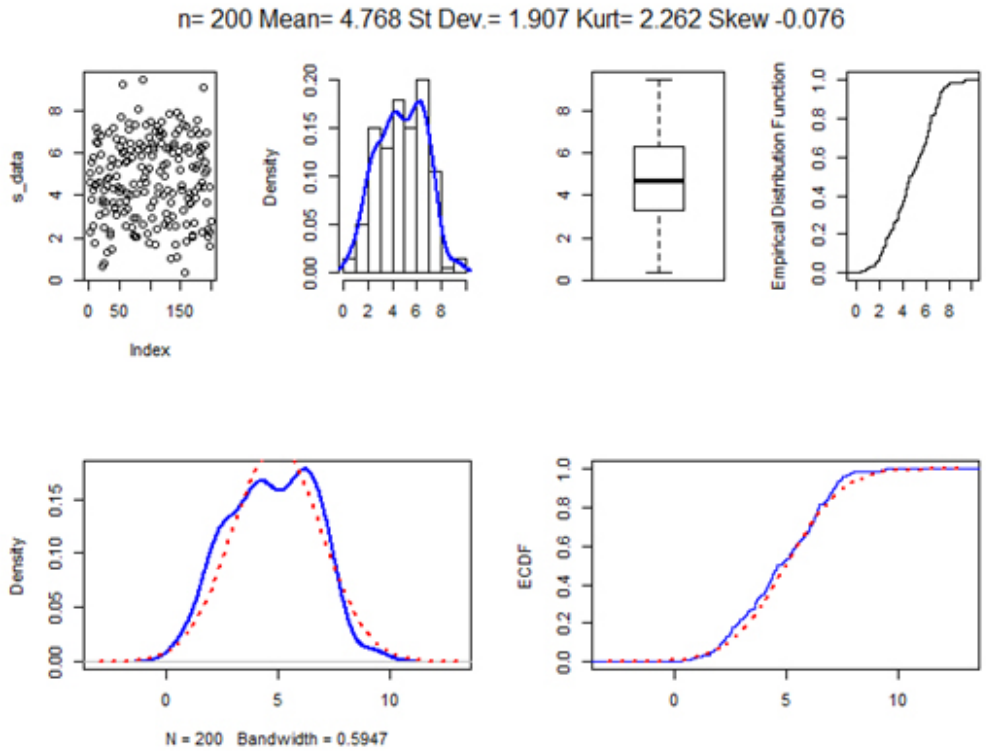


Figure 55

(top left to right) Scatter plot of random numbers generated for sample n=200, histogram of same random number generated sample with KDE overlaying histogram, boxplot of same data, and empirical cumulative distribution function of the sample.

(bottom left to right) Theoretical distribution overlain by KDE, and theoretical cumulative distribution function overlain by empirical distribution function.

D. The q-q plot is simple in concept and easy to interpret once you know the "rules" but perhaps a little confusing to construct. The R script in section #5 walks you through the construction of the plot. Spend some time working through the code line by line and comparing the output. R's built in functions `qqnorm` and `qqline` do all the work for you but it may be more informative to look at the "by hand" code. Open up the accompanying Excel spreadsheet and look at the construction. Match the Excel functions to the R functions. Confirm in Section 4 that the computed quantiles match between Excel and R.

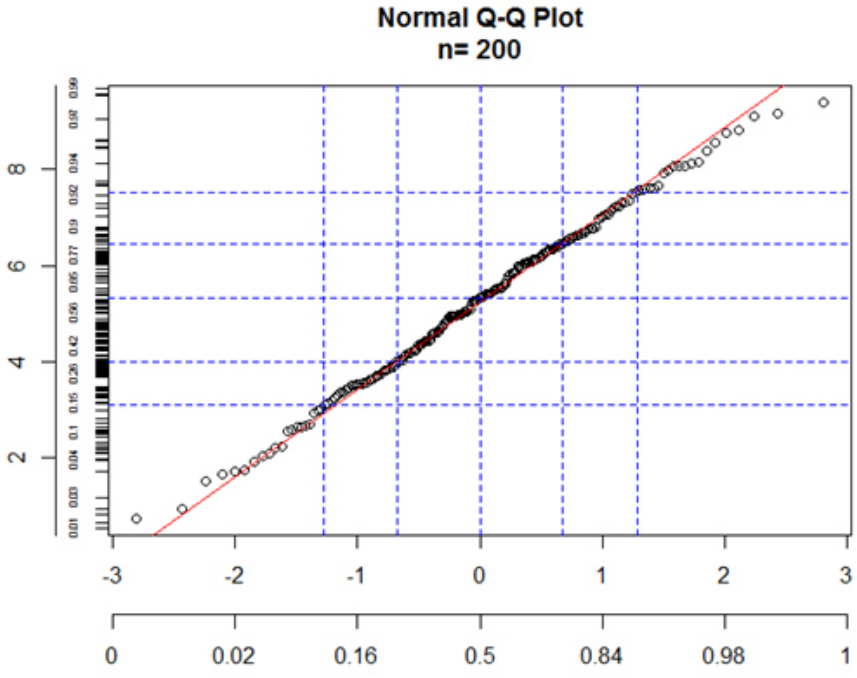


Figure 56 Quantile-Quantile plot of the random number generated sample n=200

E. APPLICATION

You have been asked if the average speed observed in male and female cyclists is normally distributed and if so, what are the parameters that best describe this distribution from the class database. To do this, write a query to select all data from the bicycle performance data frame. Graphically diagnose whether the male and female observed average speed data are normally distributed. Show the empirical data in a histogram, KDE, and fit an approximate normal to these data, as well as show the empirical cumulative density function and corresponding normal cumulative density function. Of course, include the q-q plot. For each variable prepare a short discussion showing these figures. Annotate the figure with your observations.

Deliverable

Prepare a short write up of your discovery. Submit a PDF copy to the class dropbox.

Assessment

This activity is a short response activity. Your score will be based on the quality and depth of discussion. The response expected differs by question as described in the following rubric:

Activity 30 Grading Rubric

	Excellent (10)	Good (8)	Poor (6)	NONE
Discussion	Insightful discussion or commentary relating to the question at hand demonstrating student understanding of the task.	Discussion was competent regarding the lessons, but lacked in insightful discussion.	The discussion did not address the lessons or applicability of the activity.	Did not submit
Graphic(s)	Graphic correctly demonstrated the distribution of the data set.	Graphic did not accurately demonstrated the distribution of the data.	No graphics were present in the discussion.	Did not submit
Quality	Document is typed, formatted, contains appropriate grammar and language.	Document has minor grammatical errors or inappropriate language.	Document was unorganized, contained inappropriate language and/or grammatical errors.	Did not submit

Student Notes

Depicting the Distribution Involving Discrete Variables

ACTIVITY 31

Sometimes demonstrating the differences between multiple variables using a single graphic can have more benefit than multiple graphics, especially if graphics count against word counts in professional or academic papers. This reading activity introduces the various ways to visually compare different discrete variables against themselves, or discrete variables against a continuous variable.

Purpose

The purpose of this activity is to introduce the methodology of graphically displaying multiple variables on the same graphic.

Learning Objective

Recognize the method and benefits of using a single graphic to display information contained within a data set

Required Resources

- Chapters 8 and 9 in *Graphics for Statistics and Data Analysis with R* in *Introductory statistics with R* (Keen)

Time Allocated

50 minutes out-of-class

Tasks

A. Read Chapters 8 and 9 in Keen

Deliverable

Complete the short quiz on the class course management site before coming to the next class session.

Assessment

Short quiz

Student Notes

Depicting the Distribution of Two Continuous Variables

ACTIVITY 32

A benefit of discrete variables is the tidiness that categorical data can provide in the presentation of the graphics. However, with continuous variables, a melting pot of values can be contained within a data set, requiring advanced plotting methods to present the two dimensions of two continuous variables. This activity introduces a few methods contained with the standard R packages.

Purpose

The purpose of this activity is to introduce the plots that effectively present two continuous variables.

Learning Objective

Understand effective plotting techniques of two continuous variables

Required Resources

- Chapter 10 in *Graphics for Statistics and Data Analysis with R* in *Introductory statistics with R* (Keen)

Time Allocated

50 minutes out-of-class

Tasks

A. Read Chapter 10 in Keen

Deliverable

Complete the short quiz on the class course management site before coming to the next class session.

Assessment

Short quiz

Student Notes

Introduction of Final Project Topics

ACTIVITY 33

We will explore the the six available final topics addressing questions and developing the expectations for the final project.

Purpose

The purpose of this activity is to familiarize you with the various topics available for the final project

Learning Objective

Determine and share expectations regarding the final project

Required Resources

none

Time Allocated

110 minutes in class

Tasks

There are no specific tasks in this activity.

Deliverable

No deliverables are required.

Assessment

Participation points (10) are available and assessed on active participation in any resulting discussion.

Student Notes

One and Two Sample Sets

ACTIVITY 34

This activity introduces R functions pertaining to one and two sample tests for both samples assumed to be normally distributed and distribution-free. The key tests presented in this reading assignment are the t-test (assumption of a normally distributed population) and the Wilcoxon signed rank test (distribution-free).

Purpose

The purpose of this activity is to help you develop familiarity with the R functions for one and two sample tests for use in future activities.

Learning Objective

Recognize the differences between distribution-free and normally distributed one and two sample tests.

Required Resources

- Chapter 5 in *Introductory Statistics with R* (2nd Ed.) (Dalgaard)

Time Allocated

50 minutes out-of-class

Tasks

A. Read chapter 5 in the Dalgaard text.

Deliverable

Respond to short quiz on the class course management site.

Assessment

Short quiz

Student Notes

Exploring Confidence Intervals and Simple Hypothesis Testing

ACTIVITY 35

A confidence interval describes the amount of uncertainty associated with a sample estimate of a population parameter. 95% confidence level means that 95% of the interval estimates are expected to include the population parameter. A simple hypothesis testing is an assumption which specifies the population distribution completely. It examines a random sample from a population to see if the sample data are consistent with statistical hypothesis. If not, the null hypothesis is rejected. This activity focuses on the steps of processing confidence intervals and performing statistical hypothesis testing in R.

Purpose

The purpose of this activity is give you the opportunity to learn how to compute confidence intervals and perform simple hypothesis testing in R

Learning Objective

Learn statistical function in R and diagnose the difference of two sample datasets in plots.

Required Resources

- R, R Studio
- Sample script file from class web site

Time Allocated

90 minutes in class
40 minutes out-of-class

Tasks

This activity begins assuming R Studio is open.
If not, please start R Studio and open the sample script for the activity (if provided).

Browse the metadata for the bike performance data to familiarize yourself with the dataset. Use phpPgadmin to browse through *bicycle.performance* table. Connect to database via RODBC and extract the full file that will be using for data analysis set later. Extract the average speed `'set1'` and `'set2'` of this data set grouped by levels of grade (0 = no grade; 1 = grade).

Ultimately we are going to compare the means and attempt to answer the following question: Does grade have an effect on average observed speed? And if so, what is it?

A. Diagnostic Plots

In my experience, the best thing to do before running any statistical tests or computing confidence intervals is to do summary diagnostic plots. In this exercise, four plots in different distribution functions and two statistical measures will be applied for making contrast and comparison of *set1* and *set2*.

Fortunately you now have a lot of easy tools you can use. Follow along in the sample R script provided for the today's activity. A helpful diagnostic function to produce summaries is:

```
summary(set1,na.rm=TRUE)
```

where `na.rm=TRUE` is for missing data. If `na.rm=TRUE` then missing values are removed before computation proceeds. But this doesn't tell the whole story.

1. Review the plots which follow (Figures 57 and 58) and make a short note describing your interpretation of the plots, including your "hunch" about the effect of grade on average speed.

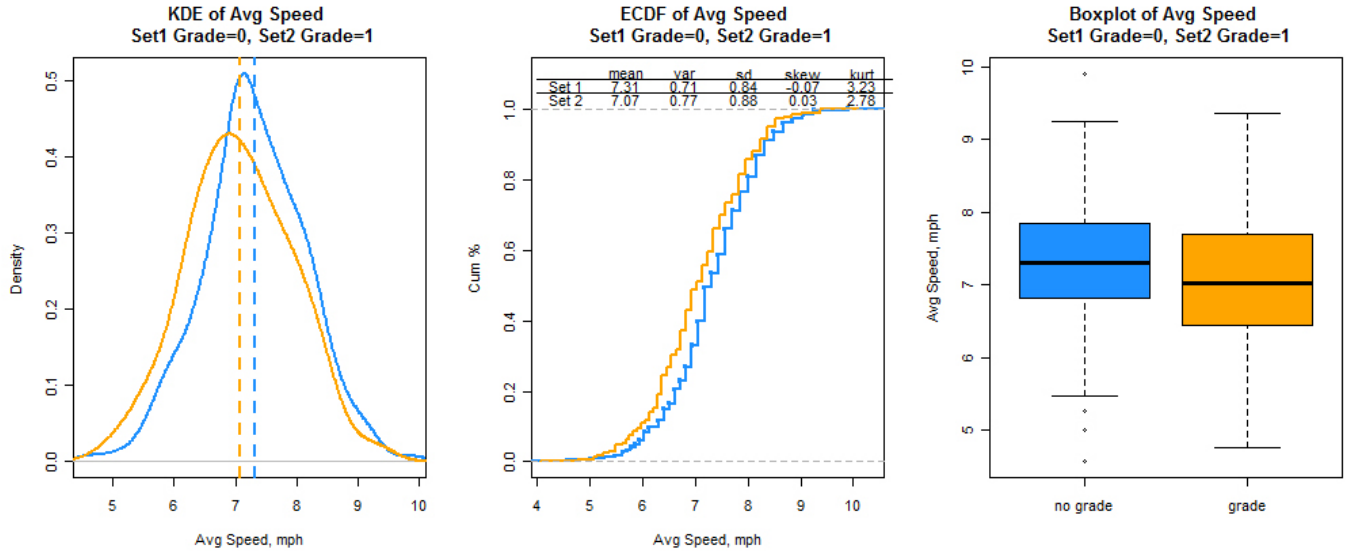


Figure 57 Summary Plots Comparing the Distributions

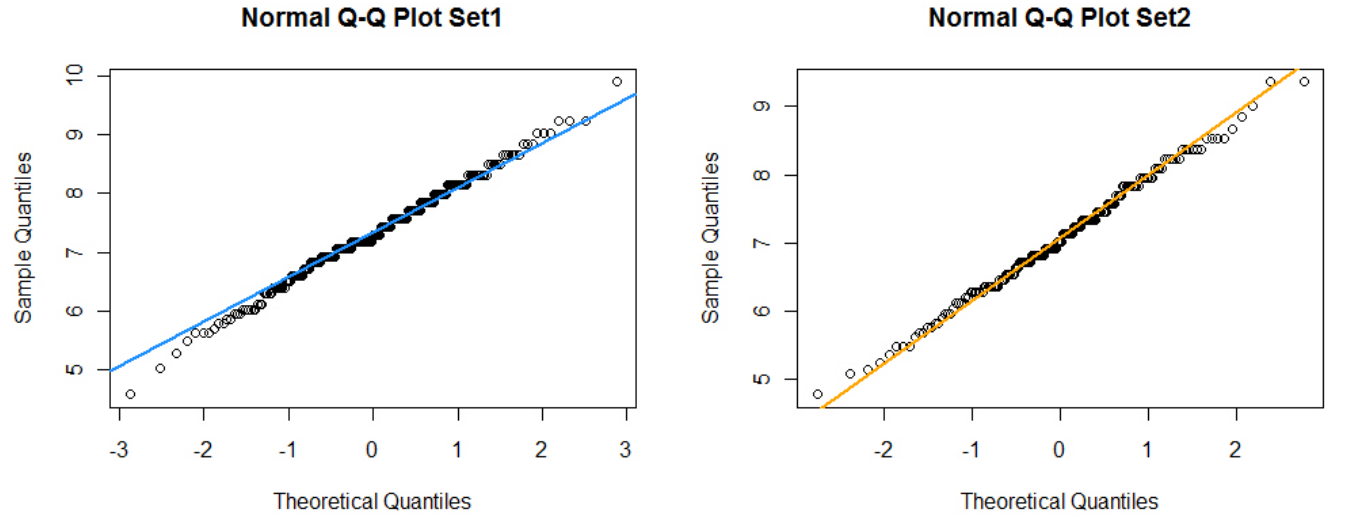


Figure 58 Q-Q Plots of the Distributions

B. Calculate Confidence Intervals Around the Mean

When average speed data are assumed in normal distribution, quantiles of normal distribution, the sample mean and standard error of the mean can be used to calculate approximate confidence intervals for the mean.

`qt` quantile function for t distribution with `df` degrees of freedom;

`qnorm` quantile function for the standard normal distribution (Z), with mean=0 and sd=1

We can do either the Z test or *t* distributions to determine our confidence interval of the mean. For samples this large, it makes little difference (as you will see later). But, let's follow Dalgaard and use the student's *t*.

The standard error of the mean is equal to standard deviation divided by square root of sample size.

$$SEM = (Standard\ deviation) / \sqrt{n}$$

To estimate the two-tailed confidence interval:

$$Upper\ confidence\ limit = \bar{x} + \left(SEM * t_{\alpha/2} \right)$$

$$Lower\ confidence\ limit = \bar{x} - \left(SEM * t_{\alpha/2} \right)$$

Where x bar is the mean and α is the chosen confidence interval by the analyst (you!). To determine the t statistic we also need the degrees of freedom, which here is equal to (n − 1).

Certainly the most common confidence level is the 95th percentile. By selecting this we are estimating a confidence interval for the mean that includes with 95% probability the true mean. We are accepting a small probability (5%) that true mean is not in this interval. Since the distribution is symmetrical, this 5% can be on either tail (as seen in the figure below). For the 95th percentile, α = 0.05 and α/2=0.025. Thus we need the cumulative probability that corresponds to the value of 1.00 − 0.025 = 0.975. To return $t_{0.975}$ for the degrees of freedom then use the `quantile()` function to return the value that captures 97.5% of the cumulative probability. Use this R syntax;

```
qt(0.975, n-1)
```

The sample R script generates the probability density function (PDF) and cumulative density function (CDF) of the t-distribution and shows the quantile of the t-distribution that corresponds to the given alpha. The figures below show that for set 1 with 249 degrees of freedom, 95% of the cumulative probability is between −1.969576 and 1.969576 (area under the curve in the PDF) and there is a 5% chance (2.5% on each tail) that the true mean is outside these bounds. This is shown in the CDF as 97.5% of the cumulative probability (since we are excluding the left tail we add 95% + 2.5% = 97.5%.

Change the alpha level in the script and notice what happens to the location of the t-statistic and the tails of the distribution. Make a connection between the alpha level and results shown in these 2 plots (see Figure 59).

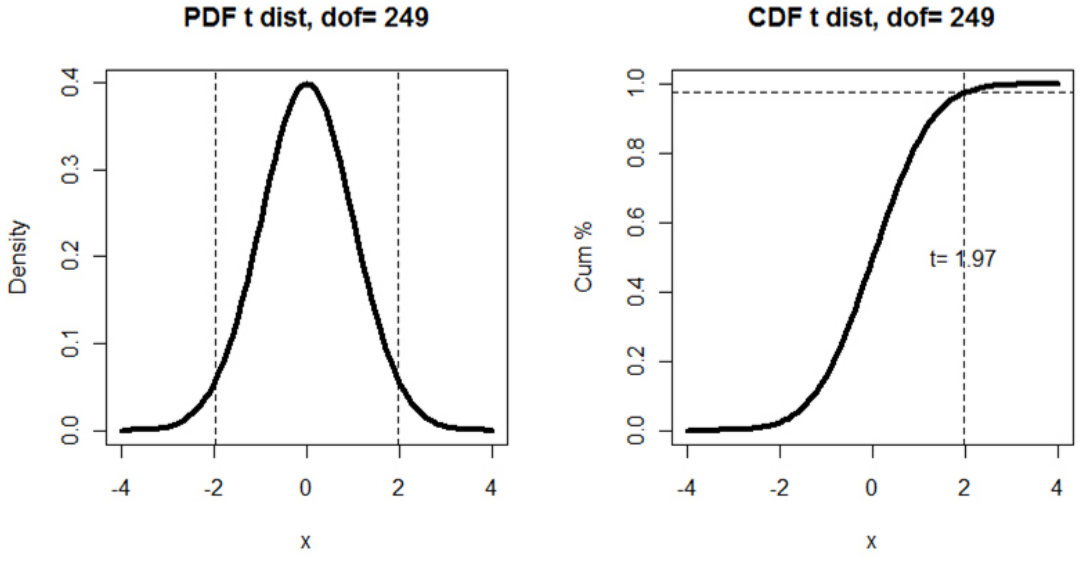


Figure 59 Plots of PDF and CDF for t-distribution with dof=249

Now, we can calculate the upper and lower confidence intervals with:

```
# use t statistic
stat1 <- qt(1-alpha/2, n-1)  # use t for set 1
stat2 <- qt(1-alpha/2, n2-1) # use t for set 2
#Calculate the bars
error <- stat1*stddev/sqrt(n)
lower <- mu-error
upper <- mu+error
error2 <- stat2*stddev2/sqrt(n2)
lower2 <- mu2-error2
upper2 <- mu2+error2
```

These are shown graphically in Figure 60: *set1* statistics are shown in blue; *set2* statistics are in orange. "+" is the upper bound of confidence interval; "-"is the lower bound of confidence interval; dot is the mean.

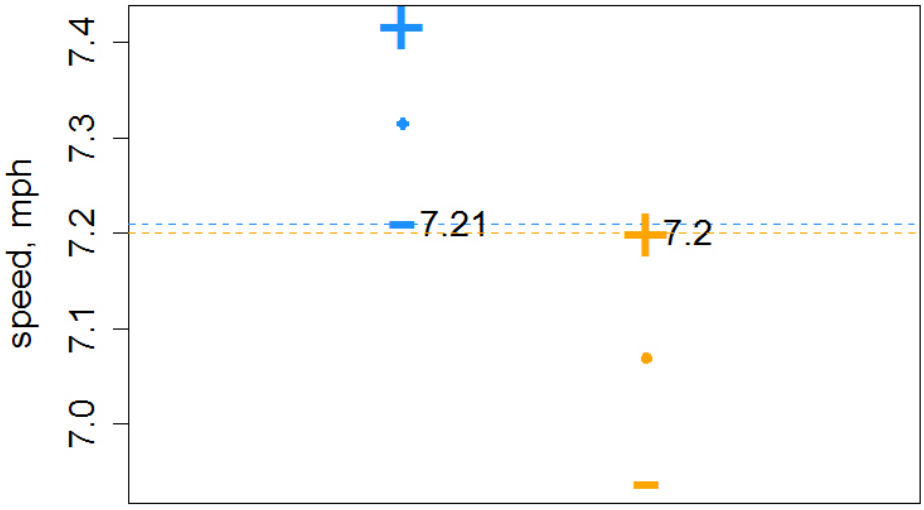


Figure 60 Plot of Mean and 95th Percentile Confidence Interval

2. Now, make 3 similar plots assuming a 85, 90, 95 and 97% confidence intervals and plot them in a `c(1,4)` arrangement. Below these 4 plots, include the CDF distribution plot as shown in Figure 60. Write a show description of your interpretation. Can you explain the differences in the confidence intervals that you see in the plots?

C. t-test

Of course the next step is to test whether there is a difference between the mean values of these sets of speed data. The test requires 2 assumptions:

- that the data being tested are normally distributed and
- the variance of the sample data being tested are equal.

The t-test is robust to some departures in the assumption 1 and there is a version of the test if you are not willing to assume that the variances are equal. First, comparing the Q-Q plots reveals that these data are reasonably normally distributed.

The variance assumption is more difficult to establish, but we can use the F test for testing underlying assumption of homogeneity of variances. Let's take *set1* and *set2* as an example to test equality of variance.

```
> var.test (set1, set2)
   F test to compare two variances
data:  set1 and set2
F = 0.9193, num df = 248, denom df = 172, p-value = 0.5429
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.6952998 1.2071398
sample estimates:
ratio of variances
         0.9193456
```

The alternative hypothesis is that the true ratio of variances is not equal to 1. The p-value of the test is 0.5429. Since this is greater than our selected alpha of 0.05, there is insufficient evidence to reject the null hypothesis that the ratios of the variance are equal to 1. Thus we conclude that the variances are equal. Alternatively, we could run the t-test with the Welch correction (see Dalgaard).

Now, let's perform the t test. The output is printed below for both variance assumptions:

```
> t.test (set2, set1, var.equal=TRUE)
   Two Sample t-test
data:  set2 and set1
t = -2.8951, df = 420, p-value = 0.003988
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.41188434 -0.07876422
sample estimates:
mean of x mean of y
 7.069296  7.314620
> t.test (set2, set1, var.equal=FALSE)
   Welch Two Sample t-test
data:  set2 and set1
t = -2.8733, df = 360.028, p-value = 0.004303
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.41323353 -0.07741503
sample estimates:
mean of x mean of y
 7.069296  7.314620
```

Both tests have very small p-values, leading us to reject the null hypothesis that the means are equal and accept the alternative hypothesis "`true difference in means is not equal to 0`". You can check your interpretation by inspecting the 95% confidence interval of the difference between the means. Note that this does not include zero.

2. Now, subset the data by MALE and FEMALE and conduct a t-test of the difference in means.

D. When does the t distribution looks like Z distribution

You often see testing for equality of means or the calculation of confidence intervals based on either distribution. The Z distribution is the "standard normal" that we have seen so much already. Use of the Z distribution in hypothesis testing and confidence interval calculations requires a large sample size and a known ***population*** standard deviation. Since we are nearly always estimating the population standard deviation from the ***sample***, this second assumption introduces additional uncertainty. The t-distribution has larger tails and helps account for this uncertainty especially with small sample sizes. But what is a small sample? A rule of thumb is that a sample size larger than 20 is sufficiently large so that use of the Z distribution is acceptable. Can we confirm this rule of thumb in R? The parameter of the t-distribution is the degrees of freedom. A loop was set up for plotting t distribution under different degree of freedoms. The standard normal distribution is shown in heavy black line.

```
for (i in 2:length(dof)) {
t <-  pt (x, dof[i])
lines (x, z-t, col=colorvector[i])
}
```

We see that the higher degree of freedom, t distribution is more closed to z distribution as shown in Figure 61. A simple subtraction of the densities is shown in the right panel, confirming that as the degrees of freedom approaches 20, the t and Z distributions are converging.

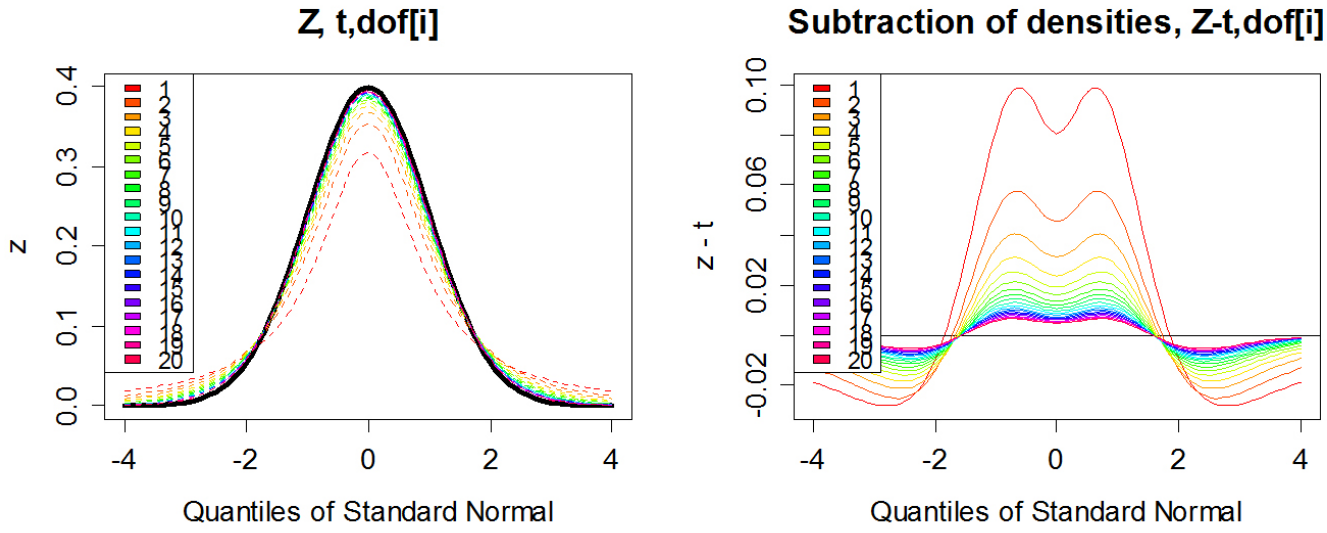


Figure 61 Z and t distributions with varying degrees of freedom

Now, should I be confused? The purpose of this is to show you the convergence of the 2 common distributions. The best advice is to use the t-distribution approaches whenever possible (note R does not have a Z-test function in the standard build) since it is more conservative. Also, when interpreting the outputs of linear regression or other outputs, pay close attention to whether the parameters are estimated by either Z or t distributions.

E. Wilcoxson Non-parametric Test

There are tests that do not require the assumption of distribution to be applied. These are called non-parametric, since the underlying distribution is not specified.

3. Use Dalgaard as a reference and run the Wilcoxson test of means on the male/female and grade/no grade data sets.

Deliverable

Summarize the results of all the numbered questions in this activity. If instructor assigns "BONUS" question in the script file, complete as well. Submit an electronic PDF of your write up to the dropbox folder on the course management system.

Assessment

This activity is a short response activity. The score that you will receive will be based on the quality and depth of discussion. The response expected differs by question as described in the following rubric:

Activity 35 Grading Rubric

	Excellent (10)	Good (8)	Poor (6)	NONE
Discussion	Insightful discussion or commentary relating to the question at hand demonstrating student understanding of the task.	Discussion was competent regarding the lessons, but lacked in insightful discussion.	The discussion did not address the lessons or applicability of the activity.	Did not submit
Graphic(s)	Graphic correctly demonstrated the distribution of the data set.	Graphic did not accurately demonstrated the distribution of the data.	No graphics were present in the discussion.	Did not submit
Quality	Document is typed, formatted, contains appropriate grammar and language.	Document has minor grammatical errors or inappropriate language.	Document was unorganized, contained inappropriate language and/or grammatical errors.	Did not submit

Student Notes

An Application of Hypothesis Testing

ACTIVITY 36

The previous activity focused on confidence intervals and hypothesis testing. This activity is an application of the lessons from the previous activities as well as an introduction to the R scripting of the ANOVA test used for testing the means of a continuous variable across multiple categories.

Purpose

The purpose of this activity is to give you the opportunity to apply multivariate hypothesis testing in R

Learning Objective

Recognize statistical functions in R and diagnose the difference of multivariate data

Required Resources

- R, R Studio

Time Allocated

70 minutes out-of-class

Tasks

A. Testing Means

For this activity, we need the incident records located in the incidents schema within the class database. Go ahead and load the whole data set into R. *duration* is a column within the incidents website in the form HH:MM:SS, but is not recognized by R as a difference in time. This generates errors when attempting to make anything coherent out of the duration values. The simple solution is to regenerate the values by generating a loop that calculates the value duration using the incident start time and the incident last updated time.

```
for (i in 1:nrow(a)){a$dur[i]<-(as.numeric(a$lastupdatetime[i])
                                -as.numeric(a$starttime[i]))}
```

Or, an alternative way of generating a new column from existing data is

```
c$dur<-(as.numeric(c$lastupdatetime[1:nrow(c)]) -as.numeric(c$starttim
e[1:nrow(c)]))
```

Another alternative is to use the *lubridate* package `hms()function` to define the time value as seconds.

Now that the duration values are in seconds and sensible, we can look over the *incidents* data and determine which columns are discrete and can be used to develop subsets of the data. An appropriate choice is the number of lanes affected in the incidents which has heading *numlanesaffected*.

a. What are the different values within this column?

Go ahead and subset the data frame by the different values within *c$numlanesaffected* and generate the summary statistics accompanied by a boxplot. Normally full distribution diagnostics would be in order, but for the purposes of this activity we will negate this to focus on hypothesis testing.

b. Are there any noticeable differences in the means of the durations for the respective affected lanes?

Performing a two-sample hypothesis test requires the function `t.test()` and can be used to test if there are differences between the means of two samples within the same population. Since there are

more than two categories within the *numlanesaffected* column multiple tests are required to test if the means are the same or not the same.

c. Generate the two-sample tests for all sensible combinations, and discuss your findings by accepting or rejecting H_0. Use $\alpha = 0.05$.

Although R makes processing multiple hypothesis tests relatively easy and fast with large numbers of "categories", other methods exist that directly relate to hypothesis testing of multiple categories. In the case of the number of lanes affected by an incident, a statistical test of the means referred to as ANOVA, or Analysis of Variance exists which can test three or more means simultaneously to see if they are all equal or not all equal. The important distinction in the null and alternative hypothesis is that H_0: $\mu1=\mu2=\mu3$ and H_1: The means are not all equal. Thus, the ANOVA test only tests if there exists a difference between the means and does not decipher the difference between the means. The ANOVA test is run using the function `aov()` and using an ~ to separate the categorical variable from the continuous variable. But first we need to ensure that the categorical values are being read as such, which means we are required to force the structure from integer to a simple factor using the `as.factor()` function and the generation of a new column.

```
c$lanes<-as.factor(c$numlanesaffected)
```

Now that the lanes column has been established the `aov()` function can be called. Or,

```
duranova<-aov(dur~lanes, data=c)
summary(duranova)
```

Which returns the following table ($\alpha = 0.05$),

```
              Df     Sum Sq   Mean Sq F value    Pr(>F)
lanes          3 7.8564e+08 261881218  6.4838 0.0002589 ***
Residuals    520 2.1003e+10  40390039
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The F-value, 6.4838, is defined as a 6.4838 times greater variation between the groups than within the groups. Using an F-distribution table the upper critical value is between 2.6049 (df = ∞) and 2.6802 (df = 120) which our F-value and the corresponding Pr(>F) or adjusted p-value states is significant and thus we reject H_0 finding significant evidence that the means differ. However, which means differ still remains an issue and can be found using post-hoc tests when there is significant evidence to reject H_0.

A common post-hoc test for the ANOVA is the `TukeyHSD()`, and is used to expose means which are significantly different. The call is simple and is as follows,

```
TukeyHSD(duranova)
```

Returning the following table,

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = dur ~ lanes, data = c)
```

```
$lanes
         diff         lwr       upr     p adj
1-0  2124.282    607.5456  3641.019 0.0019031
2-0  3606.428    745.8330  6467.023 0.0067358
3-0  1264.833  -8241.2457 10770.912 0.9861203
2-1  1482.146  -1453.8869  4418.178 0.5626325
3-1  -859.449 -10388.5004  8669.602 0.9955727
3-2 -2341.595 -12174.4647  7491.276 0.9276890
```

This table describes the categorical variables 0-3 by providing an adjusted p-value describing the whether or not a relationship exists, and again using $\alpha = 0.05$ significant difference is found between 0,1 and 0,2.

Please annotate your script with the appropriate answers and comment on any differences or similiarities between question 3 and the ANOVA and Tukey HSD tests. As always, make sure the script executes.

B. Complete an ANOVA analysis for the duration of incidents by *incidenttypeid*, *detectiontypeid*, *impacttypeid*, and *incidentlevel*. Interpret your results and prepare prepare a write up complete with plots and descriptions of your analysis. Be sure to comment on your interpretation of the plots and the ANOVA output. Try to add some explanation as to why one category may have longer or shorter duration.

Deliverable

This is a discovery activity. That means that it puts together some things that you have learned to date. You don't need to expound too much on why it worked, but upload your R code (cleaned up and commented upon) to the class dropbox.

Assessment

This activity will be graded on the following criteria:

Activity 36 Grading Rubric

	Excellent (10)	Good (8)	Poor (6)	NONE
Script	Organized, complete, accurate and executes.	Missing minor parts, but executes and is otherwise organized and accurate.	Missing significant portions of the activity, unorganized, inaccurate, but executes.	Code does not execute
Annotation	Annotations are complete and describe what the code is accomplishing.	Some annotations are incomplete or do not describe what the code is accomplishing.	No annotations were provided.	Code does not execute
Commentary	Demonstrated active engagement with exploring the various options of the functions within the activity.	Demonstrated some minor changes to the instructors code.	No changes and thus no commentary that demonstrates active engagement with exploring the options within the activity.	Code does not execute

Student Notes

CHAPTER 5 Data Exploration for Understanding

"The real voyage of discovery consists not in seeking new landscapes, but in having new eyes."

Marcel Proust, French Novelist 1871-1922

How data is collected is as important as what data is selected for use from large populations. This chapter explores random sampling and mining for data quality concluding with an activity presenting a flow chart for use in determining the appropriate method of statistical analysis for a given variable.

Activity List

Activity Type	Number and Title	Assessment Type
In Class	Activity 37: Advanced Multivariate Continuous Displays and Diagnostics	
Out-of-Class	Activity 38: Introduction to Random Sampling	Response

Student Notes

Advanced Multivariate Continuous Displays and Diagnostics

ACTIVITY 37

This activity explores advanced multivariate plotting. With the advanced diagnostic functions, output graphs are more readable and useful for analyzing complex data structures.

Purpose

The purpose of this activity is to help you become familiar with customizing advanced arguments.

Learning Objective

Be familiar with some advanced graphical methods in R for continuous data, as well as some techniques for simple data mining

Required Resources

- R, R Studio
- Sample script file from class web site
- R Libraries (`RODBC`, `moments`, `lattice`, `MASS`, `classInt`, and `RColorBrewer`)

Time Allocated

110 minutes in class

Tasks

This activity begins assuming R Studio is open. If not, please start R Studio and load libraries `RODBC`, `moments`, `lattice`, `classInt`, `RColorBrewer`, *and* `MASS`.

A. Scatterplots

The scatterplot below (Figure 62) shows the relation between speed vs. volume at lower left and relation between speed vs. occupancy at loop detector ID 1594.

Next, we will break occupancy into certain amount of color intervals. R contains a function called `cut()` that divides the range of x into defined intervals. The intervals can then be attached to the dataframe by adding a column that defines each records interval. The idea of establishing intervals is to be able to assign different colors to the intervals providing a way to depict the variation in occupancy. To complete the addition of the 3rd dimension a vector of colors is required and can be completed using the `terrain.colors()` function which only requires the input of the number of colors that should be used to assign to the intervals.

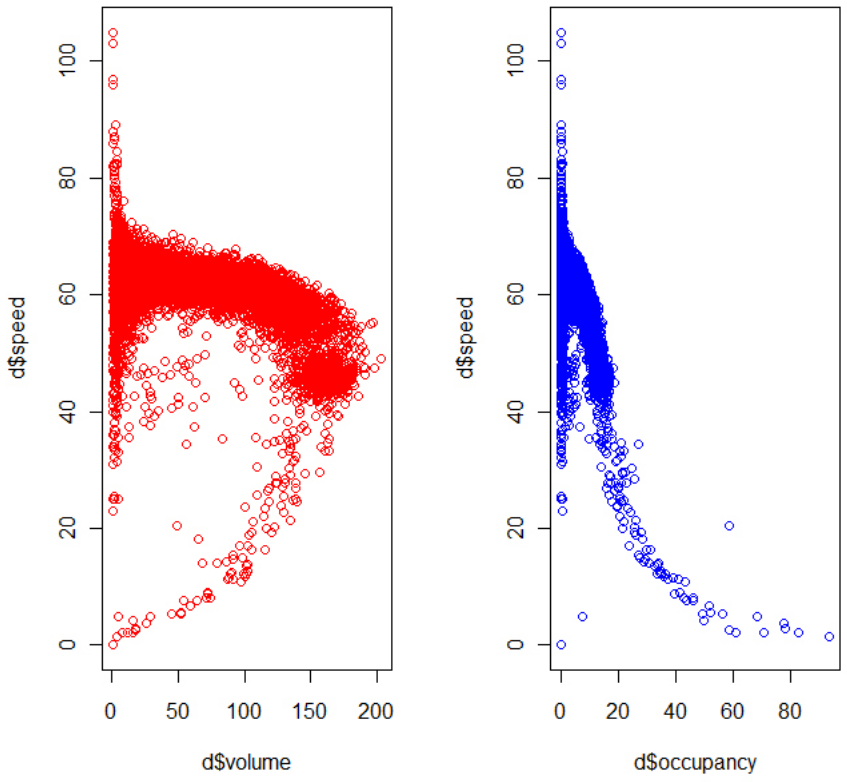


Figure 62 Scatterplots of speed versus volume (left), and speed versus occupancy (right)

Plotting the scatterplot only requires the call to plot as would normally be done, however the color argument is called using the numeric representation of the intervals, or `collist[as.numeric(d$col_occ)]` in the case of the accompanying script. Doing this establishes that the first interval, `[0,5]` receives the first color in the vector `collist` which was previously defined as the `terrain.colors()` function with 8 different colors. The resultant call returns Figure 63.

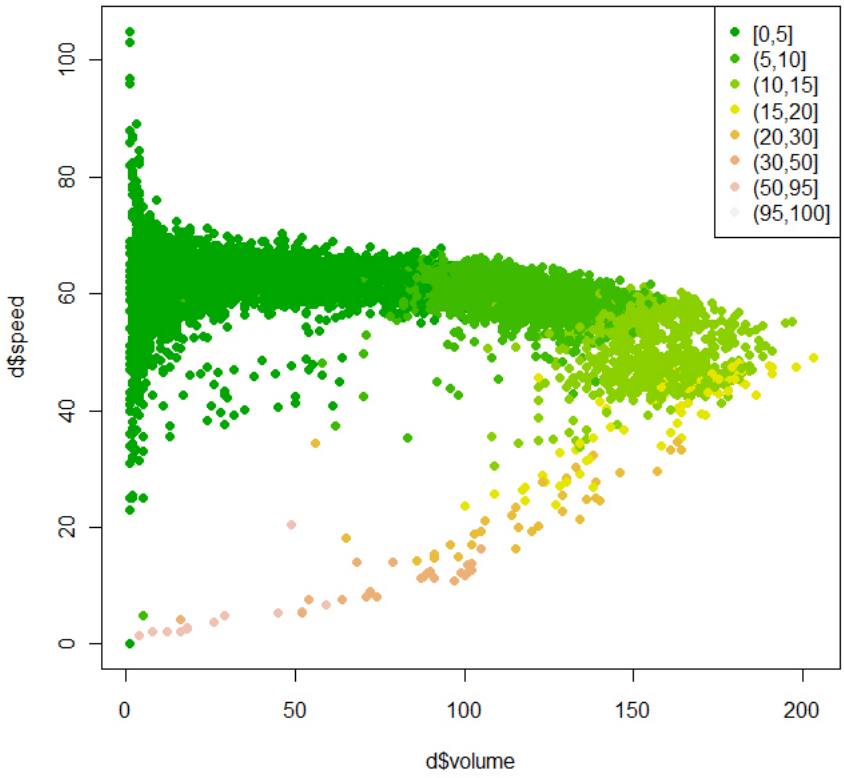


Figure 63 Scatterplot with third dimension as color

Similar to the change of color described in the previous discussion, other factors can be changed using functions found within the *graphics* package. In the case of Figure 64, the size of the data points are defined in the plot call using the function `symbols()`. Again, volume is plotted on the x-axis and speed on the y-axis with occupancy being added as the third dimension and reflected as the size of the data point with center (x, y).

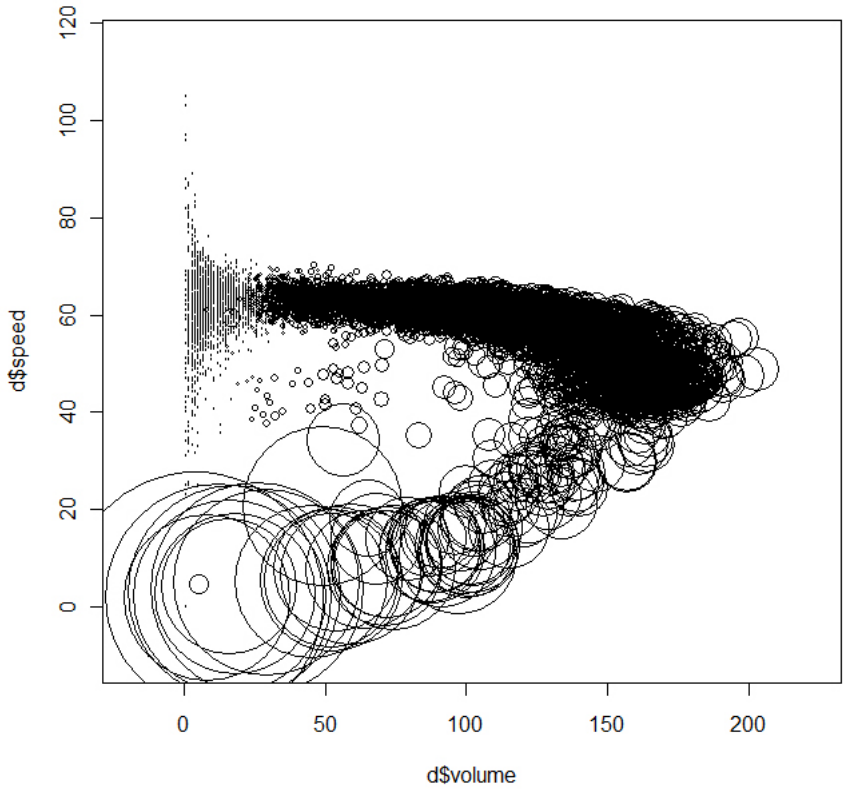


Figure 64 Scatterplot with third dimension as size

Another variation for the addition of a 3rd dimension is the addition of a second axis as demonstrated in Figure 65. The addition of the second axis is completed by using `par(new=T)` and the call for an additional axis is done using the `axis()` function. The `new=T` argument calls the next plot to be placed on the same graphic as the previous plot. Then the second axis values are placed by defining the location (the first argument of the `axis()` function) to be at the edge of the graphic. Defining the size of the graphic output with `par(mar=c())` makes the effort to establish the right side of the limit due to the user defined limits.

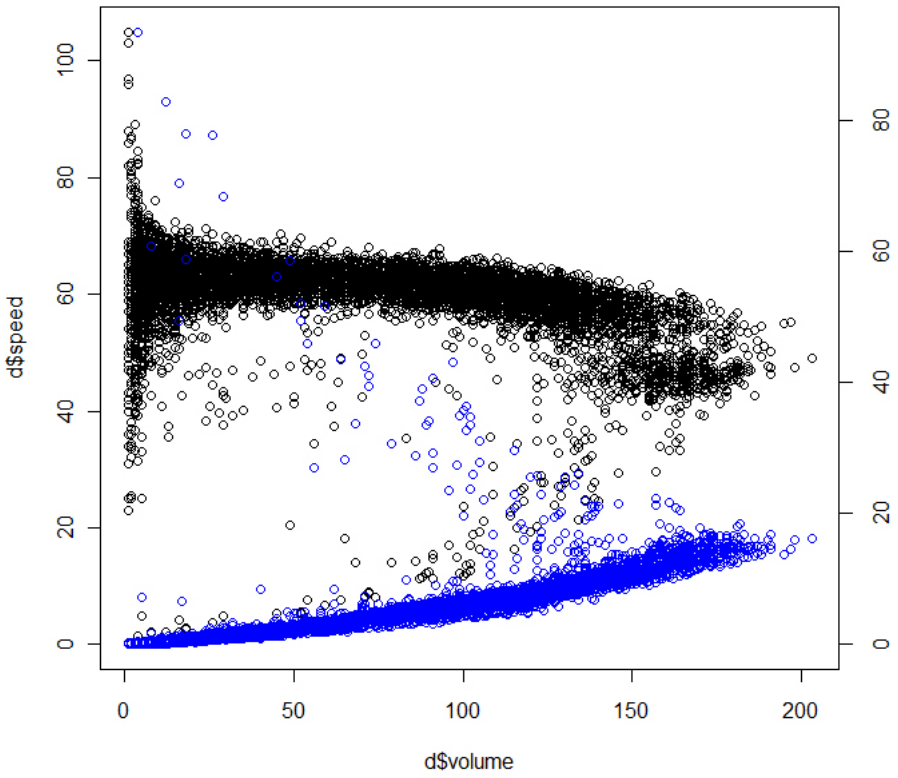


Figure 65 Scatterplot with third dimension as second axis

Earlier we established color variations using intervals. Another variation in color is to add transparency to the color as was done in Figure 66. This is accomplished by defining the color using the `rgb()` function. The key points of this function are the arguments `a=`, `r=`, `g=`, and `b=` which defines the level of the red, blue, and green primary colors. The alpha argument (`a=`) determines the level of transparency, the higher the value the more transparent. Take a few minutes to explore the variations available regarding this useful function.

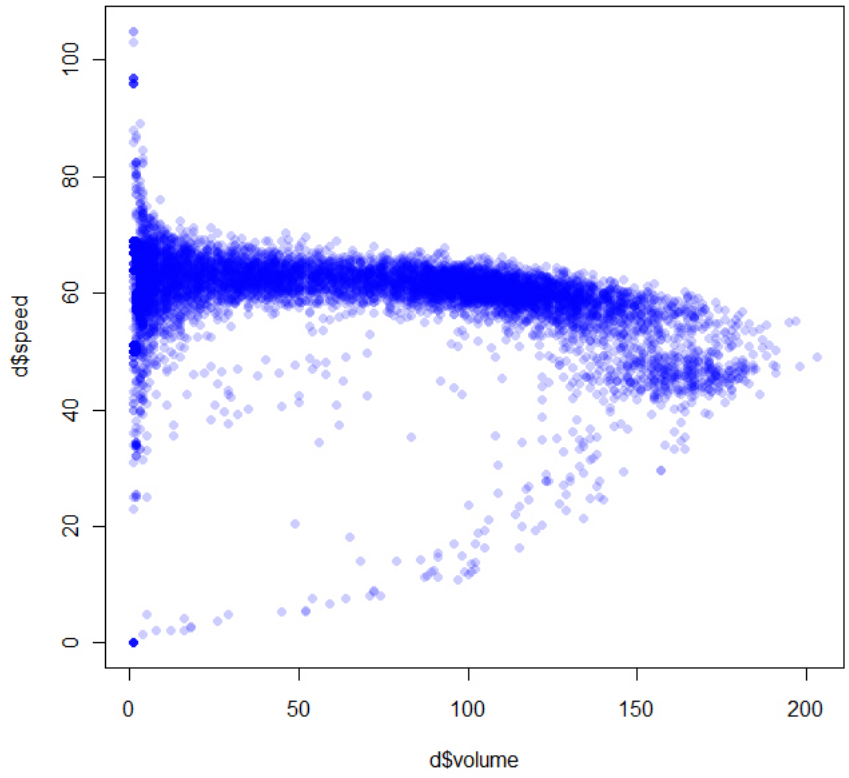


Figure 66 Scatterplot - overlay points with transparency

Another symbolic representation of a third dimension is the sunflower plot which plots a scatter plot of the non-duplicated values then over-plots the graphic demonstrating the shared value between multiple data records. For each repetition of value the function adds an additional 'petal' to depict the density of that data value. Figure 67 is a plot of the `sunflower()` function The call to plot is the function `sunflowerplot()` with similar arguments of the `plot()` function. The most notable argument is the ability to generate a rotation of the points based on a randomly assigned direction. This is an attempt to further limit what the sunflower plot attempts to negate which is confused overplotting.

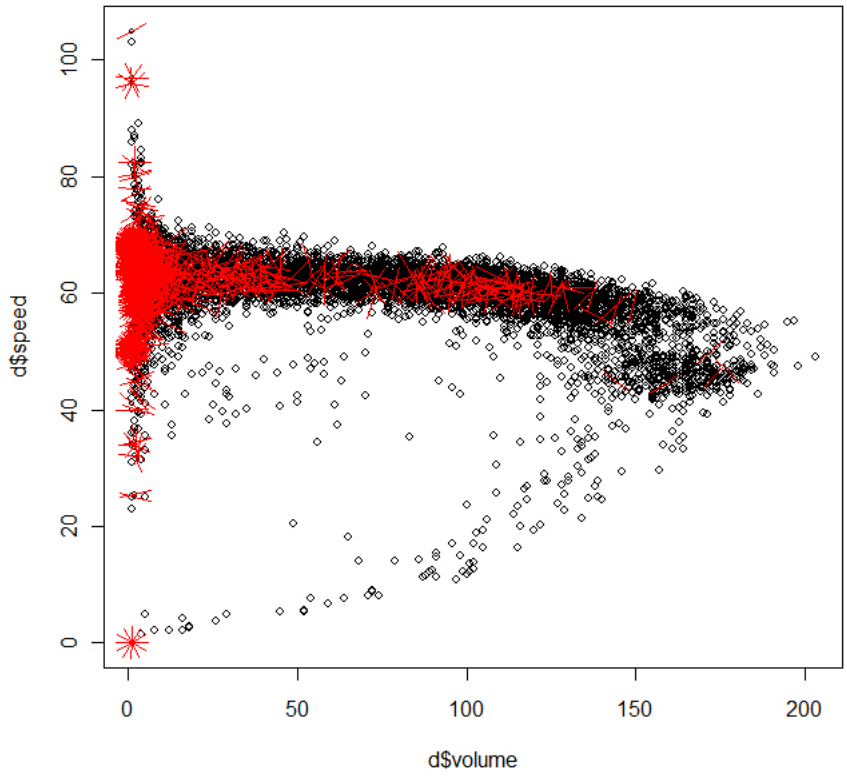


Figure 67 Scatterplot with sunflower overlay describing multiplicity of data points

As was noticed with the Figure 63 method of establishing breaks and assigning colors, the ability of the `terrain.color` function was limited regarding adjusting the range of colors. Using `brewer.pal()` allows substantial variability in the selection of colors. This function is found in the *RColorBrewer* library. Take a few moments to explore the color palette using the code from R Help that demonstrates the variability of the `brewer.pal()`.

B. Bivariate Scatterplots

This task primarily represents an introduction to the plotting method for bivariate graphs. Let's first use the basic plot function to plot simple scatter plots to compare these output plots. Use the accompanying script to generate the two 5 × 5 plots of the loop data set seen in Figure 68. Plotting the data in a minimal amount of graphic windows allows for quick comparison of the corridor, but does not provide detail regarding the specific sensors.

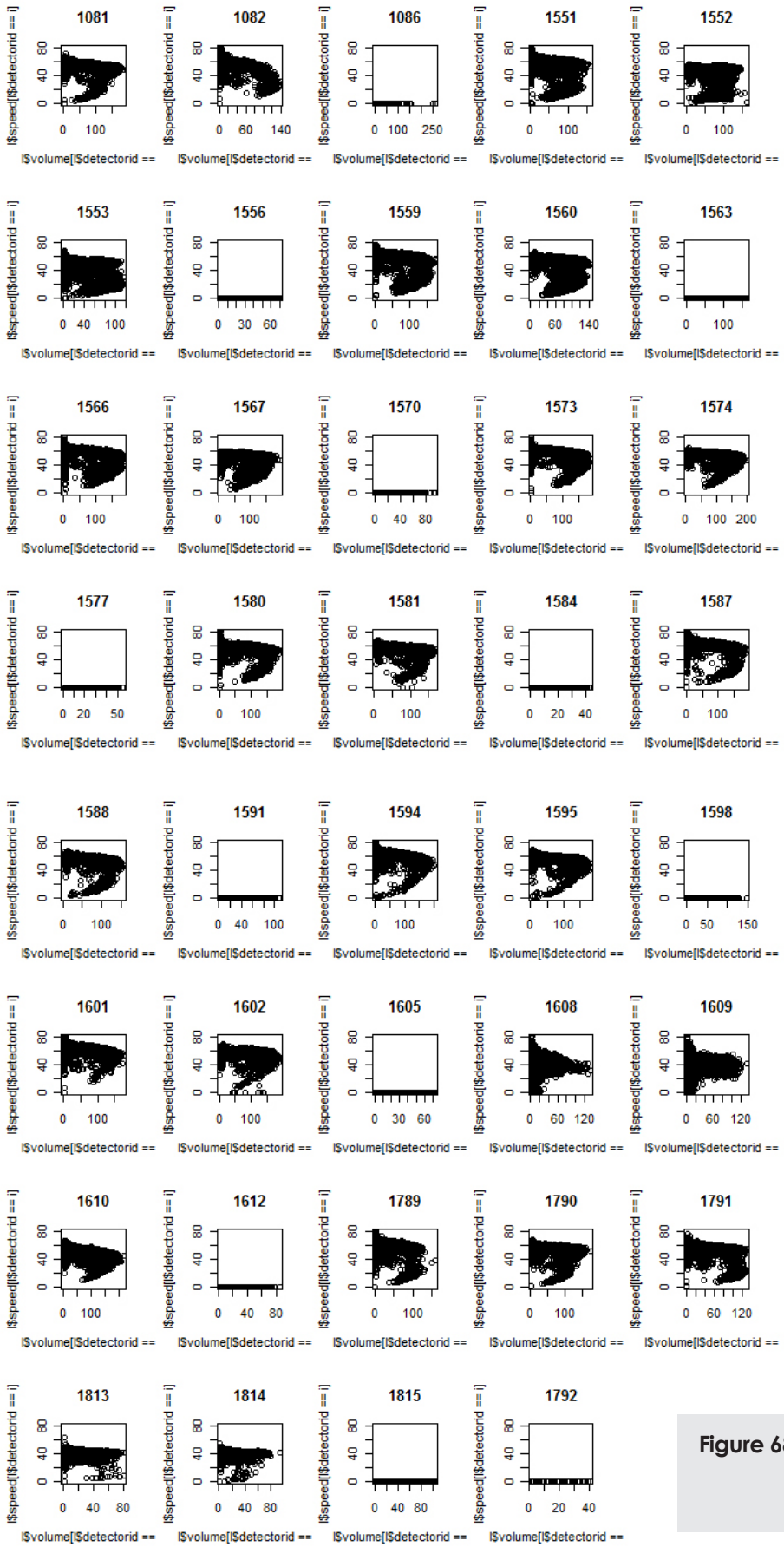


Figure 68 Bivariate plots of volume and speed from the loop dataset

A more refined graphic can be generated by xyplot within the *lattice* package. This function combines a set of arguments that include layout, aspects ratio, legends, axis annotation and many other details, and represents bivariate relation in a consistent way. An example of this function can be seen in Figure 69 (below) which generates scatterplots for the same data set as Figure 68.

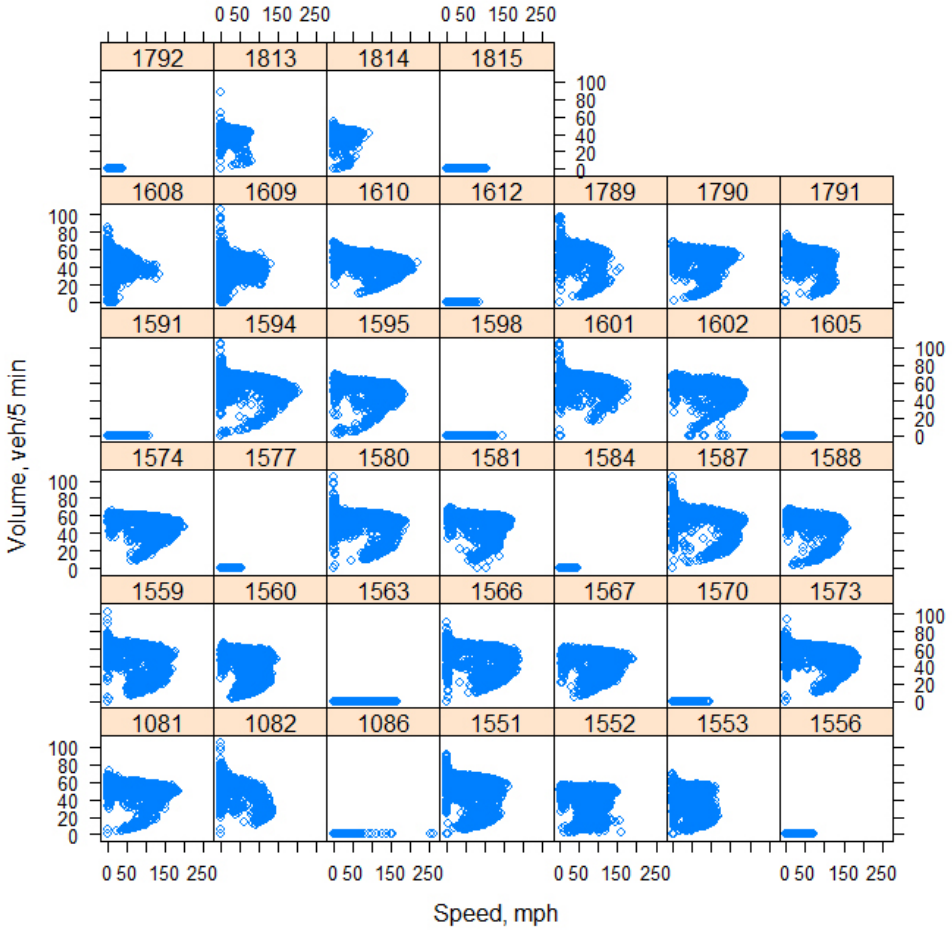


Figure 69 Graphic of speed and volume data from loop dataset using xyplot() in the lattice package

C. Breaks

The accompanying script further demonstrates the methods for establishing intervals by using other functions that have the same intention as the manual process presented and demonstrated by Figure 63. A key benefit to `classInterval` is the addition of variability regarding the calculation of breaks for the interval. The function calculates the break points based on user selected methodologies built into the function. Some of the available methodologies include: fixed, pretty based style, equal, and quantile.

The accompanying script also introduces the `findInterval()` function which takes the user defined breaks or the `classIntervals()` defined breaks and assigns each data point a bin number reflecting the interval that the data point has been defined to fall within.

D. Advanced Graphical Multivariate Displays

The final task within this activity is the introduction of more complex contour plots found within the *graphics* package. The first plot function is `image()` and generates a plot of the data in similar concept to that of a transparent graphic. Instead of adding the third dimension of the graphic using a third variable the graphic used a two-dimensional kernel density estimate function to establish a three dimensional matrix with the z variable being the estimated density of recurring values within the dataset.

The second and third plots are simplistic contour plots that demonstrate the 2-D kernel density estimation using contour lines. The `contour()` function generates a plot similar to a topographic map with the value of the line printed in the line. The number of levels can be controlled or where the levels occur using either the `nlevels=` or `levels=` arugments within the function. Similarly the levels can be controlled within the `filled.contour()` function using the same arguments. The concept of the filled contour and the simple contour are same, but the presentation of the filled contour lacks values within the graphic, only in the legend and the difference between the levels is defined by color (see Figure 70).

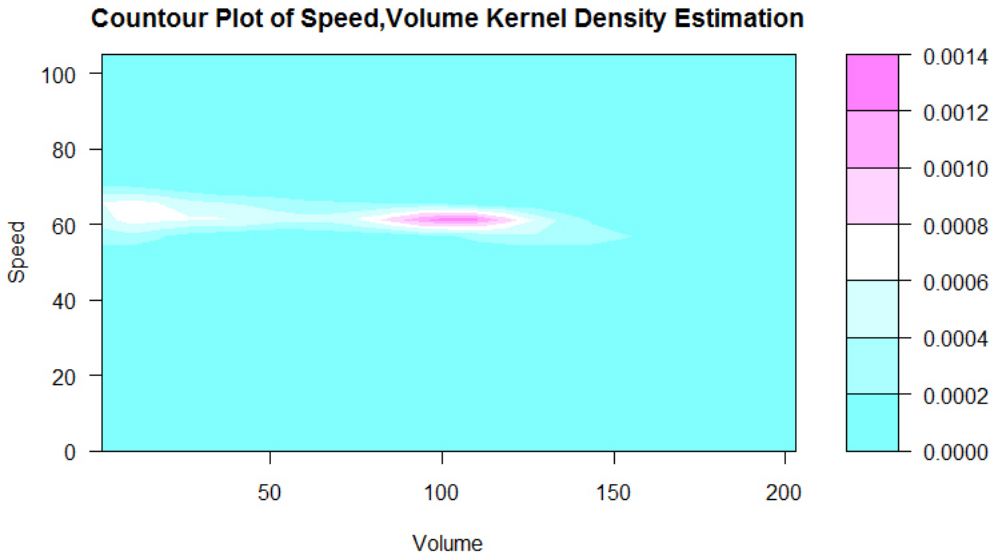


Figure 70 Filled contour plot of volume versus speed with third dimension of kernel density estimation

Finally, make sure to annotate your code and demonstrate your exploration of the functions within this activity. As always make sure your script executes.

Deliverable

This is a discovery activity (it helps you put together some things that you have learned to date). You don't need to expound too much on why it worked, but upload your R code cleaned and commented to the class dropbox.

Assessment

Your score is based on the following criteria:

Activity 37 Grading Rubric

	Excellent (10)	Good (8)	Poor (6)	NONE
Script	Organized, complete, accurate and executes.	Missing minor parts, but executes and is otherwise organized and accurate.	Missing significant portions of the activity, unorganized, inaccurate, but executes.	Code does not execute
Annotation	Annotations are complete and describe what the code is accomplishing.	Some annotations are incomplete or do not describe what the code is accomplishing.	No annotations were provided.	Code does not execute
Discussion / Commentary	Insightful discussion or commentary relating to the question at hand demonstrating student understanding of the task.	Discussion or commentary was incomplete.	Minimal to no discussion or commentary.	Code does not execute

Introduction to Random Sampling

ACTIVITY 38

A random sample provides equal probabilities for all subsets of this data frame. Each subject in the sample has an equal probability of selection. This lowers sampling bias. This independent exercise will help you learn how to generate simple random samples, and use the t-test to examine random samples. Furthermore, you will develop an understanding of the effects of the number of sampling elements on confidence interval.

Purpose

This activity will help you to combine lessons from descriptive statistics with an exploration of data.

Learning Objective

Use graphical analysis and scripting to explore the simple random sampling design.

Required Resources

- R, R Studio
- PostgreSQL ODBC driver
- Direct connection to PostgreSQL with R
- Sample R script

Time Allocated

90 minutes in class

Tasks

A. Introduction to Random Sampling

With a sample application of R, it is easy to begin to develop more advanced numerical methods such as simulations, bootstrapping, and Monte-Carlo type analysis. The most basic function is `sample()` which samples (randomly) *n* elements from a data frame *x*. Let's apply this method to show how the mean of a dataset is just an estimate of the true value. The larger the sample, the more likely the estimate is true but there is always some element of chance involved with drawing conclusions on statistically tested samples.

This activity is accompanied with a script that contains a simple loop, where the sample size has been set to 150. In each t-test, 100 random samples are selected separately from *Set1* and *Set2*, then a t-test of the equality of the means are returned. The 100 p-values of each t test are plotted in Figure 71 as well as a histogram. It indicates in most tests, the null hypothesis is rejected or, the difference in means of two random samples is not zero.

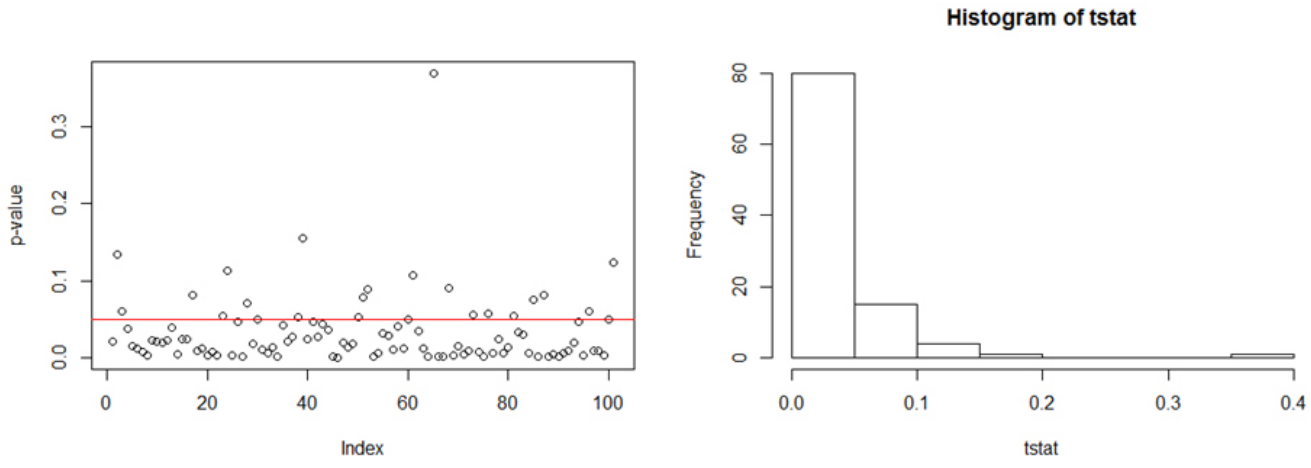


Figure 71 Random sample graphics of p-values for 100 t-tests

Since the sampling is random, your plot will not necessarily look like those in Figure 74. Rerun the code to see the plots changing. Reduce the sample size and see how this dramatically changes the outcome. This is primarily because the difference we are trying to detect is fairly small (between the means) and with smaller samples, the confidence interval becomes larger and more tests lead to the conclusion that there is no difference of the mean.

B. Effect of n on confidence interval

Use the sampling procedure shown above for the data from *set 1* to demonstrate the effect of *n*, or the number of sampling elements, on the confidence interval estimate. Write a loop that calculates the mean and 95th confidence interval for samples of the size 1:249. The last calculation (of all elements in set 1) will be what is shown in Figure 71. Make a plot with the x-axis being the number of elements in the sample and the upper interval, mean, and lower interval plotted on the y-axis. Comment on your observation and interpretation of this plot.

Deliverable

Prepare a short write up of your discovery. Submit printed copy and attach graphs.

Assessment

Participation points (10)

Activity 38 Grading Rubric

	Excellent (10)	Good (8)	Poor (6)
Graphic	Graphics are complete and reflect the discussion regarding the effect of sample size on the confidence interval.	The graphic lacks visual components that assist in the viewer understanding the purpose of the graphic.	Graphics cannot be understood and do not reflect the learning concepts of the activity.
Discussion/ Commentary	Insightful discussion or commentary relating to the question at hand demonstrating student understanding of the task.	Discussion or commentary was partially incomplete.	Minimal to no discussion or commentary.

CHAPTER 6 Putting it All Together: An Independent Structured Analysis

"Anyone who has never made a mistake has never tried anything new."

Albert Einstein, Physicist 1879-1955

Your final task is to prepare a short report documenting your analysis (no more than 4 pages, not including cover or appendices). Include your R code in the Appendix. Prepare a short Powerpoint show to present to your peers during final exam week. This presentation should not exceed 8 minutes.

Research Topics List

Topic	Title	Data Set
1	Assessing The Accessibility of Trimet Bus Stops	Transit
2	Assessing Trimet Bus Headway Reliability	Transit
3	Bicycle Performance	Bicycle
4	Freeway Data And Incidents	Freeway
5	Freeway Data And Weather	Freeway
6	WIM Data – Side By Side Loadings	WIM

Student Notes

Assessing The Accessibility of TriMet Bus Stops

TOPIC 1

Research Question

This project involves using the TriMet Bus Dispatch System (BDS) data for Route 19 and information about bus stop amenities to evaluate accessibility to stops and ease of use by riders in need of ADA-compliant facilities. Given the limited resources available to augment bus stops, which stop locations along Route 19 should be given priority for amenity and construction upgrades?

Background

Public transportation systems must be accessible to many types of users with varying physical abilities. While most transit passengers may be able to traverse less-than-ideal surfaces to access a transit stop, those with physical impairments can find it challenging unless a paved sidewalk of adequate width is provided. Concrete landing pads provide a stable surface for wheelchair-bound passengers to wait for and board a transit bus. Although these types of transit stop accessibility measures are legally required for handicapped persons, they make it easier for all users to reach the stop. Modern bus stop designs focus on "universal design", rather than on specific specifications from the Americans with Disabilities Act (ADA), which are considered to be minimum standards. Universal design recognizes the variety of bus stop users and their abilities and gears designs towards the ease of use by all users, and ultimately provides a higher level of access to those users targeted by ADA designs. While the ADA sets forth specific design standards for cross slopes, approach paths, landing pad dimensions, and other design parameters, this project focuses on the basic provision of ADA-related amenities for bus stop accessibility.

In evaluating the TriMet data for bus stop accessibility, you may want to consider the following:

- Using the warrants of lift usage, daily passenger boardings, and daily passenger alightings given in TriMet's Bus Stops Guidelines, identify bus stop locations along Route 19 which are deficient for front (ADA) landing pads (see the ADA and ABA Accessibility Guidelines for Buildings and Facilities, §209.2.3), rear landing pads, and shelters.
- How are alightings, boardings, and lift deployments distributed throughout the day?
- Are there locations with frequent lift deployment or high boarding activity which do not have either a landing pad and/or a sidewalk? (In some locations, sidewalks adjacent to the curb may serve as landing pads.)
- Are there locations with high alighting counts that do not have a rear landing pad to accompany the front ADA landing pad?
- Which locations should be considered for seating upgrades or shelters?
- How would you prioritize the bus stop locations in need of ADA-compliant amenities?

References

Assessing TriMet Bus Headway Reliability

TOPIC 2

Research Question

This project involves using the TriMet Bus Dispatch System (BDS) data for Route 19 to evaluate schedule adherence and identify locations and times of day when schedule adherence problems arise and bus bunching is an issue. How do bus bunching problems increase as buses travel along the route?

Background

Bus "bunching" occurs when buses on a route have problems maintaining the pre-determined headway and "bunch" together along the route. These schedule adherence problems can stem from a variety of operating conditions, but usually relate to traffic conditions or unexpected transit demands. When bus bunching occurs, inconsistent bus loading patterns can result—often the first bus will be more heavily loaded than normal and subsequent buses will have fewer passengers since insufficient time has elapsed for new passengers to gather at stops. The heavily-loaded lead bus will frequently bypass stops with waiting passengers to allow lightly-loaded following buses to serve the stop. This pattern can frustrate passengers—those on board the lead bus are uncomfortably crowded and those being bypassed at stops do not necessarily recognize that the next bus is moments away. Overall, bus bunching can lead to passenger dissatisfaction with the service, so it is to the transit operator's advantage to maximize schedule adherence along a route.

Select one non-"schedule" stop (see route/schedule map on TriMet's website) toward the outer end of the route (near Gateway Transit Center – such as 92nd and Glisan) and one closer to downtown (MLK & Couch may be a good choice). For inbound trips on Mondays, evaluate and compare the schedule adherence at these two stops. You should do some exploration to find days with bus bunching problems (where trajectory strings overlap).

In evaluating the TriMet BDS data for schedule adherence and bus bunching problems, you may want to consider the following:

- Describe the distribution of headways for buses on Route 19. How do the actual headways compare to the scheduled headway? How does this affect bus loading, assuming passenger arrivals at stops are random events?
- According to the Route 19 data, do schedule adherence problems tend to occur at particular times of day? How and why does bus bunching at the beginning of the route differ from that as the bus approaches downtown?
- Is it reasonable to expect that other similar routes face similar schedule adherence problems under these conditions?
- Part 3 of the Transit Capacity and Level of Service Manual provides guidelines for determining schedule-based level of service for bus systems.

References

Bicycle Performance

TOPIC 3

Research Question

What are the performance differences of bicyclists by gender, season, and grade?

Background

The schema *bicycle* contains two tables: performance and location. The table performance contains data were extracted from video data collected at two intersections in Portland, OR by Dr. Figliozzi and his students. The table location defines the location of the data collection efforts:

- Madison/NE Grand (Flat Grade) Winter Collection Date and Time: 12/4/08, 7:30-10:30 AM; Summer Collection Dates and Time: 7/3/08-7/17/08, 7:30-9:30 AM
- Vancouver/Wiedler (Grade) Winter Collection Date and Time: 12/5/08, 3:00-6:30 PM and Summer Collection Dates and Time: 7/3/08-7/21/08, 3:00-7:00 PM

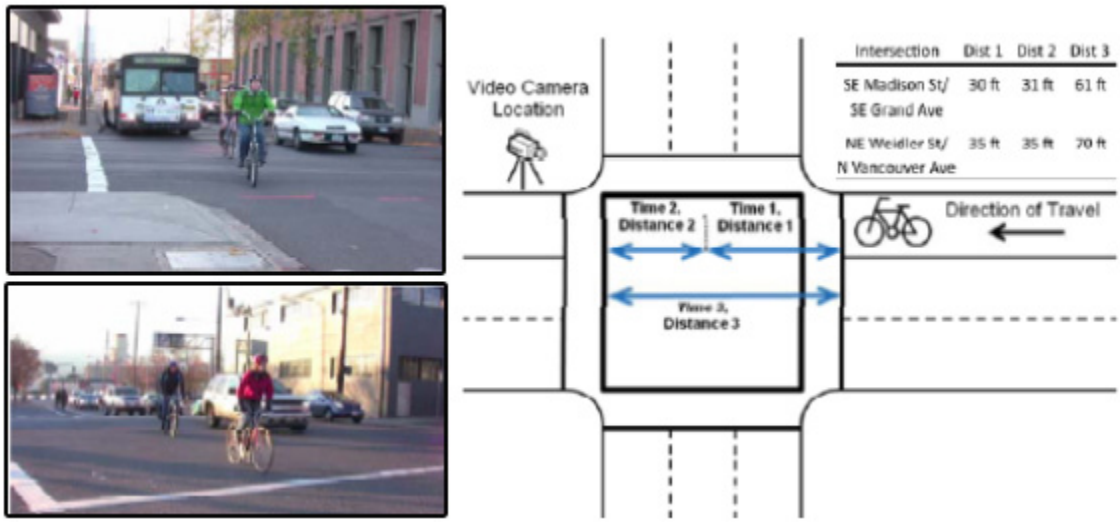


Figure 72

Estimate the descriptive statistics of delay, both numerically and graphically. Statistically quantify these results.

What are the performance differences of bicyclists by gender, season, and grade? Consider a regression model or at the very minimum, an ANOVA analysis of these data. It is not sufficient to just replicate some of the in-class activities.

References

Freeway Data and Incidents

TOPIC 4

Research Question

What are the relationships between the incident types and data?

Could you quantify an incident's impact on delay (i.e., for every one minute of incident duration you can expect xx minutes of additional delay)?

Background

The schema *incidents* contains the table *incidents_217_2009* as well as various look-up tables for the incident data. Begin by developing qualitative descriptive of the incident data fields. Explore the relationships between these variables.

The schema *loop* in the class database contains data obtained from inductive loop sensors on OR-217 SB at 5-minute aggregations for calendar year 2009. The loop sensors collect volume, speed, occupany and metrics on vehicle-miles-traveled, vehicle-hours-traveled, delay and information about bad detector readings in table loop.loopdata_5min_217sb_2009. The loop schema also contains tables *highways*, *detectors*, *ramps*, and *stations*. Detectors are related to ramps and stations.

Next process the *loop* data for some simple data quality metrics to identify a segment that you wish to consider in more detail. Ideally, this segment would have high-quality data. This segment should also have some number of incidents within the segment or downstream of the segment.

Estimate the descriptive statistics of delay, both numerically and graphically. Statistically quantify these results.

Could you quantify an incident's impact on delay (i.e., for every one minute of incident duration you can expect xx minutes of additional delay)?

References

Freeway Data and Weather

TOPIC 5

Research Question

How does weather affect the capacity of a selected segment of OR-217?

Background

The schema *loop* in the class database contains data obtained from inductive loop sensors on OR-217 SB at 5-minute aggregations for calendar year 2009. The loop sensors collect volume, speed, occupancy and metrics on vehicle-miles-traveled, vehicle-hours-traveled, delay and information about bad detector readings in table loop.loopdata_5min_217sb_2009. The loop schema also contains tables *highways*, *detectors*, *ramps*, and *stations*. Detectors are related to ramps and stations.

Begin by processing some simple data quality metrics to identify a segment that you wish to consider in more detail. Ideally, this segment would have high-quality data. Contour plots of data quality would be an excellent tool to explore this quickly. Next, the table *weather.metar_2009* contains hourly precipitation data. You will need to define "capacity" in a repeatable manner for a selected time period, then process the data to generate these measures. R's ability to process large amounts of data can be a limiting if you don't plan properly. One suggestion would be to write SQL code that produces average hourly volumes and speeds that you could join to the weather data. Assign weather information to these observations, then graphically explore the differences in speed-flow or other curves for each weather category. Statistically quantify these results. Consider t-test of the means or ANOVA analysis (See Dalgaard for reference).

References

WIM Data: Side by Side Loadings

TOPIC 6

Research Question

How often and to what magnitude are side-by-side loadings on a bridge structure likely to occur?

Background

The schema *wim* in the class database contains data from 1 month period (August 2009) from the 22 reporting weight-in-motion stations around the state of Oregon. The wim stations are described in table *wim.stations*. The actual sensor readings are given in table *wim.wimdata*. The data include gross vehicle weight, axle weight and spacing, transponder and a timestamp.

A critical loading event for a bridge or structure is the occurrence of two heavily-loaded trucks passing over at nearly the exact same time in a side-by-side event.

First, identify the stations that have WIM sensors in two lanes. Select one station to use, ideally the one with lower flows to make the problem more tractable. Use the data from the selected station to characterize the distributions of interarrival times of trucks, the gross vehicle weight, and speed distributions in both WIM lanes (not all WIM stations have two lanes, only station 9 (Woodburn, NB), station 10 (Woodburn, SB) has two lanes in one direction at WIM station) . You will need to explore how to characterize these distributions by time-of-day. Explore the distribution of these variables using graphical techniques to identify a likely theoretical distribution. Use R's fitting function to finally define the parameters of the theoretical distribution from the empirical data.

While you could certainly attempt to solve this question with a deterministic approach using probability theory, your challenge is to attempt to solve this by creating a simulation using R's ability to randomly sample from a distribution. This approach is a called a Monte-Carlo numerical solution.

In essence, develop a synthesized (simulated) truck flow with gross vehicle weight for each lane. Forecast the arrival of these trucks at the downstream bridge. Identify simultaneous arrivals and the total weight of these arrivals. If you want to add a wrinkle, identify the next downstream significant structure as your target loading span (forcing you to assume trucks will travel at a constant speed from measurement at the WIM to bridge structure, though you could "solve" this assumption with some randomness).

Assume that August data is representative for a longer time period. Ignore day of the week variations in traffic flow but develop or explore time-of-day variations. Depending on how quickly you devise a solution to the problem, you may be constrained on the length of your simulation. Ideally, at least 3 months would be simulated.

References

APPENDIX Dataset Narratives

(A) Bicycle Dataset

Tables

performance and *location*

The schema bicycle contains two tables: *performance* (422 rows) and *location* (2 rows). The table performance contains data were extracted from video data collected at 2 intersections in Portland, OR by Dr. Figliozzi and his students. The table location defines the location of the data collection efforts:

- Madison/NE Grand (Flat Grade) Winter Collection Date and Time: 12/4/08, 7:30-10:30 AM; Summer Collection Dates and Time: 7/3/08-7/17/08, 7:30-9:30 AM
- Vancouver/Wiedler (Grade) Winter Collection Date and Time: 12/5/08, 3:00-6:30 PM and Summer Collection Dates and Time: 7/3/08-7/21/08, 3:00-7:00 PM

Figure 74 describes the data collection set up and the elements recorded in the *performance* table.

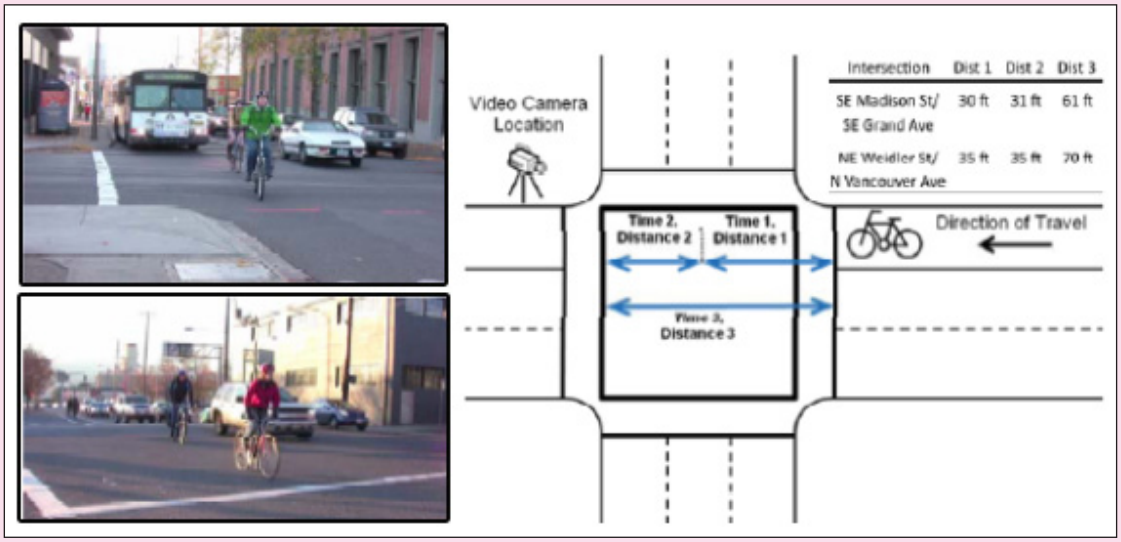


Figure 74

Information

A couple of notes about the data:

1) The rider numbers are not continuous; there are gaps in the numbering within the data that were used for analysis. This is because each rider was numbered in the field. Upon review of the data in video, some riders were excluded from analysis for various reasons (for example, if they stopped in the crosswalk instead of at the first crosswalk line). In order to be able to track a rider from analysis to our field notes, we did not change the numbering of the cyclist. This is why you may notice some rider numbers missing.
2) For the summer analysis, we did not gather the data from the video on baggage, bicycle type, or pedal type.
3) For the summer analysis, we attempted to also collect crossing time data from riders further back in the queue (the second or fifth rider back), when they were present. This is why the rider numbers in the summer are labeled with an "A": to correspond to the 1st rider. In the data I have linked to above, the 2nd and 5th rider's crossing times are NOT included (these rider numbers would be followed by "B" or "E").

(B) Bluetooth Dataset

Tables	*sensors* and *stations*
Information	The schema *bluetooth* in the class database contains data obtained during a 1-week collection period (May 10-17, 2010) on Powell Boulevard in Portland, OR. The bluetooth sensors are placed at intersections described in table *bluetooth.stations*. The actual sensor readings are given in table *bluetooth.sensors*. The sensors read MAC addresses for devices (cell phones mostly) broadcasting to another device wirelessly. This unique signal has been truncated to 7 characters but is sufficient to be considered a unique match. Matching unique MAC addresses between 2 stations can be assumed to be the travel time between those 2 stations.

(C) Incidents Dataset

Tables	*incidents_217_2009* and various lookup tables
Information	The schema *incidents* contains the table *incidents_217_2009* as well as various look-up tables for the incident data. The incident data is generated from Computer aided dispatch(CAD) from ODOT's TMOC. Portland's ATMS includes a comprehensive incident management system, which in turn generates a large computer-aided dispatch (CAD) database, which then contains nformation about all recorded incidents on Portland's freeway system. This information includes the type of incident, the lanes that were blocked as a result of the incident, and the start and end time of the incident. The TMOC can record the X–Y coordinates of an incident, but it is required only if the incident was severe enough to cause significant delay.

(D) Loop Dataset

Tables	*loopdata_5min_217sb_2009* and stations, ramps, highways, detectors [a document describing all tables in this schema]
Information	The schema *loop* in the class database contains data obtained from inductive loop sensors on OR-217 SB at 5-minute aggregations for calendar year 2009. The loop sensors collect volume, speed, occupany and metrics on vehicle-miles-traveled, vehicle-hours-traveled, delay and information about bad detector readings in table *loop.loopdata_5min_217sb_2009*. The loop schema also contains tables *highways*, *detectors*, *ramps*, and *stations*. Detectors are related to ramps and stations.

(E) Weather Dataset

Table	*metar_2009*
Information	The table *weather.metar_2009* contains hourly precipitation data gathered the METAR weather observations provided the National Oceanic and Atmospheric Administration (NOAA). METAR is the international standard code format for hourly surface weather observations which is analogous to the SA coding currently used in the US. The acronym roughly translates from French as Aviation Routine Weather Report. Weather observations are collected from three points in the Portland region: Portland International Airport and the suburbs of Hillsboro and Troutdale, located west and east, respectively, of the city of Portland.

(F) TriMet Dataset

Tables	*route19* and *trimet_stop_amenities*
Information	TriMet Bus Dispatch System (BDS) data for Route 19 for 2009 is incluced in this schema. The BDS includes automatic vehicle location via a satellitebased Global Positioning System (GPS) on all buses as well as automatic passenger counters installed on a majority of the existing fleet and all new bus acquisitions. The BDS records contain the route number, direction, trip number, date, vehicle number, operator identification, bus stop identification, stop arrival time, stop departure time, boardings, alightings, and passenger load. Also recorded is whether the doors of the bus were opened, whether the lift was used, the dwell time, maximum speed between previous and current stop, and GPS coordinates.

(G) Weigh-in-motion (WIM) Dataset

Tables	*wim* and *stations*
Information	The schema *wim* in the class database contains data from 1 month period (August 2009) from the 22 reporting weight-in-motion stations around the state of Oregon. At each of the Green Light stations, approaching trucks are directed into the appropriate lane on the mainline highway. At a location upstream from the static weigh station, if a truck is equipped with a transponder, it is identified by the reader. The unique aspect of Oregon's system is that this unique transponder identification and subsequent data are recoded together. At this same location, the vehicles are weighed in motion by load cells in the pavement. The observation consists of left and right axle weights as well as the spacing of these axles. The data also include speed, a timestamp and the lane of observation (some stations are multilane), length, gross vehicle weight, and a count of the number of axles. As part of the proprietary control program by International Road Dynamics (IRD), a sieved-based classification algorithm uses the axle spacing parameters to classify vehicles.

Student Notes