





WORKING WITH TIME IN R

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.

 <h3>PURPOSE</h3> <p>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.</p>	 <h3>LEARNING OBJECTIVE</h3> <p>To develop an understanding of how R stores and displays time-based data and some techniques for manipulating, displaying and modifying.</p>
 <h3>REQUIRED RESOURCES</h3> <ul style="list-style-type: none"> o R, R Studio o R code on working with time o Packages: RODBC and lubridate 	 <h3>TIME ALLOCATED</h3> <p>60 minutes out-of-class</p>

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(l1$starttime, l1$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.

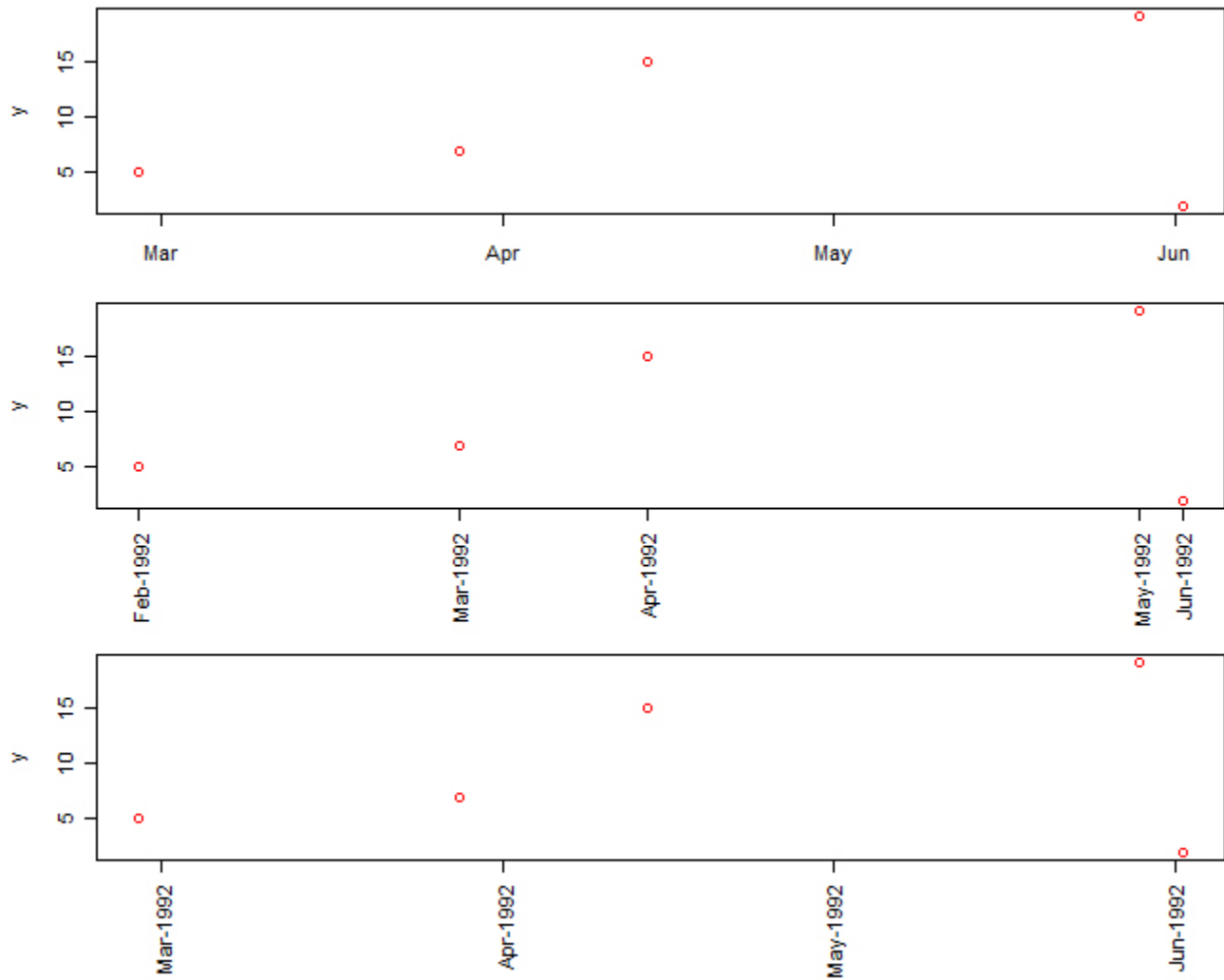


Figure 43 Plots showing R time

```

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 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:

```

l11<-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

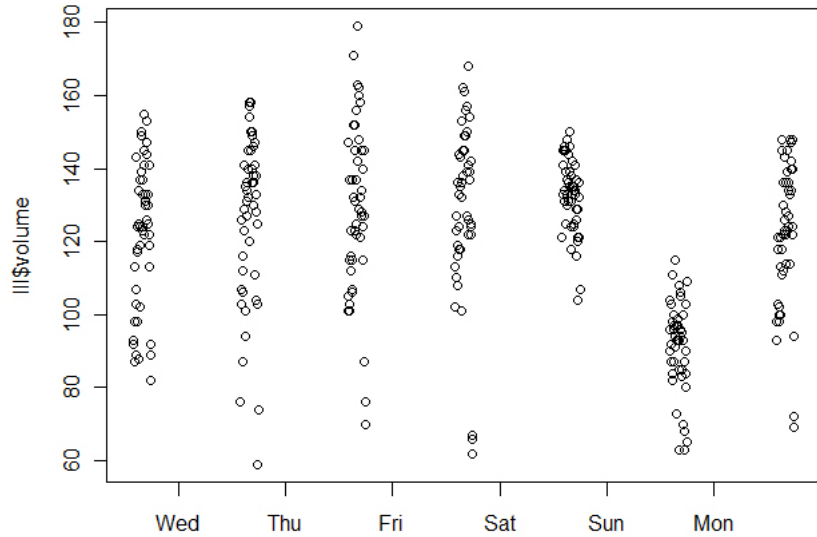


Figure 44 Volume on SB OR-217 at OR-10 2:00PM-6:00PM in five minute intervals for a seven day period

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