Interactive Assignment 3: The Tidyverse and Data Manipulation

Interactive Assignment 3: The tidyverse

This assignment explores the use of the "tidyverse", or a set of R packages designed to import, format, and analyze data in "tidy" formats.

In this lab, we will:

1. How to use `filter()` to select part of your data
2. How to arrange and select data using the `arrange()` and `select()` funtions
3. How to use `mutate()` to make new columns in a dataframe.

Remember for this assignment to create a R Notebook with all your answers. The R commands should be in code chunks and you should have annotations so that you tell me what questions you are answering with each of the code chunks. These R notebooks are going to be the notes for this class, so make sure you note what the code chunks are doing in a way that both you and I can understand.

For this assignment, you are going to work through some questions which are similar to Chapter 5.1-5.5 of R for Data Science. For more background, you may want to read this link, which has some good examples of the techniques used here. http://r4ds.had.co.nz/transform.html

To start with the tidyverse, you need to install the package 'tidyverse', which is a collection of useful packages used to analyze data. You will also need to install the package 'nycflights13' To install the package, go to Tools -> Install Packages and then type 'tidyverse' in the window to install the package. Then do the same for 'nycflights13'. When you do this, some dialog boxes may appear. If that is the case, just select yes, or approve them.

Step 1: Create your R notebook in a new folder on your computer. After you do this, you should save the R notebook and load the `tidyverse` and `nycflights13` library with the following code:

```
library('tidyverse')
library('nycflights13')
```

This activity will use the data available in the `nycflights13` package, which is a package containing data on all the flights which departed from New York City in 2013. Install and load this package by using the following commands.

When loading data that are built into R, you do not have to load them to your workspace. The `nycflights13` library will load the data we need as the `flights` dataframe. To view it, you can type `View(flights)` into your Console. Don't type a new code chunk because this may cause errors when you try to knit it. To view a summary of the data in the output, you can simply type `flights`.

The tidyverse functions for manipulating data work using the following logic:

```
output = command(dataframe, option1, option2, ... )
```

There are several commands, each with a different purpose. Each command has a series of arguments. The first argument is the dataframe you are going to manipulate. The second argument and each following argument describes an option about what you can do with the data frame. The output from this command is a new data frame. If you do not have an output variable specified, it will just output the results to the console.

Using `filter()` to select rows

The first command we will use is called `filter()` and it is used to select a subset of rows from a data frame.

The `filter()` command selects a subset of the rows from a data frame. The first argument is the data frame itself. Each argument afterward is a *comparison* expression which says which rows to select. Here is one example, which selects all the flights which left in March:

```
filter(flights, month == 3)
```

```
## # A tibble: 28,834 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     3     1        4           2159       125      318
##  2  2013     3     1       50           2358        52      526
##  3  2013     3     1      117           2245       152      223
##  4  2013     3     1      454            500        -6      633
##  5  2013     3     1      505            515       -10      746
##  6  2013     3     1      521            530        -9      813
##  7  2013     3     1      537            540        -3      856
##  8  2013     3     1      541            545        -4     1014
##  9  2013     3     1      549            600       -11      639
## 10  2013     3     1      550            600       -10      747
## # ... with 28,824 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Step 2: Go ahead and create a code chunk which filters only the flights selected in March.

Typing this outputs the answer to the screen, or in this case, a small portion of the answer to the screen, since there are thousands of flights from March.

If you would want to save the result to a new dataframe, you would have to specify a variable for the output:

```
march = filter(flights, month == 3)
```

The arguments for `filter()` are conditionals. Comparisons are expressions which use logic in order to select rows based on a certain criterion. Here are a few examples below:

- `==` This selects rows where the value in the variable is equal to the value selected. Note that this is a double equals, or the equal character twice. For example, to select rows where the variable `origin` is "JFK", I would type `origin = "JFK"`, remembering to put "JFK" in quotes since it is a string.
- `!=` This means *not equal*, and is the opposite of the `==` operator. This selects rows where the value is not equal to the given variable. For instance, typing `month != 4` would select all flights that were not in April, the fourth month.
- `>` and `<` These are greater than and less than. To do greater than or equal to, you would type `>=` and less than or equal to would be `<=`. Note, these can only be used on numeric values.
- You can separate two or more expressions with the `|` (vertical line, press shift and backslash) to select conditions where one **or** the other conditions are true. For instance, to select flights which left in January or February, you could type `select(flights, month == 1 | month == 2)`

Step 3: Using the `filter()` command , find the following:

- Had an arrival delay of two or more hours (remember that the delay variable is listed in minutes)
- Flew to Dulles International Airport (IAD)
- Were operated by United, American, or Delta

- Departed in summer (July, August, and September)

List each of these as code chunks with annotations telling me what questions you are trying to answer.

You may want to select rows that fit more than one condition. If you add more than one argument to `filter()`, the command will select rows where all of the conditions are true. For instance, if I want to select flights which left on July 4th, I want flights where `month == 7` AND `day == 4`. To do this, I just list the two arguments, separating them with commas. There is no limit as to how many of these I can list.

For example, the following selects all the flights which left from JFK on July 4th in the morning, before noon.

```
filter(flights, month == 7, day == 4, origin == "JFK", dep_time < 1200)
```

```
## # A tibble: 104 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     7     4       11           2359        12      400
##  2  2013     7     4       59           2359        60      501
##  3  2013     7     4      538            540        -2      835
##  4  2013     7     4      539            545        -6      918
##  5  2013     7     4      553            600        -7      659
##  6  2013     7     4      556            600        -4      834
##  7  2013     7     4      602            605        -3      830
##  8  2013     7     4      609            610        -1      902
##  9  2013     7     4      611            615        -4      754
## 10  2013     7     4      615            611         4      952
## # ... with 94 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

When selecting data, you may wonder when you have an AND situation versus an OR situation. Here is a way to remember it. AND situations make your output smaller. A given row has to fit every condition that you give, so each additional condition makes your output smaller and smaller. In contrast, OR situations make your output bigger. Each OR you add will increase the size of your data. AND situations are very specific (like flights leaving the morning of July 4th) whereas OR situations are not as specific (like flights leaving in the winter).

Step 4: Create code chunks to answer the following questions. Make sure you have annotations to tell me what you are doing.

- Flights that left on time or early and had an arrival delay
- Flights that were delayed in December
- Flights that had a time in air (the air_time variable) between 6 hours and 8 hours

The final thing you may want to do with the `filter()` command is to remove or look at missing data. Missing data is listed in R as `NA`. However, you cannot choose observations that are missing by using a comparison operator. Typing `data == NA` will not work. R has the `is.na()` command to test for missing data.

For instance, to choose flights with a missing departure time, I would type:

```
filter(flights, is.na(dep_time))
```

```
## # A tibble: 8,255 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
```

```
##  1  2013     1     1       NA           1630        NA       NA
##  2  2013     1     1       NA           1935        NA       NA
##  3  2013     1     1       NA           1500        NA       NA
##  4  2013     1     1       NA            600        NA       NA
##  5  2013     1     2       NA           1540        NA       NA
##  6  2013     1     2       NA           1620        NA       NA
##  7  2013     1     2       NA           1355        NA       NA
##  8  2013     1     2       NA           1420        NA       NA
##  9  2013     1     2       NA           1321        NA       NA
## 10  2013     1     2       NA           1545        NA       NA
## # ... with 8,245 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

If I want to remove these values, I want to choose the opposite, which are flights that do not have an `NA` for dep_time. In R, the exclamation point represents NOT. To choose flights without a missing dep_time, I would type:

```
filter(flights, !is.na(dep_time))
```

Step 5: Removing missing data (or data with NAs) is important. How would I remove all the flights with missing arrival times.

Sorting with the `arrange()` command

The `arrange()` command allows us to sort data. It uses the same format as the filter command, with the arguments listing the variables that will be used to sort the data, starting by sorting by the first variable, then the second variable, and so forth.

Sorting is ascending by default, starting with the lowest values for numbers and the first letters, for strings.

For instance, if we want to sort the data by destination, we would type:

```
arrange(flights, dest)
```

If we wanted to sort flights by destination, then departure time, we would type:

```
arrange(flights, dest, dep_time)
```

If we wanted to sort a variable in descending order, or reverse the normal order, we have to use the `desc()` command. So if we wanted to sort the flights by months in reverse, starting with the flights in December, then November, etc., we would type:

```
arrange(flights, desc(month))
```

Step 6: Type code chunks with annotations to do the following:

- sort the flights by scheduled departure time?
- sort the flights so that the ones with the longest arrival delays are first
- sort the flights by carrier and then origin

Selecting columns with the `select()` command

The `select()` command allows us to select columns in a data frame and create a smaller data frame. This is useful if we have data frames with many columns and want to select a smaller subset of those columns. The

power of select is that it can use several different functions to select columns, based on the column names. For instance, if we want to select the month, day, and dep_time variables, we would type:

```
select(flights, month, day, dep_time)
```

- starts_with("string"): selects variables whose names begin with "string".
- ends_with("string"): selects all variables whose names end with the string "string".
- contains("str"): selects all variables which contain the letters "str".

For example, we can use the following command to select month, year, and all the variables which deal with arrival times and delays.

```
select(flights, month, year, starts_with("arr"))
```

Step 7: Create code chunks to do the following: * all the variables which involve delays * all the variables which involve dates and times

Creating new variables using `mutate()`

The classic way to create new variables using mathematical functions or other functions is to use the `dataframe$variable` notation and create math functions based on this.

An example might be below:

```
flights$flight_time = flights$arr_time - flights$dep_time
```

The dplyr function as part of the tidyverse uses a different way to do this and uses the mutate function. In this section, I will mention how you could use both, and you can choose which one you would prefer.

For instance, I may want to create a new variable which lists the minutes since midnight each flight takes off. The flights dataframe has the variable `hour` which lists the hour that each flight took off and the variable `minute` which lists the minutes.

The mutate command uses the following logic. It starts with the data frame, then has a statement which is the name of the new variable equal to a command. To calculate the number of minutes since midnight using the mutate command, I would type:

```
mutate(flights, mins_midnight = hour*60 + minute)
```

```
## # A tibble: 336,776 x 20
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 336,766 more rows, and 13 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, mins_midnight <dbl>
```

I can use the mutate command to create multiple variables by adding them as extra arguments with commas. The `transmute()` command is like mutate, but it only contains the variables which are calculated and removes the rest of the variables. For instance, if I want to keep the month, day, and mins_midnight variable, I would type:

```
transmute(flights, month, day, mins_midnight = hour*60 + minute)
```

Step 8: Create a new variable called flight_time which is the arrival time minus departure time. Compare your new variable flight_time with the current variable air_time. In the annotations, answer the following questions: What do you expect to see? What do you see?

Step 9: Compare dep_time, sched_dep_time, and dep_delay. How would you expect those three numbers to be related?

Combining operations

When we edit data, we may want to do several operations. One way to do this is to create temporary dataframes. This type of strategy is inefficient, but it's simple to think about. There are better ways to do this which we will do later, but I want to illustrate this here.

One thing we might want to do is only look at the flights in December and then create new variables. If for instance we wanted to only look at the time between sched_dep_time and arr_time (the time when a flight is scheduled to leave and when it actually arrives), we might do the following.

Step 10: Add the following two steps as one code chunk.

First we create a dataframe containing only rows in December:

```
flights_dec = filter(flights, month == 12)
```

Then we use the mutate command on the new dataframe we've created.

```
flights_dec = mutate(flights_dec, total_time = arr_time - sched_dep_time)
```

This type of approach may come in handy later, when doing multiple data comparisons, or when we use the data manipulation commands before creating a graph.

Summary

We covered a lot in this lab. Much of this seems like busywork now, but this will be very helpful when you're cleaning your own data. You could take the old-fashioned approach of editing spreadsheets by hand, which is laborious, error-prone, not reproducible, and really boring. Or you could learn these commands to analyze your data.

When I do analyses, I usually have several commands I use to clean my data before actually analyzing it. I would say that 80% of the R commands I use are involved in manipulating data. Once you learn these commands well, you will be able to really analyze your data.

After completing this lab, you should know the following

- What is tidy data and why this is important (covered in the reading and lecture)
- How to use the `filter()` command to select a subset of rows
- How to use the `mutate()` command to create new variables
- How to use the `select()` command to select a subset of columns from a dataframe