Midterm II Winter 2021 Key

Data Science for Managers

Concepts and Code Interpretation

# Exam parameters:

* Closed book
* No time limit
* Complete in one sitting
* Submit before starting the second part of the exam
* Due: 23 March before class
* Submit a word document

# Concepts

1. Explain why pivot\_longer and pivot\_wider are opposite operations.

The function pivot\_longer stacks chosen columns from a wide tibble on top of each other in a new tibble. This function has at least three arguments. The first identifies the columns that we want to stack. The second names a column whose values correspond to the variable names. The third names a column where the values of the variable are stored. Each row in the new long tibble contains the name of the original variable and the original value that corresponds to that variable.

The function pivot\_wider converts a long tibble into one with a wide structure. This function takes two arguments. The first is names\_from. This identifies the categorical variable that contains the names of the variables to be converted into columns. The second is values\_from gives the column in the long tibble that contains the values for the new columns.

In summary, the function pivot\_longer converts a wide tibble into one with a long structure. The function pivot\_wider does just the opposite because it converts a long tibble into one with a wide structure.

1. Compare and contrast mutating and filtering joins.

These two concepts are similar to filtering and mutating when using the respective dplyr commands. The filter command chooses a subset of rows from a tibble. The mutate command creates new variables in a tibble. A mutating join adds columns or new information to a tibble from another related tibble. Filtering joins chooses a subset of observations from a tibble.

1. First define factors and explain what is meant by a factor’s level and label.

Factors are categorical variables. The categorical variables can be either nominal or ordinal. The number of categories should be a limited and manageable.

The levels are the possible values for the categories. The levels can be integers or characters. Sometimes, levels are codes that are difficult to remember and interpret. Labels are descriptions that are associated with each level. This makes factors easier to understand and interpret. Statistical software usually prints out the labels rather than levels when reporting the results of an analysis.

1. Explain when and why you should write an R function.

Our textbook states that

“Functions allow you to automate common tasks in a more powerful and general way than copy-and-pasting. Writing a function has three big advantages over using copy-and-paste:

* You can give a function an evocative name that makes your code easier to understand.
* As requirements change, you only need to update code in one place, instead of many.
* You eliminate the chance of making incidental mistakes when you copy and paste (i.e. updating a variable name in one place, but not in another).

You should consider writing a function whenever you’ve copied and pasted a block of code more than twice (i.e. you now have three copies of the same code).”

1. Explain the relationship between the cut function and if else conditional execution.

The cut function is used to create categories based on numerical values in a vector. We first define the ranges for each category. Then we sort each observation into the appropriate category.

The cut function takes a numerical vector and classifies each observation according to the value of the numerical vector. For example, let’s consider the case where we have a vector of ages for our sampled population. We can classify each person as young, middle, or old by the following:



This would be done with

cut(x, breaks = c(0, 29, 64, +Inf), labels = c(“young”, “middle”, “old”), right = TRUE).

Inherent in this logic is conditional execution. If the person is less than 30, they are “young”, if the are 65 or older, then they are “old”. All other ages are “middle”. This logic can be put into an if else framework using the following code:

if(age <= 29){

"young"

} else if (age >= 65){

"old"

} else{

"middle"

}

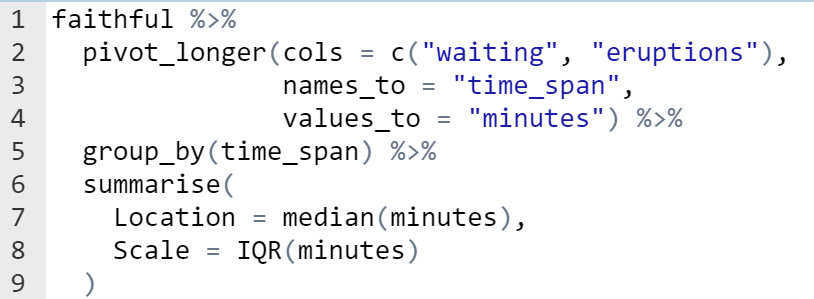
# Code Interpretation

As you interpret the following code, feel free to copy and paste it into RStudio so you can experiment with the outcomes for each line of code. Make sure that you explain your code by identifying each line of code you are interpreting. Make sure that your explanation focuses on what the code is accomplishing and doesn’t just repeat the code.

## Code Chunk #1

### Interpret lines 2 thru 8

Please recall that the faithful tibble has two columns or variables. They are the waiting time between eruptions (waiting) and the duration of the eruption of Old Faithful (eruptions).



### Output

## # A tibble: 2 x 3  
## time\_span Location Scale  
## <chr> <dbl> <dbl>  
## 1 eruptions 4 2.29  
## 2 waiting 76 24

Line 2 - 4: Converts a wide tibble into a long tibble. Faithful only contains two variables (eruptions and waiting) so we can stack them on top of each other. The key variable time\_span contains the names of the variables being stacked. The values of the variables are put into a variable called minutes.

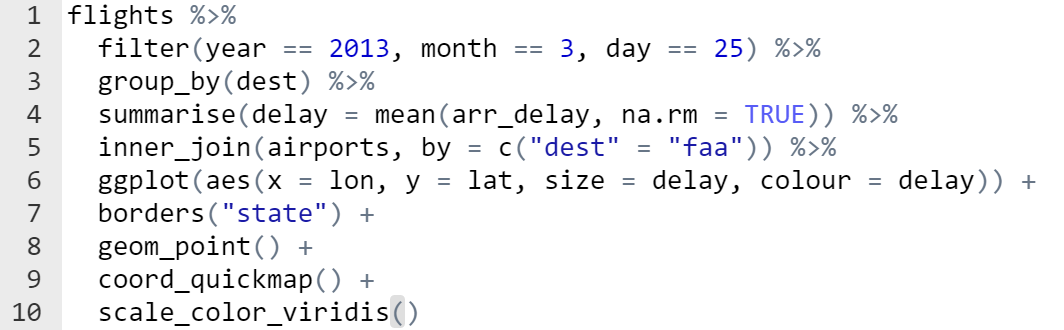
Line 5: Anticipates the calculations of the median and interquartile range in the next steps. The group\_by statement allows us to calculate the median and IQR for each of the time\_spans (eruptions and waiting).

Lines 6 - 8: Summarise collapses the tibble into two rows and two columns. We have the location (median) and the scale (interquartile range) for the two variables: waiting and eruptions.

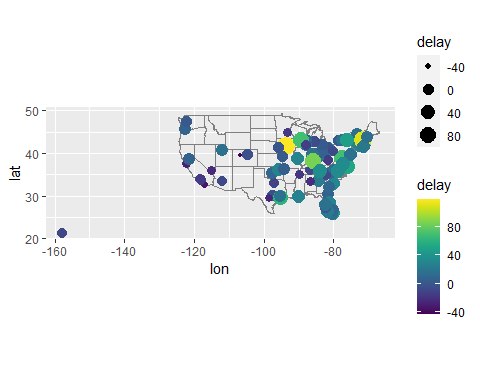
## Code Chunk #2

### Interpret lines 2 thru 5

Please remember that the flights tibble reports all flights leaving from one of the three New York City airports during the year 2013. The airports tibble include information about each of the airports in the FAA database.



### Output



Line 2: Choose those observations for March 25, 2013.

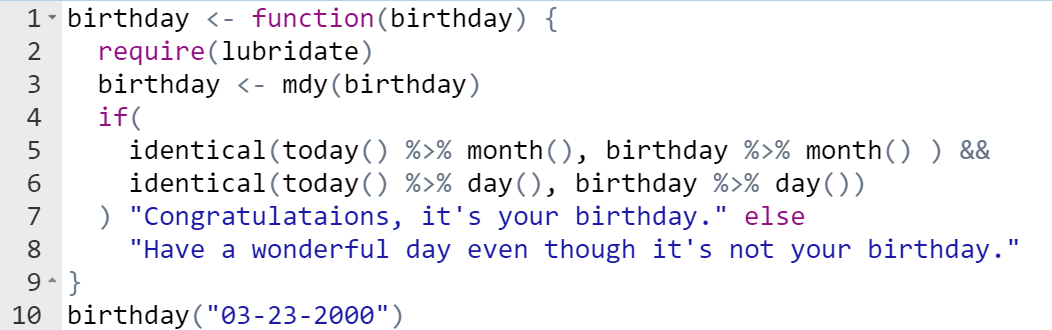
Line 3: We are going to calculate the mean delay for each destination. Therefore, we must group our observations by destination for the ensuing mean calculation to work.

Line 4: Creates a new variable called delay, which is the mean of the arrival delay with missing observations removed.

Line 5: Adds the full name of each destination airport. It first replicates the tibble created in lines 3 through 6 and then adds information from the airports tibble where the dest from the new tibble matches the faa variable in the airports tibble. Because this is an inner join, only those rows from both tibbles where the keys match are included.

## Code Chunk #3

### Interpret lines 1 thru 10



### Output

## [1] "Have a wonderful day even though it's not your birthday."

Line 1: defines a new function named birthday. It takes an argument of birthday. Having a function name the same as the argument isn’t a very good idea and I therefore shouldn’t have done that. It works ok but could make debugging difficult.

Line 2: Loads the lubridate library so that we have access to all the function in that library.

Line 3: Converts the birthday value, which came into the function as a character argument, into a date object. The birthday date object is created by interpreting a string as month, day, and year. It uses the mdy() function from lubridate.

Line 4: Initializes the block of if logic that follows

Line 5: Utilizes the today() and month() from the lubridate library. The today() function returns the current date and time from the computer. The month() function then extracts the month from that variable. Similarly month() extracts the month from the birthday variable as an integer. The identical function compares the month of the computer time with the month from the birthday.

Lines 5 and 6: The && at the end of line 5 means that the condition in line 5 and line 6 must both be TRUE in order for the whole condition to evaluate as TRUE.

Line 6: This is identical to line 5 except we are extracting the day rather than the month

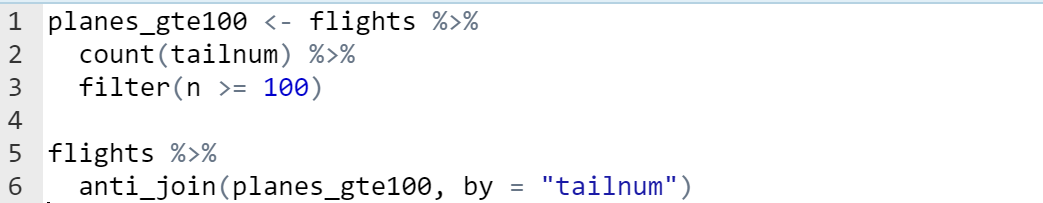
Line 7: If the month and day of the computer time match the month and day of the birthday variable, then the function returns the value “Congratulations, it’s your birthday.

Line 8: If the month and day of the computer time do not match the month and day of the birthday variable, then the function returns the value “Have a wonderful day even though it’s not your birthday.”

Line 10: Executes the birthday function and passes in the argument: “3-23-2000” to see if it matches the date from the computer’s clock.

Code Chunk #4

### Interpret lines 2 thru 6



### Output

## # A tibble: 105,874 x 19  
## year month day dep\_time sched\_dep\_time dep\_delay arr\_time sched\_arr\_time  
## <int> <int> <int> <int> <int> <dbl> <int> <int>  
## 1 2013 1 1 542 540 2 923 850  
## 2 2013 1 1 554 600 -6 812 837  
## 3 2013 1 1 558 600 -2 753 745  
## 4 2013 1 1 558 600 -2 924 917  
## 5 2013 1 1 559 600 -1 941 910  
## 6 2013 1 1 602 610 -8 812 820  
## 7 2013 1 1 606 610 -4 858 910  
## 8 2013 1 1 611 600 11 945 931  
## 9 2013 1 1 622 630 -8 1017 1014  
## 10 2013 1 1 623 610 13 920 915  
## # ... with 105,864 more rows, and 11 more variables: 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>

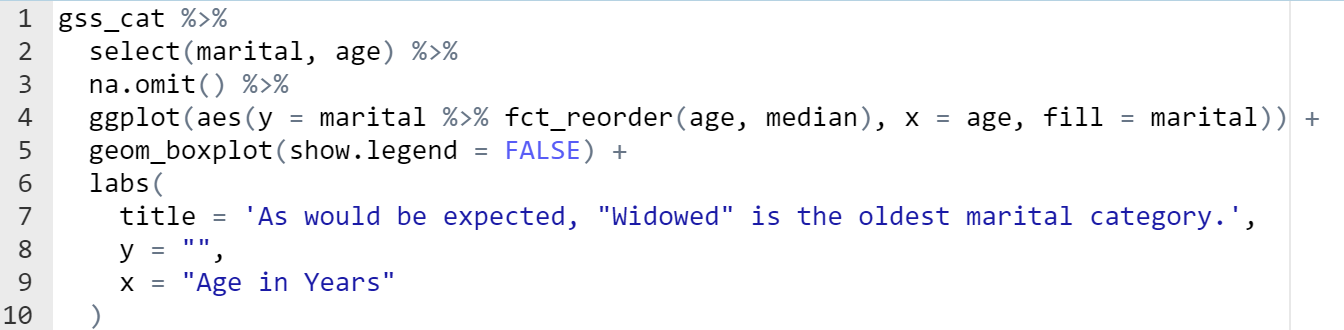
Line 2: Creates a new tibble that has two variables. The variables are the tail number for each aircraft in the flights tibble and the number of times that plane made a flight.

Line 3: Chooses only those planes than made 100 flights or more.

Line 6: anti\_join is a filtering join. The anti-join selects only those rows from flights that don’t have a matching tail number in the planes\_gte100 tibble. This selects only the flights for planes that flew less than 100 times.

## Code Chunk #5

### Interpret lines 2 thru 4



Line 2: Excludes all of the variables from the tibble except marital and age

Line 3: Excludes the entire row where any of the variables have a missing observation

Line 4: Anticipates the boxplot in line 5. A boxplot requires both a categorical and numerical variable. In this case we are putting the categorical variable marital on the y-axis and the numerical variable age on the x-axis. We want to order the marital categories by the size of their median age. For this reason, we use fct\_reorder to define the order of the marital categorical variable based on the median of age. This puts the category with the smallest median (Never married) at the bottom of the graph and the category with the largest median (Widowed) at the top of the graph.

### Output

