Data Manipulation and Transformation

Learning Objectives

Participate should be able to Manipulate/Reshape Data Frame Using some tidyverse functions

Packages and Data

We will be using the tidyverse package for this class. The data file for this class has been provided in the data folder, it's named mbta.xlsx. It is a data on passengers boarding and alighting at all stations on all lines of the Massachusetts Bay Transportation Authority (MBTA) commuter rail system.

Package

For this lesson, we will be using the tidyverse package. Tidyverse is a collection of essential R packages for data science created by Hadley Wickham.

The following packages are included in the core tidyverse: ggplot2, dplyr, tidyr, readr, purrr , tibble, stringr and forcats. You can install the tidyverse package by running the following code

Reading the Data

```
library(readxl)

library(tidyverse)

dta<-read_excel("data/mbta.xlsx",skip = 1, range = cell_cols(2:60))

View(dta)
```

Skip: this is use to skip the first page of an excel file if at all the numbers of page in the excel is more than one

Range: this is used to select the numbers of column

Gathering the years

The gather function in tidyr package helps in gathering multiple columns and collapses the columns into key-value pairs as seen below.

```
dta_tidy <- dta %>% gather(`2007-01`:`2011-10`,
    key = "year", value = "passengers")
View(dta_tidy)
```

Separating year

The separate function in tidyr turns a single character column into multiple columns as seen below

```
dta_tidy <- dta_tidy %>% separate(year, into = c("year", "month"))

View(dta_tidy)
```

Spreading mode of transportation

The pivot_wider function helps in spreading a key-value pair across multiple columns

```
dta_tidy <- dta_tidy %>% pivot_wider(mode, passengers)

dta_tidy
```

Extracting the needed columns

We may be interested in certain columns, we can apply our knowledge of subsetting to select the needed columns for our analysis

```
dta_tidy <- dta_tidy %>% .[,c(1:2,6:8)]

dta_tidy
```

Gather rail modes

After successful selecting the columns we are interested in, then we gather columns into a single column using the gather function as seen below.

```
dta_tidy <- dta_tidy %>% gather(`Commuter Rail`:`Light
    Rail`, key="rail_type", value = passengers)
dta_tidy
```

Data transformation using dplyr

For data transformation, we will be using hflights data , the dataset contains all flights departing from Houston airports IAH (George Bush Intercontinental) and HOU (Houston Hobby).

```
# install.packages("hflights")

library(hflights)

data(hflights)

View(hflights)
```

Major dplyr core functions

filter() : Subset rows using column values

arrange() : Arrange rows by column values

select() : Subset columns using their names and types

pull() : Extract a single column

mutate() : creating new variables from existing variables

summarize() : Obtaining summary of variables

group_by() : convert existing tables into a grouped table

rename() : renaming columns

distinct() : finding distinct rows

transmute() : adds new variables and drops existing ones

%>% : is pipeline through which a set of function can be perform

%in% : operator is used for matching values

In dplyr functions, the first argument is always data frame and the returned value is always a data frame as well.

filter() function

The filter() is used to choose rows/cases where conditions are true, basically used in subsetting a dataframe based on their row values.

Let’s select all flights of february, 2011.

```
library(hflights)

data("hflights")

filter(hflights, Year == 2011, Month == 2)
```

filter() function

Following operators work with filter() function

- Comparison != (not equal)
- == (equal)
- >(greater than)
- >=(greater than or equal)
- <(less than)
- <=(less than or equal to)
- Boolean & (“and”), | (“or”), ! (“not”)

We can also use syntax such x %in% y, this is telling filter() to

select every row where x is part of the values of y

filter()

```
fil<-filter(hflights, Dest %in% c("FLL", "IAH"))

View(fil)
```

Using between in the filter() function

We can also use between to specify the particular range of values we are interested in. It takes the following form between (x, left, right) which is equivalent to x >= left & x <= right

Let's filter all flights that covered distance between 224 and 944 miles

```
filter(hflights, between(Distance, 224,944))[,11:16]
```

Task 1

Find all flights that

a. Departed in April, 2011

b. Operated by AA and WN

Using the arrange() function

arrange() function is used to order a dataframe by a set of columns.

Let's arrange the flights data by Year, Month

```
arr_1 <- arrange(hflights, Year, Month)
```

arrange() in descending order

By default arrange() sorts values in ascending order. Use desc()

to re-order by a column in descending order.

```
arrange(hflights, desc(ArrDelay))
```

Using select() function in dplyr

select() can be used to extract variables from a dataframe.

```
select(hflights, Year, Month, FlightNum, AirTime)[1:4,]
```

select() helper functions

select() has various helper functions:

- everything(): selects all variables.
- starts_with("def"): matches names that begin with “def”.
- ends_with("xyz"): matches names that end with “xyz”.
- contains("ijk"): matches names that contain “ijk”.

select() function

select() can be used to rename variables

```
sel_2  <- select(hflights, tail_num =TailNum)[1:5,]
```

This will rename TailNum and drop all variables except tail_num.

For renaming, it advisable to use rename()

```
ren_1 <- rename(hflights, tail_num = TailNum)
```

select()

We can select variables that starts with Dep and Arr

```
select(hflights, starts_with("Dep"), starts_with("Arr"))
```

select() Cont’d

- vars <- c("Year", "Month", "DayofMonth", "DayofMonth", "ArrTime")
- s_1<-select(hflights, one_of(vars))

head(s_1)[1:6,]

mutate() Cont’d

mutate() adds new variables using the existing ones, it also preserves existing variables.

```
hflights %>%
    select(ends_with("Delay"), Distance, AirTime) %>%
    mutate(time_gain = ArrDelay - DepDelay, speed = Distance / AirTime * 60)
```

summarize()

summarize() function creates one or more scalar variables summarizing the variables of an existing table.

```
summarise(hflights, Delay = sum(DepDelay, na.rm = TRUE))
```

summarize() with group_by()

summarize() with group_by() will result in one row in the output for each group

To summarize average delay by day

```
hflights %>%
    group_by(Year, Month, DayofMonth) %>%
    summarise(delay = mean(DepDelay, na.rm = TRUE))
```

summarize() count

For aggregations it is generally a good idea to include a count n(). For example, let's look at the (not cancelled) planes that have the highest average delays:

```
hflights %>%
    group_by(Year, Month, DayofMonth) %>%
    summarise(DepDelay =n())
```

useful functions for

summarize()

- Measures of location: mean(x) , sum(x) , median(x) .
- Measures of spread: sd(x) , IQR(x) , mad(x) .
- Measures of rank: min(x) , quantile(x, 0.25) , max(x) .
- Measures of position: first(x) , nth(x, 2) , last(x) .
- Counts: n() .

Task 2

Do an analysis on the The AirTime, ActualElapsedTime, numbers of flight with UniqueCarrier WN, AA, and CO, plot a histogram and a barchart and a prove to show that the dataset of those unique value are normally distributed. With the summary statistics and boxplot