

DPLYR Library Vignette

Oscar A. Trevizo

2023-May-09

Contents

1	Purpose	2
2	Libraries	3
3	Exploratory Data Analysis for flights {nycflights13} dataset	3
4	Load the library	3
5	Load the data	4
6	Select	4
6.1	Select example 1	5
6.2	Select example 2	5
6.3	Select example 3	6
6.4	Select example 4	6
7	Filter	7
7.1	Filter example 1	7
7.2	Filter example 2	8
7.3	Filter example 3	8
7.4	Filter example 4	9
7.5	Filter example 5	10
8	Arrange	10
8.1	Arrange example 1	10
8.2	Arrange example 2	11
9	Summarize or Summarise	12
9.1	Summarize example 1	12
9.2	Summarize example 2	12

10 Group By	13
10.1 Group_by example 1	13
10.2 Group_by example 2	14
10.3 Group_by example 3	15
10.4 Group_by example 4	16
11 Mutate (filter to remove observations)	17
11.1 Mutate example 1	17
12 Mutate: From documentation	18
12.1 Mutate (doc) example 1	18
13 Exploratory Data Analysis for possum {DAAG} dataset	18
13.1 Pipes	20
13.2 Select	20
13.3 Filter	22
13.4 Arrange	23
13.5 Summarise	23
13.6 Group By	24
13.7 Example: arrange() on that table created by group_by()	24
13.8 Create a new table (mutate)	24
13.9 Join	25
13.9.1 Example 1	25
13.9.2 Example 2	26
13.10Relocate: Move columns around	26
14 References	27

1 Purpose

This vignette aims to introduce you dplyr library. It is here for learning purposes.

- {dplyr} (pronounced d - plier dataset plier... pliers to trim data)

Most of this code came from Harvard STAT 109 class, Prof. Bharatendra Rai. Material used here for educational purposes. It is available in YouTube and GitHub. See links under references. I expanded the material with my own notes and R documentation and I plan to continue adding examples overtime.

2 Libraries

- Load dplyr

Tip: Shortcut select a word and click quotations to automate.

It has several functions or methods

- Pipes %>% to chain commands

3 Exploratory Data Analysis for flights {nycflights13} dataset

From Harvard STAT 109 class, Prof. Bharatendra Rai described the “*Process of Visualization*” through the following steps. Material used here for educational purposes.

1. Business question
2. Data
3. Choose visualization
4. Data preparation
5. Develop visualization
6. Develop insights
7. Next steps

This study focuses on the data. That is the *Exploratory Data Analysis (EDA)* step. We do this study using the dplyr library from R, to:

1. Select
2. Filter
3. Arrange
4. Summarize (spelling too Summarise)
5. Summarize and Group By
6. Mutate

Another Rmarkdown follows this file to cover data visualization.

4 Load the library

```
# The EDA functionality from dplyr (dee-plier).  
library(dplyr, warn.conflicts = FALSE)  
  
# Use NYC Flights 2013 library to demo EDA in this study.  
# https://www.transtats.bts.gov/Homepage.asp  
library(nycflights13)
```

5 Load the data

Display its documentation from the console:

```
>?flights
```

```
# From doc: "On-time data for all flights that departed NYC (i.e. JFK, LGA or EWR) in 2013."
data('flights')
str(flights)
```

```
## tibble [336,776 x 19] (S3: tbl_df/tbl/data.frame)
## $ year      : int [1:336776] 2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
## $ month     : int [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
## $ day       : int [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
## $ dep_time  : int [1:336776] 517 533 542 544 554 554 555 557 557 558 ...
## $ sched_dep_time: int [1:336776] 515 529 540 545 600 558 600 600 600 600 ...
## $ dep_delay : num [1:336776] 2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
## $ arr_time  : int [1:336776] 830 850 923 1004 812 740 913 709 838 753 ...
## $ sched_arr_time: int [1:336776] 819 830 850 1022 837 728 854 723 846 745 ...
## $ arr_delay : num [1:336776] 11 20 33 -18 -25 12 19 -14 -8 8 ...
## $ carrier   : chr [1:336776] "UA" "UA" "AA" "B6" ...
## $ flight    : int [1:336776] 1545 1714 1141 725 461 1696 507 5708 79 301 ...
## $ tailnum   : chr [1:336776] "N14228" "N24211" "N619AA" "N804JB" ...
## $ origin    : chr [1:336776] "EWR" "LGA" "JFK" "JFK" ...
## $ dest      : chr [1:336776] "IAH" "IAH" "MIA" "BQN" ...
## $ air_time  : num [1:336776] 227 227 160 183 116 150 158 53 140 138 ...
## $ distance  : num [1:336776] 1400 1416 1089 1576 762 ...
## $ hour      : num [1:336776] 5 5 5 5 6 5 6 6 6 6 ...
## $ minute    : num [1:336776] 15 29 40 45 0 58 0 0 0 0 ...
## $ time_hour : POSIXct[1:336776], format: "2013-01-01 05:00:00" "2013-01-01 05:00:00" ...
```

```
head(flights)
```

```
## # A tibble: 6 x 19
##   year month   day dep_time sched_dep~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##   <int> <int> <int>   <int>       <int>   <dbl>   <int>   <int>   <dbl> <chr>
## 1  2013     1     1     517         515         2     830     819     11  UA
## 2  2013     1     1     533         529         4     850     830     20  UA
## 3  2013     1     1     542         540         2     923     850     33  AA
## 4  2013     1     1     544         545        -1    1004    1022    -18 B6
## 5  2013     1     1     554         600        -6     812     837    -25 DL
## 6  2013     1     1     554         558        -4     740     728     12 UA
## # ... with 9 more variables: flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dtm>, and abbreviated variable names 1: sched_dep_time,
## #   2: dep_delay, 3: arr_time, 4: sched_arr_time, 5: arr_delay
```

6 Select

```
>?select
```

“Select (and optionally rename) variables in a data frame, using a concise mini-language that makes it easy to refer to variables based on their name (e.g. `a:f` selects all columns from `a` on the left to `f` on the right) or type (e.g. `where(is.numeric)` selects all numeric columns).

```
select(.data, ...)
```

6.1 Select example 1

List the columns one by one, separated by a comma.

```
# Put the results in another tibble to avoid printing a very long dataset.
sel_flights_method1 <- flights %>% select(month, day, origin, dest, carrier)
```

```
# Explore its structure.
str(sel_flights_method1)
```

```
## tibble [336,776 x 5] (S3: tbl_df/tbl/data.frame)
##  $ month  : int  [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
##  $ day    : int  [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
##  $ origin : chr   [1:336776] "EWR" "LGA" "JFK" "JFK" ...
##  $ dest   : chr   [1:336776] "IAH" "IAH" "MIA" "BQN" ...
##  $ carrier: chr   [1:336776] "UA" "UA" "AA" "B6" ...
```

```
head(sel_flights_method1)
```

```
## # A tibble: 6 x 5
##   month   day origin dest  carrier
##   <int> <int> <chr>  <chr> <chr>
## 1     1     1     EWR   IAH    UA
## 2     1     1     LGA   IAH    UA
## 3     1     1     JFK   MIA    AA
## 4     1     1     JFK   BQN    B6
## 5     1     1     LGA   ATL    DL
## 6     1     1     EWR   ORD    UA
```

6.2 Select example 2

Put the column names inside a vector.

```
# Put the results in another tibble to avoid printing a very long dataset.
sel_flights_method2 <- flights %>% select(c('month', 'day', 'origin', 'dest', 'carrier'))
```

```
# Explore its structure.
str(sel_flights_method2)
```

```
## tibble [336,776 x 5] (S3: tbl_df/tbl/data.frame)
##  $ month  : int  [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
##  $ day    : int  [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
##  $ origin : chr   [1:336776] "EWR" "LGA" "JFK" "JFK" ...
##  $ dest   : chr   [1:336776] "IAH" "IAH" "MIA" "BQN" ...
##  $ carrier: chr   [1:336776] "UA" "UA" "AA" "B6" ...
```

```
head(sel_flights_method2)
```

```
## # A tibble: 6 x 5
##   month   day origin dest  carrier
##   <int> <int> <chr>  <chr> <chr>
## 1     1     1   EWR    IAH    UA
## 2     1     1   LGA    IAH    UA
## 3     1     1   JFK    MIA    AA
## 4     1     1   JFK    BQN    B6
## 5     1     1   LGA    ATL    DL
## 6     1     1   EWR    ORD    UA
```

6.3 Select example 3

Do a combination of a vector containing a list of column names, and another one separate as in method 1.

```
# Put the results in another tibble to avoid printing a very long dataset.
sel_flights_method3 <- flights %>% select(c('month', 'day', 'origin', 'dest'), carrier)

# Explore its structure.
str(sel_flights_method3)
```

```
## tibble [336,776 x 5] (S3: tbl_df/tbl/data.frame)
##  $ month   : int [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
##  $ day     : int [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
##  $ origin  : chr [1:336776] "EWR" "LGA" "JFK" "JFK" ...
##  $ dest    : chr [1:336776] "IAH" "IAH" "MIA" "BQN" ...
##  $ carrier: chr [1:336776] "UA" "UA" "AA" "B6" ...
```

```
head(sel_flights_method3)
```

```
## # A tibble: 6 x 5
##   month   day origin dest  carrier
##   <int> <int> <chr>  <chr> <chr>
## 1     1     1   EWR    IAH    UA
## 2     1     1   LGA    IAH    UA
## 3     1     1   JFK    MIA    AA
## 4     1     1   JFK    BQN    B6
## 5     1     1   LGA    ATL    DL
## 6     1     1   EWR    ORD    UA
```

6.4 Select example 4

Change the order of the column names, and use a combination of a `c()` vector and individual column names.

```
# Put the results in another tibble to avoid printing a very long dataset.
sel_flights_method4 <- flights %>% select(c('month', 'day', 'carrier'), origin, c('dest'))

# Explore its structure.
str(sel_flights_method4)
```

```
## tibble [336,776 x 5] (S3: tbl_df/tbl/data.frame)
## $ month : int [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
## $ day   : int [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
## $ carrier: chr [1:336776] "UA" "UA" "AA" "B6" ...
## $ origin : chr [1:336776] "EWR" "LGA" "JFK" "JFK" ...
## $ dest   : chr [1:336776] "IAH" "IAH" "MIA" "BQN" ...
```

```
head(sel_flights_method4)
```

```
## # A tibble: 6 x 5
##   month   day carrier origin dest
##   <int> <int> <chr>   <chr> <chr>
## 1     1     1     UA      EWR   IAH
## 2     1     1     UA      LGA   IAH
## 3     1     1     AA      JFK   MIA
## 4     1     1     B6      JFK   BQN
## 5     1     1     DL      LGA   ATL
## 6     1     1     UA      EWR   ORD
```

7 Filter

From the documentation: “The `filter()` function is used to subset a data frame, retaining all rows that satisfy your conditions. To be retained, the row must produce a value of `TRUE` for all conditions. Note that when a condition evaluates to `NA` the row will be dropped, unlike base subsetting with `[.]`.”

```
filter(.data, ..., .by = NULL, .preserve = FALSE)
```

The following examples will have a `select()` before doing a filter, to simplify the resulting tibble.

7.1 Filter example 1

Simple example using the “`==`” operator to obtain records that match the value of a variable.

```
# Lets filter on a specific carrier, say 'UA' as an example.
fltr_flights_1 <- flights %>% select(month, day, origin, dest, carrier) %>%
  filter(carrier == 'UA')

str(fltr_flights_1)
```

```
## tibble [58,665 x 5] (S3: tbl_df/tbl/data.frame)
## $ month : int [1:58665] 1 1 1 1 1 1 1 1 1 1 ...
## $ day   : int [1:58665] 1 1 1 1 1 1 1 1 1 1 ...
## $ origin : chr [1:58665] "EWR" "LGA" "EWR" "JFK" ...
## $ dest   : chr [1:58665] "IAH" "IAH" "ORD" "LAX" ...
## $ carrier: chr [1:58665] "UA" "UA" "UA" "UA" ...
```

```
head(fltr_flights_1)
```

```
## # A tibble: 6 x 5
##   month   day origin dest   carrier
```

```
##   <int> <int> <chr>  <chr> <chr>
## 1     1     1 EWR    IAH    UA
## 2     1     1 LGA    IAH    UA
## 3     1     1 EWR    ORD    UA
## 4     1     1 JFK    LAX    UA
## 5     1     1 EWR    SFO    UA
## 6     1     1 EWR    LAS    UA
```

First notice how the number of rows went down as we filter on “UA” carriers.

7.2 Filter example 2

Front it with a Boolean NOT “!”. It is still a simple example using the “==” operator to obtain records that this time do not match the value of a variable.

```
# Lets filter on NOT a specific carrier, say 'UA' as an example.
fltr_flights_2 <- flights %>% select(month, day, origin, dest, carrier) %>%
  filter(!carrier == 'UA')

str(fltr_flights_2)
```

```
## tibble [278,111 x 5] (S3: tbl_df/tbl/data.frame)
##  $ month   : int [1:278111] 1 1 1 1 1 1 1 1 1 1 ...
##  $ day     : int [1:278111] 1 1 1 1 1 1 1 1 1 1 ...
##  $ origin  : chr [1:278111] "JFK" "JFK" "LGA" "EWR" ...
##  $ dest    : chr [1:278111] "MIA" "BQN" "ATL" "FLL" ...
##  $ carrier: chr [1:278111] "AA" "B6" "DL" "B6" ...
```

```
head(fltr_flights_2)
```

```
## # A tibble: 6 x 5
##   month   day origin dest  carrier
##   <int> <int> <chr>  <chr> <chr>
## 1     1     1 JFK    MIA    AA
## 2     1     1 JFK    BQN    B6
## 3     1     1 LGA    ATL    DL
## 4     1     1 EWR    FLL    B6
## 5     1     1 LGA    IAD    EV
## 6     1     1 JFK    MCO    B6
```

So now we get a complement dataset to the one we obtained in the previous example.

7.3 Filter example 3

Do more Boolean operations. If we separate by “,” commas we will be doing *AND* operations. WE can also use the “&” ampersand operator to perform *AND* Boolean operations.

```
# Lets filter on UA flights originating from Newark EWR going to Chicago ORD.
fltr_flights_3 <- flights %>% select(month, day, carrier, origin, dest, flight, dep_time) %>%
  filter(carrier == 'UA', origin == 'EWR', dest == 'ORD')

str(fltr_flights_3)
```



```
## tibble [3,822 x 7] (S3: tbl_df/tbl/data.frame)
## $ month : int [1:3822] 1 1 1 1 1 1 1 1 1 1 ...
## $ day : int [1:3822] 1 1 1 1 1 1 1 1 1 1 ...
## $ carrier : chr [1:3822] "UA" "UA" "UA" "UA" ...
## $ origin : chr [1:3822] "EWR" "EWR" "EWR" "EWR" ...
## $ dest : chr [1:3822] "ORD" "ORD" "ORD" "ORD" ...
## $ flight : int [1:3822] 1696 544 580 985 32 683 459 702 1623 1676 ...
## $ dep_time: int [1:3822] 554 715 902 1038 1356 1416 1529 1601 1716 1758 ...
```

```
head(fltr_flights_3)
```

```
## # A tibble: 6 x 7
##   month   day carrier origin dest  flight dep_time
##   <int> <int> <chr>   <chr> <chr>   <int>   <int>
## 1     1     1   UA      EWR   ORD     1696     554
## 2     1     1   UA      EWR   ORD     544     715
## 3     1     1   UA      EWR   ORD     580     902
## 4     1     1   UA      EWR   ORD     985    1038
## 5     1     1   UA      EWR   ORD      32    1356
## 6     1     1   UA      EWR   ORD     683    1416
```

7.4 Filter example 4

Do more Boolean operations. To perform *OR* operations, use the “/” straight line operator.

```
# Enclosing the Boolean within parentheses was really optional, but it is a safe approach.
fltr_flights_4 <- flights %>% select(month, day, origin, dest, carrier) %>%
  filter((carrier == 'UA' | carrier == 'AA'), (origin == 'JFK' | origin == 'EWR'), dest == 'ORD')

str(fltr_flights_4)
```

```
## tibble [4,187 x 5] (S3: tbl_df/tbl/data.frame)
## $ month : int [1:4187] 1 1 1 1 1 1 1 1 1 1 ...
## $ day : int [1:4187] 1 1 1 1 1 1 1 1 1 1 ...
## $ origin : chr [1:4187] "EWR" "EWR" "EWR" "EWR" ...
## $ dest : chr [1:4187] "ORD" "ORD" "ORD" "ORD" ...
## $ carrier: chr [1:4187] "UA" "UA" "UA" "UA" ...
```

```
head(fltr_flights_4)
```

```
## # A tibble: 6 x 5
##   month   day origin dest  carrier
##   <int> <int> <chr>   <chr> <chr>
## 1     1     1   EWR   ORD   UA
## 2     1     1   EWR   ORD   UA
## 3     1     1   EWR   ORD   UA
## 4     1     1   EWR   ORD   UA
## 5     1     1   EWR   ORD   UA
## 6     1     1   EWR   ORD   UA
```

7.5 Filter example 5

Do more Boolean operations. To perform *greater than* / *smaller than* operations, use the “> or <” operators.

```
# Enclosing the Boolean within parentheses was really optional, but it is a safe approach.
fltr_flights_5 <- flights %>% select(month, day, origin, dest, carrier) %>%
  filter(month > 6)
```

```
str(fltr_flights_5)
```

```
## tibble [170,618 x 5] (S3: tbl_df/tbl/data.frame)
## $ month : int [1:170618] 10 10 10 10 10 10 10 10 10 10 ...
## $ day   : int [1:170618] 1 1 1 1 1 1 1 1 1 1 ...
## $ origin : chr [1:170618] "EWR" "EWR" "JFK" "LGA" ...
## $ dest   : chr [1:170618] "CLT" "IAH" "MIA" "IAH" ...
## $ carrier: chr [1:170618] "US" "UA" "AA" "UA" ...
```

```
head(fltr_flights_5)
```

```
## # A tibble: 6 x 5
##   month    day origin dest  carrier
##   <int> <int> <chr>  <chr> <chr>
## 1     10     1 EWR    CLT    US
## 2     10     1 EWR    IAH    UA
## 3     10     1 JFK    MIA    AA
## 4     10     1 LGA    IAH    UA
## 5     10     1 JFK    SJU    B6
## 6     10     1 JFK    BQN    B6
```

8 Arrange

From the documentation: “*arrange()* orders the rows of a data frame by the values of selected columns.

Unlike other dplyr verbs, arrange() largely ignores grouping; you need to explicitly mention grouping variables (or use .by_group = TRUE) in order to group by them, and functions of variables are evaluated once per data frame, not once per group.

8.1 Arrange example 1

Simple example to demonstrate how *arrange()* will sort data in a certain order.

The default is *ascending* order.

```
# Enclosing the Boolean within parentheses was really optional, but it is a safe approach.
arr_flights_1 <- flights %>% select(month, day, origin, dest, air_time, carrier) %>%
  arrange(air_time)
```

```
str(arr_flights_1)
```

```
## tibble [336,776 x 6] (S3: tbl_df/tbl/data.frame)
## $ month : int [1:336776] 1 4 12 2 2 2 3 3 3 3 ...
```

```
## $ day      : int [1:336776] 16 13 6 3 5 12 2 8 18 19 ...
## $ origin   : chr [1:336776] "EWR" "EWR" "EWR" "EWR" ...
## $ dest     : chr [1:336776] "BDL" "BDL" "BDL" "PHL" ...
## $ air_time: num [1:336776] 20 20 21 21 21 21 21 21 21 21 ...
## $ carrier  : chr [1:336776] "EV" "EV" "EV" "EV" ...
```

```
head(arr_flights_1)
```

```
## # A tibble: 6 x 6
##   month   day origin dest   air_time carrier
##   <int> <int> <chr>  <chr>     <dbl> <chr>
## 1     1     16 EWR    BDL         20 EV
## 2     4     13 EWR    BDL         20 EV
## 3    12      6 EWR    BDL         21 EV
## 4     2      3 EWR    PHL         21 EV
## 5     2      5 EWR    BDL         21 EV
## 6     2     12 EWR    PHL         21 EV
```

8.2 Arrange example 2

Simple example to demonstrate how *arrange()* will sort data in a certain order.

To make it into *descending* order, one needs to add *desc()* within *arrange()*.

Who wants to go to Honolulu?

```
# Enclosing the Boolean within parentheses was really optional, but it is a safe approach.
arr_flights_2 <- flights %>% select(month, day, origin, dest, air_time, carrier) %>%
  arrange(desc(air_time))

str(arr_flights_2)
```

```
## tibble [336,776 x 6] (S3: tbl_df/tbl/data.frame)
## $ month   : int [1:336776] 3 2 3 3 3 2 11 3 11 3 ...
## $ day     : int [1:336776] 17 6 15 17 16 5 12 14 20 15 ...
## $ origin  : chr [1:336776] "EWR" "JFK" "JFK" "JFK" ...
## $ dest    : chr [1:336776] "HNL" "HNL" "HNL" "HNL" ...
## $ air_time: num [1:336776] 695 691 686 686 683 679 676 676 675 671 ...
## $ carrier : chr [1:336776] "UA" "HA" "HA" "HA" ...
```

```
head(arr_flights_2)
```

```
## # A tibble: 6 x 6
##   month   day origin dest   air_time carrier
##   <int> <int> <chr>  <chr>     <dbl> <chr>
## 1     3     17 EWR    HNL         695 UA
## 2     2      6 JFK    HNL         691 HA
## 3     3     15 JFK    HNL         686 HA
## 4     3     17 JFK    HNL         686 HA
## 5     3     16 JFK    HNL         683 HA
## 6     2      5 JFK    HNL         679 HA
```

9 Summarize or Summarise

From the documentation: *summarise()* creates a new data frame. It returns one row for each combination of grouping variables; if there are no grouping variables, the output will have a single row summarising all observations in the input. It will contain one column for each grouping variable and one column for each of the summary statistics that you have specified.

summarise() and *summarize()* are synonyms.

9.1 Summarize example 1

This example include standard *mean()* calculations. Pay close attention to setting up argument *na.rm = TRUE*.

This example includes the *n()* contextual function to provide the group size (number of observation).

```
sumze_example_1 <- flights %>% summarize(AVG_air_time = mean(air_time, na.rm = TRUE),
                                         AVG_distance = mean(distance, na.rm = TRUE),
                                         No_of_flights = n())

str(sumze_example_1)
```

```
## tibble [1 x 3] (S3: tbl_df/tbl/data.frame)
##  $ AVG_air_time : num 151
##  $ AVG_distance : num 1040
##  $ No_of_flights: int 336776
```

```
sumze_example_1
```

```
## # A tibble: 1 x 3
##   AVG_air_time AVG_distance No_of_flights
##       <dbl>       <dbl>       <int>
## 1       151.       1040.       336776
```

9.2 Summarize example 2

Add pipe operations as above. Then run *summarize()* to see how the data is impacted.

```
# Lets filter on UA flights originating from Newark EWR going to Chicago ORD.
# And summarize the avg time, sd, also for the distance... does the distance change?
sumze_example_2 <- flights %>% select(month, day, origin, dest, carrier, distance, air_time) %>%
  filter(carrier == 'UA', origin == 'EWR', dest == 'ORD') %>%
  summarize(AVG_air_time = mean(air_time, na.rm = TRUE),
            SD_air_time = sd(air_time, na.rm = TRUE),
            AVG_distance = mean(distance, na.rm = TRUE),
            SD_distance = sd(distance, na.rm=TRUE),
            No_of_flights = n())

str(sumze_example_2)
```

```
## tibble [1 x 5] (S3: tbl_df/tbl/data.frame)
##  $ AVG_air_time : num 114
##  $ SD_air_time  : num 10.1
##  $ AVG_distance : num 719
##  $ SD_distance  : num 0
##  $ No_of_flights: int 3822
```

```
sumze_example_2
```

```
## # A tibble: 1 x 5
##   AVG_air_time SD_air_time AVG_distance SD_distance No_of_flights
##   <dbl>        <dbl>        <dbl>        <dbl>        <int>
## 1      114.         10.1          719           0          3822
```

Verify the distance is always the same, and it is since $\sigma = 0$. The number of flights coincides with the previous calculation when we were focused on the filter. The difference here is the entire pipe return a small tibble with only the summaries.

10 Group By

From the documentation: *Most data operations are done on groups defined by variables. group_by() takes an existing tbl and converts it into a grouped tbl where operations are performed “by group”. ungroup() removes grouping.*

10.1 Group_by example 1

Let `group_by()` be followed by a `summarize()`. At the end add `arrange()` to capture our table in a certain order.

```
grpby_example_1 <- flights %>% group_by(origin) %>%
  summarize(AVG_air_time = mean(air_time, na.rm = TRUE),
            SD_air_time = sd(air_time, na.rm = TRUE),
            AVG_distance = mean(distance, na.rm = TRUE),
            SD_distance = sd(distance, na.rm=TRUE),
            No_of_flights = n()) %>%
  arrange(desc(AVG_air_time))

str(grpby_example_1)
```

```
## tibble [3 x 6] (S3: tbl_df/tbl/data.frame)
##  $ origin      : chr [1:3] "JFK" "EWR" "LGA"
##  $ AVG_air_time : num [1:3] 178 153 118
##  $ SD_air_time  : num [1:3] 113.8 93.3 49.4
##  $ AVG_distance : num [1:3] 1266 1057 780
##  $ SD_distance  : num [1:3] 896 730 372
##  $ No_of_flights: int [1:3] 111279 120835 104662
```

```
grpby_example_1
```

```
## # A tibble: 3 x 6
##   origin AVG_air_time SD_air_time AVG_distance SD_distance No_of_flights
##   <chr>      <dbl>      <dbl>      <dbl>      <dbl>      <int>
## 1 JFK          178.        114.        1266.        896.       111279
## 2 EWR          153.         93.3        1057.        730.       120835
## 3 LGA          118.         49.4         780.        372.       104662
```

We get a nice table with rows represented by the *unique* values from the variables entered in *group_by()*. In this case I chose variable *origin* because we have only three airports: Newark, JFK, and La Guardia.

The result is a table with three rows. The first column is for the *origin*, the airport name. Then the other columns represent the values calculated in *summarize()*.

10.2 Group_by example 2

Let's focus on other variables in this new example.

```
grpby_example_2 <- flights %>% group_by(origin) %>%
  summarize(AVG_air_time = mean(air_time, na.rm = TRUE),
            SD_air_time = sd(air_time, na.rm = TRUE),
            AVG_distance = mean(distance, na.rm = TRUE),
            SD_distance = sd(distance, na.rm=TRUE),
            AVG_dep_time = mean(sched_dep_time, na.rm = TRUE),
            SD_dep_time = sd(sched_dep_time, na.rm = TRUE),
            No_of_flights = n()) %>%
  arrange(desc(AVG_air_time))

str(grpby_example_2)
```

```
## tibble [3 x 8] (S3: tbl_df/tbl/data.frame)
## $ origin      : chr [1:3] "JFK" "EWR" "LGA"
## $ AVG_air_time : num [1:3] 178 153 118
## $ SD_air_time  : num [1:3] 113.8 93.3 49.4
## $ AVG_distance : num [1:3] 1266 1057 780
## $ SD_distance  : num [1:3] 896 730 372
## $ AVG_dep_time : num [1:3] 1402 1322 1308
## $ SD_dep_time  : num [1:3] 482 465 447
## $ No_of_flights: int [1:3] 111279 120835 104662
```

```
grpby_example_2
```

```
## # A tibble: 3 x 8
##   origin AVG_air_time SD_air_time AVG_distance SD_dist~1 AVG_d~2 SD_de~3 No_of~4
##   <chr>      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>      <int>
## 1 JFK          178.        114.        1266.        896.       1402.       482.     111279
## 2 EWR          153.         93.3        1057.        730.       1322.       465.     120835
## 3 LGA          118.         49.4         780.        372.       1308.       447.     104662
## # ... with abbreviated variable names 1: SD_distance, 2: AVG_dep_time,
## #   3: SD_dep_time, 4: No_of_flights
```

10.3 Group_by example 3

Let's use two variables in the `group_by()` command.

```
grpby_example_3 <- flights %>% group_by(origin, dest) %>%
  summarize(AVG_air_time = mean(air_time, na.rm = TRUE),
            SD_air_time = sd(air_time, na.rm = TRUE),
            AVG_distance = mean(distance, na.rm = TRUE),
            SD_distance = sd(distance, na.rm=TRUE),
            AVG_dep_time = mean(sched_dep_time, na.rm = TRUE),
            SD_dep_time = sd(sched_dep_time, na.rm = TRUE),
            No_of_flights = n()) %>%
  arrange(desc(origin), desc(dest))
```

'summarise()' has grouped output by 'origin'. You can override using the
'.groups' argument.

```
str(grpby_example_3)
```

```
## group_df [224 x 9] (S3: grouped_df/tbl_df/tbl/data.frame)
## $ origin      : chr [1:224] "LGA" "LGA" "LGA" "LGA" ...
## $ dest        : chr [1:224] "XNA" "TYS" "TVC" "TPA" ...
## $ AVG_air_time : num [1:224] 173.2 97.8 94.6 146 38.1 ...
## $ SD_air_time  : num [1:224] 15.91 8.52 6.49 10.99 3.52 ...
## $ AVG_distance : num [1:224] 1147 647 655 1010 198 ...
## $ SD_distance  : num [1:224] 0 0 0 0 0 0 0 0 0 ...
## $ AVG_dep_time : num [1:224] 1316 1992 1396 1280 1726 ...
## $ SD_dep_time  : num [1:224] 451 135 404 437 455 ...
## $ No_of_flights: int [1:224] 745 308 77 2145 293 1822 737 217 6 68 ...
## - attr(*, "groups")= tibble [3 x 2] (S3: tbl_df/tbl/data.frame)
## ..$ origin: chr [1:3] "EWR" "JFK" "LGA"
## ..$ .rows : list<int> [1:3]
## .. ..$ : int [1:86] 139 140 141 142 143 144 145 146 147 148 ...
## .. ..$ : int [1:70] 69 70 71 72 73 74 75 76 77 78 ...
## .. ..$ : int [1:68] 1 2 3 4 5 6 7 8 9 10 ...
## ..@ ptype: int(0)
## ..- attr(*, ".drop")= logi TRUE
```

```
head(grpby_example_3)
```

```
## # A tibble: 6 x 9
## # Groups:   origin [1]
##   origin dest AVG_air_time SD_air_time AVG_di~1 SD_di~2 AVG_d~3 SD_de~4 No_of~5
##   <chr>   <chr>      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>      <int>
## 1 LGA     XNA         173.        15.9        1147         0      1316.      451.       745
## 2 LGA     YYS         97.8         8.52         647         0      1992.      135.       308
## 3 LGA     TVC         94.6         6.49         655         0      1396.      404.        77
## 4 LGA     TPA         146.        11.0        1010         0      1280.      437.      2145
## 5 LGA     SYR         38.1         3.52         198         0      1726.      455.       293
## 6 LGA     STL         133.        11.1         888         0      1308.      405.      1822
## # ... with abbreviated variable names 1: AVG_distance, 2: SD_distance,
## #   3: AVG_dep_time, 4: SD_dep_time, 5: No_of_flights
```

10.4 Group_by example 4

Let's use two variables in the `group_by()` command. Change it a bit, base it on carriers this time.

Let's look only at Unites (AA), American (AA), Delta (DL), and Southwest (WN).

And make it to ORD only.

```
grpby_example_4 <- flights %>%
  filter((carrier == 'UA' | carrier == 'AA' | carrier == 'DL' | carrier == 'WN'), dest=='ORD') %>%
  group_by(carrier, origin, dest) %>%
  summarize(AVG_air_time = mean(air_time, na.rm = TRUE),
            SD_air_time = sd(air_time, na.rm = TRUE),
            AVG_distance = mean(distance, na.rm = TRUE),
            AVG_dep_time = mean(sched_dep_time, na.rm = TRUE),
            SD_dep_time = sd(sched_dep_time, na.rm = TRUE),
            No_of_flights = n()) %>%
  arrange(carrier, desc(origin), desc(dest))
```

```
## 'summarise()' has grouped output by 'carrier', 'origin'. You can override using
## the '.groups' argument.
```

```
str(grpby_example_4)
```

```
## gropd_df [4 x 9] (S3: grouped_df/tbl_df/tbl/data.frame)
## $ carrier      : chr [1:4] "AA" "AA" "UA" "UA"
## $ origin       : chr [1:4] "LGA" "JFK" "LGA" "EWR"
## $ dest         : chr [1:4] "ORD" "ORD" "ORD" "ORD"
## $ AVG_air_time : num [1:4] 116 122 115 114
## $ SD_air_time  : num [1:4] 9.91 9.1 9.72 10.13
## $ AVG_distance : num [1:4] 733 740 733 719
## $ AVG_dep_time : num [1:4] 1269 1710 1284 1287
## $ SD_dep_time  : num [1:4] 464.45 7.52 464.28 427.95
## $ No_of_flights: int [1:4] 5694 365 3162 3822
## - attr(*, "groups")= tibble [4 x 3] (S3: tbl_df/tbl/data.frame)
## ..$ carrier: chr [1:4] "AA" "AA" "UA" "UA"
## ..$ origin : chr [1:4] "JFK" "LGA" "EWR" "LGA"
## ..$ .rows  : list<int> [1:4]
## .. ..$ : int 2
## .. ..$ : int 1
## .. ..$ : int 4
## .. ..$ : int 3
## .. ..@ ptype: int(0)
## ..- attr(*, ".drop")= logi TRUE
```

```
grpby_example_4
```

```
## # A tibble: 4 x 9
## # Groups:   carrier, origin [4]
##   carrier origin dest   AVG_air_time SD_air_time AVG_di~1 AVG_d~2 SD_de~3 No_of~4
##   <chr>   <chr> <chr>         <dbl>         <dbl>     <dbl>   <dbl>   <dbl>   <int>
## 1 AA     LGA    ORD           116.           9.91        733    1269.   464.    5694
## 2 AA     JFK    ORD           122.           9.10        740    1710.    7.52     365
```



```
## 3 UA      LGA      ORD      115.      9.72      733      1284.  464.      3162
## 4 UA      EWR      ORD      114.      10.1      719      1287.  428.      3822
## # ... with abbreviated variable names 1: AVG_distance, 2: AVG_dep_time,
## # 3: SD_dep_time, 4: No_of_flights
```

11 Mutate (filter to remove observations)

Prof. Bharatendra used `group_by()` followed by `summarize()` and `filter()`

11.1 Mutate example 1

Instead of *origin*, let's use *dest* to get a good number of rows (greater than the 3 we would get if we stuck with *origin*).

```
mtate_example_1 <- flights %>% group_by(dest) %>%
  summarize(AVG_air_time = mean(air_time, na.rm = TRUE),
            SD_air_time = sd(air_time, na.rm = TRUE),
            AVG_distance = mean(distance, na.rm = TRUE),
            SD_distance = sd(distance, na.rm=TRUE),
            AVG_dep_time = mean(sched_dep_time, na.rm = TRUE),
            SD_dep_time = sd(sched_dep_time, na.rm = TRUE),
            No_of_flights = n()) %>%
  filter(AVG_air_time > 345) %>%
  arrange(desc(AVG_air_time))

str(mtate_example_1)
```

```
## tibble [4 x 8] (S3: tbl_df/tbl/data.frame)
## $ dest      : chr [1:4] "HNL" "ANC" "SJC" "SFO"
## $ AVG_air_time : num [1:4] 617 413 347 346
## $ SD_air_time  : num [1:4] 21.7 14.7 16.5 17.2
## $ AVG_distance : num [1:4] 4973 3370 2569 2578
## $ SD_distance  : num [1:4] 10 0 0 10.2
## $ AVG_dep_time : num [1:4] 1126 1618 1833 1297
## $ SD_dep_time  : num [1:4] 182.69 4.63 34.27 447.82
## $ No_of_flights: int [1:4] 707 8 329 13331
```

```
mtate_example_1
```

```
## # A tibble: 4 x 8
##   dest AVG_air_time SD_air_time AVG_distance SD_dista~1 AVG_d~2 SD_de~3 No_of~4
##   <chr>      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>      <int>
## 1 HNL          617.          21.7        4973.         10.0       1126.       183.         707
## 2 ANC          413.          14.7        3370           0       1618.         4.63           8
## 3 SJC          347.          16.5        2569           0       1833.        34.3          329
## 4 SFO          346.          17.2        2578.         10.2       1297.       448.        13331
## # ... with abbreviated variable names 1: SD_distance, 2: AVG_dep_time,
## # 3: SD_dep_time, 4: No_of_flights
```

The size of the table, number of rows, is impacted by the filter. SO we are *mutating* the table by filtering and capturing only a subset of the otherwise bigger table.

I played with the value in the filter to increase or decrease my result in terms of number of observations.

12 Mutate: From documentation

The documentation of *dplyr* identifies a function called *mutate()*: “*mutate()* creates new columns that are functions of existing variables. It can also modify (if the name is the same as an existing column) and delete columns (by setting their value to *NULL*).”

12.1 Mutate (doc) example 1

Let's reduce the number of columns first to simplify the example. Then let's run *mutate()* to basically create new columns based on calculations from the other columns.

```
mtate_example_2 <- flights %>% select(origin, dest, carrier, air_time, arr_time, dep_time) %>%
  mutate(CALC_air_time = arr_time - dep_time)

str(mtate_example_2)
```

```
## tibble [336,776 x 7] (S3: tbl_df/tbl/data.frame)
##  $ origin      : chr [1:336776] "EWR" "LGA" "JFK" "JFK" ...
##  $ dest        : chr [1:336776] "IAH" "IAH" "MIA" "BQN" ...
##  $ carrier     : chr [1:336776] "UA" "UA" "AA" "B6" ...
##  $ air_time    : num [1:336776] 227 227 160 183 116 150 158 53 140 138 ...
##  $ arr_time    : int [1:336776] 830 850 923 1004 812 740 913 709 838 753 ...
##  $ dep_time    : int [1:336776] 517 533 542 544 554 554 555 557 557 558 ...
##  $ CALC_air_time: int [1:336776] 313 317 381 460 258 186 358 152 281 195 ...
```

```
head(mtate_example_2)
```

```
## # A tibble: 6 x 7
##   origin dest  carrier air_time arr_time dep_time CALC_air_time
##   <chr> <chr> <chr>      <dbl>    <int>    <int>         <int>
## 1 EWR    IAH    UA         227      830      517           313
## 2 LGA    IAH    UA         227      850      533           317
## 3 JFK    MIA    AA         160      923      542           381
## 4 JFK    BQN    B6         183     1004      544           460
## 5 LGA    ATL    DL         116      812      554           258
## 6 EWR    ORD    UA         150      740      554           186
```

This last example would require a bit more testing. What if the plane arrive on a different date, that is if the plane left just before midnight... But that is ok for now. I will ignore those nuances.

Notice that if you subtract arrival time minus departure time you get a different result than the value seen in air time. There is a reason for that. I am not considering these are class date / time variables and cannot just subtract like that... One hour does not have 100 minutes... One needs to change the class to date.

I just wanted to prove that you can subtract and add a new column.

13 Exploratory Data Analysis for possum {DAAG} dataset

```

# Add DAAG to pull some data
library(DAAG)

# For example DAAG has 'possum'

# data()

# From the console, you can do:
# > ?datasetname

data('possum')
str(possum)

```

```

## 'data.frame': 104 obs. of 14 variables:
## $ case : num 1 2 3 4 5 6 7 8 9 10 ...
## $ site : num 1 1 1 1 1 1 1 1 1 1 ...
## $ Pop : Factor w/ 2 levels "Vic","other": 1 1 1 1 1 1 1 1 1 1 ...
## $ sex : Factor w/ 2 levels "f","m": 2 1 1 1 1 1 2 1 1 1 ...
## $ age : num 8 6 6 6 2 1 2 6 9 6 ...
## $ hdlngth : num 94.1 92.5 94 93.2 91.5 93.1 95.3 94.8 93.4 91.8 ...
## $ skullw : num 60.4 57.6 60 57.1 56.3 54.8 58.2 57.6 56.3 58 ...
## $ totlngth: num 89 91.5 95.5 92 85.5 90.5 89.5 91 91.5 89.5 ...
## $ taill : num 36 36.5 39 38 36 35.5 36 37 37 37.5 ...
## $ footlngth: num 74.5 72.5 75.4 76.1 71 73.2 71.5 72.7 72.4 70.9 ...
## $ earconch: num 54.5 51.2 51.9 52.2 53.2 53.6 52 53.9 52.9 53.4 ...
## $ eye : num 15.2 16 15.5 15.2 15.1 14.2 14.2 14.5 15.5 14.4 ...
## $ chest : num 28 28.5 30 28 28.5 30 30 29 28 27.5 ...
## $ belly : num 36 33 34 34 33 32 34.5 34 33 32 ...

```

```
summary(possum)
```

```

##      case      site      Pop      sex      age
## Min.   : 1.00   Min.   :1.000   Vic :46   f:43   Min.   :1.000
## 1st Qu.: 26.75   1st Qu.:1.000   other:58   m:61   1st Qu.:2.250
## Median : 52.50   Median :3.000
## Mean   : 52.50   Mean   :3.625
## 3rd Qu.: 78.25   3rd Qu.:6.000
## Max.   :104.00   Max.   :7.000
##                                     NA's   :2
##      hdlngth      skullw      totlngth      taill
## Min.   : 82.50   Min.   :50.00   Min.   :75.00   Min.   :32.00
## 1st Qu.: 90.67   1st Qu.:54.98   1st Qu.:84.00   1st Qu.:35.88
## Median : 92.80   Median :56.35   Median :88.00   Median :37.00
## Mean   : 92.60   Mean   :56.88   Mean   :87.09   Mean   :37.01
## 3rd Qu.: 94.72   3rd Qu.:58.10   3rd Qu.:90.00   3rd Qu.:38.00
## Max.   :103.10   Max.   :68.60   Max.   :96.50   Max.   :43.00
##
##      footlngth      earconch      eye      chest      belly
## Min.   :60.30   Min.   :40.30   Min.   :12.80   Min.   :22.0   Min.   :25.00
## 1st Qu.:64.60   1st Qu.:44.80   1st Qu.:14.40   1st Qu.:25.5   1st Qu.:31.00
## Median :68.00   Median :46.80   Median :14.90   Median :27.0   Median :32.50

```

```
## Mean      :68.46   Mean      :48.13   Mean      :15.05   Mean      :27.0   Mean      :32.59
## 3rd Qu.   :72.50   3rd Qu.   :52.00   3rd Qu.   :15.72   3rd Qu.   :28.0   3rd Qu.   :34.12
## Max.      :77.90   Max.      :56.20   Max.      :17.80   Max.      :32.0   Max.      :40.00
## NA's      :1
```

```
# Try from the console:
# > ?possum
```

13.1 Pipes

- Mac shortcut shift-command-m for %>% (that is from {dplyr})
- To be demonstrated throughout this vignette

13.2 Select

- To specific column, by column number or column 'name'

```
# Select specific columns.
#
possum %>% select(2:3, 4:7)
```

```
##      site  Pop sex age hdlngth skullw
## C3      1  Vic  m   8    94.1   60.4
## C5      1  Vic  f   6    92.5   57.6
## C10     1  Vic  f   6    94.0   60.0
## C15     1  Vic  f   6    93.2   57.1
## C23     1  Vic  f   2    91.5   56.3
## C24     1  Vic  f   1    93.1   54.8
## C26     1  Vic  m   2    95.3   58.2
## C27     1  Vic  f   6    94.8   57.6
## C28     1  Vic  f   9    93.4   56.3
## C31     1  Vic  f   6    91.8   58.0
## C32     1  Vic  f   9    93.3   57.2
## C34     1  Vic  f   5    94.9   55.6
## C36     1  Vic  m   5    95.1   59.9
## C37     1  Vic  m   3    95.4   57.6
## C39     1  Vic  m   5    92.9   57.6
## C40     1  Vic  m   4    91.6   56.0
## C45     1  Vic  f   1    94.7   67.7
## C47     1  Vic  m   2    93.5   55.7
## C48     1  Vic  f   5    94.4   55.4
## C50     1  Vic  f   4    94.8   56.3
## C54     1  Vic  f   3    95.9   58.1
## C55     1  Vic  m   3    96.3   58.5
## C58     1  Vic  f   4    92.5   56.1
## C59     1  Vic  m   2    94.4   54.9
## C60     1  Vic  m   3    95.8   58.5
## C61     1  Vic  m   7    96.0   59.0
## C63     1  Vic  f   2    90.5   54.5
## C64     1  Vic  m   4    93.8   56.8
## A1      1  Vic  f   3    92.8   56.0
```

## A2	1	Vic	f	2	92.1	54.4
## A3	1	Vic	m	3	92.8	54.1
## A4	1	Vic	f	4	94.3	56.7
## AD1	1	Vic	m	3	91.4	54.6
## BB4	2	Vic	m	2	90.6	55.7
## BB13	2	Vic	m	4	94.4	57.9
## BB15	2	Vic	m	7	93.3	59.3
## BB17	2	Vic	f	2	89.3	54.8
## BB25	2	Vic	m	7	92.4	56.0
## BB31	2	Vic	f	1	84.7	51.5
## BB33	2	Vic	f	3	91.0	55.0
## BB36	2	Vic	f	5	88.4	57.0
## BB38	2	Vic	m	3	85.3	54.1
## BB40	2	Vic	f	2	90.0	55.5
## BB41	2	Vic	m	NA	85.1	51.5
## BB44	2	Vic	m	3	90.7	55.9
## BB45	2	Vic	m	NA	91.4	54.4
## WW1	3	other	m	2	90.1	54.8
## WW2	3	other	m	5	98.6	63.2
## WW3	3	other	m	4	95.4	59.2
## WW4	3	other	f	5	91.6	56.4
## WW5	3	other	f	5	95.6	59.6
## WW6	3	other	m	6	97.6	61.0
## WW7	3	other	f	3	93.1	58.1
## BR1	4	other	m	7	96.9	63.0
## BR2	4	other	m	2	103.1	63.2
## BR3	4	other	m	3	99.9	61.5
## BR4	4	other	f	4	95.1	59.4
## BR5	4	other	m	3	94.5	64.2
## BR6	4	other	m	2	102.5	62.8
## BR7	4	other	f	2	91.3	57.7
## CD1	5	other	m	7	95.7	59.0
## CD2	5	other	f	3	91.3	58.0
## CD3	5	other	f	6	92.0	56.4
## CD4	5	other	f	3	96.9	56.5
## CD5	5	other	f	5	93.5	57.4
## CD6	5	other	f	3	90.4	55.8
## CD7	5	other	m	4	93.3	57.6
## CD8	5	other	m	5	94.1	56.0
## CD9	5	other	m	5	98.0	55.6
## CD10	5	other	f	7	91.9	56.4
## CD11	5	other	m	6	92.8	57.6
## CD12	5	other	m	1	85.9	52.4
## CD13	5	other	m	1	82.5	52.3
## BSF1	6	other	f	4	88.7	52.0
## BSF2	6	other	m	6	93.8	58.1
## BSF3	6	other	m	5	92.4	56.8
## BSF4	6	other	m	6	93.6	56.2
## BSF5	6	other	m	1	86.5	51.0
## BSF6	6	other	m	1	85.8	50.0
## BSF7	6	other	m	1	86.7	52.6
## BSF8	6	other	m	3	90.6	56.0
## BSF9	6	other	f	4	86.0	54.0
## BSF10	6	other	f	3	90.0	53.8

```
## BSF11    6 other    m    3    88.4    54.6
## BSF12    6 other    m    3    89.5    56.2
## BSF13    6 other    f    3    88.2    53.2
## BTP1     7 other    m    2    98.5    60.7
## BTP3     7 other    f    2    89.6    58.0
## BTP4     7 other    m    6    97.7    58.4
## BTP5     7 other    m    3    92.6    54.6
## BTP6     7 other    m    3    97.8    59.6
## BTP7     7 other    m    2    90.7    56.3
## BTP8     7 other    m    3    89.2    54.0
## BTP9     7 other    m    7    91.8    57.6
## BTP10    7 other    m    4    91.6    56.6
## BTP12    7 other    m    4    94.8    55.7
## BTP13    7 other    m    3    91.0    53.1
## BTP14    7 other    m    5    93.2    68.6
## BTP15    7 other    f    3    93.3    56.2
## BTP16    7 other    m    1    89.5    56.0
## BTP17    7 other    m    1    88.6    54.7
## BTP19    7 other    f    6    92.4    55.0
## BTP20    7 other    m    4    91.5    55.2
## BTP21    7 other    f    3    93.6    59.9
```

13.3 Filter

```
# This example shows a filter with multiple conditions.
#
possum %>% filter(sex == 'f', Pop == 'Vic', age < 4)
```

```
##      case site Pop sex age hdlngth skullw totlngth  tail footlngth earconch  eye
## C23      5   1 Vic  f  2   91.5   56.3    85.5  36.0    71.0    53.2 15.1
## C24      6   1 Vic  f  1   93.1   54.8    90.5  35.5    73.2    53.6 14.2
## C45     17   1 Vic  f  1   94.7   67.7    89.5  36.5    73.2    53.2 14.7
## C54     21   1 Vic  f  3   95.9   58.1    96.5  39.5    77.9    52.9 14.2
## C63     27   1 Vic  f  2   90.5   54.5    85.0  35.0    70.3    50.8 14.2
## A1      29   1 Vic  f  3   92.8   56.0    88.0  35.0    74.9    51.8 14.0
## A2      30   1 Vic  f  2   92.1   54.4    84.0  33.5    70.6    50.8 14.5
## BB17    37   2 Vic  f  2   89.3   54.8    82.5  35.0    71.2    52.0 13.6
## BB31    39   2 Vic  f  1   84.7   51.5    75.0  34.0    68.7    53.4 13.0
## BB33    40   2 Vic  f  3   91.0   55.0    84.5  36.0    72.8    51.4 13.6
## BB40    43   2 Vic  f  2   90.0   55.5    81.0  32.0    72.0    49.4 13.4
##      chest belly
## C23    28.5  33.0
## C24    30.0  32.0
## C45    29.0  31.0
## C54    30.0  40.0
## C63    23.0  28.0
## A1     24.0  32.0
## A2     24.5  33.0
## BB17   28.0  31.5
## BB31   25.0  25.0
## BB33   27.0  30.0
## BB40   29.0  31.0
```

13.4 Arrange

- A type of sort

```
# Arrange, or sort
# Here we start to pipe using multiple lines.
#
possum %>% filter(sex == 'f', Pop == 'Vic', age < 4) %>%
  arrange(desc(belly))
```

```
##      case site Pop sex age hdlngth skullw totlngth taill footlgth earconch eye
## C54      21   1 Vic  f   3   95.9   58.1     96.5   39.5    77.9    52.9 14.2
## C23       5   1 Vic  f   2   91.5   56.3     85.5   36.0    71.0    53.2 15.1
## A2       30   1 Vic  f   2   92.1   54.4     84.0   33.5    70.6    50.8 14.5
## C24       6   1 Vic  f   1   93.1   54.8     90.5   35.5    73.2    53.6 14.2
## A1       29   1 Vic  f   3   92.8   56.0     88.0   35.0    74.9    51.8 14.0
## BB17     37   2 Vic  f   2   89.3   54.8     82.5   35.0    71.2    52.0 13.6
## C45      17   1 Vic  f   1   94.7   67.7     89.5   36.5    73.2    53.2 14.7
## BB40     43   2 Vic  f   2   90.0   55.5     81.0   32.0    72.0    49.4 13.4
## BB33     40   2 Vic  f   3   91.0   55.0     84.5   36.0    72.8    51.4 13.6
## C63      27   1 Vic  f   2   90.5   54.5     85.0   35.0    70.3    50.8 14.2
## BB31     39   2 Vic  f   1   84.7   51.5     75.0   34.0    68.7    53.4 13.0
##      chest belly
## C54     30.0  40.0
## C23     28.5  33.0
## A2      24.5  33.0
## C24     30.0  32.0
## A1      24.0  32.0
## BB17    28.0  31.5
## C45     29.0  31.0
## BB40    29.0  31.0
## BB33    27.0  30.0
## C63     23.0  28.0
## BB31    25.0  25.0
```

13.5 Summarise

- You can introduce functions or equations and summarise.

```
# summarise() with multiple functions... Avg, SD...
#
possum %>% filter(sex == 'f', Pop == 'Vic', age < 4) %>%
  arrange(desc(belly)) %>%
  summarise(Avg = mean(belly),
            SD = sd(belly),
            count = n())
```

```
##      Avg      SD count
## 1 31.5 3.667424    11
```

13.6 Group By

- It creates a table.

```
# group_by() before summarising.
#
possum %>% filter(sex == 'm') %>%
  group_by(site) %>%
  summarise(Avg = mean(belly),
            SD = sd(belly),
            count = n())
```

```
## # A tibble: 7 x 4
##   site   Avg    SD count
##   <dbl> <dbl> <dbl> <int>
## 1     1  33.2  2.49    14
## 2     2  32.1  3.37     8
## 3     3  34    1.47     4
## 4     4  34.6  2.22     5
## 5     5  30.9  2.28     7
## 6     6  31.5  2.78     9
## 7     7  31.8  2.25    14
```

13.7 Example: arrange() on that table created by group_by()

```
# Add another function, descending order
possum %>% filter(sex == 'm') %>%
  group_by(site) %>%
  summarise(Avg = mean(belly),
            SD = sd(belly),
            count = n()) %>%
  arrange(desc(Avg))
```

```
## # A tibble: 7 x 4
##   site   Avg    SD count
##   <dbl> <dbl> <dbl> <int>
## 1     4  34.6  2.22     5
## 2     3  34    1.47     4
## 3     1  33.2  2.49    14
## 4     2  32.1  3.37     8
## 5     7  31.8  2.25    14
## 6     6  31.5  2.78     9
## 7     5  30.9  2.28     7
```

13.8 Create a new table (mutate)

```
# New variable TR
mytable <- possum %>%
  group_by(site) %>%
```



```

  summarise(TR = sum(taill) / sum(totlngth),
            count = n()) %>%
  arrange(desc(TR))

print(mytable)

```

```

## # A tibble: 7 x 3
##   site    TR count
##   <dbl> <dbl> <int>
## 1     6 0.445    13
## 2     7 0.440    18
## 3     5 0.433    13
## 4     4 0.431     7
## 5     2 0.426    13
## 6     3 0.423     7
## 7     1 0.406    33

```

13.9 Join

From *dplyr* documentation:

mutate-joins {dplyr} R Documentation

Mutating joins

Description

Mutating joins add columns from y to x, matching observations based on the keys. There are four mutating joins: the inner join, and the three outer joins.

Inner join: An `inner_join()` only keeps observations from x that have a matching key in y.

The most important property of an inner join is that unmatched rows in either input are not included in the result. This means that generally inner joins are not appropriate in most analyses, because it is too easy to lose observations.

Outer joins: The three outer joins keep observations that appear in at least one of the data frames:

A `left_join()` keeps all observations in x.

A `right_join()` keeps all observations in y.

A `full_join()` keeps all observations in x and y.

13.9.1 Example 1

```

# Let's do an example
students_math <- c('mary', 'john', 'paul', 'jane', 'peter')
math <- c('A', 'A', 'B', 'C', 'B')

students_english <- c('tom', 'mary', 'john', 'paul')
english <- c('C', 'B', 'C', 'A')

dfa <- data.frame(students_math, math)
dfb <- data.frame(students_english, english)

```

```
colnames(dfa) <- c('students', 'math')
colnames(dfb) <- c('students', 'english')
```

```
left <- dfa %>% left_join(dfb)
```

```
## Joining with 'by = join_by(students)'
```

```
right <- dfa %>% right_join(dfb)
```

```
## Joining with 'by = join_by(students)'
```

```
inner <- dfa %>% inner_join(dfb)
```

```
## Joining with 'by = join_by(students)'
```

13.9.2 Example 2

Joining based on two columns

```
# Let's do an example
semester_math <- c('fall', 'fall', 'fall', 'fall', 'fall', 'spring', 'spring', 'spring', 'spring', 'spring')
students_math <- c('mary', 'john', 'paul', 'jane', 'peter', 'mary', 'john', 'paul', 'jane', 'peter')
math <- c('A', 'A', 'B', 'C', 'A', 'B', 'B', 'A', 'B', 'B')

semester_english <- c('fall', 'fall', 'fall', 'fall', 'spring', 'spring', 'spring', 'spring')
students_english <- c('tom', 'mary', 'john', 'paul', 'tom', 'mary', 'john', 'paul')
english <- c('C', 'B', 'C', 'A', 'B', 'B', 'B', 'A')

dfa <- data.frame(semester_math, students_math, math)
dfb <- data.frame(semester_english, students_english, english)

colnames(dfa) <- c('semester', 'students', 'math')
colnames(dfb) <- c('semester', 'students', 'english')

left <- dfa %>% left_join(dfb)
```

```
## Joining with 'by = join_by(semester, students)'
```

```
right <- dfa %>% right_join(dfb)
```

```
## Joining with 'by = join_by(semester, students)'
```

```
inner <- dfa %>% inner_join(dfb)
```

```
## Joining with 'by = join_by(semester, students)'
```

13.10 Relocate: Move columns around

```
# # https://dplyr.tidyverse.org/reference/relocate.html  
# df <- df %>% dplyr::relocate(column_x, column_y, vector_of_columns)
```

14 References

1. Dr. Bharatendra Rai YouTube Channel (accessed Jan. 22, 2023) https://www.youtube.com/watch?v=BPR_Dkll17Y&list=PL34t5iLfZddtUUABMikey6NtL05hPAp42
2. Dr. Bharatendra Rai YouTube Channel (accessed Feb 7, 2023) <https://www.youtube.com/watch?v=rsfV57N7Uns&list=PL34t5iLfZddtUUABMikey6NtL05hPAp42&index=10>
3. Harvard STAT 109 slides by Dr. Bharatendra Rai