# Unit-2: tidyverse

## tidyverse

The tidyverse includes a number of R packages that make it easier to ask, and answer questions about data. In particular, we will use the following packages

**dplyr** - grammar for data selection and manipulation

**tidyr** - tools for "tidying" data

**ggplot2** - a data visualization package

To install:

```
install.packages('tidyverse')
library('tidyverse')
```

## dplyr functions

- **select** features (columns) of interest from the data
- **filter** out irrelevant data, keep observations (rows) of interest
- **mutate** a data set by adding more features
- **arrange** observations (rows) in a particular order
- **summarize** data in terms of aggregates
- **join** multiple datasets into a single dataframe

these functions can also be executed with base R, however dplyr functions are much easier to read and write. They are especially powerful in combination with **group_by** and tibbles as we shall see shortly!

## Illustrative example using dplyr

Let's look at historical data on US presidential elections – a dataframe with the variables: state, demVote, year, south

```
View(presidentialElections) # data from 1932 to 2016
```

| state | demVote | year | south |
|---|---:|---|---|
| Alabama | 84.76 | 1932 | TRUE |
| Arizona | 67.03 | 1932 | FALSE |
| Arkansas | 86.27 | 1932 | TRUE |
| California | 58.41 | 1932 | FALSE |
| Colorado | 54.81 | 1932 | FALSE |
| Connecticut | 47.40 | 1932 | FALSE |
| Delaware | 48.11 | 1932 | FALSE |

## select() function

To select data, pass the dataframe as the first argument followed by the columns (variables) you want.

```r
votes <- select(presidentialElections, year, demVote)
View(votes)
```

| year | demVote |
|---|---|
| 1932 | 84.76 |
| 1932 | 67.03 |
| 1932 | 86.27 |
| 1932 | 58.41 |
| 1932 | 54.81 |
| 1932 | 47.40 |
| 1932 | 48.11 |

---

Note that you do not need to pass column names as strings.

No quotation marks around year, demVote

Compare:

```r
select(presidentialElections, year, demVote)    #dplyr
```

```r
presidentialElections[, c("year","demVote")]    #base R
```

The above expressions do the exact same thing BUT dplyr has more **expressive** syntax than base R. You can also do

```r
select(presidentialElections,state:year)
```

to select a range of columns

## filter() down to get the ROWS you want

filter() takes the dataframe you want to filter followed by a comma separated list of conditions each **row** must satisfy. Column names are specified WITHOUT quotation marks.

```r
votes_2008 <- filter(presidentialElections,year==2008)
```

| state | demVote | year | south |
|---|---|---|---|
| Alabama | 38.74 | 2008 | TRUE |
| Alaska | 37.89 | 2008 | FALSE |
| Arizona | 44.91 | 2008 | FALSE |
| Arkansas | 38.86 | 2008 | TRUE |
| California | 60.94 | 2008 | FALSE |
| Colorado | 53.66 | 2008 | FALSE |
| Connecticut | 60.59 | 2008 | FALSE |

---

Multiple conditions

```r
votes_colorado_2008 <- filter(
  presidentialElections,year==2008,state=='Colorado')
```

| state | demVote | year | south |
|---|---|---|---|
| Colorado | 53.66 | 2008 | FALSE |

The mutate() function allows you to create additional features (columns/variables). We will add columns for the % of votes for other parties and the absolute % vote difference between the democrats and all other parties. The syntax

```
mutate(dataframe,newcol1=def1, newcol2 =def2)
```

returns a new dataframe with extra columns that may be functions of the old

---

```
presidentialElections <- mutate(
  presidentialElections,
  other_parties_vote = 100 - demVote,
  abs_vote_difference = abs(demVote - other_parties_vote)
)
```

| state | demVote | year | south | other_parties_vote | abs_vote_difference |
|---|---|---|---|---|---|
| Alabama | 84.76 | 1932 | TRUE | 15.24 | 69.52 |
| Arizona | 67.03 | 1932 | FALSE | 32.97 | 34.06 |
| Arkansas | 86.27 | 1932 | TRUE | 13.73 | 72.54 |
| California | 58.41 | 1932 | FALSE | 41.59 | 16.82 |
| Colorado | 54.81 | 1932 | FALSE | 45.19 | 9.62 |
| Connecticut | 47.40 | 1932 | FALSE | 52.60 | 5.20 |
| Delaware | 48.11 | 1932 | FALSE | 51.89 | 3.78 |

## Arrange() to sort the rows

You might want to sort the rows based on year, then within each year, sort the rows based on the % of votes that went to the democratic party candidate like so:

```
presidentialElections <- arrange(presidentialElections,-year,demVote)
```

| state | demVote | year | south | other_parties_vote | abs_vote_difference |
|---|---|---|---|---|---|
| West Virginia | 26.18 | 2016 | FALSE | 73.82 | 47.64 |
| Utah | 27.17 | 2016 | FALSE | 72.83 | 45.66 |
| North Dakota | 27.23 | 2016 | FALSE | 72.77 | 45.54 |
| Idaho | 27.48 | 2016 | FALSE | 72.52 | 45.04 |
| Oklahoma | 28.93 | 2016 | FALSE | 71.07 | 42.14 |
| South Dakota | 31.74 | 2016 | FALSE | 68.26 | 36.52 |

we see that west virginia did not like Hilary

## Summarize()

We can use summarize(), for example, to calculate the average % of votes cast for the deomcratic party candidate over all years and states

```
average_votes <- summarize(
  presidentialElections,
  mean_dem_vote = mean(demVote),
  mean_other_parties_vote = mean(other_parties_vote)
)
```

| mean_dem_vote | mean_other_parties_vote |
|---:|---:|
| 48.3594 | 51.6406 |

## Sequential Operations

Suppose you want to answer the question:

"*Which state had the highest % of votes for the democratic party candidate (Barack Obama) in 2008?*" To answer you need to

1- Filter down only to 2008 votes

```
votes_2008 <- filter(presidentialElections,year==2008)
```

2- Filter down to state with highest `demVote`

```
most_dem_votes <- filter(votes_2008,demVote == max(demVote))
```

3- Select the name of that state

```
most_dem_state <- select(most_dem_votes,state)
```

---

This approach clutters the work environment with variables you don't need, alternatively you can nest the functions, but this is hard to read

```
# using nested functions HARD 2 READ!!
most_dem_state <- select(
  filter(
    filter(
      presidentialElections,
      year==2008
    ),
    demVote == max(demVote)
  ),
  state
)
```

| state |
|---|
| DC |

## CLEANER solution: Pipe operator

The pipe operator `%>%` passes a dataframe as the first argument of the next function. It is possible to chain together many dplyr functions using the pipe like so:

```
most_dem_state <- presidentialElections %>%
  filter(year==2008) %>%
  filter(demVote == max(demVote)) %>%
  select(state)
```

note that the result is the same but it is easier to read and understand and doesn't create any intermediate variables

| state |
|---|
| DC |

## Analyzing by group

Using `group_by` allows us to create associations among groups of rows

```
# Group observations by state
grouped <- group_by(presidentialElections,state)
```

`group_by()` returns a **tibble**. A tibble is a special kind of dataframe which is able to keep track of groups of rows within the same column. This grouping is not visibly apparent, it does not sort the rows, BUT the tibble keeps tracks of groups for computation.

```
is_tibble(grouped)
```

```
## [1] TRUE
```

---

This is useful because now you can apply summarize(), or filter() and it will automatically be applied to each group, for example:

```
state_voting_summary <- presidentialElections %>%
  group_by(state) %>%
  summarize(
    mean_dem_vote = mean(demVote),
    mean_other_parties = mean(other_parties_vote)
  )
```

| state | mean_dem_vote | mean_other_parties |
|---|---:|---:|
| Alabama | 50.75250 | 49.24750 |
| Alaska | 37.49600 | 62.50400 |
| Arizona | 45.43545 | 54.56455 |
| Arkansas | 52.43364 | 47.56636 |
| California | 51.29955 | 48.70045 |

## Takeaway message on `group_by()`

Grouping lets you frame your analysis question in terms of *comparing groups of observations (rows)* rather than individual observations.

This makes it much easier to ask and answer questions about your data!!

## Joining data frames

Consider a fundraising campaign that tracks donations with two dataframes: `donations` and `donors`

| donor_name | amount | date |
|---|---:|---|
| Maria Franca Fissolo | 100 | 2018-02-15 |
| Yang Huiyan | 50 | 2018-02-15 |
| Maria Franca Fissolo | 75 | 2018-02-15 |
| Alice Walton | 25 | 2018-02-16 |
| Susanne Klatten | 100 | 2018-02-17 |
| Yang Huiyan | 150 | 2018-02-18 |

| donor_name | email |
|---|---|
| Alice Walton | alice.walton@gmail.com |
| Jacqueline Mars | jacqueline.mars@gmail.com |
| Maria Franca Fissolo | maria.franca.fissolo@gmail.com |
| Susanne Klatten | susanne.klatten@gmail.com |
| Laurene Powell Jobs | laurene.powell.jobs@gmail.com |
| Francoise Bettencourt Meyers | francoise.bettencourt.meyers@gmail.com |

## Why more than one dataframe?

1. **Data Storage**

   Rather than duplicate information about donors each time they make a donation, you can store that info a single time thus reducing the amount of space your data takes up

2. **Data Updates**

   If you need to update info about a donor (e.g. the donor's phone number changes) you can make that change in a *single* location

But at some point you will want to combine data from both datasets using the `join()` function

## `left_join()`

```
left_join(donations, donors, by = 'donor_name')
```

```
## Joining, by = "donor_name"
```

| donor_name | amount | date | email |
|---|---|---|---|
| Maria Franca Fissolo | 100 | 2018-02-15 | maria.franca.fissolo@gmail.com |
| Yang Huiyan | 50 | 2018-02-15 | NA |
| Maria Franca Fissolo | 75 | 2018-02-15 | maria.franca.fissolo@gmail.com |
| Alice Walton | 25 | 2018-02-16 | alice.walton@gmail.com |
| Susanne Klatten | 100 | 2018-02-17 | susanne.klatten@gmail.com |
| Yang Huiyan | 150 | 2018-02-18 | NA |

---

here is how this procedure works

1. It goes through each row in donations looking at values from shared column `donor_name`
2. For each value in the left dataframe (donations) it looks for a row in the right dataframe (donors) with the same value in the shared column
3. If it finds such a matching row, it adds any other data values from columns in donors but not in donations to the left dataframe's row in the resulting new joined dataframe If there is no match, it puts `NA`
4. It repeats for each row *in the left dataframe*

**Remember: rows in the right dataframe that do not match will simply be lost in the created left_join() table**

## more on joining dataframes

For rows to match they need to have the same data in a ALL specified shared matching columns. Here is a case you might run into that you need to be aware of

```r
# An example join in the (hypotherical) case where the dataframes
# have different identifiers: e.g. if 'donations` had a column
#`donor_name`while`donors` had a column `name`
combined_data <- left_join(
  donations, donors, by = c("donor_name"="name"))
```

There are many ways to join a table, depending on which rows you care about, each function will return different result

`right_join` - opposite of left join

`inner_join` - only keeps rows where there is a match

`full_join` - keeps all rows from both tables

## Reshaping data with Tidyr

The tidyr package is used to structure and work with data frames that follow three principles

1. Each variable is in a column
2. Each observation is a row
3. Each value is a cell

In order to answer questions you may need to change your definition of what an "observation" means. This will require a restructuring of your data set.

Consider the following dataframe where each observation (row) is a city and each feature (column) is the ticket price of a specific band

---

| city | greensky_bluegrass | trampled_by_turtles | billy_strings | fruition |
|---|---:|---:|---:|---:|
| Seattle | 40 | 30 | 15 | 30 |
| Portland | 40 | 20 | 25 | 50 |
| Denver | 20 | 40 | 25 | 40 |
| Minneapolis | 30 | 100 | 15 | 20 |

But suppose you wanted to analyze the ticket price across all concerts. You could not do it easily with this data since itis organized by city, not by concert! You would prefer that price was a single feature of a concert: e.g. a city-band combination.

You need to gather all the prices into a single new column `price` and another new column - `band` which indicates where the prices were gathered from.

---

| city | band | price |
|---|---|---|
| Seattle | greensky_bluegrass | 40 |
| Portland | greensky_bluegrass | 40 |
| Denver | greensky_bluegrass | 20 |
| Minneapolis | greensky_bluegrass | 30 |
| Seattle | trampled_by_turtles | 30 |
| Portland | trampled_by_turtles | 20 |
| Denver | trampled_by_turtles | 40 |
| Minneapolis | trampled_by_turtles | 100 |
| Seattle | billy_strings | 15 |
| Portland | billy_strings | 25 |

---

```
band_data_long <- gather(   # from `tidyr` package
  band_data_wide,
  key = band,
  value = price,
  -city
)
```

notice that the **key** argument takes the name of column that will contain as values the names of the columns the data was gathered from

the `value` argument takes the name of the column that will contain the gathered values.

The data will be gathered from all columns given in the final argument, in this case `-city` instructs to collect data from all columns except for `city`.

`gather()` may not be so easy for you to understand at first…

---

It is also possible to go in reverse from long format to wide format, that is to **`spread`** out the prices into multiple columns.

```
price_by_band <- spread(
  band_data_long,
  key = city,
  value = price,)
```

the **key** argument is where it gets the new column names and the **value** argument is where the new column values come from

| band | Denver | Minneapolis | Portland | Seattle |
|---|---|---|---|---|
| billy_strings | 25 | 15 | 25 | 15 |
| fruition | 40 | 20 | 50 | 30 |
| greensky_bluegrass | 20 | 30 | 40 | 40 |
| trampled_by_turtles | 40 | 100 | 20 | 30 |

## Basic plotting with ggplot2

just as **`dpylr`** provides a grammar of data manipulation, **`ggplot2`** provides a grammar of graphics

```
install.packages('ggplot2')
library(ggplot2)
```

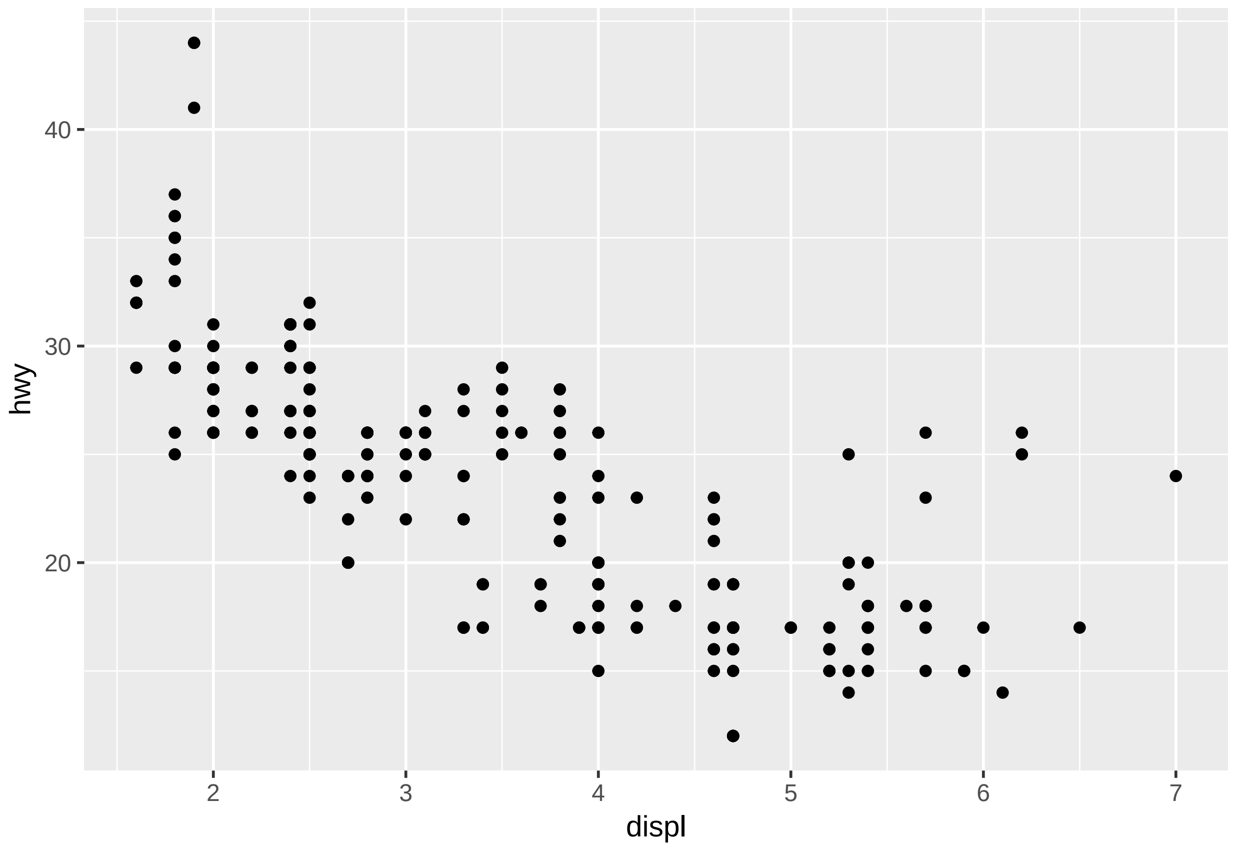
We will examine the data in the tibble dataframe `mpg` which contains data from the US environmental protection agency on 38 models of cars. Among the variables in `mpg` are:

- **displ** - a car's engine size in liters
- **hwy** - the car's highway fuel efficiency

---

To plot, run this code to put displ on x-axis and hwy on y-axis

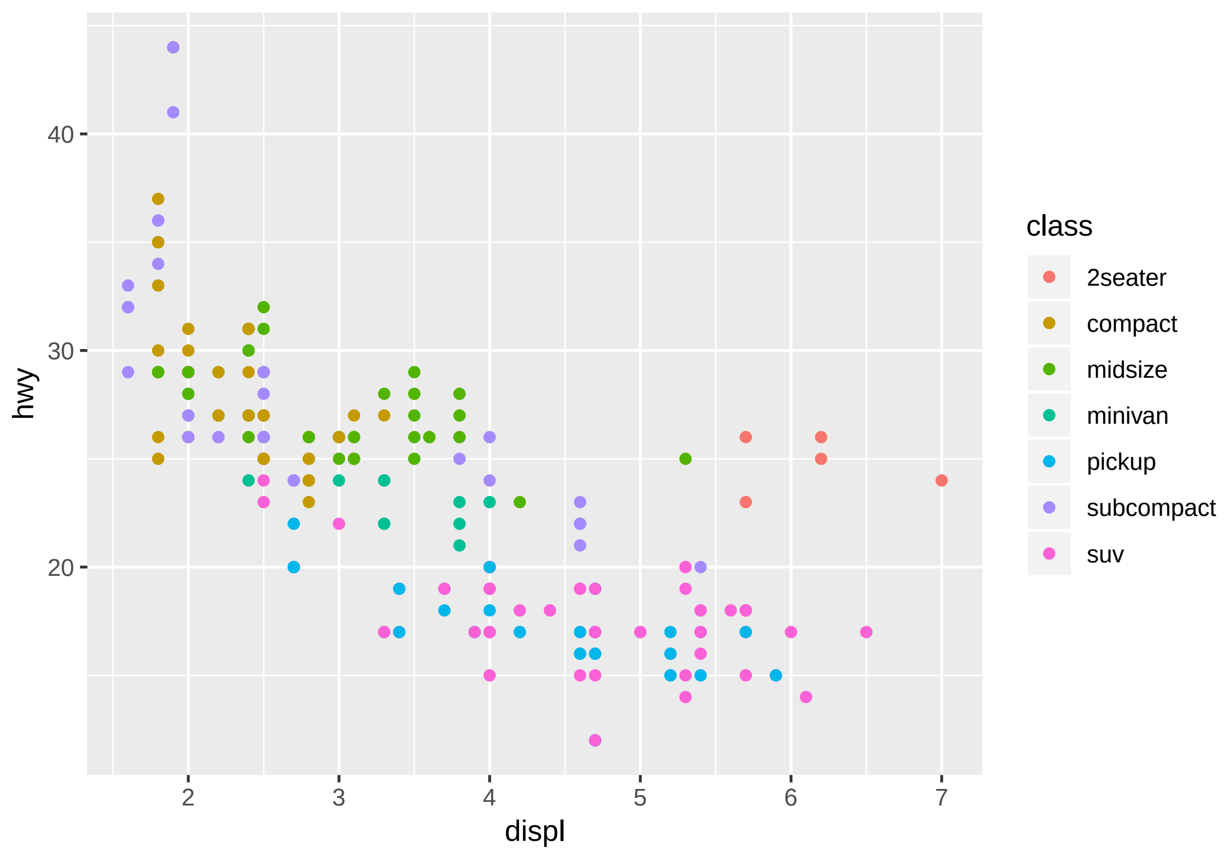
```
ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy))
```



---

map the colors of your points to reveal the class of each car

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



## Summary of the tidyverse

We will leave further exploration of data visualization with `ggplot2` to exercises

The tidyverse presents alot of excellent tools for wrangling and manipulating and visualizing data.

The main point of this class is to do mathematics and statistics with data, but we need to spend some time developing our skills at data handling in R before we can start doing more complex mathematical things with the data!

## Lab- Unit 2

*(Time allotment ~ 30 minutes)*

### Exercise 1: vehicles data

First, install the fueleconomy package:

install.packages("devtools")

devtools::install_github("hadley/fueleconomy")

Use the `library()` function to load the "fueleconomy" package

`library(fueleconomy)`

Then, install and load the "dplyr" library

You should now have access to the `vehicles` data frame You can use `View(vehicles)` to inspect it

---

### 1a) Using select()

Select the different manufacturers (makes) of the cars in this data set.

Save this vector in a variable.

Use the `distinct()` function to determine how many different car manufacturers are represented by the data set

---

### 1b) filter-arrange-mutate

Filter the data set for vehicles manufactured in 1997

Arrange the 1997 cars by highway (`hwy`) gas milage

Mutate the 1997 cars data frame to add a column `average` that has the average gas milage (between city and highway mpg) for each car

---

### 1c) Find the worst mpg

Filter the whole vehicles data set for 2-Wheel Drive vehicles that get more than 20 miles/gallon in the city. Save this new data frame in a variable.

Of the above vehicles, what is the vehicle ID of the vehicle with the worst hwy mpg?

**Hint**: filter for the worst vehicle, then select its ID.

---

### 1d) Using the Pipe operator

Which 2015 Acura model has the best hwy MGH? (Use dplyr, but without method chaining or pipes–use temporary variables!)

Which 2015 Acura model has the best hwy MPG? (Use dplyr, nesting functions)

Which 2015 Acura model has the best hwy MPG? (Use dplyr and the pipe operator)

---

### Bonus for part 1d)

Write 3 functions, one for each approach. Then,

Test how long it takes to perform each one 1000 times

you can do this by wrapping code within

`start.time <- Sys.time()`

YOUR CODE HERE

`end.time <- Sys.time() time.taken <- end.time - start.time time.taken`

---

### 1e) Optional Challenge

Write a function that takes a `year_choice` and a `make_choice` as parameters, and returns the vehicle model that gets the most hwy miles/gallon of vehicles of that make in that year.

You'll need to filter more (and do some selecting)!

What was the most efficient Honda model of 1995?

## Exercise 2: NYC flights

Install the `"nycflights13"` package. Load (`library()`) the package.

You'll also need to load `dplyr`

### 2a)

The data frame `flights` should now be accessible to you.

Use functions to inspect it: how many rows and columns does it have?

What are the names of the columns?

Use `??flights` to search for documentation on the data set (for what the columns represent)

---

### 2b)

Which airline has the highest number of delayed departures?

In order to answer this question your analysis should proceed as follows:

1. Use group_by on airline since we are interested in which airline
2. filter() down to observations where there was a delay, that is, `dep_delay` >0
3. summarize() your result by counting the number of observations using the n() function which will give you the number of delayed departures on each airline
4. determine the group with the highest count
5. select the airline from that observation

---

### 2c)

You should have found that UA is the result of exercise 2b). To get more information, we must join the `airlines` dataframe to the `flights` dataframe. Do this with a left_join() using `by = 'carrier'`. then select the name of the carrier from the joined table.

That is, find the full name of the airline with the most number of delays by selecting the name in the joined table.

---

**2d)**

What was the average departure delay in each month?

Save this as a data frame `dep_delay_by_month`

Hint: you'll have to perform a grouping operation then summarizing your data. When calculating the mean, you might also need to do `na.rm=TRUE`

---

**2e)**

Which month had the greatest average departure delay?