Classwork 1

Introduction to the Tidyverse in R

Note that this notebook relies heavily on material from the textbook R for Data Science by Hadley Wickham and Garrett Grolemund, and to a lesser extent, on the DataCamp course on the Tidyverse. Both are great resources to explore!

A. What is the Tidyverse? And how do we install it?

The Tidyverse is a collection of R packages meant to streamline data science tasks. All Tidyverse packages share an underlying design philosophy, grammar, and data structures. In this notebook, we'll learn some basics of the Tidyverse.

```
install.packages("tidyverse", dependencies = TRUE)
library(tidyverse)
```

```
Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

also installing the dependencies 'lazyeval', 'rex', 'covr', 'feather',
'mockr'


── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✔ dplyr     1.1.4     ✔ readr     2.1.5
✔ forcats   1.0.0     ✔ stringr   1.5.1
✔ ggplot2   3.5.1     ✔ tibble    3.2.1
✔ lubridate 1.9.4     ✔ tidyr     1.3.1
✔ purrr     1.0.2
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force
all conflicts to become errors
```

B. Some very basic plotting with ggplot

Let's do some plotting with the mpg dataset. mpg contains observations collected by the US Environmental Protection Agency on 38 models of car.

First, load and learn about the variables contained in this dataset. The dataset is in the ggplot2 package, which is included in the tidyverse. So, you can load the data using data(mpg).

```
data(mpg)
#help(mpg)
```

```
head(mpg)
?mpg
```

```
  manufacturer model displ year cyl trans      drv cty hwy fl class
1 audi         a4    1.8   1999 4   auto(l5)   f   18  29  p  compact
2 audi         a4    1.8   1999 4   manual(m5) f   21  29  p  compact
3 audi         a4    2.0   2008 4   manual(m6) f   20  31  p  compact
4 audi         a4    2.0   2008 4   auto(av)   f   21  30  p  compact
5 audi         a4    2.8   1999 6   auto(l5)   f   16  26  p  compact
6 audi         a4    2.8   1999 6   manual(m5) f   18  26  p  compact
```

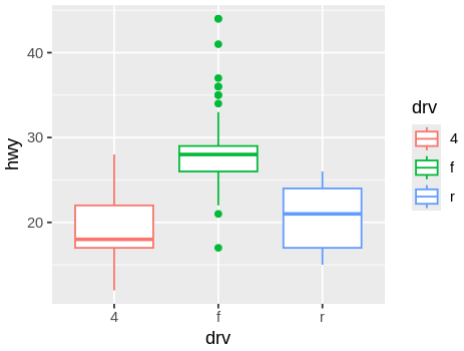
Let's look at a plot that might tell us about the relationship between drv (whether the car is front, rear, or 4-wheel drive) and hwy (highway miles per gallon).

We begin a plot with the function `ggplot()`, which creates a coordinate system that you can add layers to. Layers are created with `+ geom_boxplot()` will make a boxplot. In general, a template for creating plots would be

```
ggplot(data = DATA) +   <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

Below is the basic code for the boxplot (fill in the correct variables).

```
options(repr.plot.width=4, repr.plot.height=3) #this line just changes
the size of the boxplots
p = ggplot(data = mpg) +
    geom_boxplot(mapping = aes(x = drv , y = hwy, color = drv))
### (1) Your code here.
p
###
```

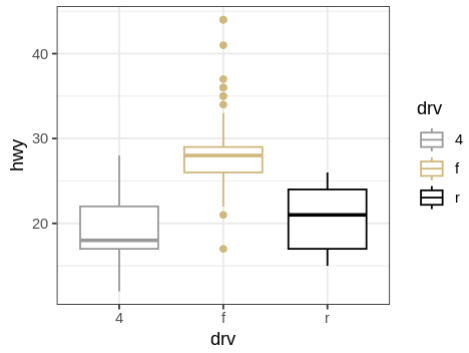


What do we notice about the relationship?

- When compared to 4-wheel drive or rear wheel drive, front-wheel is more efficient with respect to highway miles per gallon.
- There are some potential outliers in the front-wheel drive category.
- The 4-wheel drive group has a long/heavy upper tail.

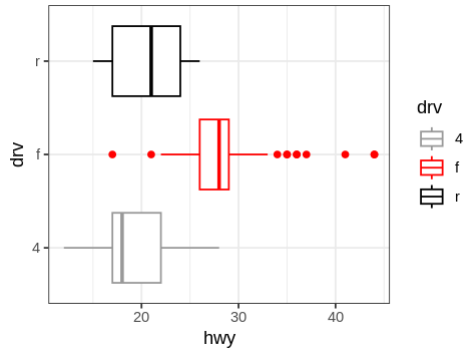
You can mess with *all* sorts of things. For example, you could change colors:

```
options(repr.plot.width=4, repr.plot.height=3)
ggplot(data = mpg) +
  geom_boxplot(mapping = aes(x = drv, y = hwy, color = drv)) +
    scale_color_manual(values=c("#999999", "#CFB87C", "black"))+
    theme_bw()
```



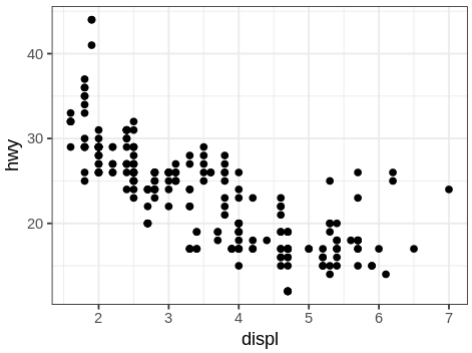
In some instances, it is helpful to swap the axes. For example, the levels of the factor along the horizontal axis might have long names. **Try flipping the axes using `coord_flip()`.**

```
### (2) Your code here
options(repr.plot.width=4, repr.plot.height=3) #this line just changes
the size of the boxplots
p = ggplot(data = mpg) +
    geom_boxplot(mapping = aes(x = drv , y = hwy, color = drv)) +
      scale_color_manual(values = c('#999999','red','black')) +
      theme_bw()+
      coord_flip()
### (1) Your code here.
p
###
###
```



Now let's try a scatterplot. **Use the template above to plot hwy (y) against displ (x).** `geom_point()` will give a scatterplot.

```
options(repr.plot.width=4, repr.plot.height=3)

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



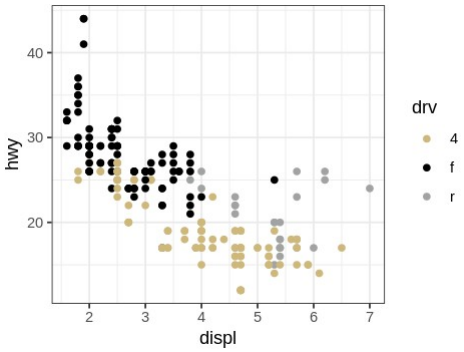
What do we notice about the relationship between engine displacement and highway miles per gallon?

There is a downward trend in `hwy` as `displ` increases. The trend appears roughly linear. However, there may be some evidence of curvature at the extremes of `displ`.

Now, let's color points based on whether they represent a vehicle that is front, rear, or 4-wheel drive (`drv`). We will map the `drv` variable to the *aesthetic* `color`. In general, an aesthetic is a "visual property of the objects in the plot" (Wickham, Section 3.3). Other aesthetics include size and shape.

Further, use the `scale_color_manual()` function to specify the CU Boulder Colors.

```
options(repr.plot.width=4, repr.plot.height=3)
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = drv)) +
    scale_color_manual(values=c("#CFB87C", "#000000", "#A2A4A3")) +
    theme_bw()
```

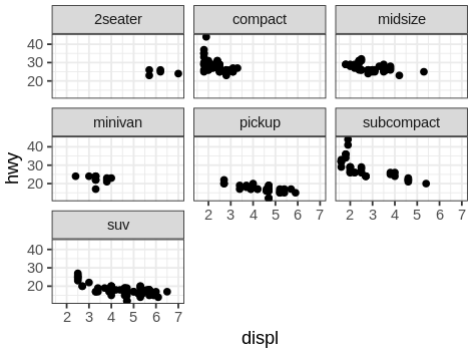


What do we notice about the relationships between these variables?

Another way to add information from categorical variables to plots is by using *facets*. Facets split a plot into subplots, where each subplot contains data for a particular level of the categorical variable. We can facet by adding `facet_wrap(~ CatVar, nrow = x)` to our ggplot, where `CatVar` is the categorical variable that we want to facet on, and `x` is the number of rows that we'd like (we could also use ncol...).

Create a facet plot where you split the data based on the class variable.

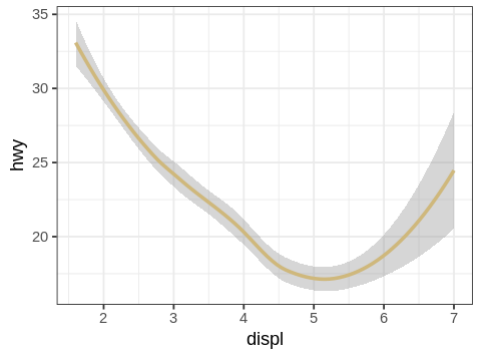
```
options(repr.plot.width=4, repr.plot.height=3)

### (4) Your code here.
p_facet = ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))+
    facet_wrap(~class,ncol = 3)+ #doing the plots based on the classes
    theme_bw()
p_facet
###
```



Instead of seeing the individual data points, we might be interested in visualizing some overall trend between displ and hwy. We could do this by substituting `geom_points()` with `geom_smooth()`. **Try it!**

```
options(repr.plot.width=4, repr.plot.height=3)
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy), color = "#CFB87C")+
    theme_bw()
```

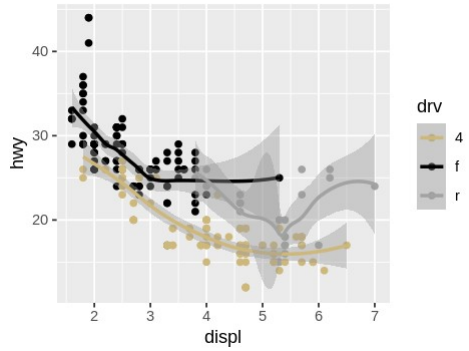
```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



And, we can layer the smooth over the scatterplot pretty easily by adding `+ geom_point()`. **Try it!**

```
ggplot(data = mpg) +
    geom_point(mapping = aes(x = displ, y = hwy, color = drv))+
    geom_smooth(mapping = aes(x = displ, y = hwy, color = drv)) +
    scale_color_manual(values=c("#CFB87C", "#000000", "#A2A4A3"))
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



C. Data Manipulation and Exploration

`dplyr` is a package in the Tidyverse that provides simple "verbs", or functions, that correspond to the most common data manipulation tasks; these verbs help you translate your thoughts into code. Let's see how some of these verbs work on the gapminder dataset. **First, if you haven't already, let's install and load the gapminder package.**

```
install.packages("gapminder")
library(gapminder)
library(dplyr)
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
```

Write a summary of the variables in this dataset.

```
data(gapminder)
head(gapminder)
```

```
  country     continent year lifeExp pop      gdpPercap
1 Afghanistan Asia      1952 28.801   8425333 779.4453
2 Afghanistan Asia      1957 30.332   9240934 820.8530
3 Afghanistan Asia      1962 31.997  10267083 853.1007
4 Afghanistan Asia      1967 34.020  11537966 836.1971
5 Afghanistan Asia      1972 36.088  13079460 739.9811
6 Afghanistan Asia      1977 38.438  14880372 786.1134
```

Filter rows with filter()

It is often useful to study a subset of your data. The verb `filter()` will easily allow you to filter rows (observations) in a data frame. Here's one possibility:

```
### (5) Your code here.
filter(gapminder,country == 'United States')
###
```

```
   country       continent year lifeExp pop       gdpPercap
1  United States Americas  1952 68.440  157553000 13990.48
2  United States Americas  1957 69.490  171984000 14847.13
3  United States Americas  1962 70.210  186538000 16173.15
4  United States Americas  1967 70.760  198712000 19530.37
5  United States Americas  1972 71.340  209896000 21806.04
6  United States Americas  1977 73.380  220239000 24072.63
7  United States Americas  1982 74.650  232187835 25009.56
8  United States Americas  1987 75.020  242803533 29884.35
9  United States Americas  1992 76.090  256894189 32003.93
10 United States Americas  1997 76.810  272911760 35767.43
11 United States Americas  2002 77.310  287675526 39097.10
12 United States Americas  2007 78.242  301139947 42951.65
```

Has the code above modified the exiting data frame or created a new one?

Not unless we store `filter(...)` back into `df`!

Filter the original dataset to show only observations where the year is later than 1987 and the life expectancy is greater than or equal to 70. Save your answer in `gapminder_filter`

```
### (6) Your code here.
gapminder_filter = filter(gapminder,year>1987,lifeExp>=70)
###
gapminder_filter
```

```
    country                 continent year lifeExp pop       gdpPercap
1   Albania                 Europe    1992 71.581   3326498   2497.438
2   Albania                 Europe    1997 72.950   3428038   3193.055
3   Albania                 Europe    2002 75.651   3508512   4604.212
```

4	Albania	Europe	2007	76.423	3600523	5937.030
5	Algeria	Africa	2002	70.994	31287142	5288.040
6	Algeria	Africa	2007	72.301	33333216	6223.367
7	Argentina	Americas	1992	71.868	33958947	9308.419
8	Argentina	Americas	1997	73.275	36203463	10967.282
9	Argentina	Americas	2002	74.340	38331121	8797.641
10	Argentina	Americas	2007	75.320	40301927	12779.380
11	Australia	Oceania	1992	77.560	17481977	23424.767
12	Australia	Oceania	1997	78.830	18565243	26997.937
13	Australia	Oceania	2002	80.370	19546792	30687.755
14	Australia	Oceania	2007	81.235	20434176	34435.367
15	Austria	Europe	1992	76.040	7914969	27042.019
16	Austria	Europe	1997	77.510	8069876	29095.921
17	Austria	Europe	2002	78.980	8148312	32417.608
18	Austria	Europe	2007	79.829	8199783	36126.493
19	Bahrain	Asia	1992	72.601	529491	19035.579
20	Bahrain	Asia	1997	73.925	598561	20292.017
21	Bahrain	Asia	2002	74.795	656397	23403.559
22	Bahrain	Asia	2007	75.635	708573	29796.048
23	Belgium	Europe	1992	76.460	10045622	25575.571
24	Belgium	Europe	1997	77.530	10199787	27561.197
25	Belgium	Europe	2002	78.320	10311970	30485.884
26	Belgium	Europe	2007	79.441	10392226	33692.605
27	Bosnia and Herzegovina	Europe	1992	72.178	4256013	2546.781
28	Bosnia and Herzegovina	Europe	1997	73.244	3607000	4766.356
29	Bosnia and Herzegovina	Europe	2002	74.090	4165416	6018.975
30	Bosnia and Herzegovina	Europe	2007	74.852	4552198	7446.299
⋮	⋮	⋮	⋮	⋮	⋮	⋮
248	Taiwan	Asia	2007	78.400	23174294	28718.277
249	Thailand	Asia	2007	70.616	65068149	7458.396
250	Tunisia	Africa	1992	70.001	8523077	4332.720
251	Tunisia	Africa	1997	71.973	9231669	4876.799
252	Tunisia	Africa	2002	73.042	9770575	5722.896
253	Tunisia	Africa	2007	73.923	10276158	7092.923
254	Turkey	Europe	2002	70.845	67308928	6508.086
255	Turkey	Europe	2007	71.777	71158647	8458.276
256	United Kingdom	Europe	1992	76.420	57866349	22705.093
257	United Kingdom	Europe	1997	77.218	58808266	26074.531
258	United Kingdom	Europe	2002	78.471	59912431	29478.999
259	United Kingdom	Europe	2007	79.425	60776238	33203.261
260	United States	Americas	1992	76.090	256894189	32003.932
261	United States	Americas	1997	76.810	272911760	35767.433
262	United States	Americas	2002	77.310	287675526	39097.100
263	United States	Americas	2007	78.242	301139947	42951.653
264	Uruguay	Americas	1992	72.752	3149262	8137.005
265	Uruguay	Americas	1997	74.223	3262838	9230.241
266	Uruguay	Americas	2002	75.307	3363085	7727.002
267	Uruguay	Americas	2007	76.384	3447496	10611.463
268	Venezuela	Americas	1992	71.150	20265563	10733.926

```
269 Venezuela              Americas  1997 72.146   22374398 10165.495
270 Venezuela              Americas  2002 72.766   24287670  8605.048
271 Venezuela              Americas  2007 73.747   26084662 11415.806
272 Vietnam                Asia      1997 70.672   76048996  1385.897
273 Vietnam                Asia      2002 73.017   80908147  1764.457
274 Vietnam                Asia      2007 74.249   85262356  2441.576
275 West Bank and Gaza     Asia      1997 71.096    2826046  7110.668
276 West Bank and Gaza     Asia      2002 72.370    3389578  4515.488
277 West Bank and Gaza     Asia      2007 73.422    4018332  3025.350
```

Important notes:

1. The arguments in `filter()` are combined with "and". To combine in other ways (e.g., "or"), use the Boolean operators (e.g., `|` is for "or").
2. Missing values: `filter()` only includes rows for which the variable is *not* `NA`. If you would like to preserve missing values, ask for them explicitly:

```
### (7) Your code here.
filter(gapminder, is.na(country) | country == 'United States')

###
```

```
   country       continent year lifeExp pop       gdpPercap
1  United States Americas  1952 68.440  157553000 13990.48
2  United States Americas  1957 69.490  171984000 14847.13
3  United States Americas  1962 70.210  186538000 16173.15
4  United States Americas  1967 70.760  198712000 19530.37
5  United States Americas  1972 71.340  209896000 21806.04
6  United States Americas  1977 73.380  220239000 24072.63
7  United States Americas  1982 74.650  232187835 25009.56
8  United States Americas  1987 75.020  242803533 29884.35
9  United States Americas  1992 76.090  256894189 32003.93
10 United States Americas  1997 76.810  272911760 35767.43
11 United States Americas  2002 77.310  287675526 39097.10
12 United States Americas  2007 78.242  301139947 42951.65
```

Arranging with arrange()

Use the `arrange()` verb, in conjunction with the code above to put the United States data (and only that data) in descending order with respect to year.

```
### (8) Your code here.
gapminder%>%
  filter(country == "United States") %>%
  arrange(desc(year))
###
```

```
   country       continent year lifeExp pop       gdpPercap
1  United States Americas  2007 78.242  301139947 42951.65
2  United States Americas  2002 77.310  287675526 39097.10
```

```
3  United States Americas  1997 76.810  272911760 35767.43
4  United States Americas  1992 76.090  256894189 32003.93
5  United States Americas  1987 75.020  242803533 29884.35
6  United States Americas  1982 74.650  232187835 25009.56
7  United States Americas  1977 73.380  220239000 24072.63
8  United States Americas  1972 71.340  209896000 21806.04
9  United States Americas  1967 70.760  198712000 19530.37
10 United States Americas  1962 70.210  186538000 16173.15
11 United States Americas  1957 69.490  171984000 14847.13
12 United States Americas  1952 68.440  157553000 13990.48
```

Selecting columns with select()

In addition to being able to filter out a subset of rows, you can also filter out a subset of columns with the `select()` verb. **Try to select just the country and year variables.**

```
### (9) Your code here.
head(select(gapminder, country, year))
###
```

```
  country     year
1 Afghanistan 1952
2 Afghanistan 1957
3 Afghanistan 1962
4 Afghanistan 1967
5 Afghanistan 1972
6 Afghanistan 1977
```

Changing columns with mutate()

We can also mutate certain columns. For example, suppose that we wanted life expectancy to be measured in months. We might write:

```
head(gapminder %>%
    mutate(lifeExp = lifeExp*12))
```

```
  country     continent year lifeExp pop      gdpPercap
1 Afghanistan Asia      1952 345.612  8425333 779.4453
2 Afghanistan Asia      1957 363.984  9240934 820.8530
3 Afghanistan Asia      1962 383.964 10267083 853.1007
4 Afghanistan Asia      1967 408.240 11537966 836.1971
5 Afghanistan Asia      1972 433.056 13079460 739.9811
6 Afghanistan Asia      1977 461.256 14880372 786.1134
```

Create a new column in the data frame that is just GDP (not GDP per capita). Store your new data frame in `gapminder_GDP`

```
### (10) Your code here.
gapminder_GDP = head(gapminder %>%
```

```
  mutate(gdp = gdpPercap * pop))
###
```

D. Exploratory Data Analysis

Let's explore a dataset about book prices from Amazon. The data consists of data on $n=325$ books and includes measurements of:

- `aprice`: The price listed on Amazon (dollars)
- `lprice`: The book's list price (dollars)
- `weight`: The book's weight (ounces)
- `pages`: The number of pages in the book
- `height`: The book's height (inches)
- `width`: The book's width (inches)
- `thick`: The thickness of the book (inches)
- `cover`: Whether the book is a hard cover of paperback.
- And other variables...

First, we'll read this data in from Github...

```
install.packages("RCurl")
library(RCurl) #a package that includes the function getURL(), which
allows for reading data from github.
library(ggplot2) #a package for nice plots!

#getURL is a nice way of reading in data from the web
url = getURL(paste0("https://raw.githubusercontent.com/bzaharatos/",
                    "-Statistical-Modeling-for-Data-Science-
Applications/",

"master/Modern%20Regression%20Analysis%20/Datasets/amazon.txt"))
#stores the data in the dataframe amazon
amazon = read.csv(text = url, sep = "\t")

#prints the names in the dataframe
names(amazon)
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

also installing the dependency ‘bitops’
```

```
Attaching package: ‘RCurl’


The following object is masked from ‘package:tidyr’:

    complete


 [1] "Title"         "Author"        "List.Price"    "Amazon.Price"
"Hard..Paper"
 [6] "NumPages"      "Publisher"     "Pub.year"      "ISBN.10"
"Height"
[11] "Width"         "Thick"         "Weight..oz."
```

Next, let's create a new data frame, called `df`, and store a subset of the variables. In addition, we'll change the names of the variables in the dataframe to something cleaner and easier to work with.

```
### (11) Your code here.
df = data.frame(aprice = amazon$Amazon.Price,
                lprice = as.numeric(amazon$List.Price),
                pages = amazon$NumPages,
                width = amazon$Width,
                weight = amazon$Weight..oz.,
                height = amazon$Height,
                thick = amazon$Thick,
                cover = amazon$Hard..Paper)

summary(df)
###
```

```
     aprice           lprice           pages           width
 Min.   :  0.77   Min.   :  1.50   Min.   : 24.0   Min.   :4.100
 1st Qu.:  8.60   1st Qu.: 13.95   1st Qu.:208.0   1st Qu.:5.200
 Median : 10.20   Median : 15.00   Median :320.0   Median :5.400
 Mean   : 13.33   Mean   : 18.58   Mean   :335.9   Mean   :5.585
 3rd Qu.: 13.13   3rd Qu.: 19.95   3rd Qu.:416.0   3rd Qu.:5.900
 Max.   :139.95   Max.   :139.95   Max.   :896.0   Max.   :9.500
                  NA's   :1        NA's   :2       NA's   :5
     weight          height          thick          cover
 Min.   : 1.20   Min.   : 5.100   Min.   :0.1000   Length:325
 1st Qu.: 7.80   1st Qu.: 7.900   1st Qu.:0.6000   Class :character
 Median :11.20   Median : 8.100   Median :0.9000   Mode  :character
 Mean   :12.49   Mean   : 8.163   Mean   :0.9077
 3rd Qu.:16.00   3rd Qu.: 8.500   3rd Qu.:1.1000
 Max.   :35.20   Max.   :12.100   Max.   :2.1000
 NA's   :9       NA's   :4        NA's   :1
```

From the summary, we can see that there are missing values in the dataset, coded as `NA`. There are many ways to deal with missing data. Suppose that sample unit i has a missing measurement for variable z_j. We could:

1. Delete sample unit i from the dataset, i.e., delete the entire row. That might be reasonable if there are very view missing values and if we think the values are missing at random.
2. Delete the variable z_j from the dataset, i.e., delete the entire column. This might be reasonable if there are many many other missing values for z_j and if we think z_j might not be neccesary for our overall prediction/explanation goals.
3. Impute missing values by substituting each missing value with an estimate.

Since most of our columns/variables are not missing values, and since these variables will be useful to us in our analysis, option 2 seems unreasonable. Let's first try option 3: impute the missing values of `lprice`, `pages`, `width`, `weight`, `height`, and `thick` with the mean of each. The following code might help you get started!

```
which(is.na(df$lprice))
df = df %>%
  mutate(lprice = replace(lprice, is.na(lprice),mean(lprice, na.rm =
TRUE)))
```

```
[1] 205
```

```
### (12) Your code here.
df = df %>%
    mutate(lprice = replace(lprice, is.na(lprice),mean(lprice, na.rm =
TRUE)))%>%
    mutate(weight = replace(weight, is.na(weight),mean(weight, na.rm =
TRUE)))%>%
    mutate(pages = replace(pages, is.na(pages),mean(pages, na.rm =
TRUE)))%>%
    mutate(height = replace(height, is.na(height),mean(height, na.rm =
TRUE)))%>%
    mutate(width = replace(width, is.na(width),mean(width, na.rm =
TRUE)))%>%
    mutate(thick = replace(thick, is.na(thick),mean(thick, na.rm =
TRUE)))
summary(df)
###
```

```
     aprice           lprice           pages           width      
 Min.   :  0.77   Min.   :  1.50   Min.   : 24.0   Min.   :4.100  
 1st Qu.:  8.60   1st Qu.: 13.95   1st Qu.:208.0   1st Qu.:5.200  
 Median : 10.20   Median : 15.00   Median :320.0   Median :5.400  
 Mean   : 13.33   Mean   : 18.58   Mean   :335.9   Mean   :5.585  
 3rd Qu.: 13.13   3rd Qu.: 19.95   3rd Qu.:416.0   3rd Qu.:5.900  
 Max.   :139.95   Max.   :139.95   Max.   :896.0   Max.   :9.500  
```

```
     weight          height          thick          cover
 Min.   : 1.20   Min.   : 5.100   Min.   :0.1000   Length:325
 1st Qu.: 7.80   1st Qu.: 7.900   1st Qu.:0.6000   Class :character
 Median :11.20   Median : 8.100   Median :0.9000   Mode  :character
 Mean   :12.49   Mean   : 8.163   Mean   :0.9077
 3rd Qu.:16.00   3rd Qu.: 8.500   3rd Qu.:1.1000
 Max.   :35.20   Max.   :12.100   Max.   :2.1000
```

Use the `summary()` function to print numerical summaries of this dataset.

```
### (13) Your code here.
summary(df)
###
```

```
     aprice           lprice           pages           width
 Min.   :  0.77   Min.   :  1.50   Min.   : 24.0   Min.   :4.100
 1st Qu.:  8.60   1st Qu.: 13.95   1st Qu.:208.0   1st Qu.:5.200
 Median : 10.20   Median : 15.00   Median :320.0   Median :5.400
 Mean   : 13.33   Mean   : 18.58   Mean   :335.9   Mean   :5.585
 3rd Qu.: 13.13   3rd Qu.: 19.95   3rd Qu.:416.0   3rd Qu.:5.900
 Max.   :139.95   Max.   :139.95   Max.   :896.0   Max.   :9.500
     weight          height          thick          cover
 Min.   : 1.20   Min.   : 5.100   Min.   :0.1000   Length:325
 1st Qu.: 7.80   1st Qu.: 7.900   1st Qu.:0.6000   Class :character
 Median :11.20   Median : 8.100   Median :0.9000   Mode  :character
 Mean   :12.49   Mean   : 8.163   Mean   :0.9077
 3rd Qu.:16.00   3rd Qu.: 8.500   3rd Qu.:1.1000
 Max.   :35.20   Max.   :12.100   Max.   :2.1000
```

Use the `arrange()` verb to rearrange the `df` dataframe in descending order with respect to `lprice` (that is, with the row corresponding to the highest `lprice` at the top, the row corresponding to the next highest `lprice` second, etc.). Do *not* rewrite the dataframe in `df`.

```
### (14) Your code here.
df %>% arrange(desc(lprice))
###
```

```
    aprice lprice pages    width weight   height   thick    cover
1   139.95 139.95 160.0000 8.200 22.40000 10.60000 0.500000 P
2    83.04 114.95 544.0000 7.300 28.80000  9.10000 0.800000 P
3    98.95  98.95 778.0000 9.500 12.48797 11.30000 1.500000 H
4    97.50  97.50 480.0000 8.900 14.40000 10.70000 0.900000 P
5    54.61  86.95 512.0000 7.400 23.20000  9.10000 0.800000 P
6    39.45  75.00 700.0000 6.500 12.48797  9.50000 0.907716 H
7    55.75  70.80 560.0000 5.800 19.20000  8.90000 0.900000 P
8    39.92  53.95 192.0000 6.100 11.20000  9.10000 0.500000 P
9    44.32  48.20 384.0000 6.000 14.40000  8.90000 0.600000 P
10   39.72  48.20 400.0000 5.900 16.00000  8.90000 0.600000 P
11   37.95  39.95 384.0000 6.600 22.40000  9.20000 0.800000 P
```

12	23.79	39.95	256.0000	5.585	12.48797	8.16324	1.000000	H
13	25.00	37.50	384.0000	7.300	12.48797	9.10000	1.400000	H
14	35.95	35.95	176.0000	6.100	9.90000	9.10000	0.500000	P
15	25.84	35.75	436.0000	7.400	25.60000	9.50000	1.000000	P
16	21.68	35.75	288.0000	5.400	12.00000	8.40000	0.500000	P
17	18.81	35.00	335.8576	6.500	12.48797	9.60000	2.100000	H
18	23.10	35.00	336.0000	6.100	17.60000	8.90000	1.300000	H
19	24.90	34.80	368.0000	6.700	19.20000	9.40000	0.900000	P
20	21.75	32.95	352.0000	6.500	25.60000	9.50000	1.200000	H
21	16.77	30.50	720.0000	5.200	22.40000	8.00000	1.400000	P
22	19.80	30.00	304.0000	6.400	19.20000	9.60000	1.100000	H
23	17.43	30.00	255.0000	6.500	16.00000	9.60000	0.900000	H
24	20.00	30.00	384.0000	9.000	32.00000	11.00000	1.000000	P
25	23.94	30.00	248.0000	5.500	12.00000	8.50000	0.600000	P
26	26.61	29.50	192.0000	6.000	9.60000	8.90000	0.500000	P
27	16.44	28.95	460.0000	6.300	32.00000	8.90000	1.700000	H
28	16.09	28.00	208.0000	6.300	19.20000	9.20000	0.800000	H
29	16.08	28.00	480.0000	6.200	25.60000	9.10000	1.700000	H
30	11.20	27.99	464.0000	6.000	1.20000	9.10000	1.700000	H
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
296	7.99	7.99	352	4.2	6.9	6.9	1.0	P
297	7.99	7.99	224	5.4	4.8	8.2	0.5	P
298	7.95	7.95	336	6.3	16.0	8.7	1.1	H
299	7.95	7.95	283	4.5	4.2	6.3	0.6	H
300	7.95	7.95	272	5.2	7.2	8.2	0.7	H
301	7.95	7.95	688	4.1	11.2	6.6	1.7	H
302	7.77	7.77	140	5.1	16.0	7.8	1.0	P
303	7.75	7.75	168	5.9	9.6	8.9	0.5	P
304	5.87	7.50	160	4.1	4.0	6.7	0.6	P
305	6.99	6.99	32	9.3	1.6	8.5	0.1	P
306	6.99	6.99	352	4.2	5.6	6.7	1.0	P
307	6.99	6.99	208	4.1	4.0	6.9	0.7	P
308	6.99	6.99	560	4.1	9.6	6.7	0.9	P
309	6.99	6.99	320	5.2	8.0	7.5	0.9	P
310	6.99	6.99	160	7.2	8.0	9.0	0.4	P
311	6.99	6.99	192	4.2	4.0	6.7	0.6	P
312	6.99	6.99	176	5.2	3.2	7.6	0.4	P
313	6.95	6.95	272	5.4	9.6	8.3	0.8	P
314	6.95	6.95	92	5.1	16.0	7.8	1.0	P
315	6.95	6.95	224	5.3	8.0	7.6	0.6	P
316	6.99	6.69	288	5.2	4.8	7.6	0.8	P
317	5.99	5.99	128	4.1	2.4	6.8	0.4	P
318	5.99	5.99	160	5.2	4.8	6.6	0.4	P
319	5.99	5.99	336	4.2	6.4	6.6	1.0	P
320	3.99	5.95	192	5.2	8.8	7.4	0.5	P
321	4.99	4.99	24	4.3	4.0	5.8	0.5	H
322	4.99	4.99	24	4.3	8.0	5.7	0.4	H
323	3.95	3.95	288	4.1	5.6	6.7	1.0	P

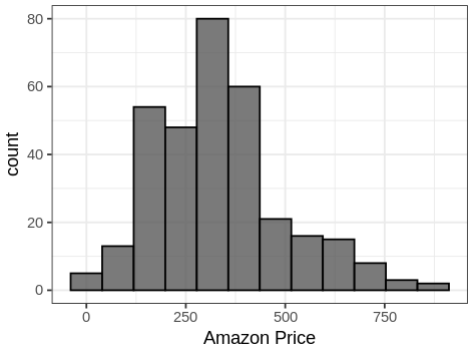
```
324 0.77   2.00   128      5.2    4.2     8.2      0.3      P
325 1.50   1.50    96      5.2    4.0     8.3      0.3      P
```

Note that you could provide more descriptive labels for the levels of this factor (note that `H` = "Hardcover" and `P` = "Paperback"). The easiest way do do this is with the `levels()` function: `levels(x) = value`.

```
levels(df$cover) = c("Hardcover","Paperback")
summary(df)
```

Use `ggplot` to create a histogram of the `pages` variable. Change the number of bins to 15. For credit, store the histogram in the variable `p_hist`. Comment on it's shape.

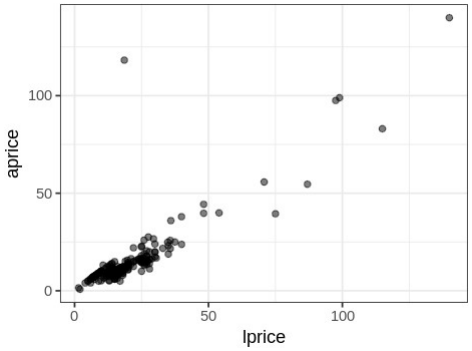
```
### (15) Your code here.
p_hist = ggplot(df) +
    geom_histogram(aes(x = pages), bins =12, alpha = 0.8, color =
"black") +
    xlab("Amazon Price") +
    theme_bw()
p_hist
###
```



The histogram is somewhat bellshaped, but there is certainly skew in the data, with a long right tail.

Use `ggplot` to create a scatterplot of `aprice` (y) against `lprice` (x). What do you notice about this plot?

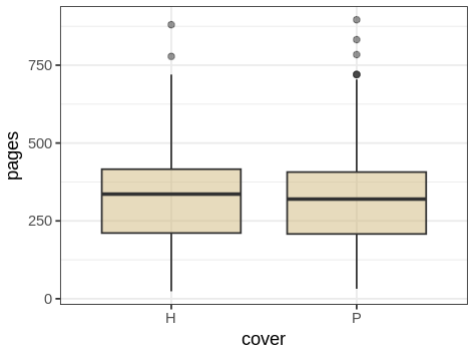
```
### (16) Your code here.
ggplot(df) +
    geom_point(aes(x = lprice, y = aprice), alpha = 0.5)+
    theme_bw()
###
```



The relationship between these variables looks linear, but there is at least one outlier at roughly $(20, 120)$. Could that be the value that we imputed?!

Use `ggplot` to produce a boxplot of `pages` conditioned on `cover`. Interpret this plot.

```
### (17) Your code here.
ggplot(df) +
    geom_boxplot(aes(x = cover, y = pages), alpha = 0.5, fill =
"#CFB87C") +
    theme_bw()
###
```



The average number of pages appears to be a bit higher in hardcover books than in paperback books. These distributions are skewed a bit, with a long tail for high values of `pages`. We note that there are two potential outliers in the `Hardcover` group, and four in the `Paperback` group.