Programming for Data Science with R

Part - I

Mohammad Wasiq

2/2/22

DSM - 1005

Programming for Data Science with R – I
DSM-1005

Table of Contents

I follow the book named **Statistical Inference via Data Science** *A ModernDive into R and the Tidyverse* by **Chester Ismay** and **Albert Y. Kim**

Teacher : **Prof. Athar Ali Khan Sir**

Writer : **Mohammad Wasiq** , $MS(Data\ \ Science)$

1 R programing for Data Science DSM-1005

In this Book we learn the R programming for Data Science at intermediate level .
We learn the following Topics :

- **Tidyverse Package**
- **Data Visualization** Using **ggplot2**
- **Data Wrangling** Using **dplyr**
- **Data Importing** & **Tidy Data**

- Useful Links to Connect me.

LinkedIn : **https://www.linkedin.com/in/mohammad-wasiq-198327217**
Twitter : **https://mobile.twitter.com/Mohammadwasiq0**
Github : **https://github.com/MohammadWasiq0786**
Kaggle : **https://www.kaggle.com/mohammadwasiqqcs**

2 About nycfights13

Here we know about the datasets of **nycglights13** packages . Flights leaving NYC (New York City) in 2013. All three major airports in New York City: Newark (origin code EWR), John F. Kennedy International (JFK), and LaGuardia (LGA)

1. ***About Flights Data*** *Description* : On-time data for all flights that departed NYC (i.e. JFK, LGA or EWR) in 2013. *Usage* : flights *Format* : Data frame with columns

- year, month, day = Date of departure.
- dep_time, arr_time = Actual departure and arrival times (format HHMM or HMM), local tz.
- sched_dep_time, sched_arr_time = Scheduled departure and arrival times (format HHMM or HMM), local tz.
- dep_delay, arr_delay = Departure and arrival delays, in minutes. Negative times represent early departures/arrivals.
- carrier = Two letter carrier abbreviation. See airlines to get name.
- flight = Flight number.
- tailnum = Plane tail number. See planes for additional metadata.
- origin, dest = Origin and destination. See airports for additional metadata.
- air_time = Amount of time spent in the air, in minutes.
- distance = Distance between airports, in miles.
- hour, minute = Time of scheduled departure broken into hour and minutes.
- time_hour = Scheduled date and hour of the flight as a POSIXct date.

2. ***About weather data*** *Description* : Hourly meterological data for LGA, JFK and EWR. *Usage* : weather *Format* : A data frame with columns:

- origin = Weather station. Named origin to facilitate merging with flights data.
- year, month, day, hour = Time of recording.
- temp, dewp = Temperature and dewpoint in F.
- humid = Relative humidity.
- wind_dir, wind_speed, wind_gust = Wind direction (in degrees), speed and gust speed (in mph).
- precip = Precipitation, in inches.
- pressure = Sea level pressure in millibars.
- visib = Visibility in miles.
- time_hour = Date and hour of the recording as a POSIXct date.

3 Data Visualization with `ggplot2`

Here we are discussion `Graphics` using **ggplot2** package. We will studing the following five(5) graphs 1. **Scatter Plot** 2. **Line Graph** 3. **Boxplot** 4. **Histogram** 5. **Barplot** The basic *Syntax* of ggplot is $\boldsymbol{ggplot(data = dataset, \quad mapping = aes(x = x - axis, \quad y = y - axis))} +$ **geom_graph_name()**

3.1 Scatter Plot or Bivariate Plots

Scatter Plot is used to visualize the relationship between two numerical variables.
ggplot(data = dataset, mapping = aes(x = x-axis, y = y-axis)) + geom_point( )

3.1.1 LC (2.3 - 2.6) :-

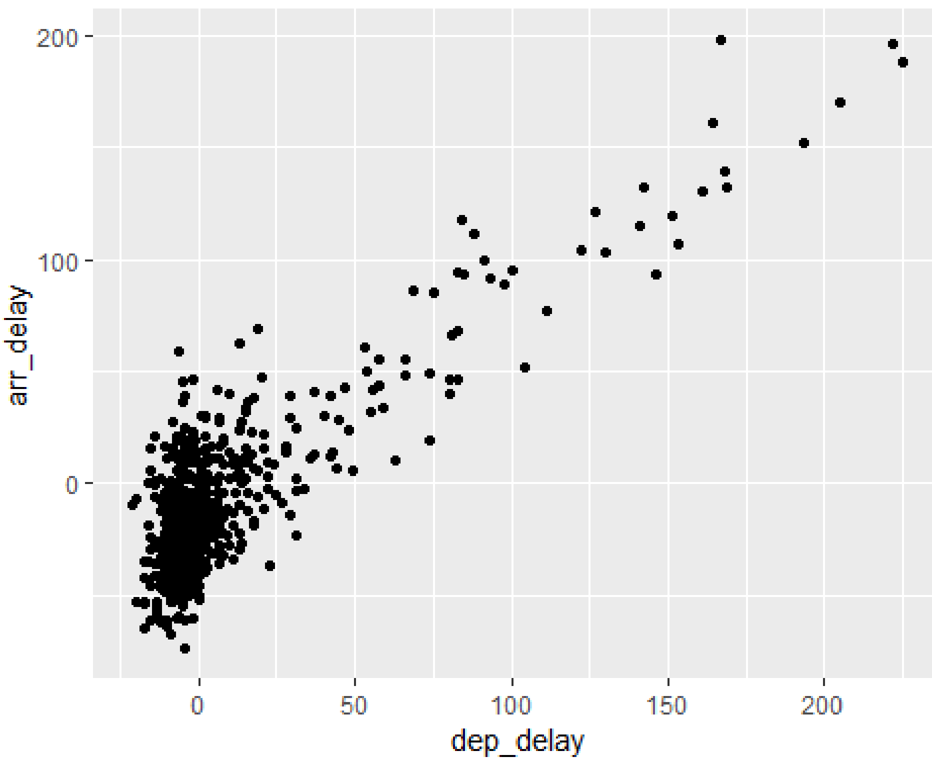
Take the flights data frame, filter the rows so that only the 714 rows corresponding to *Alaska Airlines* flights are kept, and save this in a new data frame called *alaska_flights* and make a *scatter plot* with `dep_delay` departure delay on the horizontal *"x" - axis* and `arr_delay` arrival delay on the vertical *"y" - axis* .

```
# upload the require libraries
library(ggplot2)
library(dplyr)
library(nycflights13)

# load the flights dataset and watch it's head
data(flights)
head(flights)

## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay
##   <int> <int> <int>    <int>          <int>     <dbl>
## 1  2013     1     1      517            515         2
## 2  2013     1     1      533            529         4
## 3  2013     1     1      542            540         2
## 4  2013     1     1      544            545        -1
## 5  2013     1     1      554            600        -6
## 6  2013     1     1      554            558        -4
## # ... with 13 more variables: arr_time <int>,
## #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>

# Filter the data into alaska flights using dplyr
alaska_flights <- flights %>% filter(carrier == "AS")
head(alaska_flights)

## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay
##   <int> <int> <int>    <int>          <int>     <dbl>
## 1  2013     1     1      724            725        -1
## 2  2013     1     1     1808           1815        -7
## 3  2013     1     2      722            725        -3
## 4  2013     1     2     1818           1815         3
## 5  2013     1     3      724            725        -1
## 6  2013     1     3     1817           1815         2
## # ... with 13 more variables: arr_time <int>,
```

```
## #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

```
# Plot the Scatter plot
ggplot(data = alaska_flights , mapping = aes(x = dep_delay , y = arr_delay))
+ geom_point()
```

```
## Warning: Removed 5 rows containing missing values
## (geom_point).
```



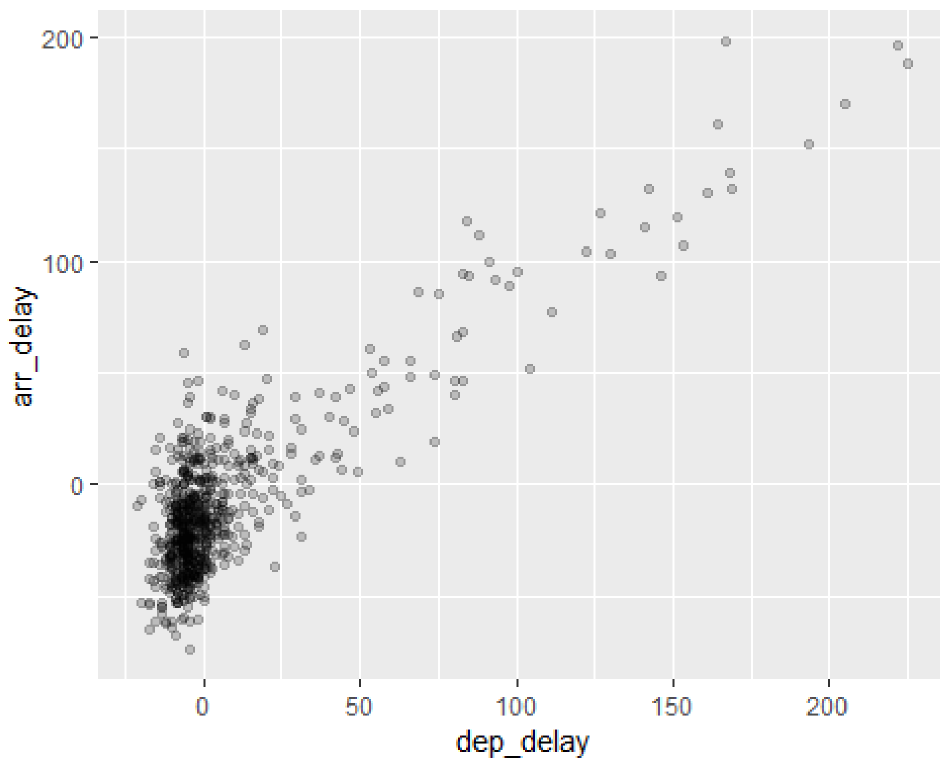
Interpretation :- From the above Scatter plot we say that there is a `positive correlation` b/w *dep_delay* and *arr_delay* and most of points lies near **(0, 0)** co-ordinates. It's means that `Most of the flights of Alaska  arrival and departure at time.` i.e. *There is good service given by airport for Alaska.*

Overplottig :- The large mass of points near (0, 0) in Figure can cause some confusion since it is hard to tell the true number of points that are plotted. This is the result of a phenomenon called **overplotting**. As one may guess, this corresponds to point being plotted on top of each other over and over again. When overplotting occurs, it is difficult to know the number of points being plotted. There are two methods to address the issue of overplotting. *Adjusting the transparency of the points* or Adding a little random "jitter", or random "nudges", to each of the points.

3.1.1.1 Method 1 : Changing the Transparency / Opacity

To change the transparency of the points by setting the `alpha` argument in `geom_point()`. We take the *alpha value between **0** and **1** , where 0 means 10*0 % transparent and 1 means *100% opaque* . By default , alpha is set 1 .
Syntax : - : ***ggplot(data , aes(x , y)) + geom_point(alpha = 0 to 1)***

```
ggplot(alaska_flights , aes(dep_delay , arr_delay)) + geom_point(alpha = 0.2)

## Warning: Removed 5 rows containing missing values
## (geom_point).
```

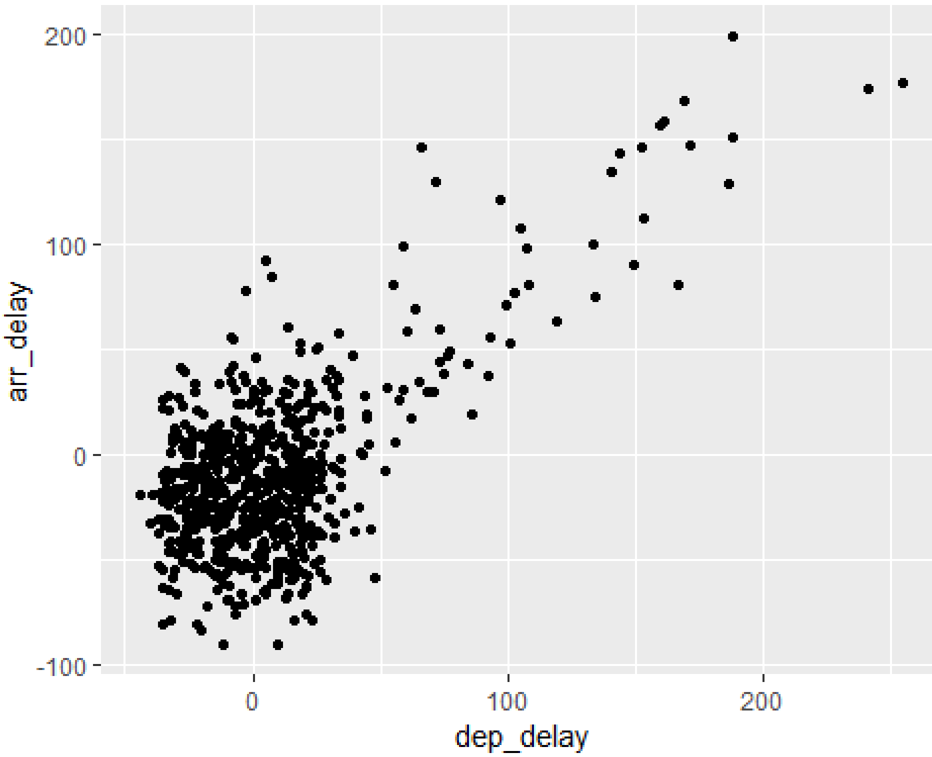


Conclusion -: Here , we can see that the highly degree overplotting are darker .

3.1.1.2 Method 2 : Jittering the Points

The second way of addressing overplotting is by *jittering* all the points. To create a jittered scatterplot, instead of using geom_point(), we use **geom_jitter()**.

```
ggplot(alaska_flights , aes(dep_delay , arr_delay)) + geom_jitter(width = 30
, height = 30)

## Warning: Removed 5 rows containing missing values
## (geom_point).
```



Conclusion -: Here , we can see that this figure is zoomed as above. we adjusted the `width` and `height` arguments to *geom_jitter().* This corresponds to how hard you'd like to shake the plot in horizontal x-axis units and vertical y-axis units, respectively . As we increase the value of `width` and `height` , the graph is zoom .

3.1.1.2.1 Summary

Scatterplots display the relationship between two numerical variables. They are among the most commonly used plots because they can provide an immediate way to see the trend in one numerical variable versus another. However, if we try to create a scatterplot where either one of the two variables is not numerical, we might get strange results. Be careful!

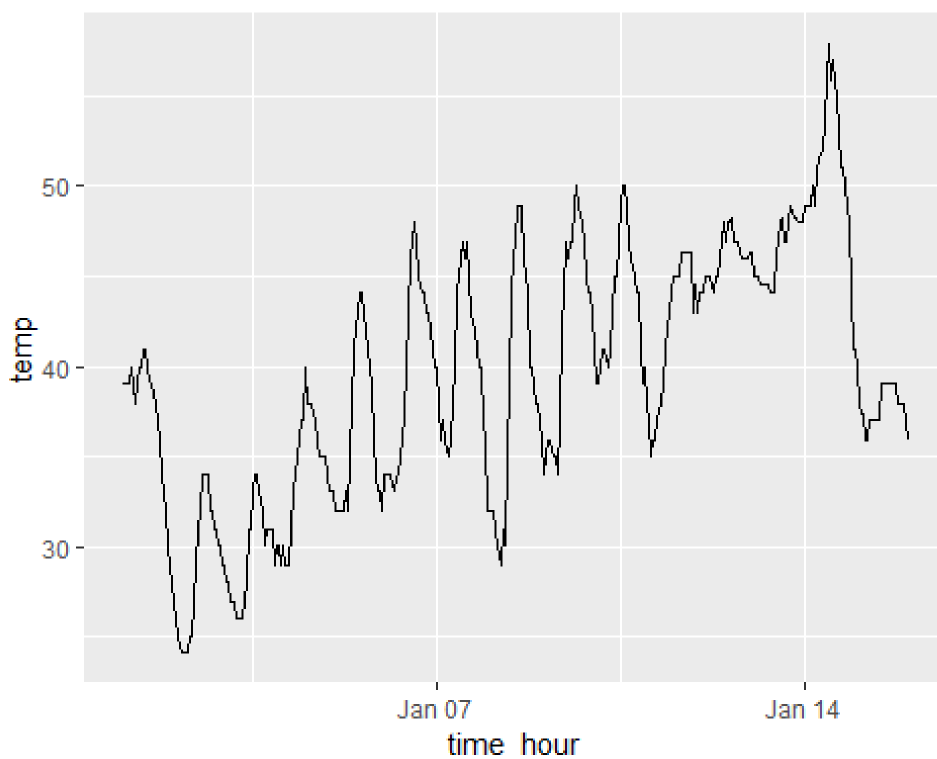
3.2 Line Graphs

Line Graphs show the relationship between two numerical variables when the variable on the x-axis , also called the **explanatoy variable** , is of a sequential nature. *OR* There is an inherent ordering to the varible . Linegraph have some notation of `time` on the `x-axis` .
*Syntax : ***ggplot(data ,mapping = aes(x, y)) + geom_line()***

3.2.1 LC (2.9 - 2.10) :-

Illustrate linegraphs using another dataset in the nycflights13 package named *weather* data frame. choose `weather` where the origin is `"EWR"`, the month is `January`, and the day is between `1` and `15` and plot a linegraph on x-axis `time_hour` and on y-axis `temp` .

```r
# Load weather data
data("weather")

# Filter the data
january_weather <- weather %>% filter(origin == "EWR" ,
month == 1 & day <= 15)

head(january_weather)

## # A tibble: 6 x 15
##   origin  year month   day  hour  temp  dewp humid wind_dir
##   <chr>  <int> <int> <int> <int> <dbl> <dbl> <dbl>    <dbl>
## 1 EWR     2013     1     1     1  39.0  26.1  59.4      270
## 2 EWR     2013     1     1     2  39.0  27.0  61.6      250
## 3 EWR     2013     1     1     3  39.0  28.0  64.4      240
## 4 EWR     2013     1     1     4  39.9  28.0  62.2      250
## 5 EWR     2013     1     1     5  39.0  28.0  64.4      260
## 6 EWR     2013     1     1     6  37.9  28.0  67.2      240
## # ... with 6 more variables: wind_speed <dbl>,
## #   wind_gust <dbl>, precip <dbl>, pressure <dbl>,
## #   visib <dbl>, time_hour <dttm>

# Plot Line Graph
ggplot(january_weather , aes(time_hour , temp)) + geom_line()
```

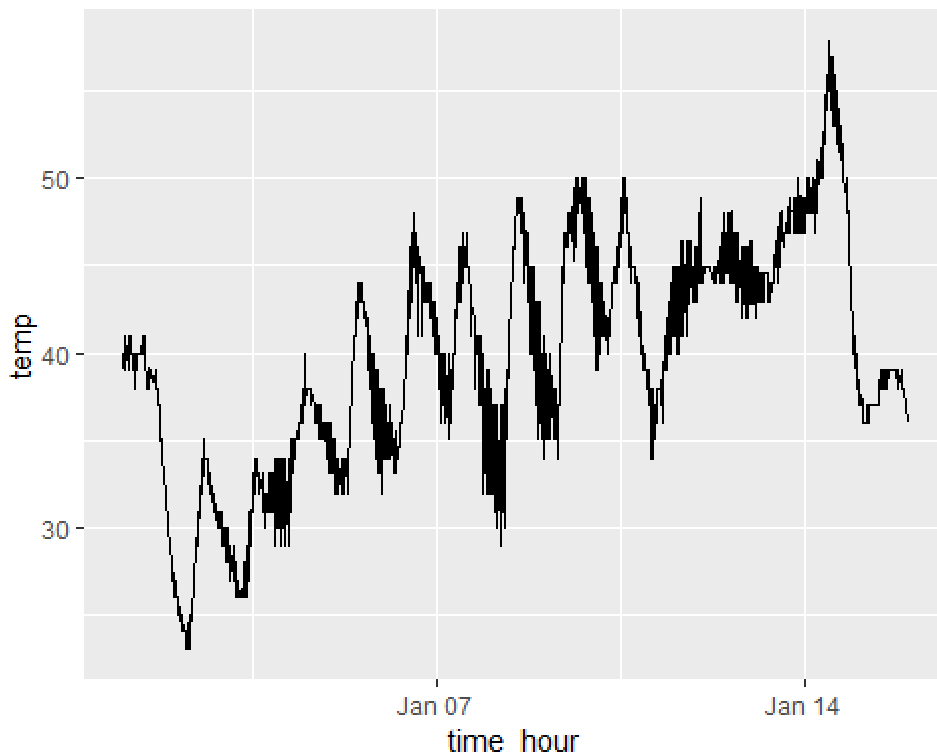


(LC 2.11 & 2.12) -: Why should linegraphs be avoided when there is not a clear ordering of the horizontal axis? *Ans-:* Because n the variable on the x-axis, also called the explanatory variable, is of a sequential nature. **OR**

Because lines suggest connectedness and ordering. Because time is sequential: subsequent observations are closely related to each other.

(LC2.13) -: Plot a time series of a variable other than temp for New York Airport in the first 15 days of January 2013 .

```
# filter the data
jan_nyc_weather <- weather %>% filter(month == 1 & day <= 15)

# Plot Line Graph
ggplot(jan_nyc_weather , aes(time_hour , temp)) + geom_line()
```



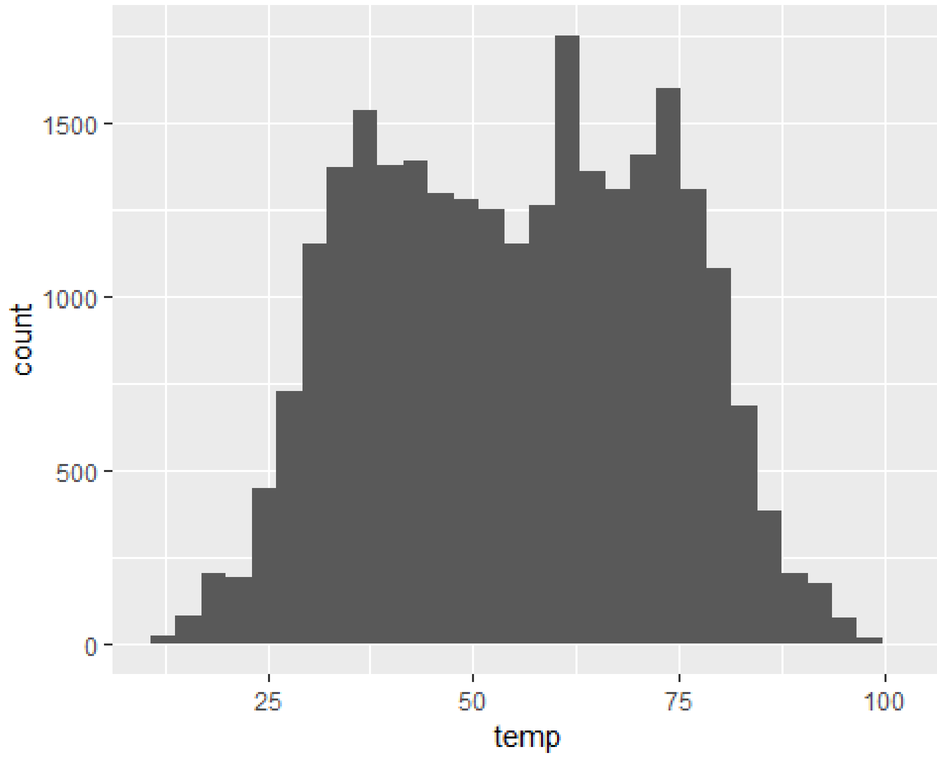
Summary -: Just like scatterplots, display the relationship between two numerical variables. However, it is preferred to use linegraphs over scatterplots when the variable on the x-axis (i.e., the explanatory variable) has an inherent ordering, such as some notion of time.

3.3 Histogram

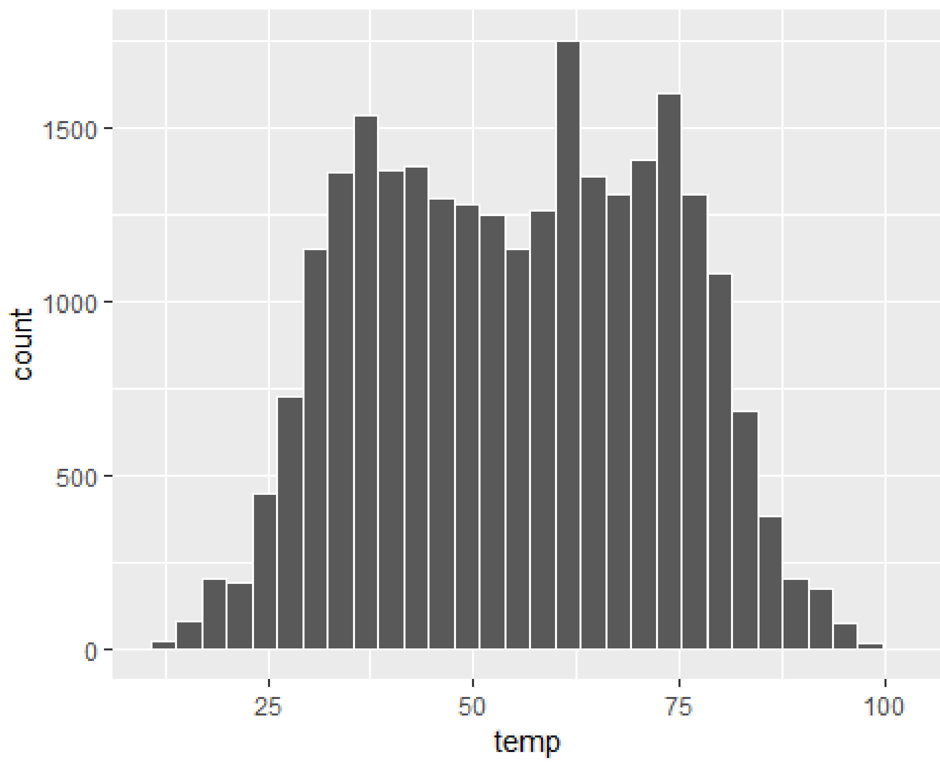
Histogram tells us that how the values of variable distribute . In other word ,we can interprete using histogram **Smallest** and **Largest** values **Center** and **Typical** values **Spread** of values **Frequent** and **Infrequent** of values **Distribution** of Values

- A histogram is plot that visualizes the distribution of a numerical value as follows :
- We first cut up the x-axis into a series of bins, where each bin represents a range of values.
- For each bin, we count the number of observations that fall in the range corresponding to that bin.
- Then for each bin, we draw a bar whose height marks the corresponding count.
- *Syntax:* ***ggplot(data , mapping = aes(x =)) + geom_histogram(bins = 40 , color = "", fill ="")***
- Make a `Histogram` of temp variable from `weather` data .

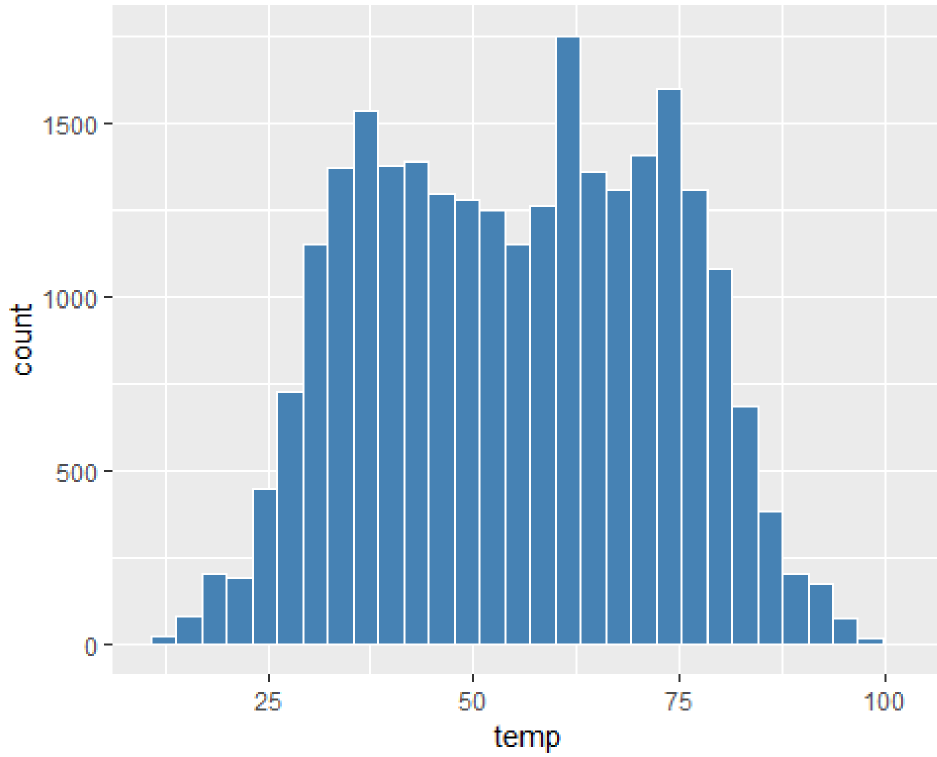
```
library(ggplot2)
library(dplyr)
library(nycflights13)

jan_temp <- weather %>% filter(month == 1)

# Histogram of temp variable from weather
p <- ggplot(weather , aes(temp))
p + geom_histogram()

## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.

## Warning: Removed 1 rows containing non-finite values
## (stat_bin).
```




```
# Histogram with different bins
p + geom_histogram(col = "white")

## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.

## Warning: Removed 1 rows containing non-finite values
## (stat_bin).
```




```r
# Histogram of Blue Color
p + geom_histogram(col = "white" , fill = "steelblue")

## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.

## Warning: Removed 1 rows containing non-finite values
## (stat_bin).
```



- **Interpretation -:** The *temperature* is measures about **1750** times which is approximately **63 C**
- This data is aprpoximately **symmetric** OR it's follows **Normal Distribution**

Note -:

- There are **657** possible colors in *R* , which can be seen by the command `colors()` .

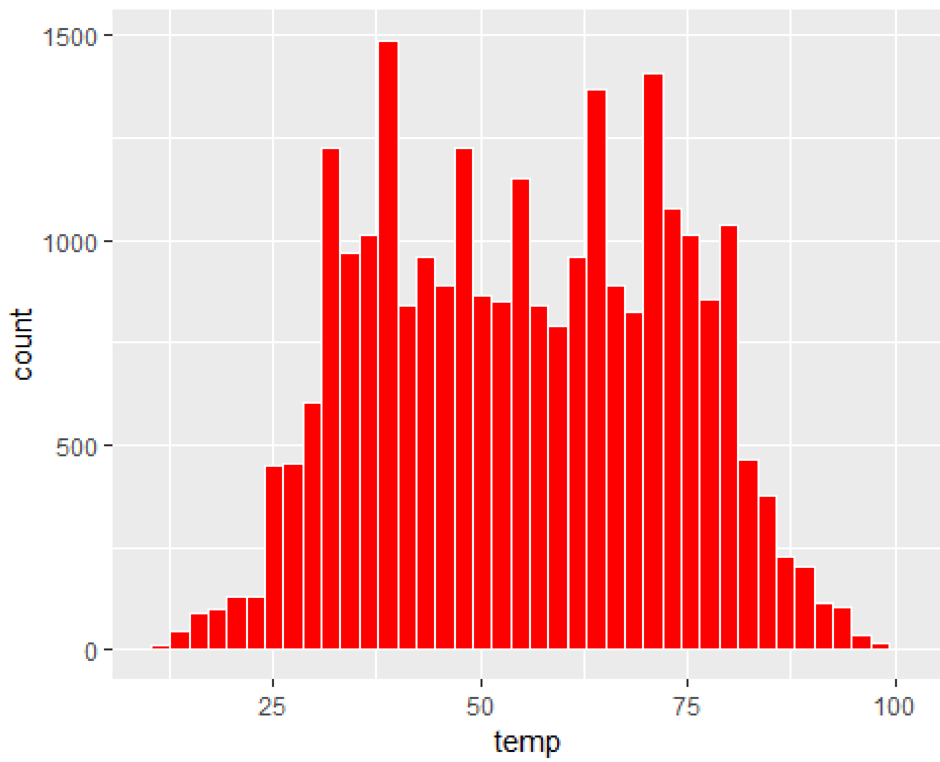
3.3.0.1 Adjusting the bins

bins is the width of a *Histogram* . By default bins is `30` .

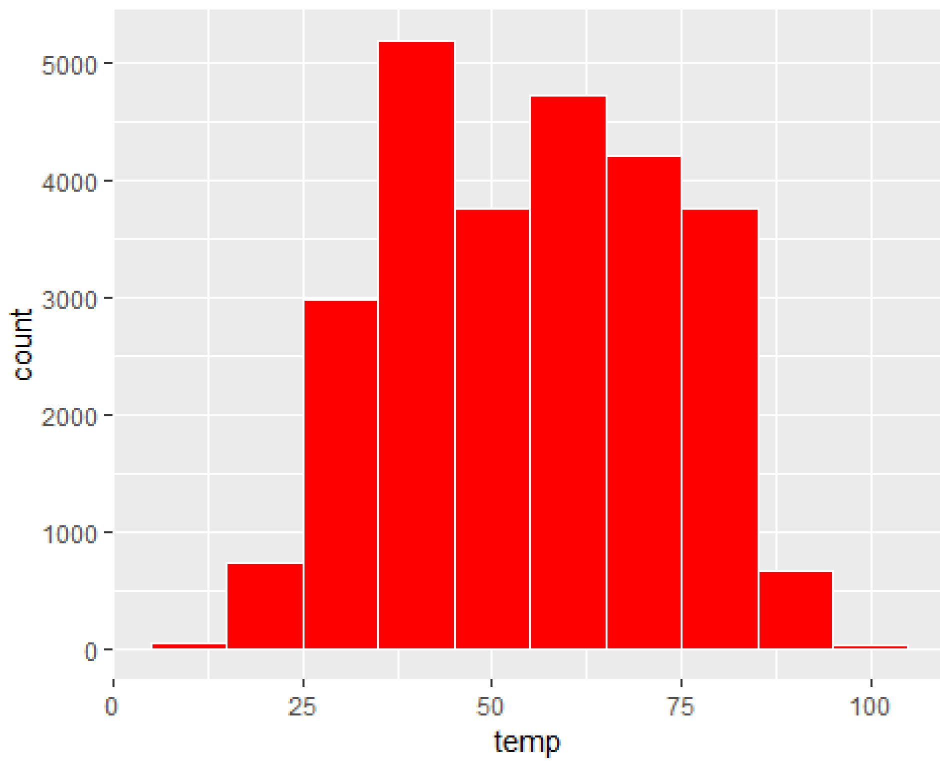
- **Task -:**

1. By adjusting the number of bins via the bins argument to `geom_histogram()`.
2. By adjusting the width of the bins via the binwidth argument to `geom_histogram()`.

```
# Histogram with 40 bins
p <- ggplot(weather , aes(temp))
p + geom_histogram(bins = 40 , col= "white" , fill = "red")

## Warning: Removed 1 rows containing non-finite values
## (stat_bin).
```




```
# Histogram with binwidth = 10
 p + geom_histogram(binwidth = 10 , col= "white" , fill = "red")

## Warning: Removed 1 rows containing non-finite values
## (stat_bin).
```



- **(LC2.14) -:** What does changing the number of bins from 30 to 40 tell us about the distribution of temperatures?
  Ans -: When we increase the *bins* from 30 to 40 increase the width of bars. Temperature is approximately **symmetric** i.e. It's follows **Normal Distribution**

- **(LC2.15) -:** Would you classify the distribution of temperatures as symmetric or skewed in one direction or another?
  Ans -: Temperature is approximately **symmetric** i.e. It's follows **Normal Distribution**

- **(LC2.16) -:** What would you guess is the "center" value in this distribution? Why did you make that choice?
  Ans -: The center value of the distribution is around **55** because in graph aound *55* is the mid value.

- **(LC2.17) -:** Is this data spread out greatly from the center or is it close? Why ?
  Ans -: This data spread colse to center because this data folows Normal distribution
  .

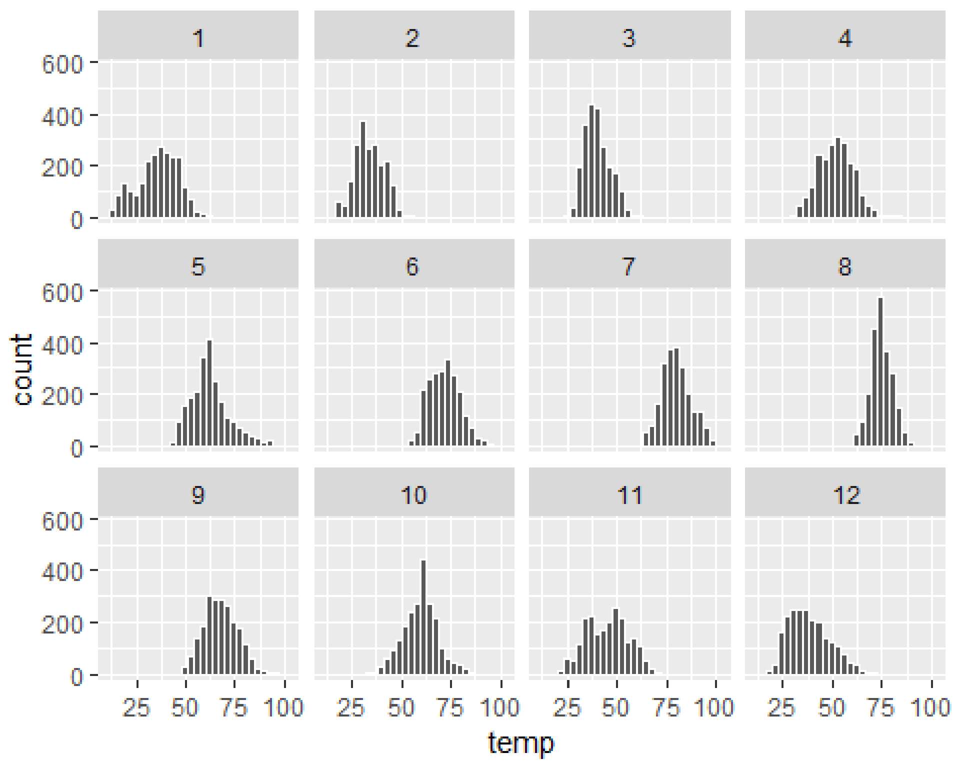
Summary :- Histograms, unlike scatterplots and linegraphs, present information on only a single numerical variable. Specifically, they are visualizations of the distribution of the numerical variable .

3.3.0.2 Facets

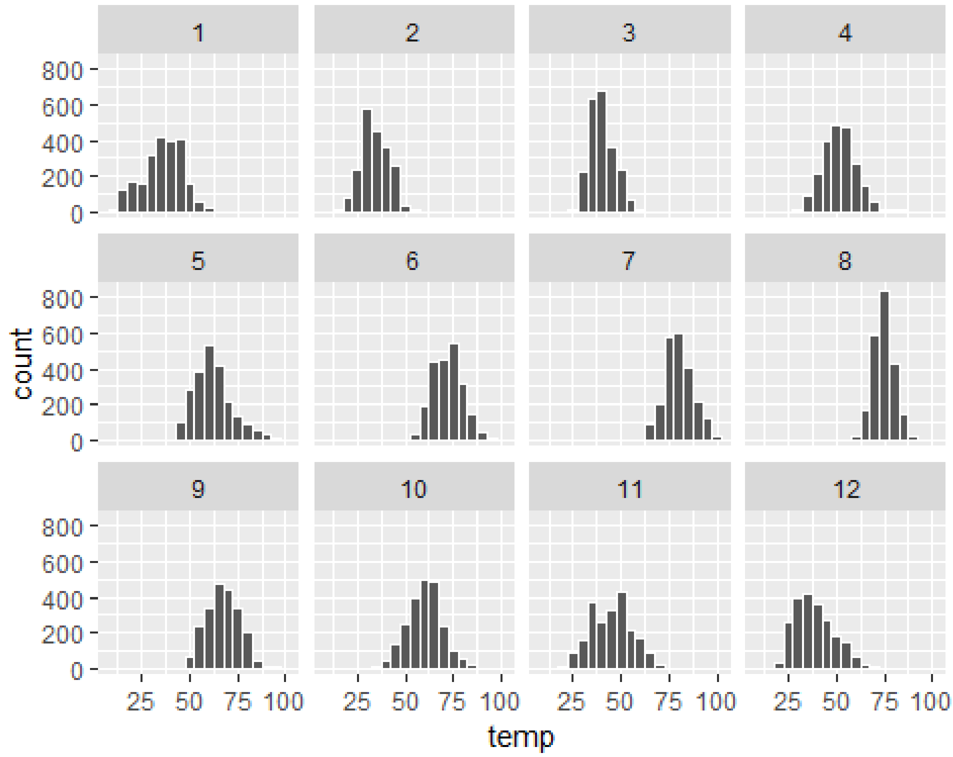
Faceting is used when we'd like to split a particular visualization by the values of another variable.

- *Syntax : **ggplot(data, mapping = aes(x)) + geom_histogram(binwidth = 5 , color = "white") + facet_wrap(~ cat_var)***
- **Task -:** Wrap the temp variable of weather data accorging to Months .

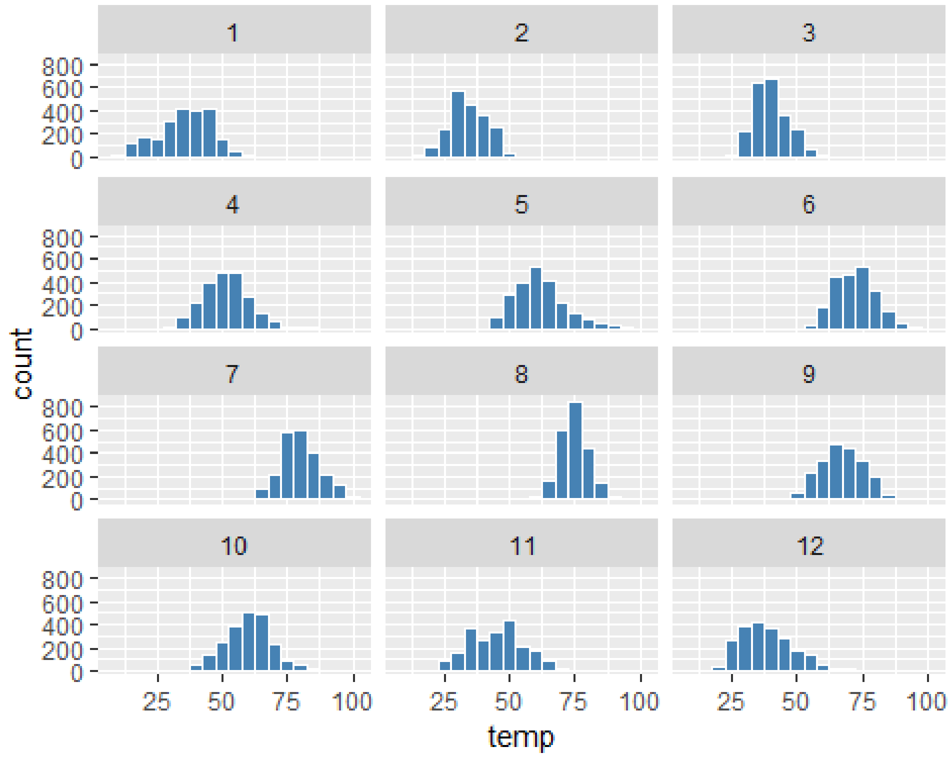
```
p <- ggplot(weather , aes(temp))
p + geom_histogram(col= "white") +
facet_wrap(~ month)

## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.

## Warning: Removed 1 rows containing non-finite values
## (stat_bin).
```




```
# Wrap with binwidth = 5
p + geom_histogram(binwidth = 5 , col= "white") +
facet_wrap(~ month)

## Warning: Removed 1 rows containing non-finite values
## (stat_bin).
```




```
# Wrap with binwidth = 5 abd 4 rows and steelblue color
p + geom_histogram(binwidth = 5 , col= "white" , fill = "steelblue") +
facet_wrap(~ month , nrow = 4)

## Warning: Removed 1 rows containing non-finite values
## (stat_bin).
```



- **(LC2.18)** What other things do you notice about this faceted plot ?
  How does a faceted plot help us see relationships between two variables ?
  Ans -: It's easy to interpretate . **OR**
  Certain months have much more consistent weather (August in particular), while others have crazy variability like January and October, representing changes in the seasons.
  Because we see *temp* recordings split by *month*, we are considering the relationship between these two variables. For example, for summer months, temperatures tend to be higher.
- **(LC2.19)** What do the numbers 1-12 correspond to in the plot ? What about 25, 50, 75, 100 ?
  Ans -: Number 1- 12 are the months. Month 1 , 6 is left skewed and 2 , 3 , 5 , 12 are right skewed and others are symmetric. 25 , 50 , 75 , 100 are *Temperature* .
- **(LC2.20)** For which types of datasets would faceted plots not work well in comparing relationships between variables ? Give an example describing the nature of these variables and other important characteristics.
  Ans -: For Numerical variable the faceted plots not work well in comparing relationships between variables .
  Ex - If we faceted by individual days rather than months, as we would have 365 facets to look at. When considering all days in 2013, it could be argued that we shouldn't care about day-to-day fluctuation in weather so much, but rather month-to-month fluctuations, allowing us to focus on seasonal trends.

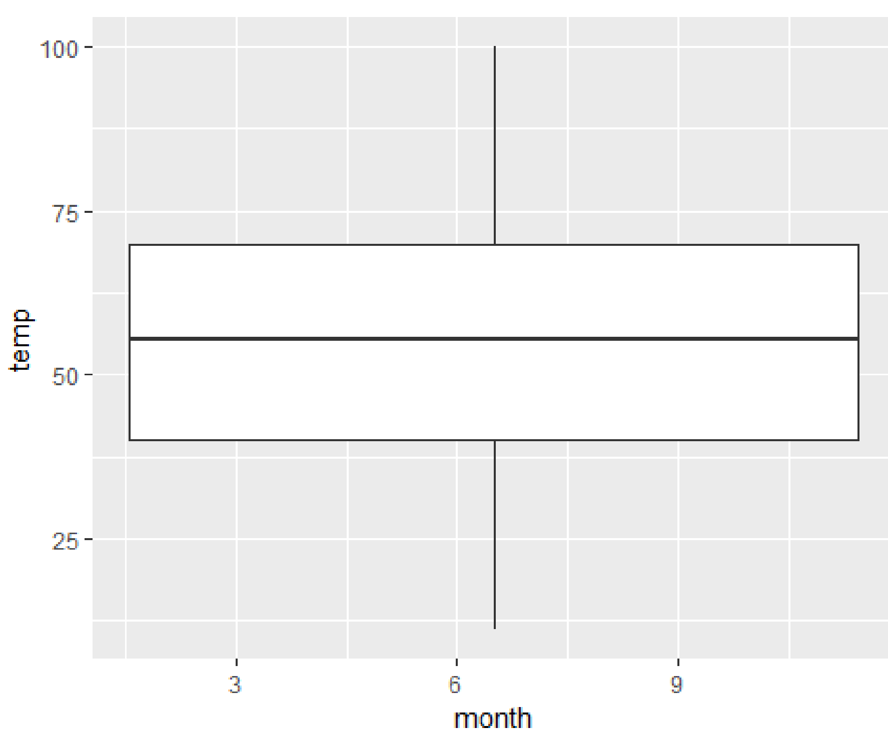
```
# p <- ggplot(weather , aes(temp))
# p + geom_histogram(col= "white" , binwidth = 5) +
# facet_wrap(~ humid)
```

- **(LC 2.21)** Does the temp variable in the weather dataset have a lot of variability ? Why do you say that ?
  Ans -: I would say yes, because in New York City, you have 4 clear seasons with different weather. Whereas in Seattle WA and Portland OR, you have two seasons: summer and rain!

3.4 Boxplots

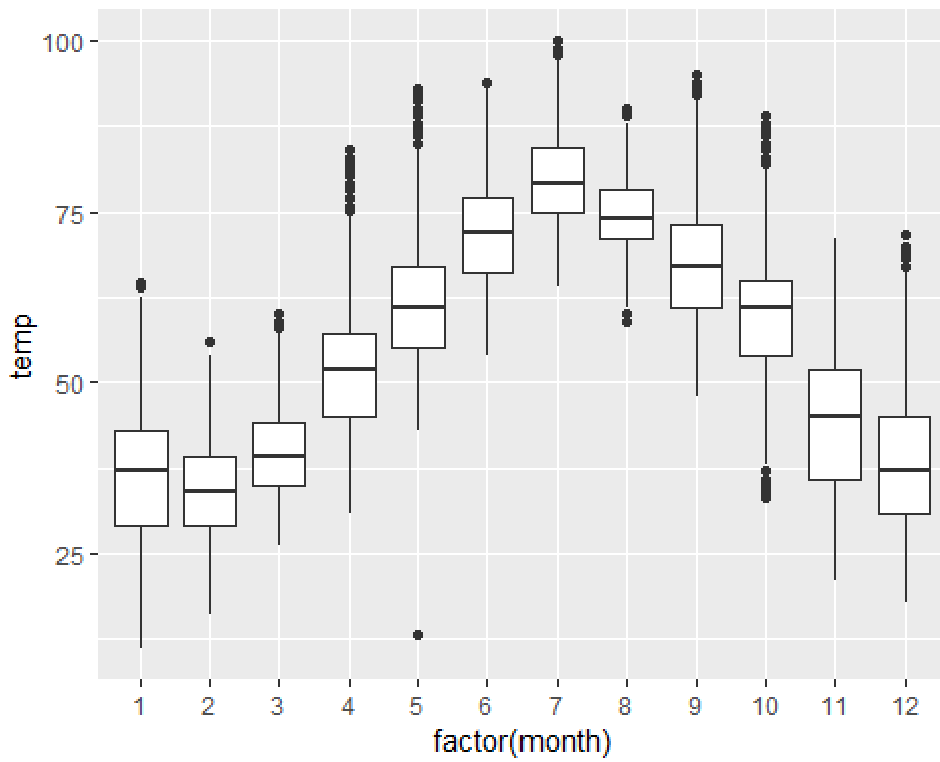
A **boxplot** is constructed from the information provided in the **five number summary** .
ggplot(data, mapping = aes(x, y)) + geom_boxplot( )
x-axis = **Categorical_Variable** , y-axis = **Numeric_Variable**

```
# Normal Boxplot
p <- ggplot(weather, aes(month , temp))
p + geom_boxplot()

## Warning: Continuous x aesthetic -- did you forget
## aes(group=...)?

## Warning: Removed 1 rows containing non-finite values
## (stat_boxplot).
```

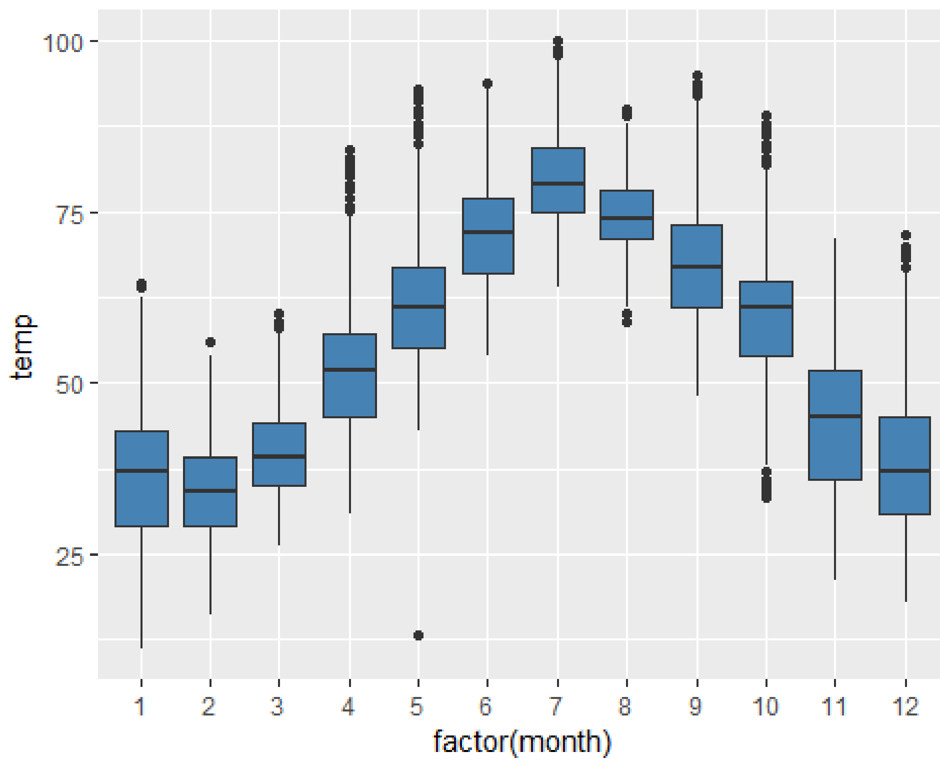


The above boxplot show error because both both aes are numeric . To remove this error We can convert the numerical variable month into a factor categorical variable by using the **`factor()`** function.

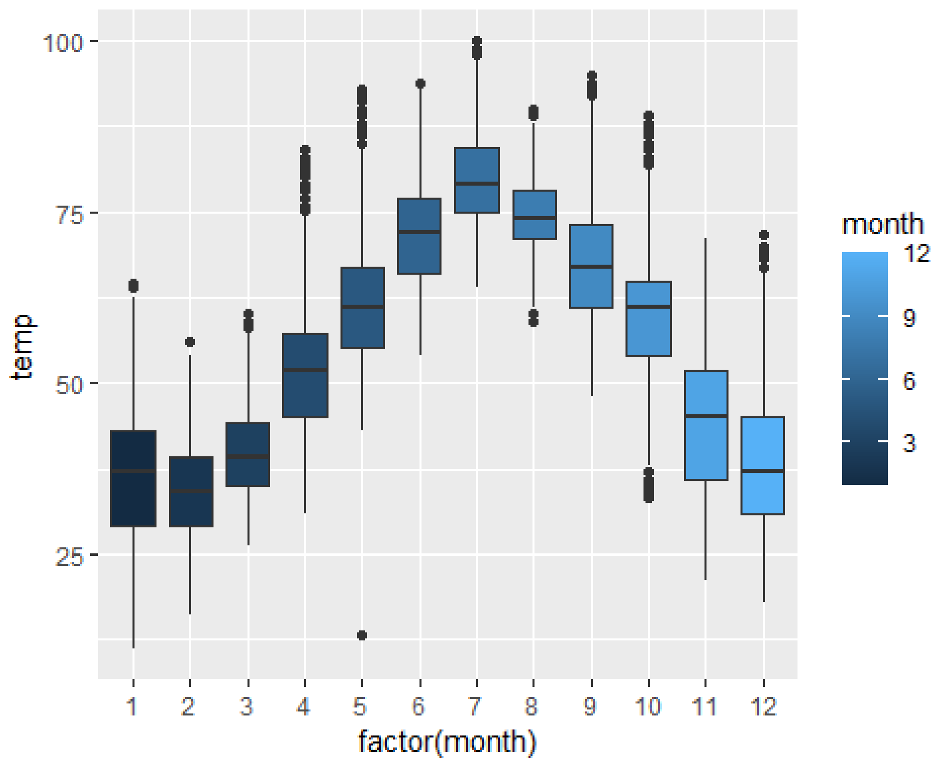
```r
# Boxplot with factor
p <- ggplot(weather, aes(factor(month) , temp))
p + geom_boxplot()

## Warning: Removed 1 rows containing non-finite values
## (stat_boxplot).
```



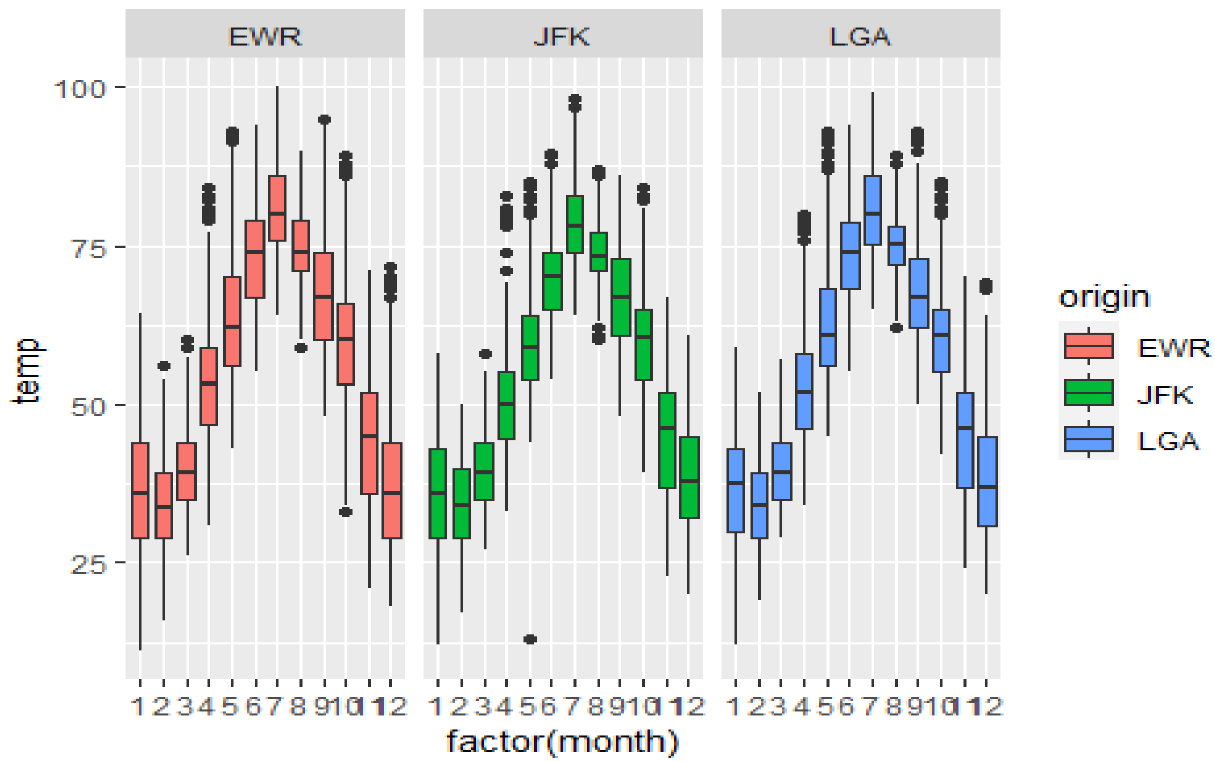

```r
# Boxplot with steelblue color
p + geom_boxplot(fill = "steelblue")

## Warning: Removed 1 rows containing non-finite values
## (stat_boxplot).
```



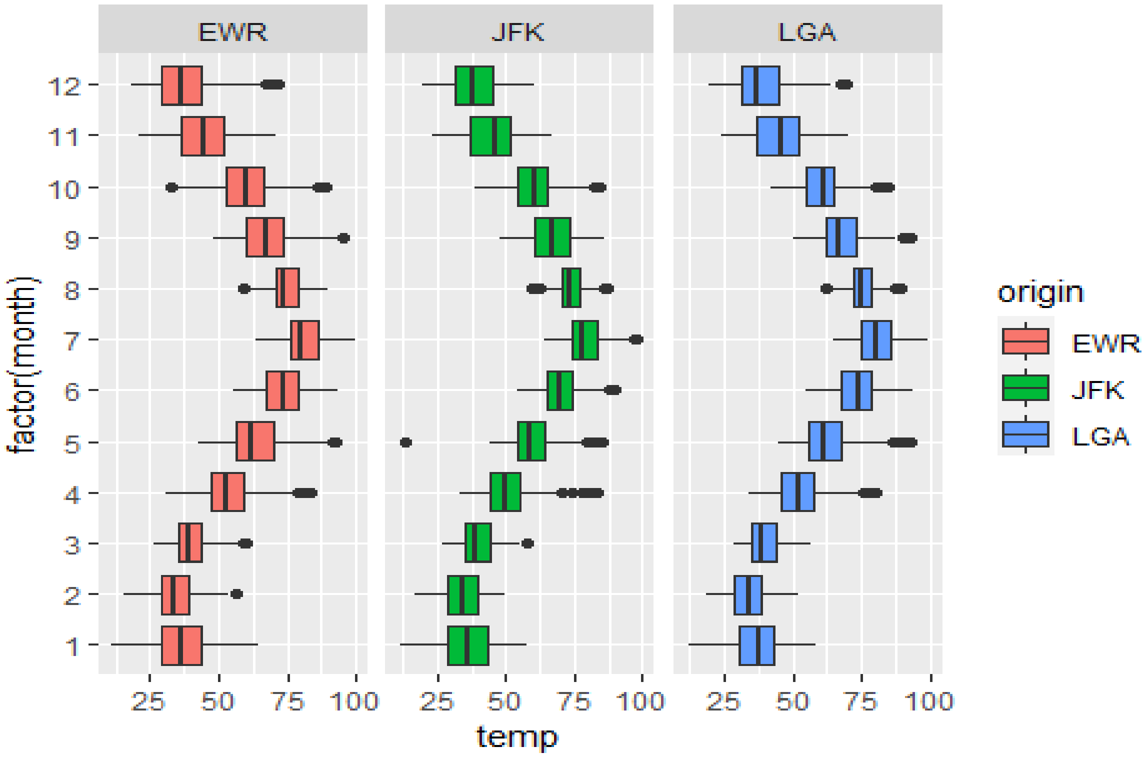

```
# Boxplot with month color
ggplot(weather, aes(factor(month) , temp , fill = month)) + geom_boxplot()

## Warning: Removed 1 rows containing non-finite values
## (stat_boxplot).
```




```r
# Boxplot with origin color & facting with origin
ggplot(weather, aes(factor(month) , temp , fill = origin)) + geom_boxplot() +
facet_wrap(~ origin)

## Warning: Removed 1 rows containing non-finite values
## (stat_boxplot).
```

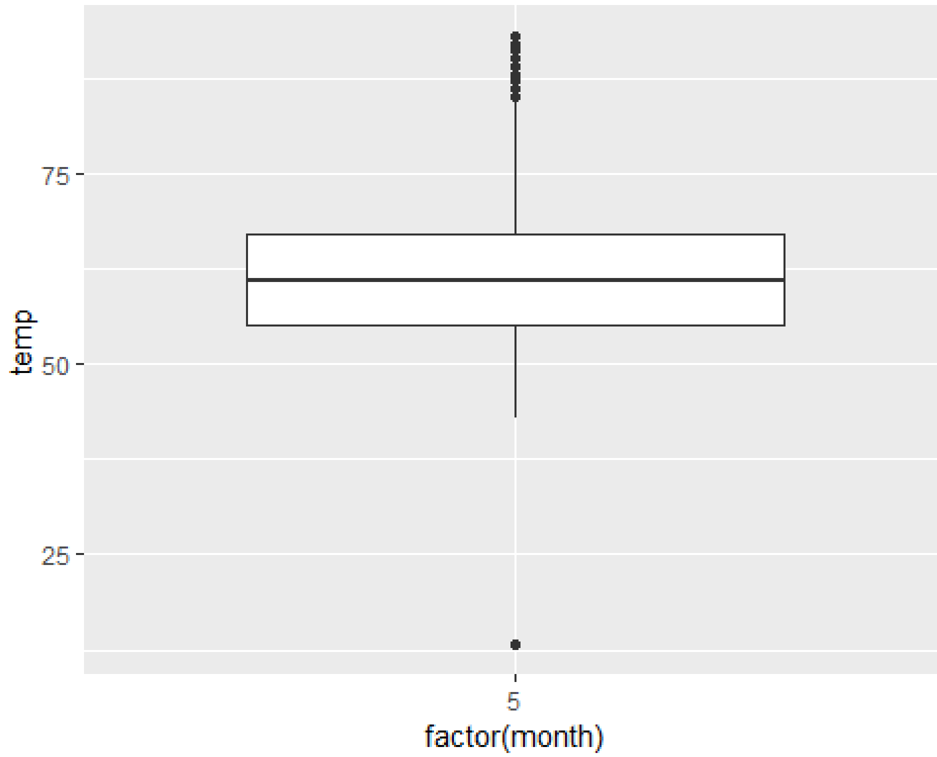



```
# Horizontal Boxplot with origin color & facting with origin
ggplot(weather, aes(temp , factor(month) , fill = origin)) + geom_boxplot() +
facet_wrap(~ origin)

## Warning: Removed 1 rows containing non-finite values
## (stat_boxplot).
```

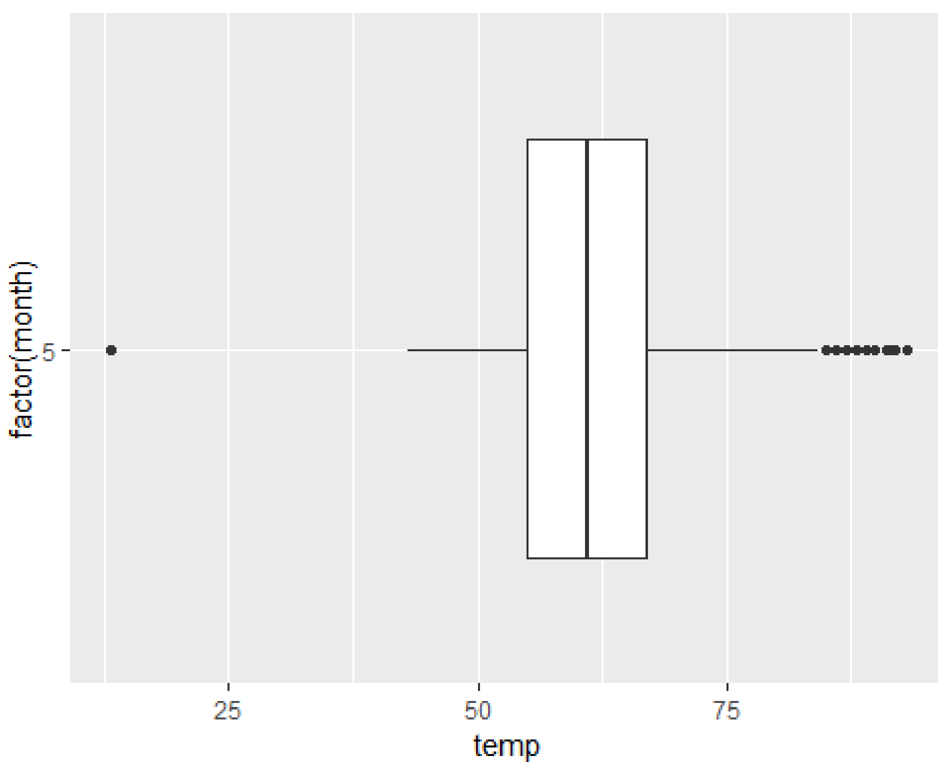


- **LC2.22)** What does the dot at the bottom of the plot for May correspond to? Explain what might have occurred in May to produce this point.

```
may_weather <- weather %>% filter(month ==5)

ggplot(may_weather, aes(factor(month), temp)) + geom_boxplot()
```




```
# Horizontal
ggplot(may_weather, aes(temp , factor(month))) + geom_boxplot()
```



Ans -: The dot at the bottom of the plot is called **Outlier**. The minimum value of *temp* variable is **12.5** and maximum value is **90** , *25 %* of temp data is below than **56** and rest is above . *50 %* of temp data is below than **60** and rest is above . *75 %* of temp data is below than **68** and rest is above .

- **(LC2.23)** Which months have the highest variability in temperature? What reasons can you give for this ?
  Ans -: We are now interested in the **spread** of the data. One measure some of you may have seen previously is the standard deviation. But in this plot we can read off the Interquartile Range (IQR) :
  * The distance from the 1st to the 3rd quartiles i.e. the length of the boxes
  * You can also think of this as the spread of the middle 50% of the data

Just from eyeballing it, it seems

- November has the biggest IQR, i.e. the widest box, so has the most variation in temperature
  August has the smallest IQR, i.e. the narrowest box, so is the most consistent temperature-wise

```
weather %>%
  group_by(month) %>%
  summarize(IQR = IQR(temp, na.rm = TRUE)) %>%
  arrange(desc(IQR))

## # A tibble: 12 x 2
##    month   IQR
##    <int> <dbl>
##  1    11 16.0
##  2    12 14.0
##  3     1 13.8
##  4     9 12.1
##  5     4 12.1
##  6     5 11.9
##  7     6 11.0
##  8    10 11.0
##  9     2 10.1
## 10     7  9.18
## 11     3  9
## 12     8  7.02
```

- **(LC2.24)** We looked at the distribution of the numerical variable temp split by the numerical variable month that we converted using the factor() function in order to make a side-by-side boxplot. Why would a boxplot of temp split by the numerical variable pressure similarly converted to a categorical variable using the factor() not be informative?
  Ans -: Without factoring , it's show error messages such as : Warning messages: *Continuous x aesthetic – did you forget aes(group=...) ?* Removed 1 rows containing non-finite values (stat_boxplot). So , factoring is important. **OR**

Because there are 12 unique values of month yielding only 12 boxes in our boxplot. There are many more unique values of pressure (469 unique values in fact), because values are to the first decimal place. This would lead to 469 boxes, which is too many for people to digest.

- **(LC2.25)** Boxplots provide a simple way to identify outliers. Why may outliers be easier to identify when looking at a boxplot instead of a faceted histogram ?
  Ans -: In a histogram, the bin corresponding to where an outlier lies may not by high enough for us to see. In a boxplot, they are explicitly labelled separately.

3.5 Barplot

The best way to visualize these different counts, also known as frequencies, is with barplots (also called barcharts).

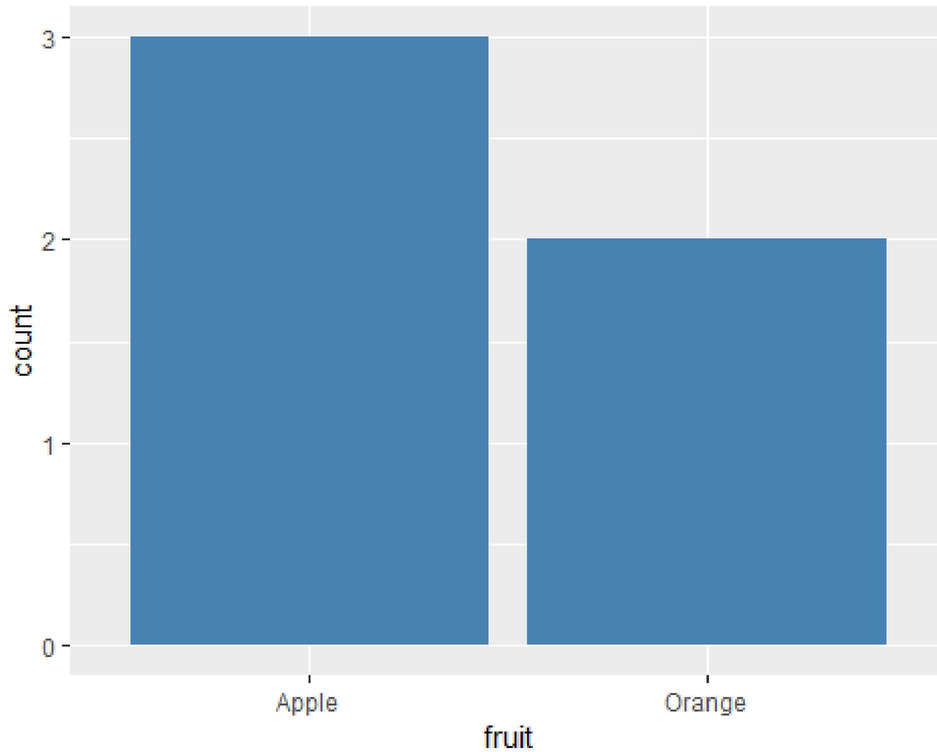
3.5.1 geom_bar() or geom_col()

In *ggplot* for *uncounted data* we use `geom_bar()` and for *counted* we use `geom_col()` .

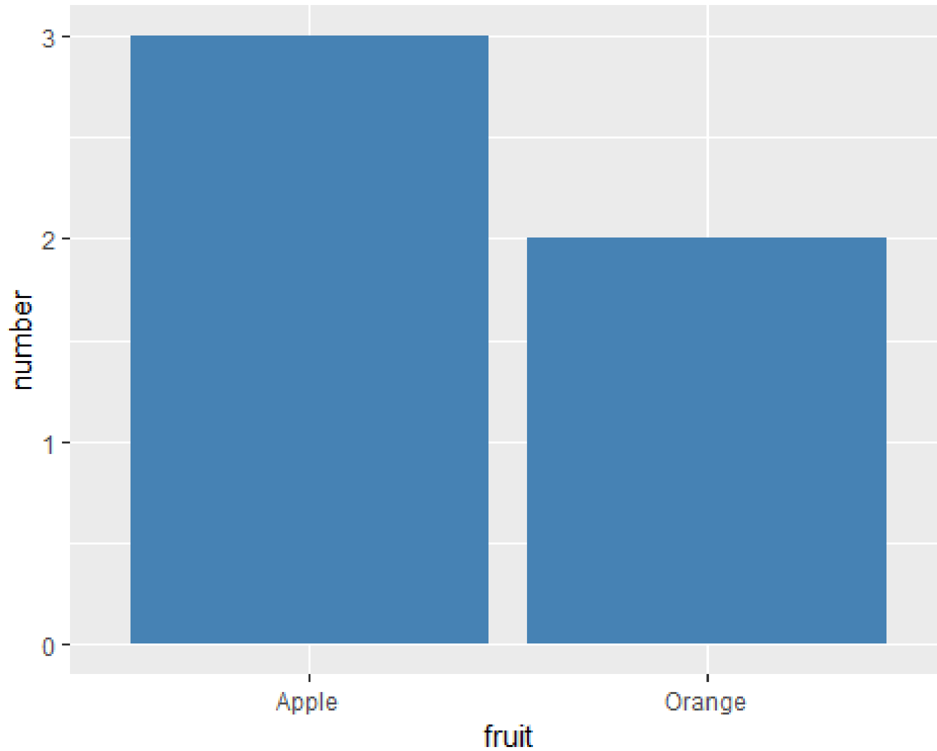
- Syntax : ***ggplot(data , mapping = aes(x)) + geom_bar( )***
- Syntax : ***ggplot(data , mapping = aes(x)) + geom_col( )***
- Creates two data frames representing a collection of fruit: 3 apples and 2 oranges.

```
fruits <- tibble(
  fruit = c("Apple" , "Apple" , "Orange" , "Apple" , "Orange")
)
fruits

## # A tibble: 5 x 1
##   fruit
##   <chr>
## 1 Apple
## 2 Apple
## 3 Orange
## 4 Apple
## 5 Orange

fruits_counted <- tibble(fruit  = c("Apple" , "Orange") ,
                         number = c(3 , 2)
                         )
fruits_counted

## # A tibble: 2 x 2
##   fruit  number
##   <chr>   <dbl>
## 1 Apple       3
## 2 Orange      2
```

```r
# Barplot via geom_bar()
ggplot(fruits , aes(fruit)) + geom_bar(fill = "steelblue")
```

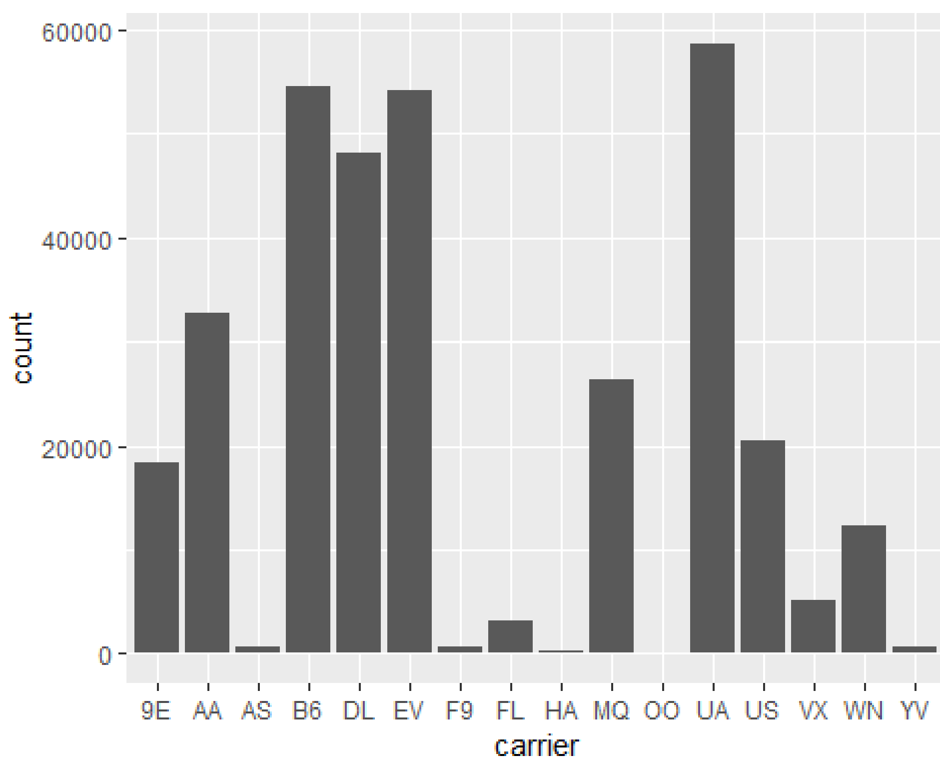



```r
# Barplot via geom_bar()
ggplot(fruits_counted , aes(fruit , number)) + geom_col(fill = "steelblue")
```

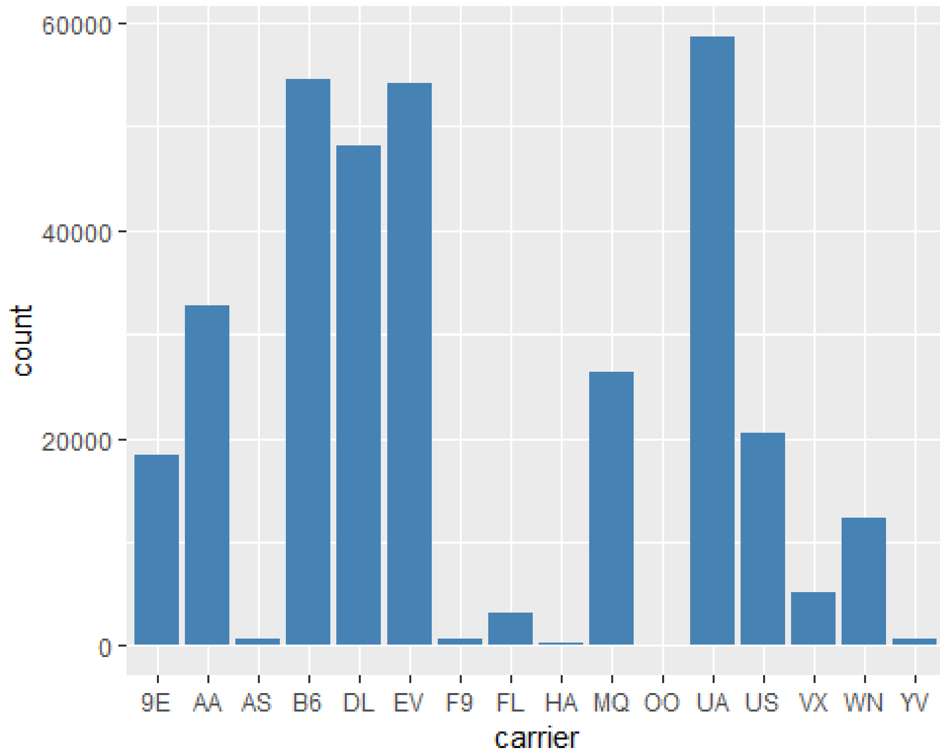


3.5.1.1 Barplot of filght data

```
ggplot(flights , aes(carrier)) + geom_bar()
```



```r
# Barplot with Color
ggplot(flights , aes(carrier))+  geom_bar(fill= "steelblue")
```



- **(LC2.26) -:** Why are histograms inappropriate for categorical variables ?
  Ans -: Histograms are for numerical variables i.e. the horizontal part of each histogram bar represents an interval, whereas for a categorical variable each bar represents only one level of the categorical variable.

- **(LC2.27) -:** What is the difference between histograms and barplots ?
  Ans -: Histograms are for numerical variables i.e. the horizontal part of each histogram bar represents an interval, whereas for a categorical variable each bar represents only one level of the categorical variable.

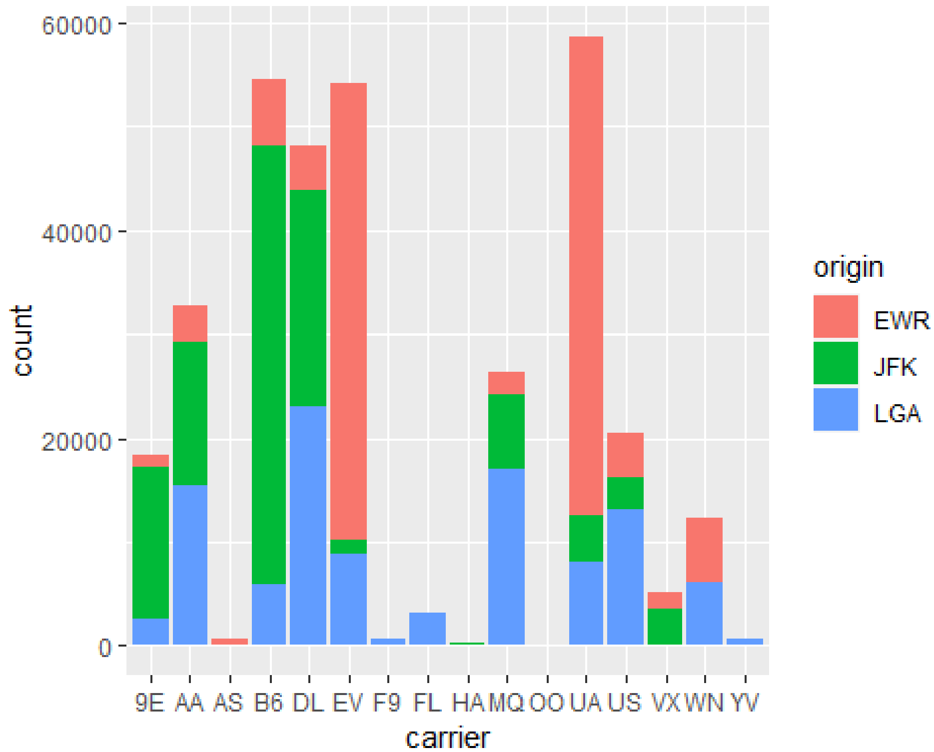
- **(LC2.28) -:** How many Envoy Air flights departed NYC in 2013 ?
  Ans -: Envoy Air is carrier code *MQ* and thus **26397** flights departed NYC in 2013.

- **(LC2.29) -:** What was the 7th highest airline for departed flights from NYC in 2013 ? How could we better present the table to get this answer quickly ?
  Ans -: US the 7th highest airline for departed flights with **20536** from NYC in 2013 . We better present the table to get this answer quickly via `Barplot` .

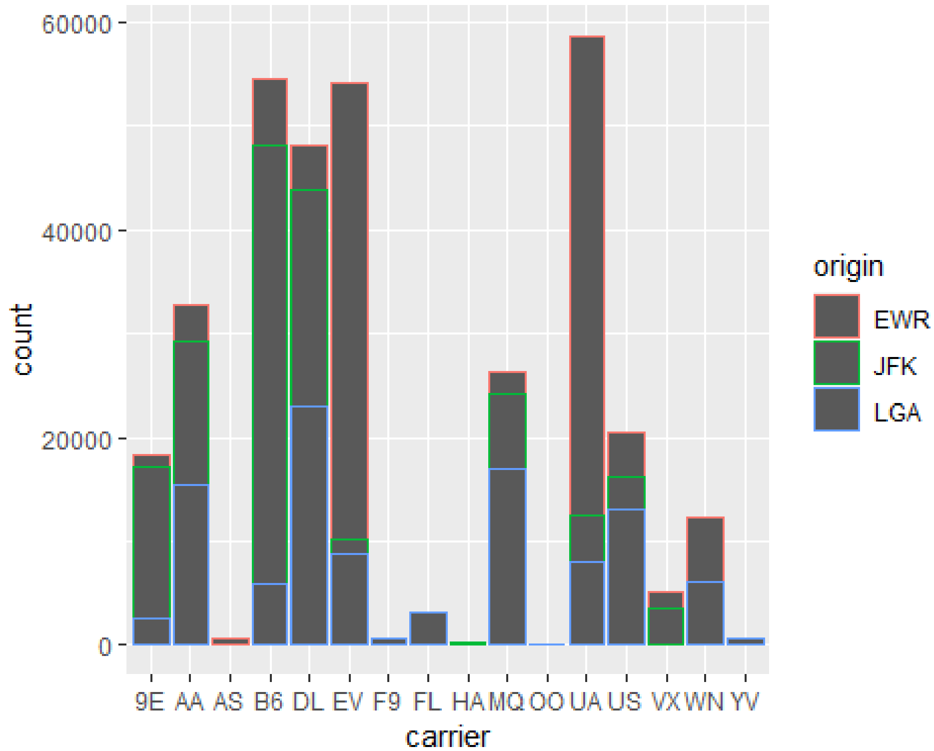
3.5.2 Two Categorical Variables Barplot

Barplots is to visualize the joint distribution of two categorical variables at the same time.
Stacked Barplot

```
# Stacked Barplot
ggplot(flights , aes(carrier , fill = origin)) + geom_bar()
```

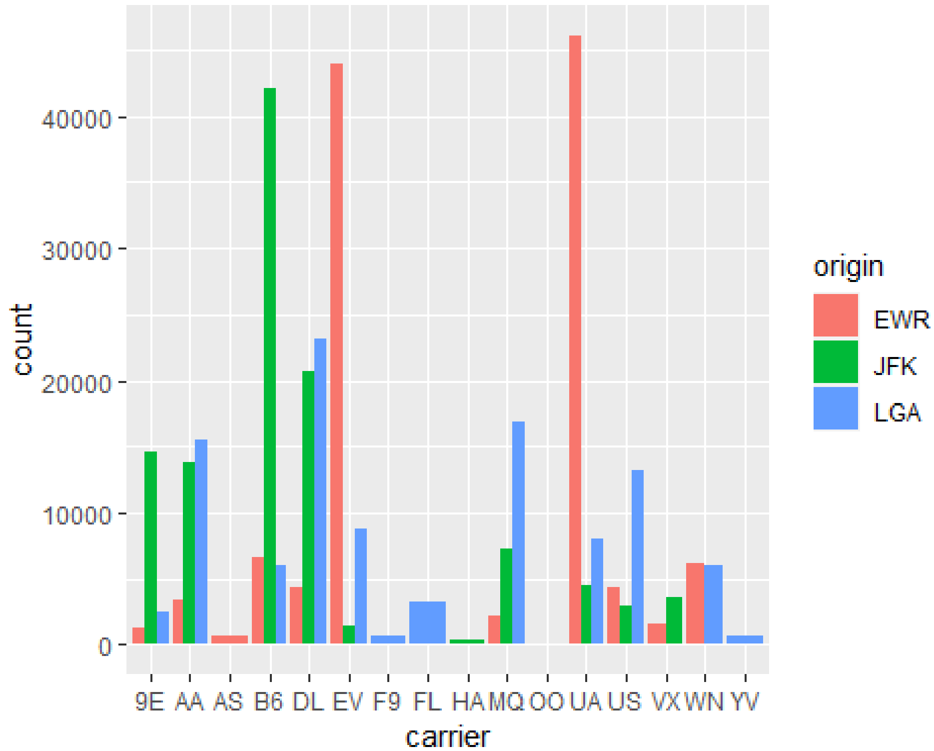



```
# Stack barplot with origin color
ggplot(flights , aes(x= carrier , color = origin)) + geom_bar()
```



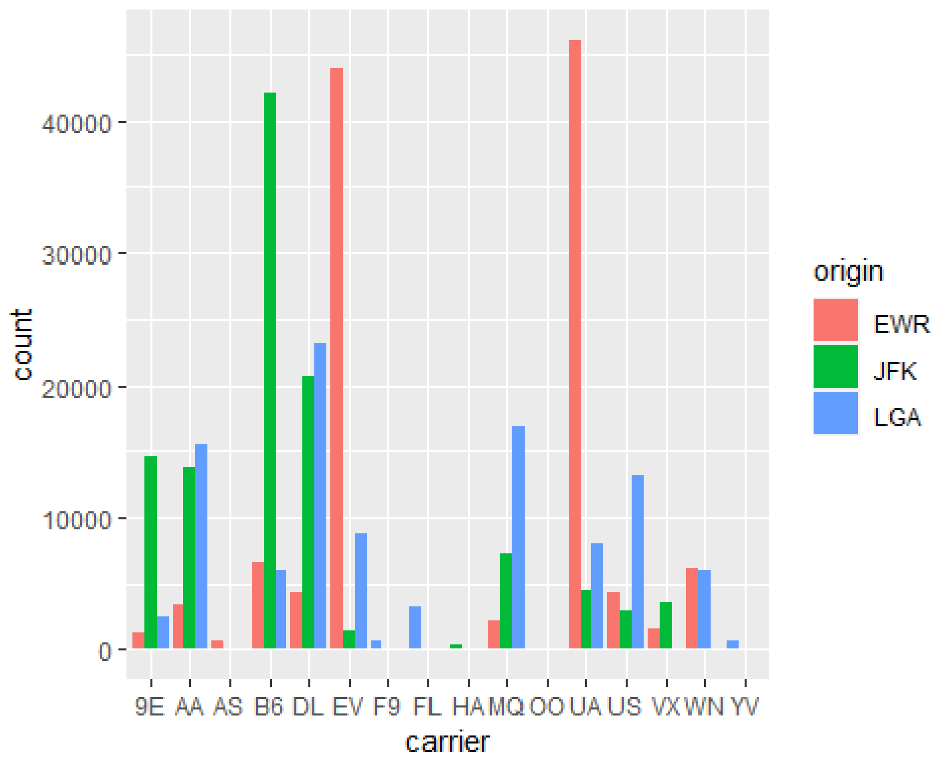
Side by Side Barplot -: To plot a side by side barplot we use **`position = "dodge"`** inside `geom_bar()`.

```
# Side by Side Barplot
ggplot(flights ,  aes(carrier, fill= origin)) + geom_bar(position = "dodge")
```



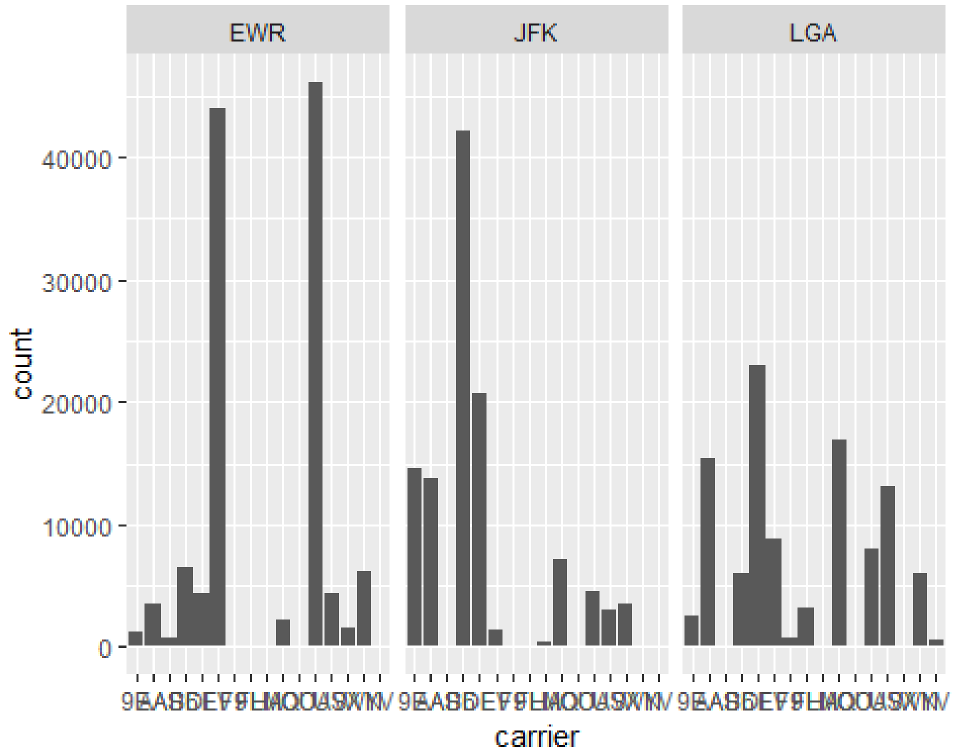
We can make one tweak to the position argument to get them to be the same size in terms of width as the other bars by using the more robust **position_dodge()** function.

```
ggplot(flights , aes(carrier , fill = origin)) + geom_bar(position =
position_dodge(preserve = "single"))
```

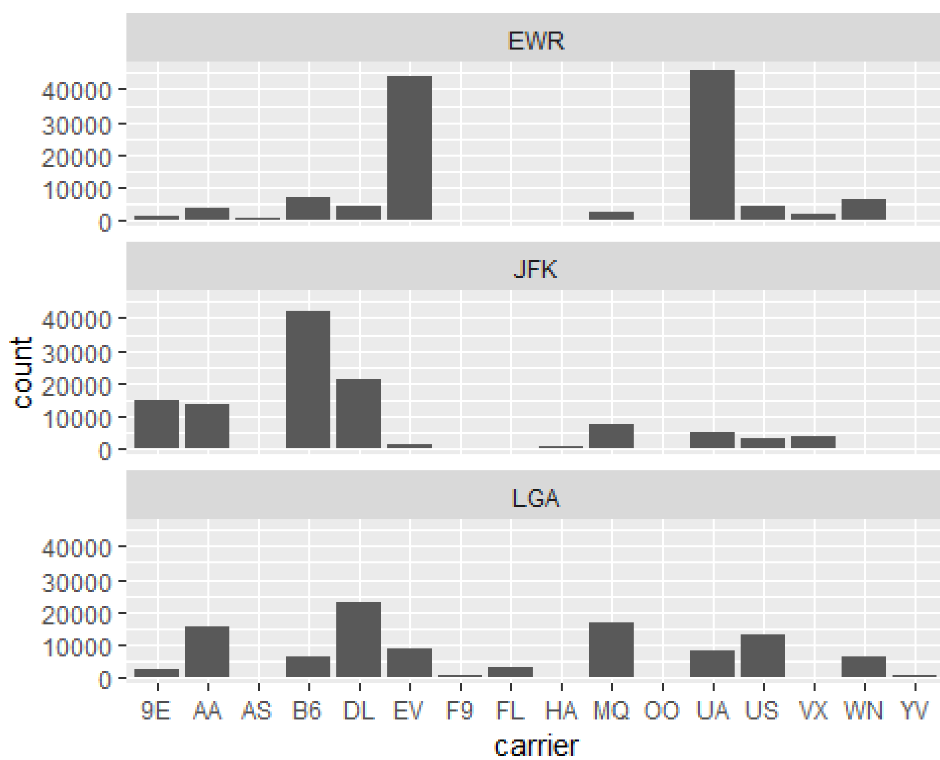


Faceting in Barplot

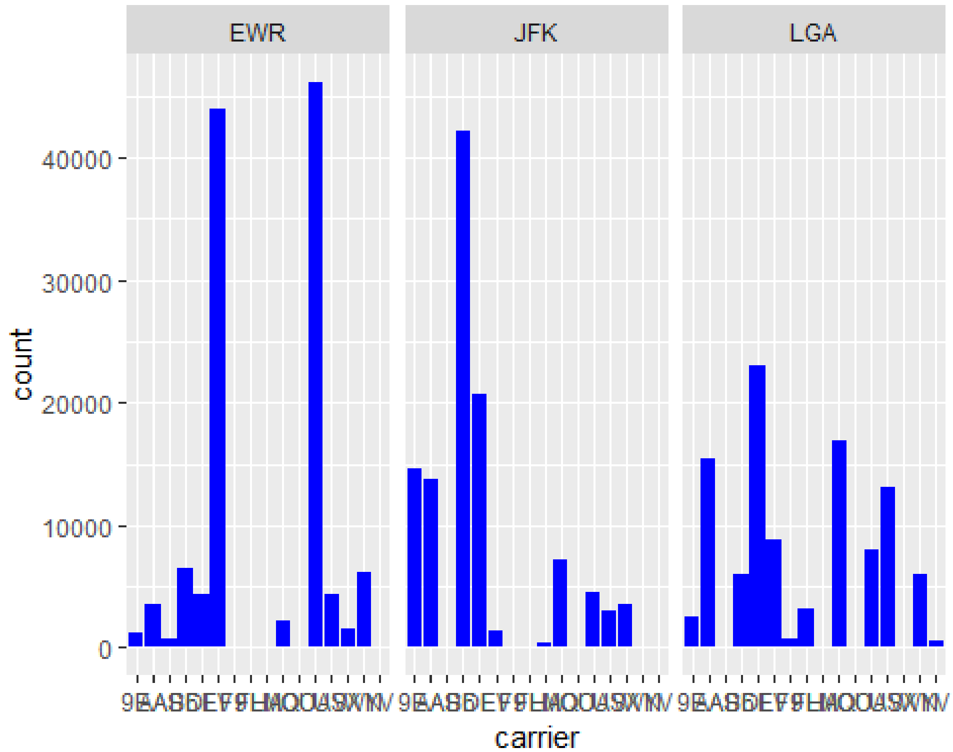
```
p <- ggplot(flights , aes(carrier)) + geom_bar()
p + facet_wrap(~ origin)
```




```r
# Barplot with 1 column
p + facet_wrap(~ origin , ncol = 1)
```



```r
# Barplot with color
ggplot(flights , aes(carrier)) + geom_bar(fill = "blue") + facet_wrap(~
origin)
```



- **(LC2.32) -:** What kinds of questions are not easily answered by looking at barplot ?
Ans -: Because the red, green, and blue bars don't all start at 0 (only red does), it makes comparing counts hard.

- **(LC2.33) -:** What can you say, if anything, about the relationship between airline and airport in NYC in 2013 in regards to the number of departing flights?
Ans -: The different airlines prefer different airports. For example, United is mostly a Newark carrier and JetBlue is a JFK carrier. If airlines didn't prefer airports, each color would be roughly one third of each bar.

- **(LC2.34) -:** Why might the side-by-side barplot be preferable to a stacked barplot in this case ?
Ans -: The side-by-side barplot be preferable to a stacked barplot because it easy to understand .

- **(LC2.35) -** What are the disadvantages of using a dodged barplot, in general ?
Ans -: It is hard to get totals for each airline.

- **(LC2.36) -:** Why is the faceted barplot preferred to the side-by-side and stacked barplots in this case ?
Ans -: Not that different than using side-by-side; depends on how we want to organize our presentation.

- **(LC2.37) -:** What information about the different carriers at different airports is more easily seen in the faceted barplot ?
  Ans -: Now we can also compare the different carriers **within** a particular airport easily too. For example, we can read off who the top carrier for each airport is easily using a single horizontal line.

4 Data Wrangling with `dplyr`

dplyr package is used to *manipulation* of *data* . *dplyr* is similar to *Database Querying Language* **SQL** and pronounced as *Sequel* ar spelled out as S. Q. L. which stands for *Structured Query Language*. `dplyr` package for data wrangling that will allow you to take a data frame and "`wrangle`" it (`transform` it) to suit your needs. Such functions include:
1. **filter()** a data frame's existing rows to only pick out a subset of them.
2. **summarize()** one or more of its columns/variables with a summary statistic.
3. **group_by()** its rows. In other words, assign different rows to be part ofthe same group. We can then combine *group_by()* with *summarize()* to report summary statistics for each group separately.
4. **mutate()** its existing columns/variables to create new ones.
5. **arrange()** its rows. or Ordering the rows .
6. **join()** it with another data frame by matching along a "key" variable. In other words, merge these two data frames together.

- **The Pipe Operator : %>%** The pipe operator *%>%* . The pipe operator allows us to combine multiple operations in R into a single sequential chain of actions. like - *h(g(f(x)))* is same as **x %>% f( ) %>% g( ) %>% h( )**

4.1 `filter()`

- **Task 1 -:** Filter the *Alaska Flights* from *flights* data .

```
# load requie libraries
library(dplyr)
library(nycflights13)

# Extract the AS flights
alaska_flights <- flights %>%
  filter(carrier == "AS")

#View(alaska_flights)
head(alaska_flights , 5)

## # A tibble: 5 x 19
##    year month   day dep_time sched_dep_time dep_delay
##   <int> <int> <int>    <int>          <int>     <dbl>
## 1  2013     1     1      724            725        -1
## 2  2013     1     1     1808           1815        -7
## 3  2013     1     2      722            725        -3
## 4  2013     1     2     1818           1815         3
## 5  2013     1     3      724            725        -1
```

```
## # ... with 13 more variables: arr_time <int>,
## #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

- **Task 2 -:** Filter only on flights from New York City to Portland, Oregon. The *dest* destination code (or airport code) for Portland, Oregon is *"PDX"*.

```
pdx_flights <- flights %>%
  filter(dest == "PDX")

#View(pdx_flights)
head(pdx_flights , 5)

## # A tibble: 5 x 19
##    year month   day dep_time sched_dep_time dep_delay
##   <int> <int> <int>    <int>          <int>     <dbl>
## 1  2013     1     1     1739           1740        -1
## 2  2013     1     1     1805           1757         8
## 3  2013     1     1     2052           2029        23
## 4  2013     1     2      804            805        -1
## 5  2013     1     2     1552           1550         2
## # ... with 13 more variables: arr_time <int>,
## #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

- **Task 3 -:** filter flights for all rows that departed from *JFK* and were heading to Burlington, Vermont (*"BTV"*) or Seattle, Washington (*"SEA"*) and departed in the months of *October, November, or December* .

```
btv_sea_flights_fall <- flights %>%
  filter(origin == "JFK" & (dest == "BTV" | dest == "SEA") & month >= 10)

# View(btv_sea_flights_fall)
head(btv_sea_flights_fall , 5)

## # A tibble: 5 x 19
##    year month   day dep_time sched_dep_time dep_delay
##   <int> <int> <int>    <int>          <int>     <dbl>
## 1  2013    10     1      729            735        -6
## 2  2013    10     1      853            900        -7
## 3  2013    10     1      916            925        -9
## 4  2013    10     1     1216           1221        -5
## 5  2013    10     1     1452           1459        -7
## # ... with 13 more variables: arr_time <int>,
## #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

- **Task 4 -:** filtering rows corresponding to flights that didn't go to Burlington, "BVT" or Seattle, "SEA".

```
not_btv_sea <- flights %>%
  filter(!(dest == "BTV" | dest == "SEA"))

# View(not_btv_sea)
head(not_btv_sea , 5)

## # A tibble: 5 x 19
##    year month   day dep_time sched_dep_time dep_delay
##   <int> <int> <int>    <int>          <int>     <dbl>
## 1  2013     1     1      517            515         2
## 2  2013     1     1      533            529         4
## 3  2013     1     1      542            540         2
## 4  2013     1     1      544            545        -1
## 5  2013     1     1      554            600        -6
## # ... with 13 more variables: arr_time <int>,
## #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Note -: Again, note the careful use of parentheses around the *(dest == "BTV" | dest == "SEA")*. If we didn't use parentheses as follows: **flights %>% filter(!dest == "BTV" | dest == "SEA")** We would be returning all flights not headed to "BTV" or those headed to "SEA", which is an entirely different resulting data frame.

- **Task 5 -:** filter for, say *"SEA", "SFO", "PDX", "BTV", and "BDL"*. We could continue to use the | (or) operator .

```
many_airports <- flights %>%
  filter(dest == "SEA" | dest == "SFO" | dest == "PDX" | dest == "BTV" | dest
== "BDL")

# View(many_airports)
head(many_airports , 5)

## # A tibble: 5 x 19
##    year month   day dep_time sched_dep_time dep_delay
##   <int> <int> <int>    <int>          <int>     <dbl>
## 1  2013     1     1      558            600        -2
## 2  2013     1     1      611            600        11
## 3  2013     1     1      655            700        -5
## 4  2013     1     1      724            725        -1
## 5  2013     1     1      729            730        -1
## # ... with 13 more variables: arr_time <int>,
## #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

%in% Operator As we progressively include more airports, this will get unwieldy to write. A slightly shorter approach uses the **%in%** operator along with the c( ) function "combines" or "concatenates" values into a single vector of values.

```
many_airports_flights <- flights %>%
  filter(dest %in% c("SEA", "SFO", "PDX", "BTV", "BDL"))

View(many_airports_flights)
head(many_airports_flights , 5)

## # A tibble: 5 x 19
##    year month   day dep_time sched_dep_time dep_delay
##   <int> <int> <int>    <int>          <int>     <dbl>
## 1  2013     1     1      558            600        -2
## 2  2013     1     1      611            600        11
## 3  2013     1     1      655            700        -5
## 4  2013     1     1      724            725        -1
## 5  2013     1     1      729            730        -1
## # ... with 13 more variables: arr_time <int>,
## #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

- **Note -:** The %in% operator is useful for looking for matches commonly in one vector/variable compared to another.

- **Note -:** we recommend that filter() should often be among the first verbs you consider applying to your data. This cleans your dataset to only those rows you care about, or put differently, it narrows down the scope of your data frame to just the observations you care about.

4.2 `summarize()`

Fromthe function **summarize()** , we get the statistical functions like - *min()* , *max()* , *mean()* , *sd()* , *sd()* , *etc* .

- **Task 1 -:** Save the results in a new data frame called summary_temp that will have two columns/variables: the *mean* and the *std_dev* from *weather* dataset.

```
data("weather")
# Summarise the data temp column from weather data by mean and standard
deviation
summary_temp <- weather %>%
  summarise(mean = mean(temp) , std = sd(temp))
summary_temp

## # A tibble: 1 x 2
##    mean   std
##   <dbl> <dbl>
## 1    NA    NA
```

It shows *NA* because there are some missing values in *weather* data . So , we use `na.rm = T` command .

```
summary_temp <- weather %>%
  summarise(Mean = mean(temp , na.rm = T) , SD = sd(temp , na.rm = T))
summary_temp

## # A tibble: 1 x 2
##    Mean    SD
##   <dbl> <dbl>
## 1  55.3  17.8

# Round it upto 3 digits
round(summary_temp , 2)

## # A tibble: 1 x 2
##    Mean    SD
##   <dbl> <dbl>
## 1  55.3  17.8
```

- **Task - 2 -:** Use summary by statistica functions like *mean() , sd() , min() , max() , IQR() , sum() , n() . etc.*

```
summ_temp <- weather %>%
  summarise(Mean = mean(temp , na.rm = T) ,
            SD = sd(temp , na.rm = T) ,
            Min = min(temp , na.rm = T) ,
            Max = max(temp , na.rm = T) ,
            IQR = IQR(temp , na.rm = T) ,
            Freq. = n())
summ_temp

## # A tibble: 1 x 6
##    Mean    SD   Min   Max   IQR Freq.
##   <dbl> <dbl> <dbl> <dbl> <dbl> <int>
## 1  55.3  17.8  10.9  100.  30.1 26115
```

- **(LC 3.4) -:**

```
# summary_temp <- weather %>%
#  summarise(Mean = mean(temp , na.rm = T)) %>%
#  summarise(SD = sd(temp , na.rm = T))
# summ_temp
```

The above codes create errors , Because after the first *summarize()*, the variable temp disappears as it has been collapsed to the value *mean*. So when we try to run the second *summarize()*, it can't find the variable *temp* to compute the standard deviation of.

4.3 group_by()

4.3.1 Grouping by One Variable

- **Task - 1 -:** *"grouping"* temperature observations by the values of another variable, in this case by the 12 values of the variable month.

```
summary_montly_temp <- weather %>%
  group_by(month) %>%
  summarise(Mean = mean(temp , na.rm = T),
            SD = sd(temp , na.rm = T) ,
            Min = min(temp , na.rm = T) ,
            Max = max(temp , na.rm = T) ,
            IQR = IQR(temp , na.rm = T) ,
            Freq = n())
summary_montly_temp

## # A tibble: 12 x 7
##    month  Mean    SD   Min   Max   IQR  Freq
##    <int> <dbl> <dbl> <dbl> <dbl> <dbl> <int>
##  1     1  35.6 10.2   10.9  64.4 13.8   2226
##  2     2  34.3  6.98  16.0  55.9 10.1   2010
##  3     3  39.9  6.25  26.1  60.1  9     2227
##  4     4  51.7  8.79  30.9  84.0 12.1   2159
##  5     5  61.8  9.68  13.1  93.0 11.9   2232
##  6     6  72.2  7.55  54.0  93.9 11.0   2160
##  7     7  80.1  7.12  64.0 100.   9.18  2228
##  8     8  74.5  5.19  59    90.0  7.02  2217
##  9     9  67.4  8.47  48.0  95   12.1   2159
## 10    10  60.1  8.85  33.1  89.1 11.0   2212
## 11    11  45.0 10.4   21.0  71.1 16.0   2141
## 12    12  38.4  9.98  18.0  71.6 14.0   2144
```

- **Task - 2-:** Use Diamond data and group by cut column

```
data(diamonds)
dia_cut <- diamonds %>%
  group_by()
dim(dia_cut)

## [1] 53940    10
```

- **Task -3-:** Find the mean of price variable by grouping cut of *diamonda* data .

```
mean_price <- diamonds %>%
  group_by(cut) %>%
  summarise(Mean = mean(price) ,
            Freq. = n())
mean_price

## # A tibble: 5 x 3
##   cut        Mean Freq.
##   <ord>     <dbl> <int>
## 1 Fair      4359.  1610
```

```
## 2 Good      3929.  4906
## 3 Very Good 3982. 12082
## 4 Premium   4584. 13791
## 5 Ideal     3458. 21551
```

- Remove this grouping structure meta-data, we can pipe the resulting data frame into the ungroup() function:

```
# diamonds %>%
# group_by(cut) %>%
# ungroup()
```

- **Task - 4 -:** Count how many flights departed each of the three airports in New York City:

```
dept_airport <- flights %>%
  group_by(origin) %>%
  summarise(Freq = n())

dept_airport

## # A tibble: 3 x 2
##   origin   Freq
##   <chr>   <int>
## 1 EWR    120835
## 2 JFK    111279
## 3 LGA    104662
```

4.3.2 Grouping by More than One Variable

- **Task - 5 -:** We want to know the number of flights leaving each of the three New York City airports for each month .

```
dept_airport_mont <- flights %>%
  group_by(origin , month) %>%
  summarise(Count = n())

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

dept_airport_mont

## # A tibble: 36 x 3
## # Groups:   origin [3]
##    origin month Count
##    <chr>  <int> <int>
##  1 EWR        1  9893
##  2 EWR        2  9107
##  3 EWR        3 10420
##  4 EWR        4 10531
##  5 EWR        5 10592
##  6 EWR        6 10175
##  7 EWR        7 10475
##  8 EWR        8 10359
```

```
##  9 EWR        9  9550
## 10 EWR       10 10104
## # ... with 26 more rows
```

```r
# View(dept_airport_mont)

by_origin_monthly_incorrect <- flights %>%
  group_by(origin) %>%
  group_by(month) %>%
  summarize(count = n())

by_origin_monthly_incorrect
```

```
## # A tibble: 12 x 2
##    month count
##    <int> <int>
##  1     1 27004
##  2     2 24951
##  3     3 28834
##  4     4 28330
##  5     5 28796
##  6     6 28243
##  7     7 29425
##  8     8 29327
##  9     9 27574
## 10    10 28889
## 11    11 27268
## 12    12 28135
```

Here is that the second *group_by(month)* overwrote the grouping structure meta-data of the earlier *group_by(origin)*, so that in the end we are only grouping by month. The lesson here is if you want to *group_by()* two or more variables, we should include all the variables at the same time in the same *group_by()* adding a comma between the variable names.

- **(LC 3.5)-:** Looked at temperatures by months in NYC. What does the standard deviation column in the summary_monthly_temp data frame tell us about temperatures in NYC throughout the year .

```r
count__monthly_temp <- weather %>%
  group_by(month) %>%
  summarise(Count = n())

count__monthly_temp
```

```
## # A tibble: 12 x 2
##    month Count
##    <int> <int>
##  1     1  2226
##  2     2  2010
##  3     3  2227
##  4     4  2159
```

```
## 5       5  2232
## 6       6  2160
## 7       7  2228
## 8       8  2217
## 9       9  2159
## 10     10  2212
## 11     11  2141
## 12     12  2144
```

- **(LC 3.6)-:** Write code would be required to get the mean and standard deviation temperature for each day in 2013 for NYC .

```
sum__monthly_temp <- weather %>%
  group_by(day) %>%
  summarise(Count = n() ,
            Mean = mean(temp , na.rm = T) ,
            SD = sd(temp , na.rm = T))

sum__monthly_temp

## # A tibble: 31 x 4
##       day Count  Mean    SD
##     <int> <int> <dbl> <dbl>
##  1      1   855  57.6  17.4
##  2      2   848  55.7  20.2
##  3      3   864  53.8  18.9
##  4      4   861  54.0  18.8
##  5      5   862  55.6  16.2
##  6      6   863  55.7  15.6
##  7      7   864  55.6  17.4
##  8      8   864  55.0  17.6
##  9      9   864  56.6  17.4
## 10     10   861  56.9  17.8
## # ... with 21 more rows
```

- **(LC 3.7)-:** Recreate by_monthly_origin, but instead of grouping via **group_by(origin, month)**, group variables in a different order **group_by(month, origin).** What differs in the resulting dataset.

```
dept_airport_month <- flights %>%
  group_by(month , origin) %>%
  summarise(Count = n())

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

dept_airport_month

## # A tibble: 36 x 3
## # Groups:   month [12]
##    month origin Count
##    <int> <chr>  <int>
```

```
## 1       1 EWR     9893
## 2       1 JFK     9161
## 3       1 LGA     7950
## 4       2 EWR     9107
## 5       2 JFK     8421
## 6       2 LGA     7423
## 7       3 EWR    10420
## 8       3 JFK     9697
## 9       3 LGA     8717
## 10      4 EWR    10531
## # ... with 26 more rows

# View(dept_airport_month)
```

group_by(month , origin) gives all three *origin ,month-wise* while **group_by(origin , month)** gives all *months , origin-wise .*

- **(LC3.8)-:** How could we identify how many flights left each of the three airports for each carrier?

```
flight_carr <- flights %>%
  group_by(origin , carrier) %>%
  summarise(Freq. = n())

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

flight_carr

## # A tibble: 35 x 3
## # Groups:   origin [3]
##    origin carrier Freq.
##    <chr>  <chr>   <int>
##  1 EWR    9E       1268
##  2 EWR    AA       3487
##  3 EWR    AS        714
##  4 EWR    B6       6557
##  5 EWR    DL       4342
##  6 EWR    EV      43939
##  7 EWR    MQ       2276
##  8 EWR    OO          6
##  9 EWR    UA      46087
## 10 EWR    US       4405
## # ... with 25 more rows
```

- **(LC3.9)-** How does the filter( ) operation differ from a group_by( ) followed by a summarize( ) ?
  Ans : `filter` picks out rows from the original dataset without modifying them, whereas `group_by %>% summarize` computes summaries of numerical variables, and hence reports new values.

4.4 `mutate()`

mutate( ) existing variables By using ***mutate***() we add/create a new variable in our data at our convenience.

- **Task - 1-:** We are more comfortable thinking of temperature in degrees Celsius (°C) instead of degrees Fahrenheit (°F). The formula to convert temperatures from °F to °C is $temp\ in\ C = \frac{temp\_in\_F - 32}{1.8}$

```
# Add a new variable named temp_in C in the weather dataset
weather <- weather %>%
  mutate(temp_in_C = (temp - 32) / 1.8)

# View(weather)
names(weather)

##  [1] "origin"     "year"       "month"      "day"
##  [5] "hour"       "temp"       "dewp"       "humid"
##  [9] "wind_dir"   "wind_speed" "wind_gust"  "precip"
## [13] "pressure"   "visib"      "time_hour"  "temp_in_C"
```

- **Task - 2-:** Compute monthly average/mean of temperatures in both °F and °C .

```
monthly_temp <- weather %>%
  group_by(month) %>%
  summarise(Mean_F = mean(temp , na.rm = T) ,
            Mean_C = mean(temp_in_C, na.rm = T))

# monthly_temp
round(monthly_temp , 2)

## # A tibble: 12 x 3
##    month Mean_F Mean_C
##    <dbl>  <dbl>  <dbl>
##  1     1   35.6   2.02
##  2     2   34.3   1.26
##  3     3   39.9   4.38
##  4     4   51.8  11.0
##  5     5   61.8  16.6
##  6     6   72.2  22.3
##  7     7   80.1  26.7
##  8     8   74.5  23.6
##  9     9   67.4  19.6
## 10    10   60.1  15.6
## 11    11   45.0   7.22
## 12    12   38.4   3.58
```

- **Task - 3-** Make a Variable named gain which is basically the difference between **dep_delay - ar_delay** of *flights* dataset .

```
flights <- flights %>%
  mutate(gain = dep_delay - arr_delay)
```

```r
# View(flights)
names(flights)
```

```
##  [1] "year"           "month"          "day"
##  [4] "dep_time"       "sched_dep_time" "dep_delay"
##  [7] "arr_time"       "sched_arr_time" "arr_delay"
## [10] "carrier"        "flight"         "tailnum"
## [13] "origin"         "dest"           "air_time"
## [16] "distance"       "hour"           "minute"
## [19] "time_hour"      "gain"
```

- **Task - 4-** Some summary statistics of the gain variable by considering multiple summary functions at once in the same summarize() .

```r
gain_summary <- flights %>%
  summarise(
    Min = min(gain , na.rm = T) ,
    Q1 = quantile(gain , 0.25 , na.rm = T),
    Q2_Median= quantile(gain , 0.50 , na.rm = T) ,
    Q3 = quantile(gain , 0.75 , na.rm = T) ,
    Max = max(gain , na.rm = T) ,
    Mean = mean(gain , na.rm = T) ,
    SD = sd(gain , na.rm = T) ,
    Missing = sum(is.na(gain)) ,
  )

gain_summary
```

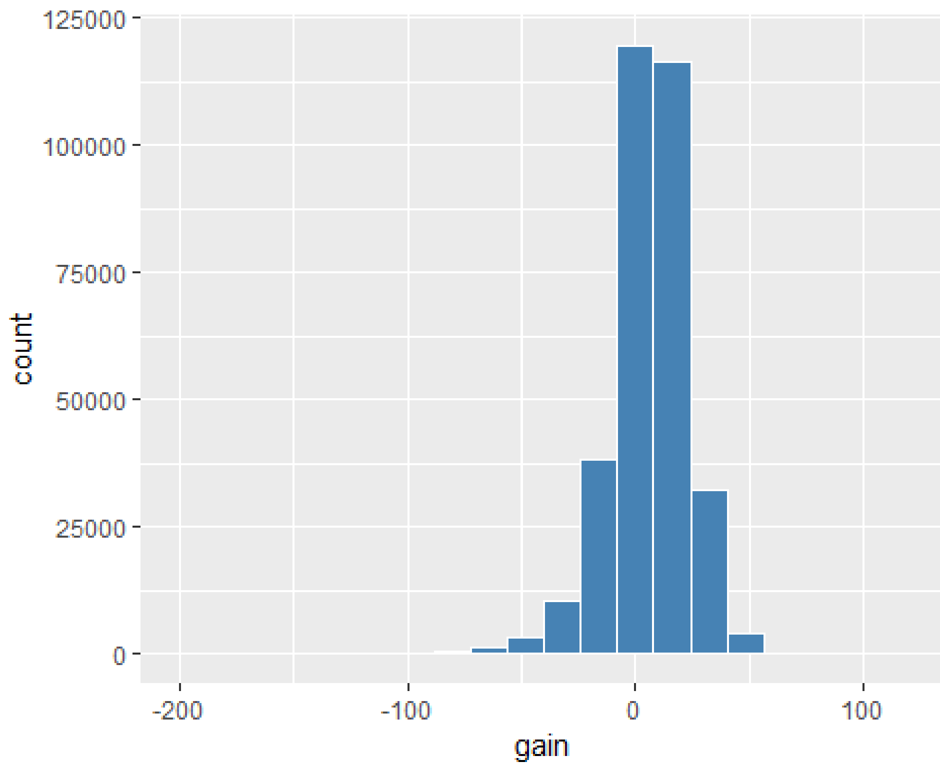
```
## # A tibble: 1 x 8
##     Min    Q1 Q2_Median    Q3   Max  Mean    SD Missing
##   <dbl> <dbl>     <dbl> <dbl> <dbl> <dbl> <dbl>   <int>
## 1  -196    -3         7    17   109  5.66  18.0    9430
```

- **(LC3.12)** Describe it in a few sentences using the plot and the gain_summary data frame values.
  Ans- There are 9430 observations in thegain variable . The minimum and maximum value of gain are −1.96 and 109. The mean of gain is 5.66 and the standard deviation is 18.04 . 7 is the value that divides the gain to equal parts . 75% of gain data is below 17 and rest 25% of data is above 17 .

- **Task - 5-** gain is a numerical variable, we can visualize its distribution using a histogram.

```r
ggplot(flights ,aes(gain)) + geom_histogram(col = "white" , fill =
"steelblue" , bins = 20)
```

```
## Warning: Removed 9430 rows containing non-finite values
## (stat_bin).
```



- **Task - 6-** Add new variables in flights data as *gain = dep_delay - arr_delay , hours = air_time/60 , gain_per_hour = gain / hours* and also rund upto 2 decimal places .

```
flights <- flights %>%
  mutate(
    gain = dep_delay - arr_delay ,
    hours = round(air_time / 60 , 2) ,
    gain_per_hour = round(gain / hours ,2)
  )

# View(flights)
names(flights)

##  [1] "year"           "month"          "day"
##  [4] "dep_time"       "sched_dep_time" "dep_delay"
##  [7] "arr_time"       "sched_arr_time" "arr_delay"
## [10] "carrier"        "flight"         "tailnum"
## [13] "origin"         "dest"           "air_time"
## [16] "distance"       "hour"           "minute"
## [19] "time_hour"      "gain"           "hours"
## [22] "gain_per_hour"
```

- **(LC 3.10)-** What do positive values of the gain variable in flights correspond to ? What about negative values? And what about a zero value ?
  Ans - Negative Values tell that there a delay or flight is late . Negative Values tells that there is no delay or flight is on it's exact time . Positive Values tells that flights are arrived before time.(On book's page - 82)

- **(LC 3.11)-** Could we create the dep_delay and arr_delay columns by simply subtracting dep_time from sched_dep_time and similarly for arrivals? Try the code out and explain any differences between the result and what actually appears in flights.

```
flights <- flights %>%
  mutate(
    dept = dep_time - sched_dep_time ,
    arr = arr_time - sched_arr_time
  )

# View(flights)
names(flights)

##  [1] "year"           "month"          "day"
##  [4] "dep_time"       "sched_dep_time" "dep_delay"
##  [7] "arr_time"       "sched_arr_time" "arr_delay"
## [10] "carrier"        "flight"         "tailnum"
## [13] "origin"         "dest"           "air_time"
## [16] "distance"       "hour"           "minute"
## [19] "time_hour"      "gain"           "hours"
## [22] "gain_per_hour"  "dept"           "arr"
```

4.5 `arrange()` and `sort()` rows

arrange() function allows us to sort/reorder a data frame's rows according to the values of the specified variable.

- **Task - 1-:** We are interested in determining the most frequent destination airports for all domestic flights departing from New York City in 2013:

```
freq_dest <- flights %>%
  group_by(dest) %>%
  summarise(freq_flights = n())

# View(freq_dest)
```

- **Task - 2-:** Sorted from the most to the least number of flights (freq_flights) instead

```
# Arrange in Ascending Order
asc_freq_dest <- freq_dest %>%
  arrange(freq_flights)

# We get 105 x 2 Mtx. so
head(asc_freq_dest)

## # A tibble: 6 x 2
##   dest  freq_flights
##   <chr>        <int>
## 1 LEX              1
## 2 LGA              1
## 3 ANC              8
```

```
## 4 SBN              10
## 5 HDN              15
## 6 MTJ              15

# Arrange in Descending Order
desc_freq_flights <- freq_dest %>%
  arrange(desc(freq_flights))

# We get 105 x 2 Mtx. so
head(desc_freq_flights)

## # A tibble: 6 x 2
##   dest  freq_flights
##   <chr>        <int>
## 1 ORD          17283
## 2 ATL          17215
## 3 LAX          16174
## 4 BOS          15508
## 5 MCO          14082
## 6 CLT          14064
```

4.6 join data frame

"joining" or "merging" two different datasets . A key variable to match the rows of the two data frames. Key variables are almost always identification variables that uniquely identify the observational units .

4.6.1 Matching "key" variable names

4.6.1.1 inner_join()

We **use *inner_join()*** function to join the two data frames
Task - 1- Join the *flights* and *airlines* data frames , the *key* variable we want to *join/merge/match* the rows by has the same name **carrier** .

```
dim(flights)

## [1] 336776     24

dim(airlines)

## [1] 16  2

flights_joined <- flights %>%
  inner_join(airlines , by = "carrier")

# View(flights_joined)
dim(flights_joined)

## [1] 336776     25
```

4.6.2 Multiple "key" Varables

- **Task- 2-** Join the flights and weather data frames, we need more than one key variable: *year, month, day, hour, and origin.* This is because the combination of these 5 variables act to uniquely identify each observational unit in the weather data frame: hourly weather recordings at each of the 3 NYC airport

```
dim(flights)

## [1] 336776     24

names(flights)

##  [1] "year"           "month"          "day"
##  [4] "dep_time"       "sched_dep_time" "dep_delay"
##  [7] "arr_time"       "sched_arr_time" "arr_delay"
## [10] "carrier"        "flight"         "tailnum"
## [13] "origin"         "dest"           "air_time"
## [16] "distance"       "hour"           "minute"
## [19] "time_hour"      "gain"           "hours"
## [22] "gain_per_hour"  "dept"           "arr"

dim(weather)

## [1] 26115    16

names(weather)

##  [1] "origin"     "year"       "month"      "day"
##  [5] "hour"       "temp"       "dewp"       "humid"
##  [9] "wind_dir"   "wind_speed" "wind_gust"  "precip"
## [13] "pressure"   "visib"      "time_hour"  "temp_in_C"

flights_weather_join <- flights %>%
  inner_join(weather , by = c("year" , "month" , "day" , "hour" , "origin"))

# View(flights_weather_join)

dim(flights_weather_join)

## [1] 335220     35

names(flights_weather_join)

##  [1] "year"           "month"          "day"
##  [4] "dep_time"       "sched_dep_time" "dep_delay"
##  [7] "arr_time"       "sched_arr_time" "arr_delay"
## [10] "carrier"        "flight"         "tailnum"
## [13] "origin"         "dest"           "air_time"
## [16] "distance"       "hour"           "minute"
## [19] "time_hour.x"    "gain"           "hours"
## [22] "gain_per_hour"  "dept"           "arr"
## [25] "temp"           "dewp"           "humid"
```

```
## [28] "wind_dir"        "wind_speed"      "wind_gust"
## [31] "precip"          "pressure"        "visib"
## [34] "time_hour.y"     "temp_in_C"
```

Note - The common/key variables are count once in dimensions.

- **Task - 3-:** In airports the airport code is in faa, whereas in flights the airport codes are in origin and dest.

```
flights_with_airport_names <- flights %>%
inner_join(airports, by = c("dest" = "faa"))
#View(flights_with_airport_names)

#Let us see with details and rename
named_dests <- flights %>%
  group_by(dest) %>%
  summarize(num_flights = n()) %>%
  arrange(desc(num_flights)) %>%
  inner_join(airports, by = c("dest" = "faa")) %>%
  rename(airport_name = name)
# View(named_dests)
head(named_dests)

## # A tibble: 6 x 9
##   dest  num_flights airport_name       lat    lon   alt    tz
##   <chr>       <int> <chr>            <dbl>  <dbl> <dbl> <dbl>
## 1 ORD         17283 Chicago Ohare I~  42.0  -87.9   668    -6
## 2 ATL         17215 Hartsfield Jack~  33.6  -84.4  1026    -5
## 3 LAX         16174 Los Angeles Intl  33.9 -118.    126    -8
## 4 BOS         15508 General Edward ~  42.4  -71.0    19    -5
## 5 MCO         14082 Orlando Intl      28.4  -81.3    96    -5
## 6 CLT         14064 Charlotte Dougl~  35.2  -80.9   748    -5
## # ... with 2 more variables: dst <chr>, tzone <chr>
```

4.6.3 Normal Forms

The process of decomposing data frames into less redundant tables without losing information is called normalization.

- **Task - 1-:** The names of the airline companies are included in the name variable of the airlines data frame. In order to have the airline company name included in flights.

```
joined_flights <- flights %>%
  inner_join(airlines , by = "carrier")

# View(joined_flights)
head(joined_flights)

## # A tibble: 6 x 25
##    year month   day dep_time sched_dep_time dep_delay
##   <int> <int> <int>    <int>          <int>     <dbl>
```

```
## 1  2013     1     1      517            515         2
## 2  2013     1     1      533            529         4
## 3  2013     1     1      542            540         2
## 4  2013     1     1      544            545        -1
## 5  2013     1     1      554            600        -6
## 6  2013     1     1      554            558        -4
## # ... with 19 more variables: arr_time <int>,
## #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, gain <dbl>, hours <dbl>,
## #   gain_per_hour <dbl>, dept <int>, arr <int>, name <chr>
```

4.7 Other Verbs

4.7.1 select()

select() only a subset to variables / columns .

- **Task - 1-:** variables the two named *carrier* and *flight* from *flights* data .

```
s_c_f <- flights %>%
  select(carrier , flight)

# View(s_c_f)
head(s_c_f)

## # A tibble: 6 x 2
##   carrier flight
##   <chr>    <int>
## 1 UA        1545
## 2 UA        1714
## 3 AA        1141
## 4 B6         725
## 5 DL         461
## 6 UA        1696
```

- **Task - 2-:** Drop or De-select the *year* column from flights data .

```
no_year <- flights %>%
  select(-year)

# View(no_year)
names(no_year)

##  [1] "month"          "day"            "dep_time"
##  [4] "sched_dep_time" "dep_delay"      "arr_time"
##  [7] "sched_arr_time" "arr_delay"      "carrier"
## [10] "flight"         "tailnum"        "origin"
## [13] "dest"           "air_time"       "distance"
## [16] "hour"           "minute"         "time_hour"
```

```
## [19] "gain"           "hours"          "gain_per_hour"
## [22] "dept"           "arr"
```

- **Task- 3-:** Selecting columns/variables is by specifying a range of columns.

```
select_range <- flights %>%
  select(month : day , arr_time : sched_arr_time)

# View(select_range)
glimpse(select_range)

## Rows: 336,776
## Columns: 4
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, ~
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, ~
```

The above code select() all columns between month and day, as well as between arr_time and sched_arr_time, and drop the rest.

- **Note- The `select()`** function can also be used to reorder columns when used with the **`everything()`** helper function.

- **Task- 4-:** We want the hour, minute, and time_hour variables to appear immediately after the year, month, and day variables, while not discarding the rest of the variables.

```
flights_recorder <- flights %>%
  select(year , month , day , hour , minute , time_hour , everything())

# View(flights_recorder)
names(flights_recorder)

##  [1] "year"           "month"          "day"
##  [4] "hour"           "minute"         "time_hour"
##  [7] "dep_time"       "sched_dep_time" "dep_delay"
## [10] "arr_time"       "sched_arr_time" "arr_delay"
## [13] "carrier"        "flight"         "tailnum"
## [16] "origin"         "dest"           "air_time"
## [19] "distance"       "gain"           "hours"
## [22] "gain_per_hour"  "dept"           "arr"

glimpse(flights_recorder)

## Rows: 336,776
## Columns: 24
## $ year           <int> 2013, 2013, 2013, 2013, 2013, 2013, ~
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, ~
## $ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0~
```

```
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-01-01 05:~
## $ dep_time       <int> 517, 533, 542, 544, 554, 554, 555, 5~
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 6~
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2,~
## $ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, ~
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, ~
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -~
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA", ~
## $ flight         <int> 1545, 1714, 1141, 725, 461, 1696, 50~
## $ tailnum        <chr> "N14228", "N24211", "N619AA", "N804J~
## $ origin         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "~
## $ dest           <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "~
## $ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158, 5~
## $ distance       <dbl> 1400, 1416, 1089, 1576, 762, 719, 10~
## $ gain           <dbl> -9, -16, -31, 17, 19, -16, -24, 11, ~
## $ hours          <dbl> 3.78, 3.78, 2.67, 3.05, 1.93, 2.50, ~
## $ gain_per_hour  <dbl> -2.38, -4.23, -11.61, 5.57, 9.84, -6~
## $ dept           <int> 2, 4, 2, -1, -46, -4, -45, -43, -43,~
## $ arr            <int> 11, 20, 73, -18, -25, 12, 59, -14, -~
```

Note - The helper functions ***starts_with(), ends_with()*** and ***contains()*** can be used to select variables/columns that match those conditions.

4.7.1.1 starts_with()

starts_with(a) returns the columns which starts with letter **"a"**.

```
start_flight <- flights %>%
  select(starts_with("a"))

# View(start_flight)
names(start_flight)

## [1] "arr_time"  "arr_delay" "air_time"  "arr"
```

4.7.1.2 ends_with()

ends_with(a) returns the columns which ends with letter **"a"**.

```
ends_flights <- flights %>%
  select(ends_with("delay"))

# View(ends_flights)
names(ends_flights)

## [1] "dep_delay" "arr_delay"
```

4.7.1.3 contains()

contains(a) returns the columns which contains letter **"a"**.

```r
contain_flights <- flights %>%
  select(contains("time"))

# View(contain_flights)
names(contain_flights)

## [1] "dep_time"       "sched_dep_time" "arr_time"
## [4] "sched_arr_time" "air_time"       "time_hour"
```

4.7.2 rename()

By **rename( )** command , we can change the name of variable .

- **Task - 1-** We want to only focus on *dep_time* and *arr_time* and *change* dep_time and arr_time to be *departure_time* and *arrival_time* instead in the flights_time data frame.

```r
flights_time_new <- flights %>%
  select(dep_time , arr_time) %>%
  rename(departure_time = dep_time , arrival_time = arr_time)

# View(flights_time_new)
names(flights_time_new)

## [1] "departure_time" "arrival_time"
```

4.7.3 top_n()

top_n() returns the Top n values of a variable. *Syntax -:* $top_n(n = ?, wt = col)$ n: is the number . wt: is the column name which we want.

- **Task -1-** Find the top 10 destination airport of flights data.

```r
top_dest <- named_dests %>%
  top_n(n = 10 , wt = num_flights)

top_dest

## # A tibble: 10 x 9
##    dest  num_flights airport_name      lat    lon   alt    tz
##    <chr>       <int> <chr>            <dbl>  <dbl> <dbl> <dbl>
##  1 ORD         17283 Chicago Ohare ~   42.0  -87.9   668    -6
##  2 ATL         17215 Hartsfield Jac~   33.6  -84.4  1026    -5
##  3 LAX         16174 Los Angeles In~   33.9 -118.    126    -8
##  4 BOS         15508 General Edward~   42.4  -71.0    19    -5
##  5 MCO         14082 Orlando Intl      28.4  -81.3    96    -5
##  6 CLT         14064 Charlotte Doug~   35.2  -80.9   748    -5
##  7 SFO         13331 San Francisco ~   37.6 -122.     13    -8
##  8 FLL         12055 Fort Lauderdal~   26.1  -80.2     9    -5
##  9 MIA         11728 Miami Intl        25.8  -80.3     8    -5
## 10 DCA          9705 Ronald Reagan ~   38.9  -77.0    15    -5
## # ... with 2 more variables: dst <chr>, tzone <chr>
```

- **Task - 2-** Arrange the above result .

```
# Arrange in Ascending Order
named_dests %>%
  top_n(n = 5 , wt = num_flights) %>%
  arrange(num_flights)

## # A tibble: 5 x 9
##   dest  num_flights airport_name        lat    lon   alt    tz
##   <chr>       <int> <chr>             <dbl>  <dbl> <dbl> <dbl>
## 1 MCO         14082 Orlando Intl       28.4  -81.3    96    -5
## 2 BOS         15508 General Edward ~   42.4  -71.0    19    -5
## 3 LAX         16174 Los Angeles Intl   33.9 -118.    126    -8
## 4 ATL         17215 Hartsfield Jack~   33.6  -84.4  1026    -5
## 5 ORD         17283 Chicago Ohare I~   42.0  -87.9   668    -6
## # ... with 2 more variables: dst <chr>, tzone <chr>

# Arrange in Descending Order
named_dests %>%
  top_n(n = 5 , wt = num_flights) %>%
  arrange(desc(num_flights))

## # A tibble: 5 x 9
##   dest  num_flights airport_name        lat    lon   alt    tz
##   <chr>       <int> <chr>             <dbl>  <dbl> <dbl> <dbl>
## 1 ORD         17283 Chicago Ohare I~   42.0  -87.9   668    -6
## 2 ATL         17215 Hartsfield Jack~   33.6  -84.4  1026    -5
## 3 LAX         16174 Los Angeles Intl   33.9 -118.    126    -8
## 4 BOS         15508 General Edward ~   42.4  -71.0    19    -5
## 5 MCO         14082 Orlando Intl       28.4  -81.3    96    -5
## # ... with 2 more variables: dst <chr>, tzone <chr>
```

- **(LC3.19)-** Create a new data frame that shows the top 5 airports with the largest arrival delays from NYC in 2013.

```
flights %>%
  top_n(n = 5 , wt = arr_delay)

## # A tibble: 5 x 24
##    year month   day dep_time sched_dep_time dep_delay
##   <int> <int> <int>    <int>          <int>     <dbl>
## 1  2013     1     9      641            900      1301
## 2  2013     1    10     1121           1635      1126
## 3  2013     6    15     1432           1935      1137
## 4  2013     7    22      845           1600      1005
## 5  2013     9    20     1139           1845      1014
## # ... with 18 more variables: arr_time <int>,
## #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, gain <dbl>, hours <dbl>,
## #   gain_per_hour <dbl>, dept <int>, arr <int>
```

4.8 Summary of the verbs in dplyr

Verb	Data wrangling operation
`filter()`	Pick out a subset of rows
`summarize()`	Summarize many values to one using a summary statistic function like mean(), median(), etc.
`group_by()`	Add grouping structure to rows in data frame. Note this does not change values in data frame, rather only the meta-data
`mutate()`	Create new variables by mutating existing ones
`arrange()`	Arrange rows of a data variable in ascending (default) or descending order
`inner_join()`	Join/merge two data frames, matching rows by a key variable
`select()`	Subset of Variables / columns
`select(starts_with(a))`	Subset of columns which stats with "a"
`select(ends_with(a))`	Subset of columns which ends with "a"
`select(contains(a))`	Subset of columns which contains "a"
`rename()`	Rename the column name
`top_n(n , wt)`	Top n obs. of wt column

5 Data Importing & Tidy Data

5.1 Importing Data

- Comma Seperated Values **.csv** Excel Spreadsheet **.xlsx** Google sheet

5.1.1 Using the console

- The .csv file dem_score.csv contains ratings of the level of democracy in different countries spanning 1952 to 1992 and is accessible at https://moderndive.com/data/dem_score.csv. Make sure that we must connect with *Internet*

```
# Load Require Packages
library(dplyr)
library(ggplot2)
library(readr)
library(tidyr)
library(nycflights13)
library(fivethirtyeight)

# library(readr)
# Load the .csv file from internet
dem_score <- read_csv("https://moderndive.com/data/dem_score.csv")

## Rows: 96 Columns: 10
```

```
## -- Column specification ------------------------------------
## Delimiter: ","
## chr (1): country
## dbl (9): 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, ...

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this
message.

head(dem_score)

## # A tibble: 6 x 10
##   country    `1952` `1957` `1962` `1967` `1972` `1977` `1982`
##   <chr>       <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 Albania        -9     -9     -9     -9     -9     -9     -9
## 2 Argentina      -9     -1     -1     -9     -9     -9     -8
## 3 Armenia        -9     -7     -7     -7     -7     -7     -7
## 4 Australia      10     10     10     10     10     10     10
## 5 Austria        10     10     10     10     10     10     10
## 6 Azerbaijan     -9     -7     -7     -7     -7     -7     -7
## # ... with 2 more variables: 1987 <dbl>, 1992 <dbl>
```

5.1.2 Using RStudio's Interface

- **Read .xlsx file**

```
library(readxl)
data <- read_excel("C:/Users/ACER/Downloads/DP-
02/Project/Merged_Resolution_Time.xlsx")

## New names:
## * `` -> ...1

head(data)

## # A tibble: 6 x 6
##    ...1 SL_No Volume Team_Experience Domain_Expertise
##   <dbl> <dbl>  <dbl>           <dbl>            <dbl>
## 1     0     1     69              20               15
## 2     1     2     84              25               20
## 3     2     3     72              21               15
## 4     3     4     79              23               18
## 5     4     5     20              13                9
## 6     5     6     NA              20               15
## # ... with 1 more variable: Resolution_Time <dbl>
```

5.2 Tidy Data

A data with the following features 1. Each variables forms a row . 2. Each observation forms a row . 3. Each type of observational unit forms a table .

```r
# library(fivethirtyeight)
dim(drinks)

## [1] 193   5

names(drinks)

## [1] "country"
## [2] "beer_servings"
## [3] "spirit_servings"
## [4] "wine_servings"
## [5] "total_litres_of_pure_alcohol"

head(drinks)

## # A tibble: 6 x 5
##   country           beer_servings spirit_servings wine_servings
##   <chr>                     <int>           <int>         <int>
## 1 Afghanistan                   0               0             0
## 2 Albania                      89             132            54
## 3 Algeria                      25               0            14
## 4 Andorra                     245             138           312
## 5 Angola                      217              57            45
## 6 Antigua & Barbuda           102             128            45
## # ... with 1 more variable:
## #   total_litres_of_pure_alcohol <dbl>

drinks_smaller <- drinks %>%
  filter(country %in% c("USA", "China", "Italy", "Saudi Arabia")) %>%
  select(-total_litres_of_pure_alcohol) %>%
  rename(beer = beer_servings, spirit = spirit_servings, wine =
wine_servings)

drinks_smaller # a tibble of 4*4

## # A tibble: 4 x 4
##   country       beer spirit  wine
##   <chr>        <int>  <int> <int>
## 1 China           79    192     8
## 2 Italy           85     42   237
## 3 Saudi Arabia     0      5     0
## 4 USA            249    158    84
```

5.2.1 Converting to 'Tidy' data

If we original data frame is in wide (non-"tidy") format and you would like to use the ggplot2 or dplyr packages, we will first have to convert it to "tidy" format. To do so, we recommend using the **`pivot_longer()`** function in the *tidyr* package

- We convert it to "tidy" format by using the pivot_longer() function .

```
drinks_smaller_tidy <- drinks_smaller %>%
  pivot_longer(names_to = "type" ,
               values_to = "serving" ,
               cols = -country)

drinks_smaller_tidy # tibble 12 x 3

## # A tibble: 12 x 3
##    country      type    serving
##    <chr>        <chr>     <int>
##  1 China        beer         79
##  2 China        spirit      192
##  3 China        wine          8
##  4 Italy        beer         85
##  5 Italy        spirit       42
##  6 Italy        wine        237
##  7 Saudi Arabia beer          0
##  8 Saudi Arabia spirit        5
##  9 Saudi Arabia wine          0
## 10 USA          beer        249
## 11 USA          spirit      158
## 12 USA          wine         84
```

We set the arguments to *pivot_longer()* as follows:

1. names_to here corresponds to the name of the variable in the new “tidy”/long data frame that will contain the column names of the original data. Observe how we set names_to = “type”. In the resulting drinks_smaller_tidy, the column type contains the three types of alcohol beer, spirit, and wine. Since type is a variable name that doesn’t appear in drinks_smaller, we use quotation marks around it. You’ll receive an error if you just use names_to = type here.

2. values_to here is the name of the variable in the new “tidy” data frame that will contain the values of the original data. Observe how we set values_to = “servings” since each of the numeric values in each of the beer, wine, and spirit columns of the drinks_smaller data corresponds to a value of servings. In the resulting drinks_smaller_tidy, the column servings contains the 4 × 3 = 12 numerical values. Note again that servings doesn’t appear as a variable in drinks_smaller so it again needs quotation marks around it for the values_to argument.

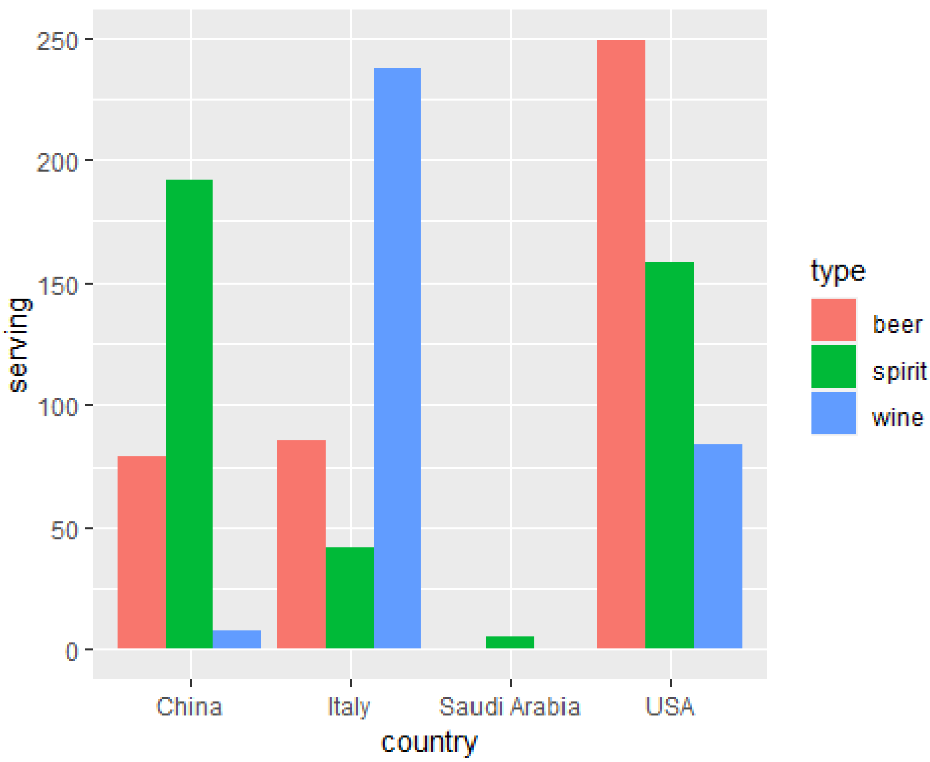
3. The third argument cols is the columns in the drinks_smaller data frame you either want to or don’t want to “tidy.” Observe how we set this to -country indicating that we don’t want to “tidy” the country variable in drinks_smaller and rather only beer, spirit, and wine. Since country is a column that appears in drinks_smaller we don’t put quotation marks around it.

- The third argument here of cols is a little nuanced, so let’s consider code that’s written slightly differently but that produces the same output

```
drinks_smaller %>%
  pivot_longer(names_to = "type" ,
               values_to = "servings" ,
               cols = c(beer , spirit , wine))

## # A tibble: 12 x 3
##    country      type   servings
##    <chr>        <chr>     <int>
##  1 China        beer         79
##  2 China        spirit      192
##  3 China        wine          8
##  4 Italy        beer         85
##  5 Italy        spirit       42
##  6 Italy        wine        237
##  7 Saudi Arabia beer          0
##  8 Saudi Arabia spirit        5
##  9 Saudi Arabia wine          0
## 10 USA          beer        249
## 11 USA          spirit      158
## 12 USA          wine         84

# Same as above
drinks_smaller %>%
  pivot_longer(names_to = "type",
               values_to = "servings",
               cols = beer:wine)

## # A tibble: 12 x 3
##    country      type   servings
##    <chr>        <chr>     <int>
##  1 China        beer         79
##  2 China        spirit      192
##  3 China        wine          8
##  4 Italy        beer         85
##  5 Italy        spirit       42
##  6 Italy        wine        237
##  7 Saudi Arabia beer          0
##  8 Saudi Arabia spirit        5
##  9 Saudi Arabia wine          0
## 10 USA          beer        249
## 11 USA          spirit      158
## 12 USA          wine         84
```

- Barplots that we use geom_col() and not geom_bar(), since we would like to map the “pre-counted” servings variable to the y-aesthetic of the bars.

```
ggplot(drinks_smaller_tidy, aes(x = country, y = serving, fill = type)) +
  geom_col(position = "dodge")
```



5.3 Case Study : Democracy in Guatemala

Convert a data frame that isn't in "tidy" format ("wide" format) to a data frame that is in "tidy" format ("long/narrow" format).

```
library(moderndive)
library(tidyr)
guat_dem <- dem_score %>%
  filter(country == "Guatemala")

guat_dem

## # A tibble: 1 x 10
##   country   `1952` `1957` `1962` `1967` `1972` `1977` `1982`
##   <chr>      <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 Guatemala      2     -6     -5      3      1     -3     -7
## # ... with 2 more variables: 1987 <dbl>, 1992 <dbl>
```

- **Task -1-** Take the values of the columns corresponding to years in guat_dem and convert them into a new "names" variable called year. Furthermore, we need to take the democracy score values in the inside of the data frame and turn them into a new "values" variable called democracy_score.

```
guat_dem_tidy <- guat_dem %>%
  pivot_longer(names_to = "year",
               values_to = "democracy_score",
               cols = -country,
               names_transform = list(year = as.integer)) # to covert into
cha to integer
guat_dem_tidy
```

```
## # A tibble: 9 x 3
##   country    year democracy_score
##   <chr>     <int>           <dbl>
## 1 Guatemala  1952               2
## 2 Guatemala  1957              -6
## 3 Guatemala  1962              -5
## 4 Guatemala  1967               3
## 5 Guatemala  1972               1
## 6 Guatemala  1977              -3
## 7 Guatemala  1982              -7
## 8 Guatemala  1987               3
## 9 Guatemala  1992               3
```

- Make a **Line Chart**

```
ggplot(guat_dem_tidy, aes(x = year, y = democracy_score)) +
  geom_line() +
  labs(x = "Year", y = "Democracy Score")
```

- **(LC4.5)** Read in the life expectancy data stored at https://moderndive.com/data/le_mess.csv and convert it to a "tidy" data frame.

```
le_mess <- read_csv("https://moderndive.com/data/le_mess.csv")

## Rows: 202 Columns: 67

## -- Column specification --------------------------------------
## Delimiter: ","
## chr  (1): country
## dbl (66): 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958,...

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this
message.

# names(le_mess)
dim(le_mess)

## [1] 202  67

head(le_mess)

## # A tibble: 6 x 67
##   country    `1951` `1952` `1953` `1954` `1955` `1956` `1957`
##   <chr>       <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 Afghanist~   27.1   27.7   28.2   28.7   29.3   29.8   30.3
## 2 Albania      54.7   55.2   55.8   56.6   57.4   58.4   59.5
## 3 Algeria      43.0   43.5   44.0   44.4   44.9   45.4   45.9
## 4 Angola       31.0   31.6   32.1   32.7   33.2   33.8   34.3
## 5 Antigua a~   58.3   58.8   59.3   59.9   60.4   60.9   61.4
## 6 Argentina    61.9   62.5   63.1   63.6   64.0   64.4   64.7
## # ... with 59 more variables: 1958 <dbl>, 1959 <dbl>,
## #   1960 <dbl>, 1961 <dbl>, 1962 <dbl>, 1963 <dbl>,
## #   1964 <dbl>, 1965 <dbl>, 1966 <dbl>, 1967 <dbl>,
## #   1968 <dbl>, 1969 <dbl>, 1970 <dbl>, 1971 <dbl>,
## #   1972 <dbl>, 1973 <dbl>, 1974 <dbl>, 1975 <dbl>,
## #   1976 <dbl>, 1977 <dbl>, 1978 <dbl>, 1979 <dbl>,
## #   1980 <dbl>, 1981 <dbl>, 1982 <dbl>, 1983 <dbl>, ...

afg_mess <- le_mess %>%
  filter(country == "Afghanistan")

afg_mess

## # A tibble: 1 x 67
##   country    `1951` `1952` `1953` `1954` `1955` `1956` `1957`
##   <chr>       <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 Afghanist~   27.1   27.7   28.2   28.7   29.3   29.8   30.3
## # ... with 59 more variables: 1958 <dbl>, 1959 <dbl>,
## #   1960 <dbl>, 1961 <dbl>, 1962 <dbl>, 1963 <dbl>,
```

```
## #   1964 <dbl>, 1965 <dbl>, 1966 <dbl>, 1967 <dbl>,
## #   1968 <dbl>, 1969 <dbl>, 1970 <dbl>, 1971 <dbl>,
## #   1972 <dbl>, 1973 <dbl>, 1974 <dbl>, 1975 <dbl>,
## #   1976 <dbl>, 1977 <dbl>, 1978 <dbl>, 1979 <dbl>,
## #   1980 <dbl>, 1981 <dbl>, 1982 <dbl>, 1983 <dbl>, ...

afg_mess_tidy <- afg_mess %>%
  pivot_longer(names_to = "year",
               values_to = "democracy_score",
               cols = -country,
               names_transform = list(year = as.integer)) # to covert into
cha to integer

# View(afg_mess_tidy)
dim(afg_mess_tidy)

## [1] 66  3

head(afg_mess_tidy)

## # A tibble: 6 x 3
##   country      year democracy_score
##   <chr>       <int>           <dbl>
## 1 Afghanistan  1951            27.1
## 2 Afghanistan  1952            27.7
## 3 Afghanistan  1953            28.2
## 4 Afghanistan  1954            28.7
## 5 Afghanistan  1955            29.3
## 6 Afghanistan  1956            29.8
```

5.4 Connecting a DataBase

```
library(dplyr)
#install.packages("RSQLite")
#install.packages("learnrbook")
require(learnrbook)
con <- DBI::dbConnect(RSQLite::SQLite(), dbname = ":memory:")

copy_to(con, weather_wk_25_2019.tb, "weather",
        temporary = FALSE,
        indexes = list(
          c("month_name", "calendar_year", "solar_time"),
          "time",
          "sun_elevation",
          "was_sunny",
          "day_of_year",
          "month_of_year")
        )

weather.db <- tbl(con, "weather")
```

```r
colnames(weather.db)
```

```
##  [1] "time"          "PAR_umol"       "PAR_diff_fr"
##  [4] "global_watt"   "day_of_year"    "month_of_year"
##  [7] "month_name"    "calendar_year"  "solar_time"
## [10] "sun_elevation" "sun_azimuth"    "was_sunny"
## [13] "wind_speed"    "wind_direction" "air_temp_C"
## [16] "air_RH"        "air_DP"         "air_pressure"
## [19] "red_umol"      "far_red_umol"   "red_far_red"
```

6 Next

Next Part - II of this series is on next file