

# Basic Statistics Using R

*Feb 6, 2020*

## Review: What you have learned

- Install and use R and Rstudio
- Read/write data from menu or script, `help()`, `install.packages()` (Many useful user-written R packages in CRAN or Github!)
- Common variable types: integer/numeric, character, factor, logic
- Common data types: vector, matrix, **data frame**, list
- Operators: arithmetic, logical
- Conditions: if/else;
- Loops: for/while()
- Functions: use/write your own functions to modularize your program and reduce repetition/errors

## Note for R code windows

- Either in PC or server, you can open the .rmd file, then highlight one or a few line of code, then use “run selected line” or use the short cut key of control+ enter.
- You can also run an entire chunk of code. In other use, you can also copy the code to run in the console, or run the code from a R script window.
- R markdown file (.rmd) is a combination of R code (in code chunks) and notes/outout/summary reports. (search “R markdown” and “Get started”).

## Descriptive statistics (Sec 2.1)

- Descriptive statistics are useful to summarize the overall feature of the data distribution such as center and spread.
- “Table 1” of most research paper is about descriptive statistics of the study population...
- To measure the center, one may use the **mean**, the **median** or the **mode**.

## Mean

The **arithmetic mean** (or **mean** or **average**),  $\bar{x}$  (read *x bar*), is the mean of the  $n$  values  $x_1, x_2, \dots, x_n$ .<sup>[2]</sup>

The arithmetic mean is the most commonly used and readily understood measure of central tendency in a [data set](#). In statistics, the term [average](#) refers to any of the measures of central tendency. The arithmetic mean of a set of observed data is defined as being equal to the sum of the numerical values of each and every observation divided by the total number of observations. Symbolically, if we have a data set consisting of the values  $a_1, a_2, \dots, a_n$ , then the arithmetic mean  $A$  is defined by the formula:

$$A = \frac{1}{n} \sum_{i=1}^n a_i = \frac{a_1 + a_2 + \dots + a_n}{n}$$

(See [summation](#) for an explanation of the [summation operator](#)).

For example, consider the monthly salary of 10 employees of a firm: 2500, 2700, 2400, 2300, 2550, 2650, 2750, 2450, 2600, 2400. The arithmetic mean is

$$\frac{2500 + 2700 + 2400 + 2300 + 2550 + 2650 + 2750 + 2450 + 2600 + 2400}{10} = 2530.$$

If the data set is a [statistical population](#) (i.e., consists of every possible observation and not just a subset of them), then the mean of that population is called the **population mean**. If the data set is a [statistical sample](#) (a subset of the population), we call the statistic resulting from this calculation a **sample mean**.

## Median: ('order statistics')

The **median** is the value separating the higher half from the lower half of a data [sample](#) (a [population](#) or a [probability distribution](#)). For a data set, it may be thought of as the "middle" value. For example, in the data set {1, 3, 3, 6, 7, 8, 9}, the median is 6, the fourth largest, and also the fourth smallest, number in the sample. For a [continuous probability distribution](#), the median is the value such that a number is equally likely to fall above or below it.

The median is a commonly used measure of the properties of a data set in [statistics](#) and [probability theory](#). The basic advantage of the median in describing data compared to the [mean](#) (often simply described as the "average") is that it is not [skewed](#) so much by a small proportion of extremely large or small values, and so it may give a better idea of a "typical" value. For example, in understanding statistics like household income or assets, which vary greatly, the mean may be skewed by a small number of extremely high or low values. [Median income](#), for example, may be a better way to suggest what a "typical" income is.

Because of this, the median is of central importance in [robust statistics](#), as it is the most [resistant statistic](#), having a [breakdown point](#) of 50%: so long as no more than half the data are contaminated, the median will not give an arbitrarily large or small result.

1, 3, 3, **6**, 7, 8, 9  
Median = **6**

1, 2, 3, **4, 5**, 6, 8, 9  
Median = (4 + 5) ÷ 2  
= **4.5**

Finding the median in sets of data with an odd and even number of values

## Mode

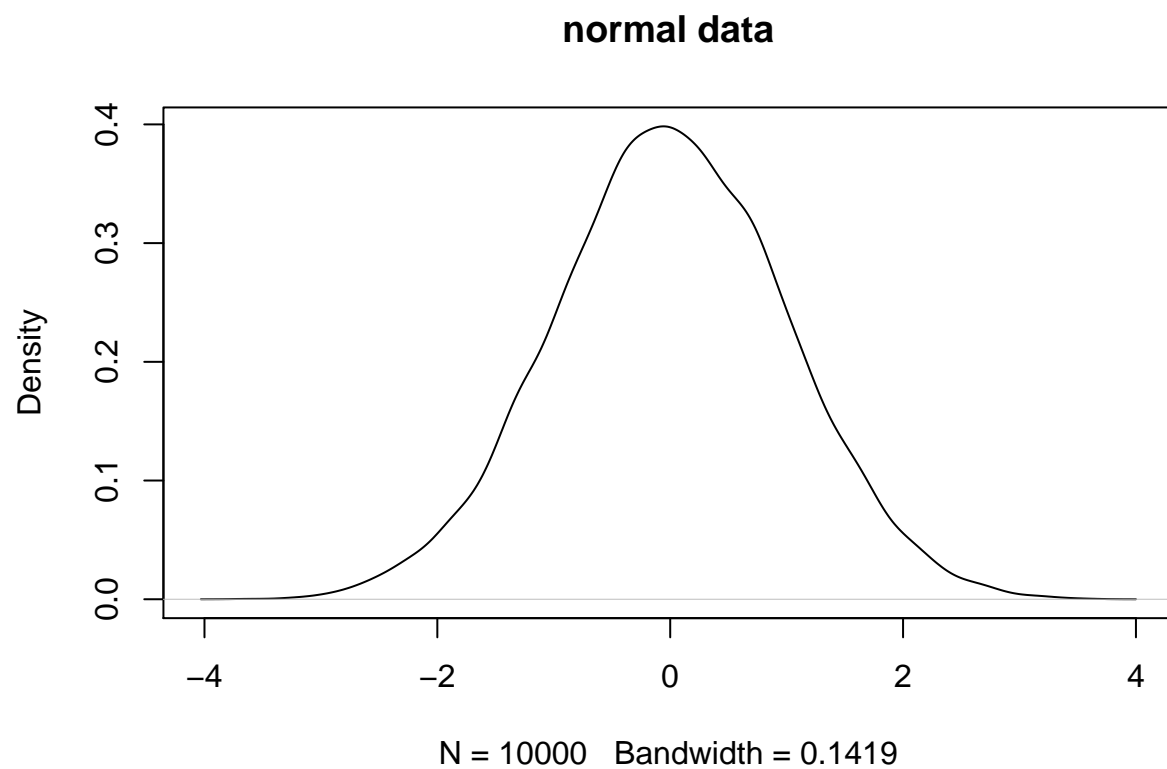
The **mode** of a set of data values is the value that appears most often.<sup>[1]</sup> If  $X$  is a discrete random variable, the mode is the value  $x$  (i.e.,  $X = x$ ) at which the [probability mass function](#) takes its maximum value. In other words, it is the value that is most likely to be sampled.

Like the statistical [mean](#) and [median](#), the mode is a way of expressing, in a (usually) single number, important information about a [random variable](#) or a [population](#). The numerical value of the mode is the same as that of the mean and median in a [normal distribution](#), and it may be very different in highly [skewed distributions](#).

## Center of Some Distributions ('population')

- For normal (or Gaussian) distribution, with a symmetric, bell shape with one peak, the mean, median and mode are the same.

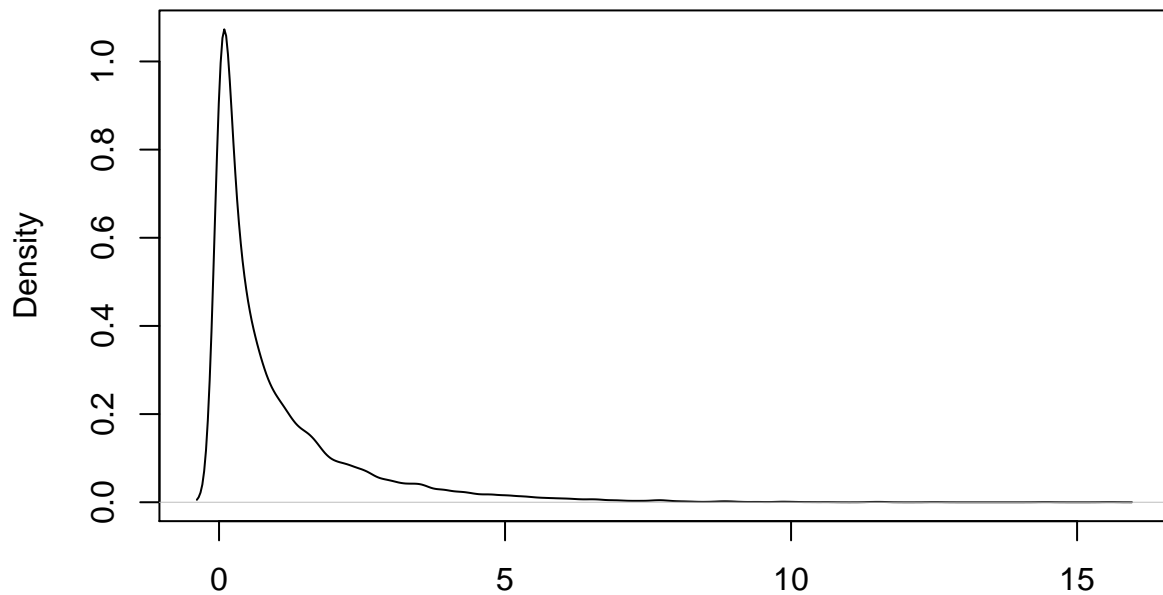
```
set.seed(321)
NormalData<- rnorm(10000) #standard N(0,1)
plot(density(NormalData), main='normal data') # one center at 0
```



- For asymmetric distribution, the mean, median and mode may be all different.

```
set.seed(321)
ChiData<- rchisq(10000, df=1) #chisquare distribution , right skewed
plot(density(ChiData), main='chi-square data')
```

## chi-square data



N = 10000 Bandwidth = 0.1293

## Data Examples (1)



1. Check the data description: (a commonly used data example)

```
##?iris
```

```
# data Structure  
str(iris)
```

```
## 'data.frame':  150 obs. of  5 variables:
```

```
## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
## $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
## $ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

```
#show the first few lines of the transcriptome data
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1         5.1         3.5         1.4         0.2  setosa
## 2         4.9         3.0         1.4         0.2  setosa
## 3         4.7         3.2         1.3         0.2  setosa
## 4         4.6         3.1         1.5         0.2  setosa
## 5         5.0         3.6         1.4         0.2  setosa
## 6         5.4         3.9         1.7         0.4  setosa
```

```
#tail(iris) #last 6
```

```
#show the sample names of the transcriptome data
names(iris)
```

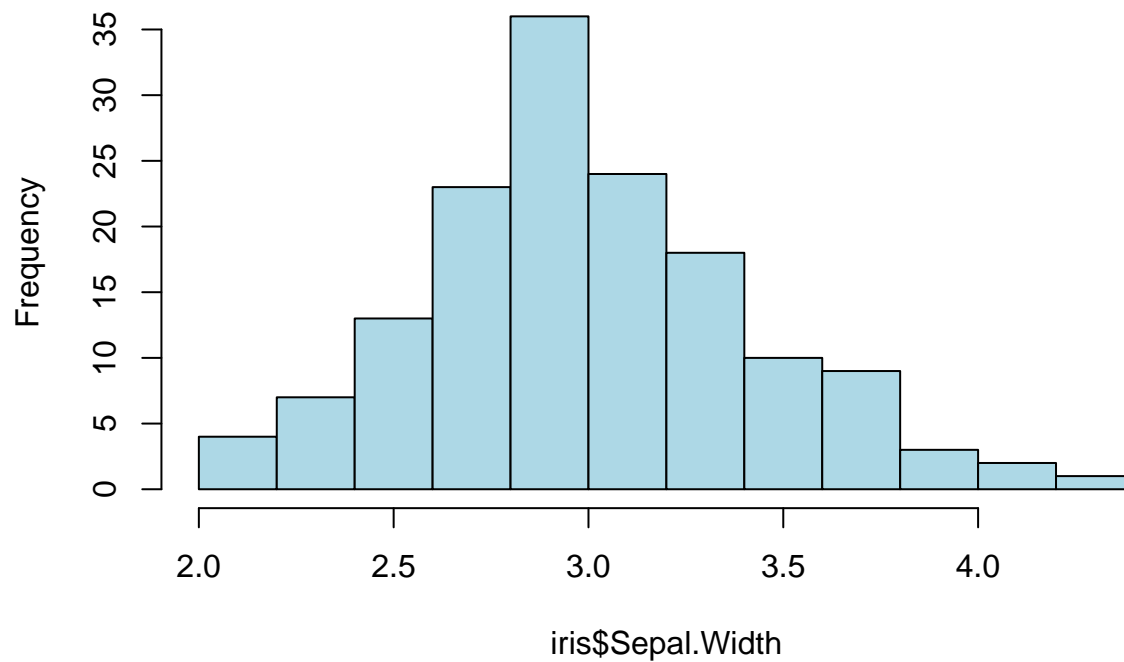
```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

## Data Examples (2)

### 2. Getting the mean, median and mode of one variable

```
# It is good idea to always plot the data to do some 'visual inspection' first
hist(iris$Sepal.Width, col='light blue')
```

## Histogram of iris\$Sepal.Width



```
#mean  
mean(iris$Sepal.Width)
```

```
## [1] 3.057333
```

```
#median  
median(iris$Sepal.Width)
```

```
## [1] 3
```

```
#mode using mfv - most frequent value, part of modeest package  
library(modeest)
```

```
## Warning: package 'modeest' was built under R version 3.6.2
```

```
mfv(iris$Sepal.Width)
```

```
## [1] 3
```

### Note:

Sometimes, there are missing value, we use **na.rm=T** as an extra argument to removing missing at calculation.

## Try our randomly generated data

### 1. Normal data (same mean/median)

```
#set.seed(321) ; NormalData<- rnorm(10000) #standard N(0,1)
mean(NormalData)
```

```
## [1] 0.004038458
```

```
median(NormalData)
```

```
## [1] -0.0002606666
```

### 2. Chi-square data (different mean/median, mean is pulled to the skewed side)

```
# set.seed(321); ChiData<- rchisq(10000, df=1) #chisquare distribution , right skewed
mean(ChiData)
```

```
## [1] 0.988585
```

```
median(ChiData)
```

```
## [1] 0.4480839
```

Note: If you have truly continuous data, or each discrete data points are different, there will be no unique or meaningfully ‘mode’.

## Measures of Data Spread and Variability (sec 2.2)

### 1. Min , max and range

```
#min, max, range
min(iris$Sepal.Width)
```

```
## [1] 2
```

```
max(iris$Sepal.Width)
```

```
## [1] 4.4
```

```
range(iris$Sepal.Width)
```

```
## [1] 2.0 4.4
```

2. Variance =  $E(X - \mu)^2$ , and standard deviation,  $\sigma = \sqrt{\text{variance}}$

In probability theory and statistics, **variance** is the expectation of the squared deviation of a random variable from its mean. Informally, it measures how far a set of (random) numbers are spread out from their average value. Variance has a central role in statistics, where some ideas that use it include descriptive statistics, statistical inference, hypothesis testing, goodness of fit, and Monte Carlo sampling. Variance is an important tool in the sciences, where statistical analysis of data is common. The variance is the square of the standard deviation, the second central moment of a distribution, and the covariance of the random variable with itself, and it is often represented by  $\sigma^2$ ,  $s^2$ , or  $\text{Var}(X)$ .

```
#variance
var(iris$Sepal.Width)
```

```
## [1] 0.1899794
```

```
var(iris$Petal.Width)
```

```
## [1] 0.5810063
```

```
#standard deviation
sd(iris$Sepal.Width)
```

```
## [1] 0.4358663
```

```
sd(iris$Petal.Width)
```

```
## [1] 0.7622377
```

### 3. Coefficient of Variation (CV)

The coefficient of variation (CV) is defined as the ratio of the standard deviation  $\sigma$  to the mean  $\mu$ :<sup>[1]</sup>  $c_v = \frac{\sigma}{\mu}$ . It shows the extent of variability in relation to the mean of the population. The coefficient of variation should be computed only for data measured on a ratio scale, as these are the measurements that allow the division operation. The coefficient of variation may not have any meaning for data on an interval scale.<sup>[2]</sup> For example, most temperature scales (e.g., Celsius, Fahrenheit etc.) are interval scales with arbitrary zeros, so the coefficient of variation would be different depending on which scale you used. On the other hand, Kelvin temperature has a meaningful zero, the complete absence of thermal energy, and thus is a ratio scale. While the standard deviation (SD) can be meaningfully derived using Kelvin, Celsius, or Fahrenheit, the CV is only valid as a measure of relative variability for the Kelvin scale because its computation involves division.



- CV is a relative measure of dispersion. Sometimes, inter-assay and intra-assay CV to check for precision and performance of the assay.

```
#coefficient of variation
(sd(iris$Sepal.Width)/mean(iris$Sepal.Width))*100
```

```
## [1] 14.25642
```

```
(sd(iris$Petal.Width)/mean(iris$Petal.Width))*100
```

```
## [1] 63.55511
```

#### 4. Standard Error or SEM, $SE = SD/\sqrt{n}$

The **standard error (SE)** of a **statistic** (usually an estimate of a **parameter**) is the **standard deviation** of its **sampling distribution**<sup>[1]</sup> or an estimate of that standard deviation. If the parameter or the statistic is the mean, it is called the **standard error of the mean (SEM)**.

```
(sd(iris$Sepal.Width)/(sqrt(length(iris$Sepal.Width))))
```

```
## [1] 0.03558833
```

```
(sd(iris$Petal.Width)/(sqrt(length(iris$Petal.Width))))
```

```
## [1] 0.06223645
```

## Measures of Data Spread and Variability(2) (sec 2.2)

### 5. Quantiles/Percentiles, quartiles (25%-Q1, median-Q2, 75%-Q3)

We commonly used Mean +/- SEM to summarize the variable if it is approximately normal or use Median (Interquartile Range, IQR, Q1 to Q3) to summarize the variable if it is skewed.

In statistics and probability **quantiles** are cut points dividing the range of a **probability distribution** into continuous intervals with equal probabilities, or dividing the observations in a sample in the same way. There is one fewer quantile than the number of groups created. Thus **quartiles** are the three cut points that will divide a dataset into four equal-sized groups. Common quantiles have special names: for instance quartile, decile (creating 10 groups: see below for more). The groups created are termed halves, thirds, quarters, etc., though sometimes the terms for the quantile are used for the groups created, rather than for the cut points.

```
#quantiles
quantile(iris$Sepal.Width)
```

```
##    0%   25%   50%   75%  100%
##   2.0   2.8   3.0   3.3   4.4
```

```
quantile(iris$Sepal.Width, prob=c(1/3, 2/3)) #tercile
```

```
## 33.33333% 66.66667%  
##      2.9      3.2
```

```
#summary: R function to provide 6 summary stat for each numeric var and freq for categorical var,  
summary(iris$Sepal.Width)
```

```
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.  
##      2.000   2.800   3.000   3.057   3.300   4.400
```

```
#summary: apply to a data frame to summarize every variable  
summary(iris)
```

```
##      Sepal.Length      Sepal.Width      Petal.Length      Petal.Width  
##      Min.      :4.300      Min.      :2.000      Min.      :1.000      Min.      :0.100  
##      1st Qu.:5.100      1st Qu.:2.800      1st Qu.:1.600      1st Qu.:0.300  
##      Median :5.800      Median :3.000      Median :4.350      Median :1.300  
##      Mean   :5.843      Mean   :3.057      Mean   :3.758      Mean   :1.199  
##      3rd Qu.:6.400      3rd Qu.:3.300      3rd Qu.:5.100      3rd Qu.:1.800  
##      Max.    :7.900      Max.    :4.400      Max.    :6.900      Max.    :2.500  
##      Species  
##      setosa      :50  
##      versicolor:50  
##      virginica  :50  
##  
##  
##
```

We can apply summary in several subgroups:

```
by(iris$Sepal.Width, iris$Species, summary)
```

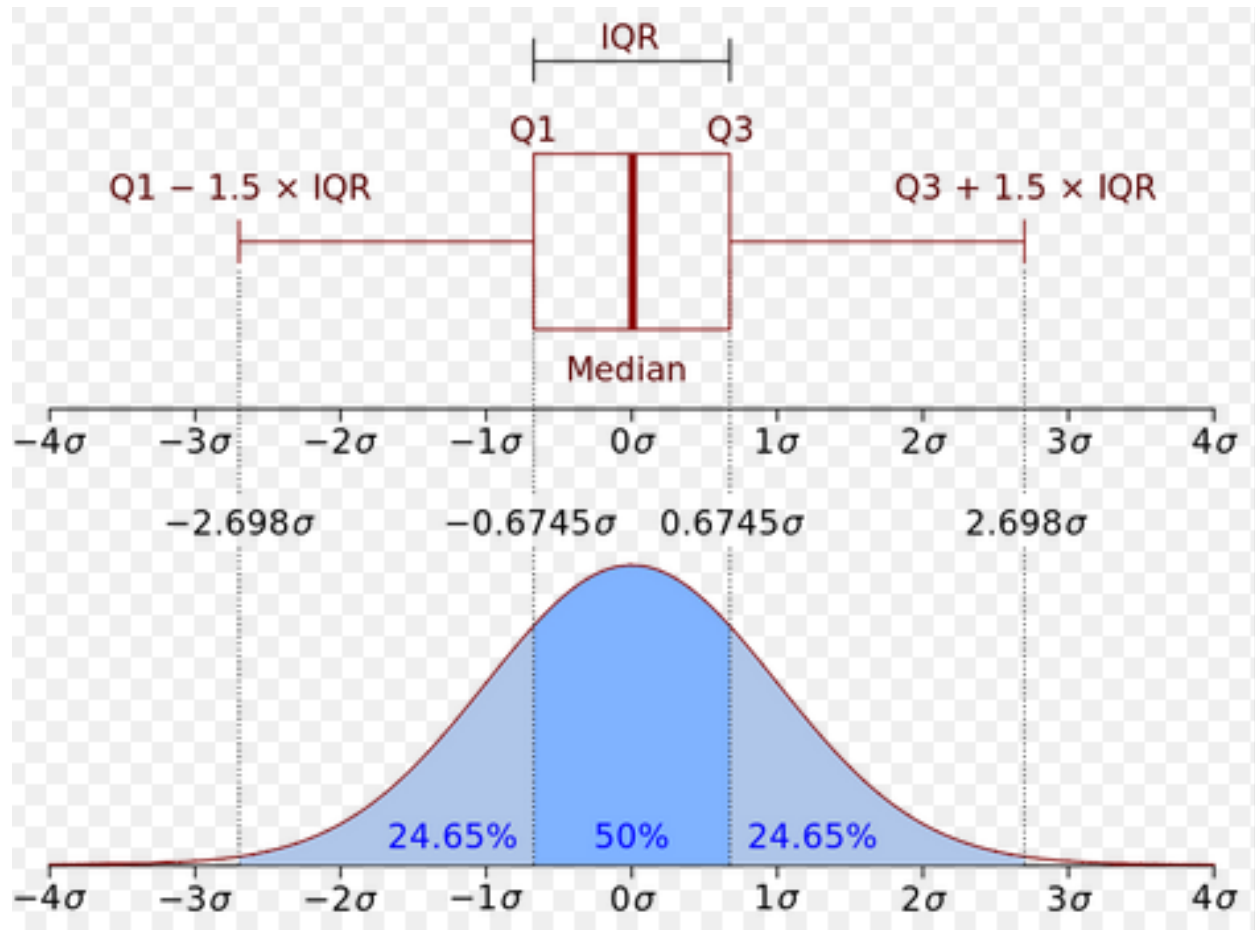
```
## iris$Species: setosa  
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.  
##      2.300   3.200   3.400   3.428   3.675   4.400  
## -----  
## iris$Species: versicolor  
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.  
##      2.000   2.525   2.800   2.770   3.000   3.400  
## -----  
## iris$Species: virginica  
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.  
##      2.200   2.800   3.000   2.974   3.175   3.800
```

```
by(iris$Sepal.Width, iris$Species, sd)
```

```
## iris$Species: setosa  
## [1] 0.3790644  
## -----
```

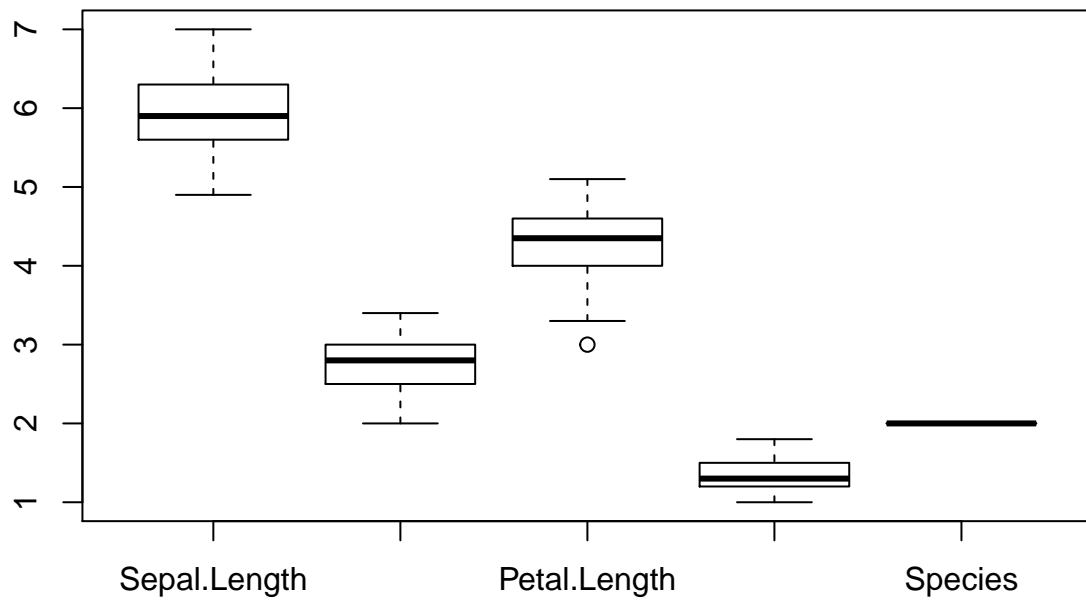
```
## iris$Species: versicolor
## [1] 0.3137983
## -----
## iris$Species: virginica
## [1] 0.3224966
```

Box-plot or Box-Whisker plot

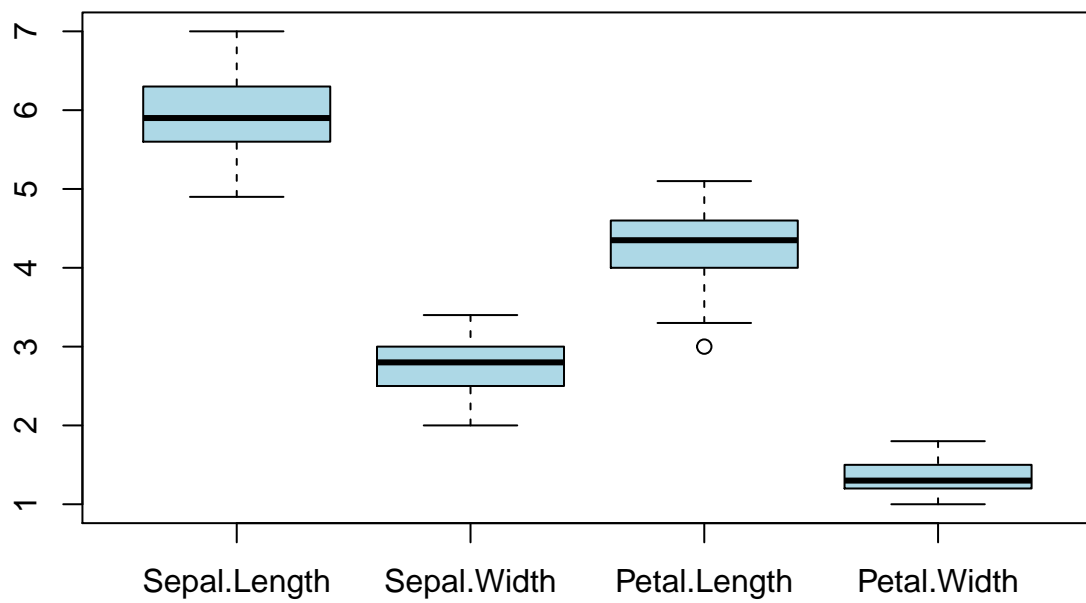


Note: It is useful to use boxplot to check the distribution of a variable : is it symmetric? Any outliers?

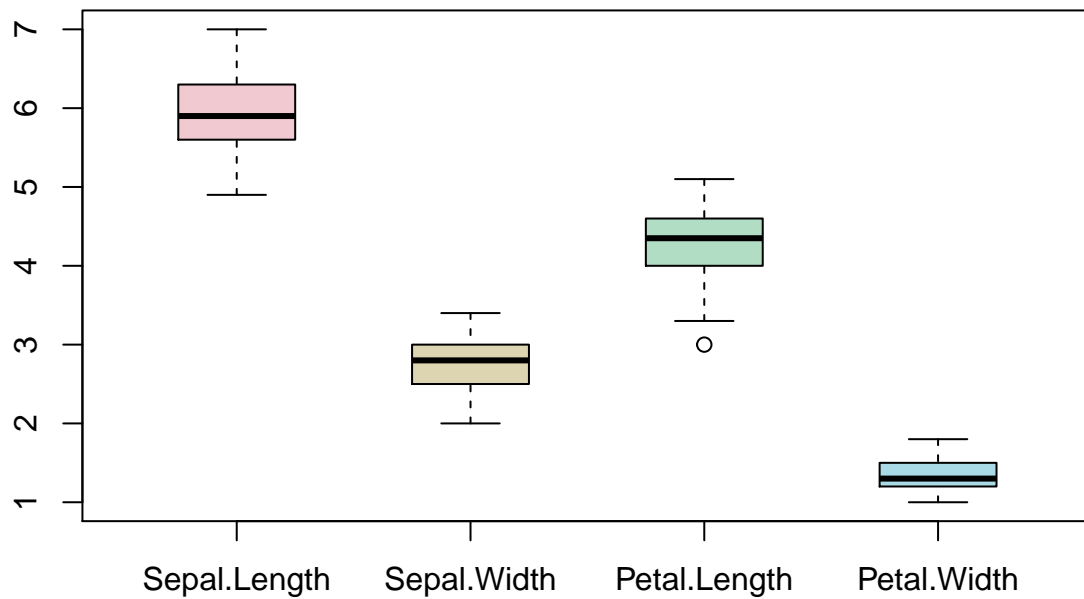
```
#boxplot - This include the factor 'Species' var
boxplot(subset(iris,Species=='versicolor'))
```



```
# better way  
boxplot(subset(iris,Species=='versicolor')[,1:4], col='lightblue')
```



```
library(colorspace)
boxplot(subset(iris,Species=='versicolor')[,1:4], col= rainbow_hcl(5, alpha=0.5), boxwex = 0.5)
```



## Descriptive stat for categorical variable

For binary or categorical variables, we are mostly interested in frequency and percentage , n (%).

```
# n
table(iris$Species)
```

```
##
##      setosa versicolor  virginica
##      50         50         50
```

```
##
prop.table(table(iris$Species))
```

```
##
##      setosa versicolor  virginica
## 0.3333333 0.3333333 0.3333333
```

```
#round off
round(prop.table(table(iris$Species))*100, 1)
```

```
##
##      setosa versicolor  virginica
##      33.3      33.3      33.3
```

## Common Data Distribution: (Sec 3)

There are many statistical distributions. We will discuss a few of common used ones today.

**Normal Distribution** is the most important distribution because

1. The fundamental central limit theorem: if you take sufficiently large independent random samples from the population, then the distribution of the sample means will be approximately normally distributed. This will hold true regardless of whether the source population is normal or skewed, continuous or discrete, provided the sample size is sufficiently large (usually  $n > 30$ ).
2. Many statistical theories and tests are based on normal assumption. It is only characterized by two parameters.

The density function for a normal random variable  $Y$  is:

$$f(Y) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{Y - \mu}{\sigma}\right)^2\right] \quad -\infty < Y < \infty$$

where  $\mu$  and  $\sigma$  are the two parameters of the normal distribution and  $\exp(a)$  denotes  $e^a$ .

The mean and variance of a normal random variable  $Y$  are:

$$\begin{aligned} E\{Y\} &= \mu \\ \sigma^2\{Y\} &= \sigma^2 \end{aligned}$$

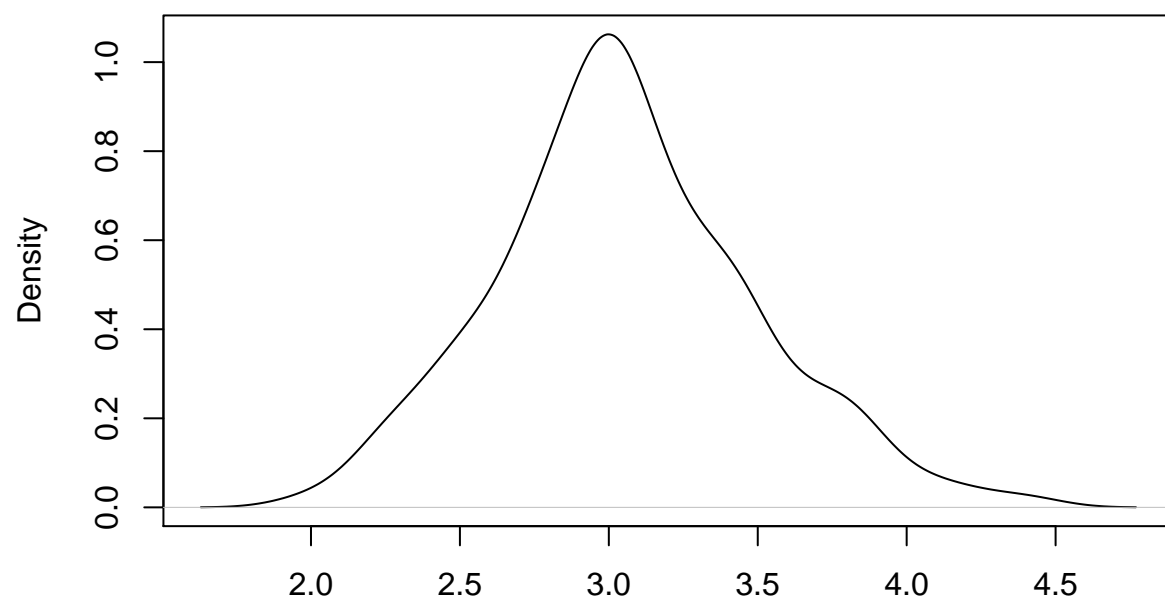
3. We can use **plot(density())** to check the shape of the distribution, use **qqnorm()** to compare with normal quantiles, or **Shapiro-Wilk Normality Test** to check normal assumption.

### Example

```
# help("Distributions")

#visualizing data density distribution
plot(density(iris$Sepal.Width))
```

**density.default(x = iris\$Sepal.Width)**

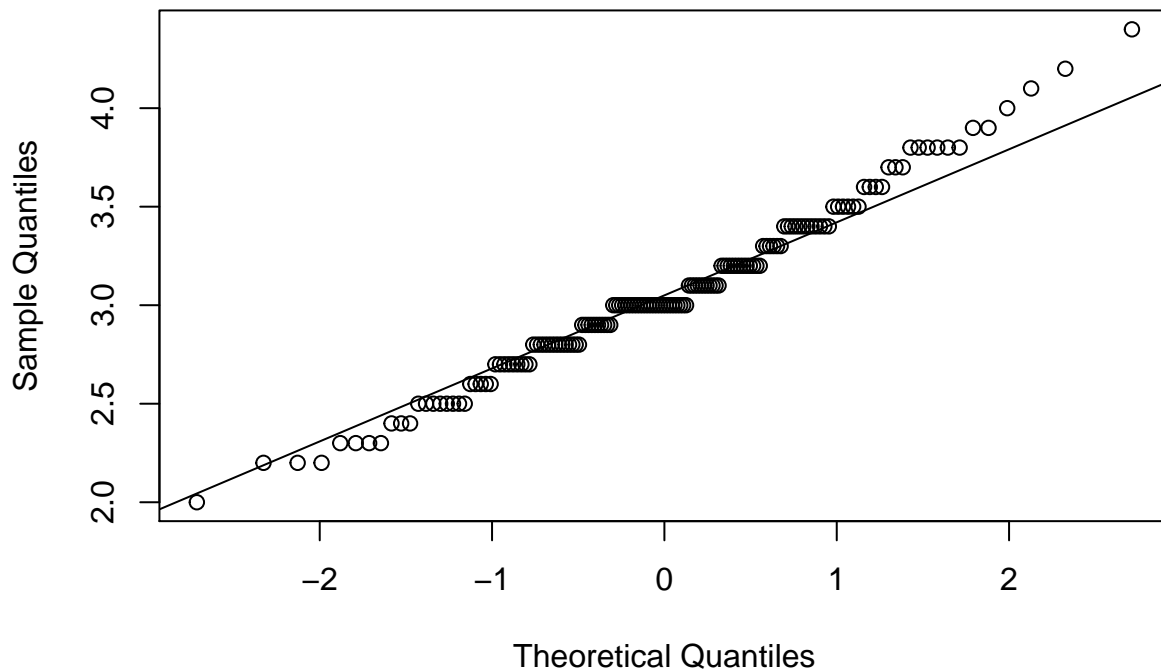


N = 150 Bandwidth = 0.1233

```
qqnorm(iris$Sepal.Width)  
qqline(iris$Sepal.Width)
```



## Normal Q-Q Plot



```
#testing for normality
#is the sepal width normally distributed ?
shapiro.test(iris$Sepal.Width)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  iris$Sepal.Width
## W = 0.98492, p-value = 0.1012
```

Note:  $p\text{-value}=0.10 > 0.05$ , and we can't reject the normality at the significance level of 0.05.

4. Assuming the underlying distribution is normal for **Sepal.Width**, we can compute certain probability of an event. `pnorm()` calculates the probability of a given normal distribution.

```
c(mean= mean(iris$Sepal.Width ),sd= sd(iris$Sepal.Width ))
```

```
##      mean      sd
## 3.0573333 0.4358663
```

```
#calculate and use the mean and sd of Sepal.Width
#what is the probability of observing a sepal width less than 2 : Pr(Y<2)
# this is very small since 2 is ~ 2.4 SD away from the mean.
pnorm(2,mean=3.057,sd=0.4358)
```

```
## [1] 0.007645248
```

```
#what is the probability of observing a sepal width 2 or more: : Pr(Y>=2)=1- Pr(Y<2)
pnorm(2,mean=3.057,sd=0.4358, lower.tail = FALSE)
```

```
## [1] 0.9923548
```

## Other Common Continuous Distributions

Later we will discuss some statistical tests: chi-square test, t-test and F-tests based on these distributions (all related to normal variables).

### 2. Chi-square distribution, df

#### $\chi^2$ Distribution

Let  $z_1, \dots, z_\nu$  be  $\nu$  independent standard normal random variables. We then define a chi-square random variable as follows:

$$\chi^2(\nu) = z_1^2 + z_2^2 + \dots + z_\nu^2 \quad \text{where the } z_i \text{ are independent}$$

The  $\chi^2$  distribution has one parameter,  $\nu$ , which is called the *degrees of freedom (df)*. The mean of the  $\chi^2$  distribution with  $\nu$  degrees of freedom is:

$$E\{\chi^2(\nu)\} = \nu$$

### 3. t-distribution, df (symmetric, long-tails than standard normal).

#### t Distribution

Let  $z$  and  $\chi^2(\nu)$  be independent random variables (standard normal and  $\chi^2$ , respectively). We then define a  $t$  random variable as follows:

$$t(\nu) = \frac{z}{\left[\frac{\chi^2(\nu)}{\nu}\right]^{1/2}} \quad \text{where } z \text{ and } \chi^2(\nu) \text{ are independent}$$

### 4. F-distribution, df1, df2

#### F Distribution

Let  $\chi^2(\nu_1)$  and  $\chi^2(\nu_2)$  be two independent  $\chi^2$  random variables. We then define an  $F$  random variable as follows:

$$F(\nu_1, \nu_2) = \frac{\chi^2(\nu_1)}{\nu_1} \div \frac{\chi^2(\nu_2)}{\nu_2} \quad \text{where } \chi^2(\nu_1) \text{ and } \chi^2(\nu_2) \text{ are independent}$$

$\nearrow$        $\nwarrow$   
 Numerator    Denominator  
 df              df

The  $F$  distribution has two parameters, the *numerator degrees of freedom* and the *denominator degrees of freedom*, here  $\nu_1$  and  $\nu_2$ , respectively.

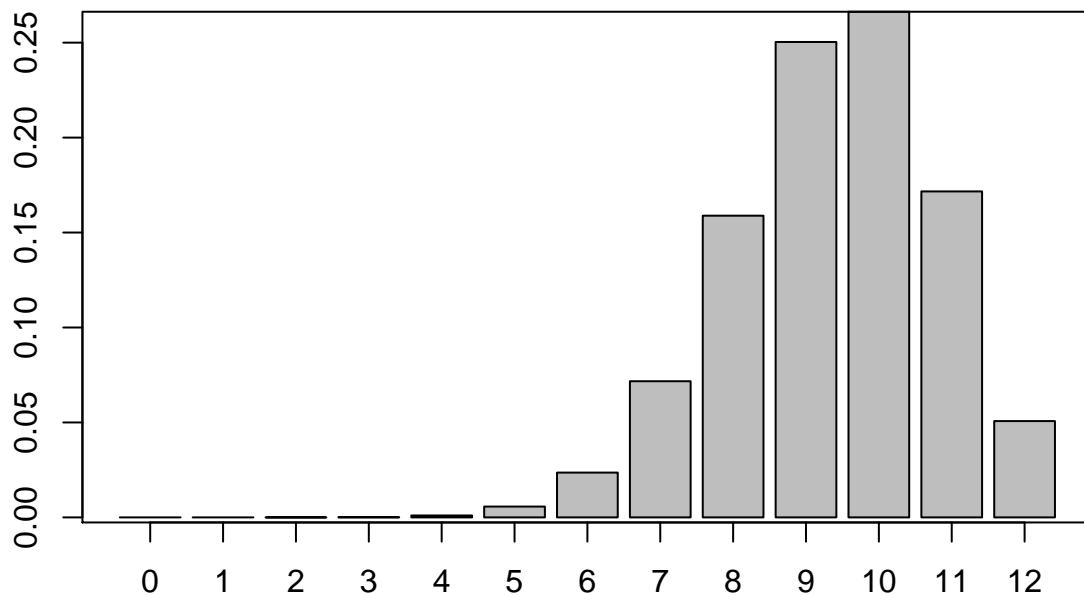
## Discrete Distributions

**Binomial Distribution** is a common used distribution for discrete variable. A variable  $X$  is considered binomially distributed, if  $X$  = the number of success of  $n$  independent trials where each trial consists of two possible results, with a fixed probability of success, e.g. flipping a coin 20 times.

### Example

Our example data shows that 78% of the students who take the AP biology test, pass the exam. If 12 new students take this test, what is the probability that exactly 8 of them will pass the test ?

```
##dbinom  
#data visualization  
#1. calculate the probability distribution for all possible outcomes  
  
passprob=dbinom(0:12,size=12,prob=0.78)  
  
#visualize the probability distribution  
barplot(passprob)  
axis(1, at=seq(0.7, 15.1, 1.2), labels=0:12)  
box()
```



```
#probability =Pr(X= 8 | N=12, p=.78)  
dbinom(8,12, 0.78)
```

```
## [1] 0.158874
```

```
#If 12 new students take this test, what is the probability that at least 8 of them will pass the test  
pbinom(7, size=12, prob = 0.78, lower.tail = FALSE)
```

```
## [1] 0.897864
```

```
#If change p=0.5 ?  
pbinom(7, size=12, prob = 0.5, lower.tail = FALSE)
```

```
## [1] 0.1938477
```

#### Note:

Distribution theories are more useful in the context of statistical inference. Often, people apply the statistical tests without knowing or checking the underlying distribution assumption, or apply the wrong test when the assumptions are not met. We will discuss more tomorrow with examples.

## Data transformation (Sec 4)

We often need to create new variables or apply certain transformation (e.g. log-transformation for skewed variable) to our data set. This can be done by the traditional way or use the new **tidyverse** package.

```
Rpackage= "tidyverse"  
if (! Rpackage %in% installed.packages()) install.packages(Rpackage)  
library(tidyverse)
```

```
## Registered S3 method overwritten by 'httr':  
##   method      from  
##   print.response rmutil
```

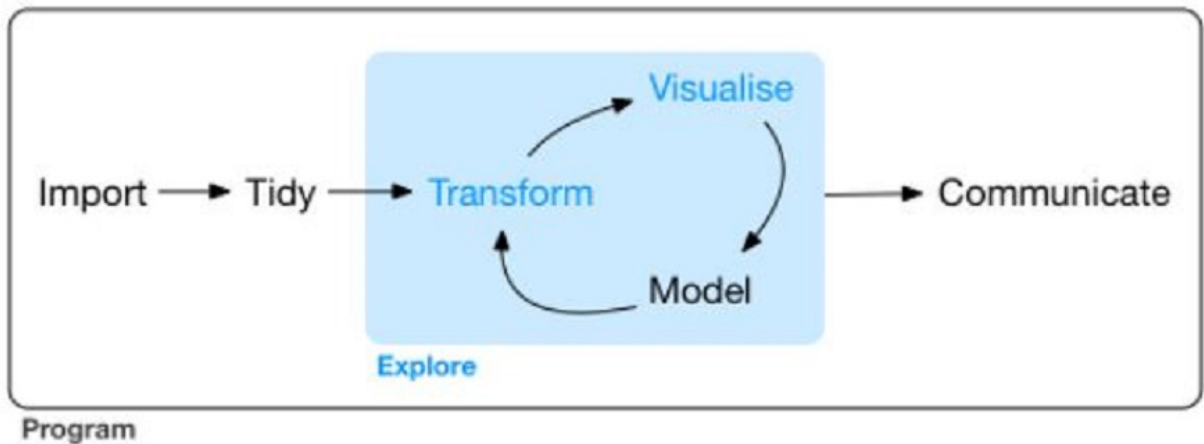
```
## -- Attaching packages ----- tidyverse 1.2.1 --
```

```
## v ggplot2 3.2.1    v purrr  0.3.2  
## v tibble  2.1.3    v dplyr  0.8.3  
## v tidyr   0.8.3    v stringr 1.4.0  
## v readr   1.3.1    v forcats 0.4.0
```

```
## -- Conflicts ----- tidyverse_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag()    masks stats::lag()
```

Some examples below are based on an online book **R for Data Science**, by Garrett Golemund and Hadley Wickham, from RStudio, also the authors of many useful new R packages.

Their data exploration flowchart is



### 3.1 Check data.

This data frame contains all 336,776 flights that departed from New York City in 2013.

```

Rpackage= "nycflights13"
if (! Rpackage %in% installed.packages()) install.packages(Rpackage)

library('nycflights13')

#? flights
data(flights)
str(flights)

```

```

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

```

```

#head(flights,3)

```

## 3.2 basic data transformation steps using dplyr

### 3.2.1 Pick observations by their values: filter()

This allows you to subset your observations based on certain criteria or conditions. Multiple conditions can be combined with. Only rows where the condition evaluates to TRUE are kept.

```
# pick all flights on certain date, note "==" for comparison, "=" for assignment
```

```
Dec25 = filter(flights, month == 12, day == 25)
Dec25
```

```
## # A tibble: 719 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>   <int>         <int>       <dbl>   <int>
## 1  2013    12    25     456           500         -4     649
## 2  2013    12    25     524           515          9     805
## 3  2013    12    25     542           540          2     832
## 4  2013    12    25     546           550         -4    1022
## 5  2013    12    25     556           600         -4     730
## 6  2013    12    25     557           600         -3     743
## 7  2013    12    25     557           600         -3     818
## 8  2013    12    25     559           600         -1     855
## 9  2013    12    25     559           600         -1     849
## 10 2013    12    25     600           600          0     850
## # ... with 709 more rows, and 12 more variables: 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 <dtm>
```

### 3.2.2 Reorder data: arrange()

It takes a data frame and a set of column names to order by. If you provide more than one column name, each additional column will be used to break ties in the values of preceding columns:

```
# Arrange rows and order by
arrange(flights, year, month, day)
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>   <int>         <int>       <dbl>   <int>
## 1  2013     1     1     517           515          2     830
## 2  2013     1     1     533           529          4     850
## 3  2013     1     1     542           540          2     923
## 4  2013     1     1     544           545         -1    1004
## 5  2013     1     1     554           600         -6     812
## 6  2013     1     1     554           558         -4     740
## 7  2013     1     1     555           600         -5     913
## 8  2013     1     1     557           600         -3     709
## 9  2013     1     1     557           600         -3     838
## 10 2013     1     1     558           600         -2     753
## # ... with 336,766 more rows, and 12 more variables: 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 <dtm>
```

```
# use desc() to re-order by a column in descending order:
# Missing values are always sorted at the end:
```

```
arrange(flights, desc(dep_time))
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>   <int>         <int>         <dbl>   <int>
## 1  2013    10    30     2400             2359           1     327
## 2  2013    11    27     2400             2359           1     515
## 3  2013    12     5     2400             2359           1     427
## 4  2013    12     9     2400             2359           1     432
## 5  2013    12     9     2400             2250          70      59
## 6  2013    12    13     2400             2359           1     432
## 7  2013    12    19     2400             2359           1     434
## 8  2013    12    29     2400             1700         420     302
## 9  2013     2     7     2400             2359           1     432
## 10 2013     2     7     2400             2359           1     443
## # ... with 336,766 more rows, and 12 more variables: 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 <dtm>
```

### 3.2.3 Pick variables by their names: select()

If you have a large data with lots of columns/variables, you can use `select()` to subset and select certain columns.

```
# Select columns by name
select(flights, year, month, day)
```

```
## # A tibble: 336,776 x 3
##   year month   day
##   <int> <int> <int>
## 1  2013     1     1
## 2  2013     1     1
## 3  2013     1     1
## 4  2013     1     1
## 5  2013     1     1
## 6  2013     1     1
## 7  2013     1     1
## 8  2013     1     1
## 9  2013     1     1
## 10 2013     1     1
## # ... with 336,766 more rows
```

```
# select a number of variables between the two specified ones: inclusive
select(flights, year:sched_dep_time)
```

```
## # A tibble: 336,776 x 5
##   year month   day dep_time sched_dep_time
##   <int> <int> <int>   <int>         <int>
## 1  2013     1     1     517           515
## 2  2013     1     1     533           529
## 3  2013     1     1     542           540
## 4  2013     1     1     544           545
## 5  2013     1     1     554           600
## 6  2013     1     1     554           558
## 7  2013     1     1     555           600
## 8  2013     1     1     557           600
## 9  2013     1     1     557           600
## 10 2013     1     1     558           600
## # ... with 336,766 more rows
```

```
# move some variables to the start of the data frame
select(flights, time_hour, air_time, everything())
```

```
## # A tibble: 336,776 x 19
##   time_hour          air_time year month   day dep_time sched_dep_time
##   <dtm>          <dbl> <int> <int> <int>   <int>         <int>
## 1 2013-01-01 05:00:00      227  2013     1     1     517           515
## 2 2013-01-01 05:00:00      227  2013     1     1     533           529
## 3 2013-01-01 05:00:00      160  2013     1     1     542           540
## 4 2013-01-01 05:00:00      183  2013     1     1     544           545
## 5 2013-01-01 06:00:00      116  2013     1     1     554           600
## 6 2013-01-01 05:00:00      150  2013     1     1     554           558
## 7 2013-01-01 06:00:00      158  2013     1     1     555           600
## 8 2013-01-01 06:00:00       53  2013     1     1     557           600
## 9 2013-01-01 06:00:00      140  2013     1     1     557           600
## 10 2013-01-01 06:00:00      138  2013     1     1     558           600
## # ... with 336,766 more rows, and 12 more variables: dep_delay <dbl>,
## #   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, distance <dbl>,
## #   hour <dbl>, minute <dbl>
```

Note. `select()` drops those variables not explicitly mentioned, but `rename()` only renames variables and keep other variables the same.

```
flights= rename(flights, tail_num = tailnum)
#view(flights)
```

### 3.2.4 Create new variables using existing variables: `mutate()`

It's often useful to add new columns that are functions of existing columns, `mutate()` always adds new columns at the end of your dataset. Remember in RStudio, the easiest way to see all the columns is `View()`. Note that you can use to columns that you've just created in `mutate`.



```

# select certain colmns
flights_sml <- select(flights,
  year:day,
  ends_with("delay"),
  distance,
  air_time
)

# adding new variables
mutate(flights_sml,
  gain = dep_delay - arr_delay,
  speed = distance / air_time*60,
  hours = air_time / 60,
  gain_per_hour = gain / hours,
  log_distance= log(distance)
)

```

```

## # A tibble: 336,776 x 12
##   year month   day dep_delay arr_delay distance air_time  gain speed
##   <int> <int> <int>     <dbl>     <dbl>     <dbl>     <dbl> <dbl> <dbl>
## 1  2013     1     1         2         11      1400      227    -9   370.
## 2  2013     1     1         4         20      1416      227   -16   374.
## 3  2013     1     1         2         33      1089      160  -31   408.
## 4  2013     1     1        -1        -18      1576      183    17   517.
## 5  2013     1     1        -6        -25       762      116    19   394.
## 6  2013     1     1        -4         12       719      150   -16   288.
## 7  2013     1     1        -5         19      1065      158  -24   404.
## 8  2013     1     1        -3        -14       229       53    11   259.
## 9  2013     1     1        -3         -8       944      140     5   405.
## 10 2013     1     1        -2         8       733      138   -10   319.
## # ... with 336,766 more rows, and 3 more variables: hours <dbl>,
## #   gain_per_hour <dbl>, log_distance <dbl>

```

### 3.2.5 Generate data summary: summarise()

summarise() is to summarize the observation with summary statistics.

```

# Summary one variable
summarise(flights, mean_delay = mean(dep_delay, na.rm = TRUE),
  sd_delay = sd(dep_delay, na.rm = TRUE),
  median_delay = median(dep_delay, na.rm = TRUE) )

## # A tibble: 1 x 3
##   mean_delay sd_delay median_delay
##   <dbl>     <dbl>         <dbl>
## 1    12.6     40.2           -2

```

It is mostly useful to pair it with group\_by(). So for the each subgroup defined by groupby, the summary statistics are provided.

```
# by destination
table(flights$dest)
```

```
##
##  ABQ  ACK  ALB  ANC  ATL  AUS  AVL  BDL  BGR  BHM  BNA  BOS
##  254  265  439    8 17215 2439  275  443  375  297 6333 15508
##  BQN  BTV  BUF  BUR  BWI  BZN  CAE  CAK  CHO  CHS  CLE  CLT
##  896 2589 4681  371 1781   36  116  864   52 2884 4573 14064
##  CMH  CRW  CVG  DAY  DCA  DEN  DFW  DSM  DTW  EGE  EYW  FLL
## 3524  138 3941 1525 9705 7266 8738  569 9384  213   17 12055
##  GRR  GSO  GSP  HDN  HNL  HOU  IAD  IAH  ILM  IND  JAC  JAX
##  765 1606  849   15  707 2115 5700 7198  110 2077   25  2720
##  LAS  LAX  LEX  LGA  LGB  MCI  MCO  MDW  MEM  MHT  MIA  MKE
## 5997 16174    1    1  668 2008 14082 4113 1789 1009 11728 2802
##  MSN  MSP  MSY  MTJ  MVY  MYR  OAK  OKC  OMA  ORD  ORF  PBI
##  572 7185 3799   15  221   59  312  346  849 17283 1536 6554
##  PDX  PHL  PHX  PIT  PSE  PSP  PVD  PWM  RDU  RIC  ROC  RSW
## 1354 1632 4656 2875  365   19  376 2352 8163 2454 2416 3537
##  SAN  SAT  SAV  SBN  SDF  SEA  SFO  SJC  SJU  SLC  SMF  SNA
## 2737  686  804   10 1157 3923 13331  329 5819 2467  284   825
##  SRQ  STL  STT  SYR  TPA  TUL  TVC  TYS  XNA
## 1211 4339  522 1761 7466  315  101  631 1036
```

```
by_dest <- group_by(flights, dest)
delay <- summarise(by_dest,
                    count = n(),
                    dist = mean(distance, na.rm = TRUE),
                    delay = mean(arr_delay, na.rm = TRUE)
)
(delay <- filter(delay, count > 20, dest != "HNL"))
```

```
## # A tibble: 96 x 4
##   dest count dist delay
##   <chr> <int> <dbl> <dbl>
## 1 ABQ    254 1826   4.38
## 2 ACK    265  199   4.85
## 3 ALB    439  143  14.4
## 4 ATL   17215 757. 11.3
## 5 AUS   2439 1514.  6.02
## 6 AVL    275  584.  8.00
## 7 BDL    443  116   7.05
## 8 BGR    375  378   8.03
## 9 BHM    297  866. 16.9
## 10 BNA   6333 758. 11.8
## # ... with 86 more rows
```

### 3.2.6 Put everything together with a pipe operator

The pipe operator, `%>%`, takes the output of one statement and makes it the input of the next statement. The above code can be shorted with pipe analysis to avoid naming new variables each time.

```
delays <- flights %>%
  group_by(dest) %>%
  summarise(
    count = n(),
    dist = mean(distance, na.rm = TRUE),
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  filter(count > 20, dest != "HNL")
delays
```

```
## # A tibble: 96 x 4
##   dest count  dist delay
##   <chr> <int> <dbl> <dbl>
## 1 ABQ    254 1826   4.38
## 2 ACK    265  199   4.85
## 3 ALB    439  143  14.4
## 4 ATL  17215  757.  11.3
## 5 AUS   2439 1514.   6.02
## 6 AVL    275  584.   8.00
## 7 BDL    443  116   7.05
## 8 BGR    375  378   8.03
## 9 BHM    297  866.  16.9
## 10 BNA   6333  758.  11.8
## # ... with 86 more rows
```

### 3.2.6 Exercises (20 mins)

- Find all flights that
  - Had an arrival delay of two or more hours
  - Flew to Houston (HOU)
  - Were operated by United or Southwest?
- Sort flights to find the most delayed flights.
- select dep\_time, dep\_delay, arr\_time, and arr\_delay from flights.

### 3.2.6 Exercises (1)

- Find all flights that
  - Had an arrival delay of two or more hours
  - Flew to Houston (HOU)
  - Were operated by United or Southwest?

Check ?flights to use variables: arr\_delay, dest and carrier?

```
(Sub <- filter(flights, arr_delay >= 2, dest == 'HOU', carrier == 'UA'))
```

```
## # A tibble: 0 x 19
## # ... with 19 variables: year <int>, month <int>, day <int>,
## #   dep_time <int>, sched_dep_time <int>, dep_delay <dbl>, arr_time <int>,
## #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tail_num <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
```

```
(Sub <- filter(flights, arr_delay >=2, dest=='HOU', carrier=='WN'))
```

```
## # A tibble: 567 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>   <int>         <int>         <dbl>   <int>
## 1  2013     1     1    1306           1300           6    1622
## 2  2013     1     1    1708           1700           8    2037
## 3  2013     1     2     734           700          34    1045
## 4  2013     1     2    1319           1305          14    1633
## 5  2013     1     2    1810           1655          75    2146
## 6  2013     1     3     704           700           4    1036
## 7  2013     1     3    1300           1300           0    1617
## 8  2013     1     3    1704           1700           4    2015
## 9  2013     1     4    1324           1300          24    1621
## 10 2013     1     5     704           700           4    1013
## # ... with 557 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tail_num <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dtm>
```

```
# check steps : very fast to filter through a large data
```

```
Sub1 <- filter(flights, arr_delay >=2)
Sub2 <- filter(Sub1, dest=='HOU')
Sub3 <- filter(Sub2, carrier=='WN')
```

```
dim(Sub1);dim(Sub2);dim(Sub3)
```

```
## [1] 127929    19
```

```
## [1] 868    19
```

```
## [1] 567    19
```

### 3.2.6 Exercises (2)

- Sort flights to find the most delayed flights.

```
# use desc() to re-order by a column in descending order: Departure delays
arrange(flights, desc(dep_delay))
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>   <int>         <int>         <dbl>   <int>
```

```
## 1 2013 1 9 641 900 1301 1242
## 2 2013 6 15 1432 1935 1137 1607
## 3 2013 1 10 1121 1635 1126 1239
## 4 2013 9 20 1139 1845 1014 1457
## 5 2013 7 22 845 1600 1005 1044
## 6 2013 4 10 1100 1900 960 1342
## 7 2013 3 17 2321 810 911 135
## 8 2013 6 27 959 1900 899 1236
## 9 2013 7 22 2257 759 898 121
## 10 2013 12 5 756 1700 896 1058
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tail_num <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dtm>
```

```
# use desc() to re-order by a column in descending order: arrival delays
arrange(flights, desc(arr_delay))
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>   <int>         <int>         <dbl>   <int>
## 1 2013     1     9     641           900         1301    1242
## 2 2013     6    15    1432          1935         1137    1607
## 3 2013     1    10    1121          1635         1126    1239
## 4 2013     9    20    1139          1845         1014    1457
## 5 2013     7    22     845          1600         1005    1044
## 6 2013     4    10    1100          1900          960    1342
## 7 2013     3    17    2321           810          911     135
## 8 2013     7    22    2257           759          898     121
## 9 2013    12     5     756          1700          896    1058
## 10 2013     5     3    1133          2055          878    1250
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tail_num <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dtm>
```

### 3.2.6 Exercises (3)

3. select dep\_time, dep\_delay, arr\_time, and arr\_delay from flights.

```
# use desc() to re-order by a column in descending order: arrival delays
select(flights, dep_time, dep_delay, arr_time, arr_delay)
```

```
## # A tibble: 336,776 x 4
##   dep_time dep_delay arr_time arr_delay
##   <int>     <dbl>   <int>     <dbl>
## 1     517         2     830         11
## 2     533         4     850         20
## 3     542         2     923         33
## 4     544        -1    1004        -18
## 5     554        -6     812        -25
## 6     554        -4     740         12
```

```
## 7      555      -5      913      19
## 8      557      -3      709     -14
## 9      557      -3      838      -8
## 10     558      -2      753       8
## # ... with 336,766 more rows
```

### 3.2.7 Data and Stat Exercise 2 : 25 min

```
Rpackage= "ggplot2"
if (! Rpackage %in% installed.packages()) install.packages(Rpackage)

library(ggplot2)
#? mpg
mpg
```

```
## # A tibble: 234 x 11
##   manufacturer model displ  year  cyl trans drv   cty   hwy fl   class
##   <chr>          <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
## 1 audi          a4      1.8  1999    4 auto~ f     18    29 p    comp~
## 2 audi          a4      1.8  1999    4 manu~ f     21    29 p    comp~
## 3 audi          a4      2    2008    4 manu~ f     20    31 p    comp~
## 4 audi          a4      2    2008    4 auto~ f     21    30 p    comp~
## 5 audi          a4      2.8  1999    6 auto~ f     16    26 p    comp~
## 6 audi          a4      2.8  1999    6 manu~ f     18    26 p    comp~
## 7 audi          a4      3.1  2008    6 auto~ f     18    27 p    comp~
## 8 audi          a4 q~    1.8  1999    4 manu~ 4     18    26 p    comp~
## 9 audi          a4 q~    1.8  1999    4 auto~ 4     16    25 p    comp~
## 10 audi         a4 q~    2    2008    4 manu~ 4     20    28 p    comp~
## # ... with 224 more rows
```

Check the data structure first. There may be different ways for each question below:

1. Summarize the highway miles per gallon (hwy) and city miles per gallon (cty)
2. Find the car with most highway miles per gallon (hwy). Find the car with most city miles per gallon (cty).
3. How many different classes of cars?
4. Summarize highway miles per gallon (hwy) data by the classes of the cars?
5. plot highway miles per gallon (hwy) data by a histogram.
6. plot highway miles in boxplot by the classes.

### 3.2.7 Answers using mostly examples learned earlier

Check Example data (1)

```
str(mpg)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame': 234 obs. of 11 variables:
## $ manufacturer: chr "audi" "audi" "audi" "audi" ...
## $ model : chr "a4" "a4" "a4" "a4" ...
## $ displ : num 1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
## $ year : int 1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
## $ cyl : int 4 4 4 4 6 6 6 4 4 4 ...
## $ trans : chr "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
## $ drv : chr "f" "f" "f" "f" ...
## $ cty : int 18 21 20 21 16 18 18 18 16 20 ...
## $ hwy : int 29 29 31 30 26 26 27 26 25 28 ...
## $ fl : chr "p" "p" "p" "p" ...
## $ class : chr "compact" "compact" "compact" "compact" ...
```

## Q1.

Summarize the highway miles per gallon (hwy) and city miles per gallon (cty)

**Method 1:** Use summary()

```
summary(mpg[, c('hwy', 'cty')])
```

```
##      hwy      cty
## Min.   :12.00 Min.   : 9.00
## 1st Qu.:18.00 1st Qu.:14.00
## Median :24.00 Median :17.00
## Mean   :23.44 Mean   :16.86
## 3rd Qu.:27.00 3rd Qu.:19.00
## Max.   :44.00 Max.   :35.00
```

**Method 2:** Use summarize()

```
mpg %>%
  summarise(min_hwy = min(hwy), mean_hwy = mean(hwy), med_hwy= median(hwy), max_hwy= max(hwy))
```

```
## # A tibble: 1 x 4
##   min_hwy mean_hwy med_hwy max_hwy
##   <int>    <dbl>   <dbl>   <int>
## 1      12     23.4     24      44
```

```
mpg %>%
  summarise(min_cty = min(cty), mean_cty = mean(cty), med_cty= median(cty), max_cty= max(cty))
```

```
## # A tibble: 1 x 4
##   min_cty mean_cty med_cty max_cty
##   <int>    <dbl>   <dbl>   <int>
## 1       9     16.9     17      35
```

## Q2.

Find the car with most highway miles per gallon (hwy). Find the car with most city miles per gallon (cty).

**Method 1:**

```
mpg[mpg$hwy== max(mpg$hwy),]
```

```
## # A tibble: 2 x 11
##   manufacturer model  displ  year  cyl trans drv   cty   hwy fl   class
##   <chr>          <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
## 1 volkswagen    jetta   1.9  1999    4 manu~ f     33   44 d    comp~
## 2 volkswagen    new b~   1.9  1999    4 manu~ f     35   44 d    subc~
```

```
mpg[mpg$cty== max(mpg$cty),]
```

```
## # A tibble: 1 x 11
##   manufacturer model  displ  year  cyl trans drv   cty   hwy fl   class
##   <chr>          <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
## 1 volkswagen    new b~   1.9  1999    4 manu~ f     35   44 d    subc~
```

Method 2:

```
filter(mpg, hwy== max(hwy))
```

```
## # A tibble: 2 x 11
##   manufacturer model  displ  year  cyl trans drv   cty   hwy fl   class
##   <chr>          <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
## 1 volkswagen    jetta   1.9  1999    4 manu~ f     33   44 d    comp~
## 2 volkswagen    new b~   1.9  1999    4 manu~ f     35   44 d    subc~
```

```
filter(mpg, cty== max(cty))
```

```
## # A tibble: 1 x 11
##   manufacturer model  displ  year  cyl trans drv   cty   hwy fl   class
##   <chr>          <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
## 1 volkswagen    new b~   1.9  1999    4 manu~ f     35   44 d    subc~
```

Method 3: order the data first by hwy and cty.

**Q3.**

How many different classes of cars in this data set? in Percentages?

```
table(mpg$class)
```

```
##
##   2seater    compact    midsize    minivan    pickup subcompact
##         5         47         41         11         33         35
##      suv
##      62
```



```
# percentage
round(prop.table(table(mpg$class))*100, 1)
```

```
##
##      2seater      compact      midsize      minivan      pickup subcompact
##        2.1        20.1        17.5         4.7        14.1        15.0
##      suv
##        26.5
```

#### Q4.

Summarize highway miles per gallon (hwy) data by the classes

Method 1: by()

```
with(mpg, by(hwy, class, summary))
```

```
## class: 2seater
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##      23.0   24.0   25.0   24.8   26.0   26.0
## -----
## class: compact
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##      23.0   26.0   27.0   28.3   29.0   44.0
## -----
## class: midsize
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##      23.00  26.00  27.00   27.29  29.00   32.00
## -----
## class: minivan
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##      17.00  22.00  23.00   22.36  24.00   24.00
## -----
## class: pickup
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##      12.00  16.00  17.00   16.88  18.00   22.00
## -----
## class: subcompact
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##      20.00  24.50  26.00   28.14  30.50   44.00
## -----
## class: suv
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##      12.00  17.00  17.50   18.13  19.00   27.00
```

Method 2: with a better data frame format

```
mpg %>%
  group_by(class)%>%
  summarise(min_hwy = min(hwy), mean_hwy = mean(hwy), med_hwy= median(hwy), max_hwy= max(hwy))
```

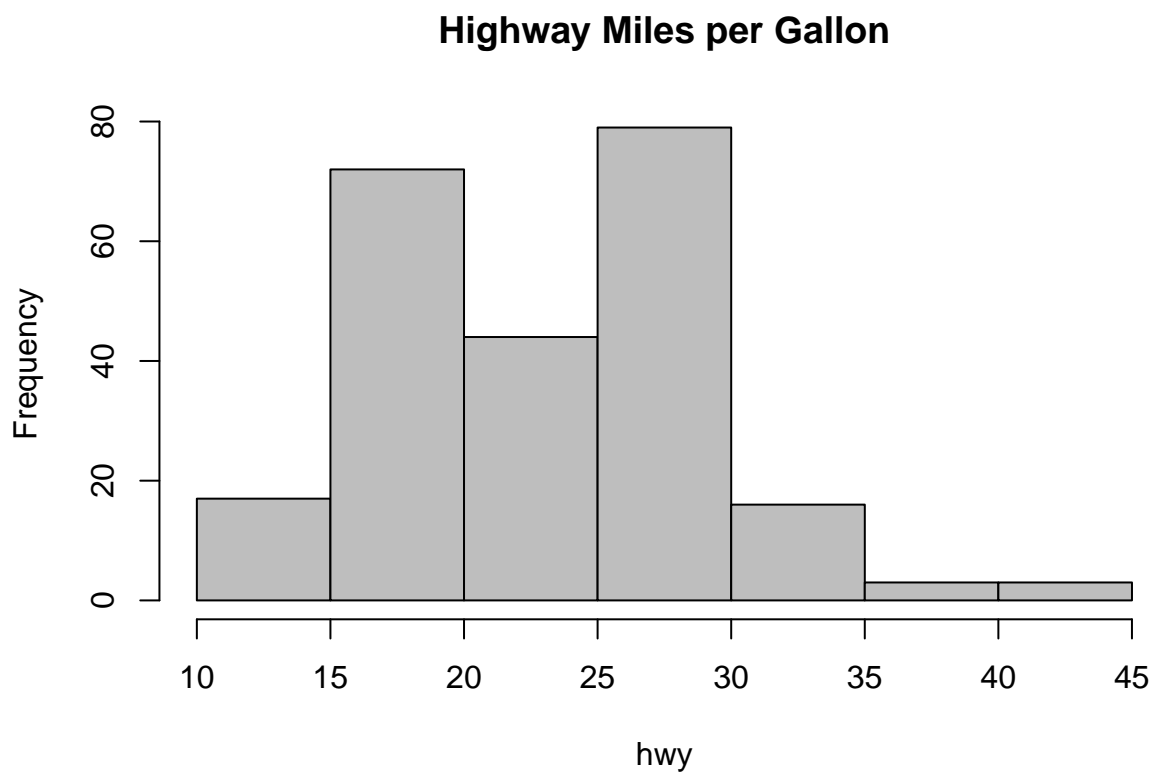
```
## # A tibble: 7 x 5
##   class      min_hwy mean_hwy med_hwy max_hwy
##   <chr>      <int>    <dbl>  <dbl>   <int>
## 1 2seater      23     24.8    25      26
## 2 compact      23     28.3    27      44
## 3 midsize      23     27.3    27      32
## 4 minivan      17     22.4    23      24
## 5 pickup       12     16.9    17      22
## 6 subcompact   20     28.1    26      44
## 7 suv          12     18.1    17.5    27
```

Q5.

Plot highway miles per gallon (hwy) data by a histogram

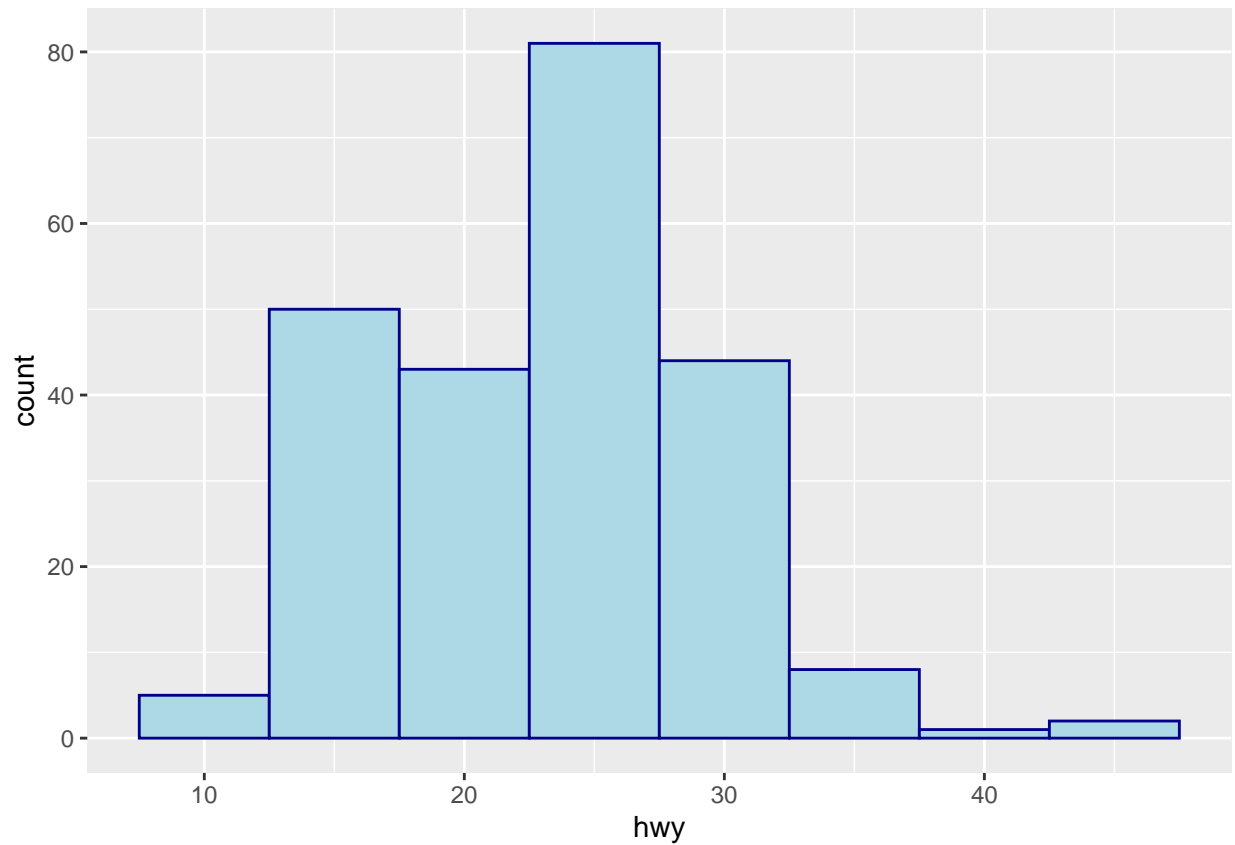
Method 1:

```
hist(mpg$hwy, col='gray', main='Highway Miles per Gallon', xlab='hwy')
```



Method 2:

```
ggplot(mpg, aes(x=hwy)) + geom_histogram(color="darkblue", fill="lightblue", binwidth=5) #binwidth=3
```

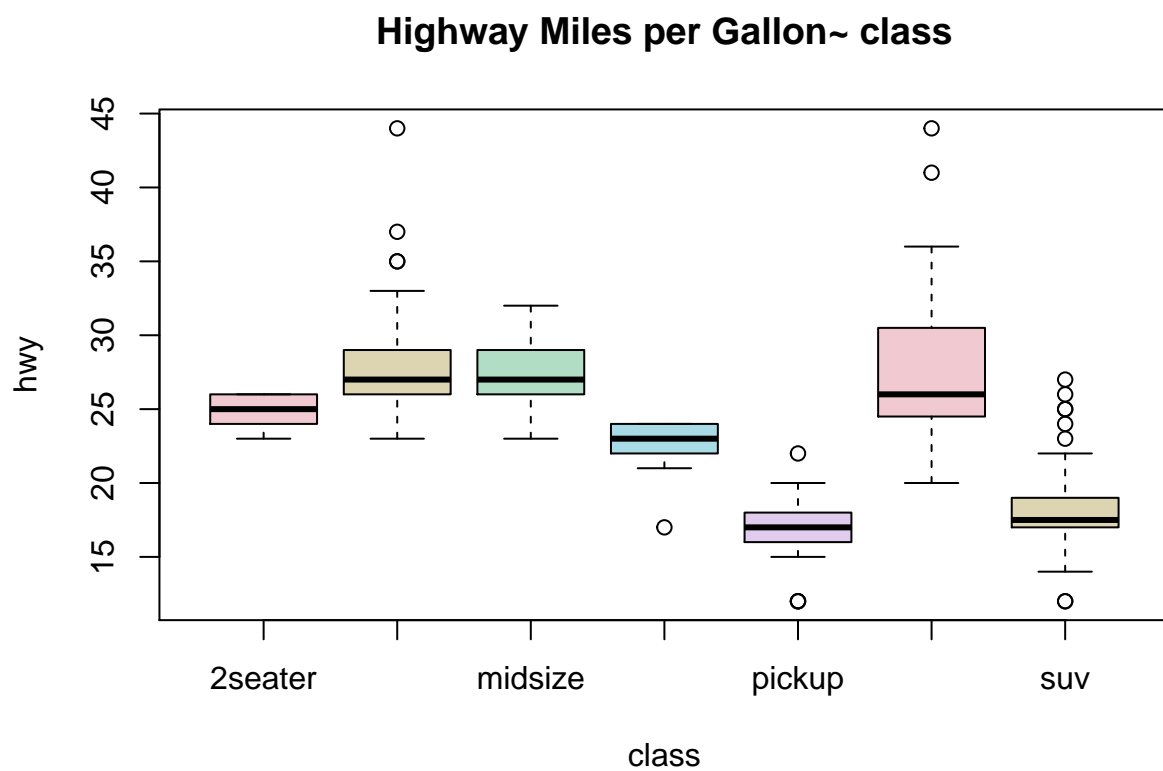


**Q6.**

plot highway miles in boxplot by the classes

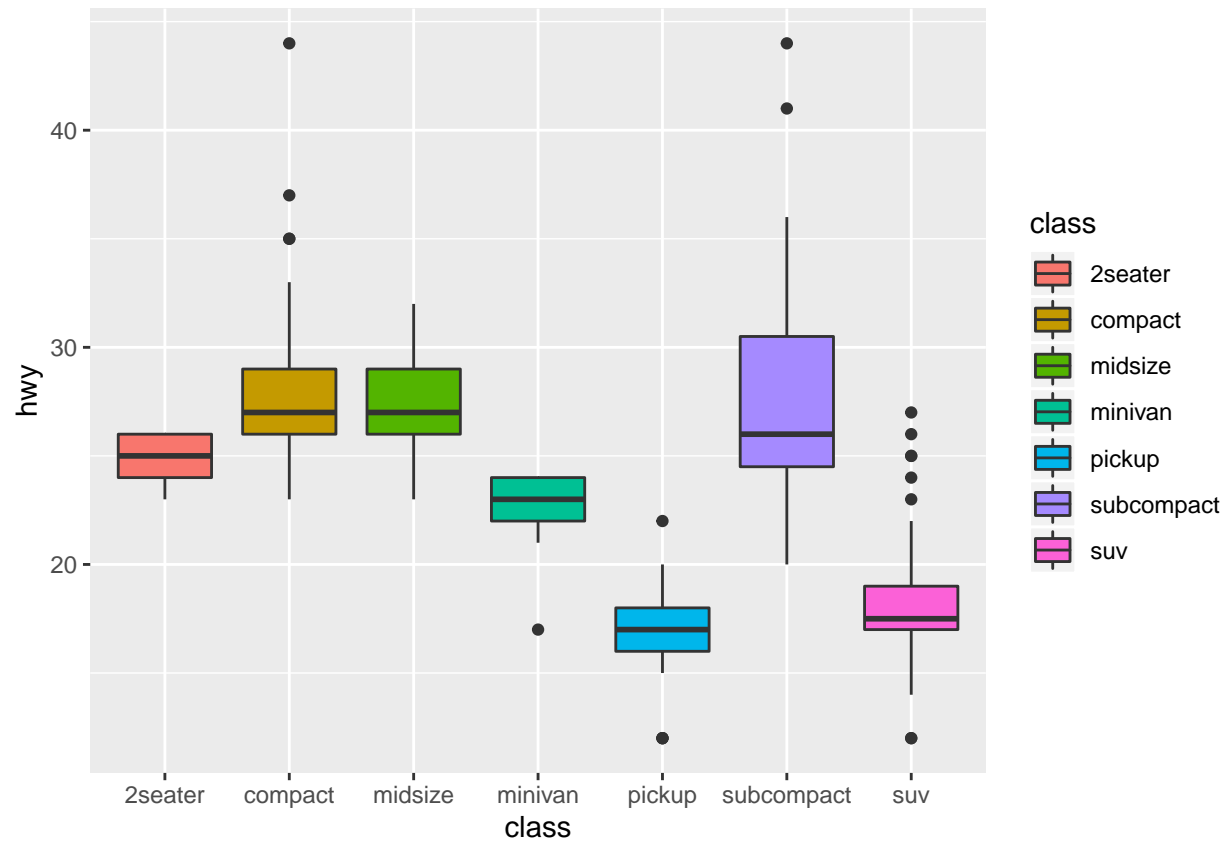
**Method 1:**

```
#library(colorspace)
boxplot(hwy ~ class, main='Highway Miles per Gallon~ class', data=mpg, col=rainbow_hcl(5, alpha=0.5))
```



Method 2:

```
ggplot(mpg, aes(x=class, y=hwy, fill=class)) +  
  geom_boxplot()
```



adding more components:

```
ggplot(mpg, aes(x=class, y=hwy, color=class)) +  
  geom_jitter(width=0.1)+  
  geom_boxplot(alpha=0.1)+  
  theme_bw() +  
  labs(  
    title = "Car Hwy by class",  
    x = "hwy",  
    y = "class"  
  )
```

