Elementary Statistics with R Programming

Table of contents

Preface

This is an R-manual that accompanies the textbook Triola (2022) for the courses STAT 2670: Elementary Statistics offered at Auburn University at Montgomery.

Credits:

- Jerome Goddard (Chapters 2 and 3)
- Yi Wang (Chapters 0, 1, 4 and 5; Overall editing)
- Wen Tang (Chapters 6 and 7)
- Jieun Park (Chapters 8 and 10)

Setting-up Computing Environment

Setting up your own computing environment on a personal computer

This is the recommended way and the advantage is that it's easy to handle files.

- Go to the website https://posit.co/download/rstudio-desktop/.
- Follow the two steps: 1) download and install R: Choose the appropriate operating system, and then choose "base" to "install R for the first time". You can simply accept all default options.

2) download Rstudio Desktop and Install it.

After installation, start R-Studio, and you are ready to use it.

Use R-Studio Cloud (No setting-up needed)

Alternatively, one can save the hassle of setting up on a personal computer and use the R-Studio Cloud for **free**. Here are the steps:

- Go to the website https://login.rstudio.cloud/.
- Either create a new account using an email address such as your AUM email or simply "Log in using Google" or click on other log-in alternative.

After log-in to your account, you are ready to use R Studio.

1 Quick Start to R Programming

1.1 1-minture R Programming Introduction

`Variables`: To create a variable, use the assignment operator `<-`. For example,

```
x <- 5
```

`Data Structures`: `R` supports various data structures such as `vectors`, `matrices`, `arrays`, `lists`, and `dataframes`. For instance, you can create a vector using `c()` like

```
my_vector <- c(1, 2, 3, 4, 5)
```

To index the second element of `my_vector`, use `[]` operator:

```
my_vector[2]
```

```
[1] 2
```

The output after running the code `my_vector[2]` is displayed as `[1] 2`, where `[1]` indicates the *first* line of the output, and `2` is the *value* of `my_vector[2]`.

To print a *formatted* output, use the built-in function `cat` to concatenate *strings* enclosed in double quotes `""`(or equivalently single quotes `' '`). Note `"\n"` represents a *newline* feed. For example:

```
cat("The first element in my_vector is:", my_vector[1], "\n")
```

```
The first element in my_vector is: 1
```

`Functions`: Functions in `R` are defined using the `function()` keyword. For example, you can create a function as follows:

```
my_sum <- function(arg1, arg2) {
  # function body
  return(arg1 + arg2)  #return the sum of arg1 and arg2
}
```

`Code comment`: A code comment starts with `#`. A comment line will not affect your code. When a R-code is executed, a comment line will be ignored by the R-code interpreter. **When you are following along with the code in this manual, you do not need to type the line starting with `#`.** They are provided to interpret the codes.

`Control Structures`: `R` supports typical control structures like `if-else` statements, `for` loops, `while` loops, etc.

`Packages`: R's functionality can be extended through packages. You can install packages using the `install.packages("package_Name")` function and load them using the `library("package_name")` function.

`Data Manipulation and Analysis`: `R` provides powerful tools for data manipulation and analysis. Packages like `dplyr` and `ggplot2` are commonly used for data manipulation and visualization, respectively.

`Help`: to access the help document, type in the R-console: `?function_name` or help(function_name). For example:, `?mean` or `help(mean)`.

2 Exploring Data with Tables and Graphs

r-function	Description
`data('dataset_name')`	Load a R built-in dataset named by `dataset_name`
`table(x)`	generate a frequency table for `x`
`length(x)`	return the length of the vector `x`
`cat`	concatenate strings and variable values for formatted print
`round (x, digits=2)`	round x element-wise with 2 decimal digits
`hist(x)`	plot histogram of the 1-D data `x`. Optional arguments: `main`: main title; `xlab`: x-label; `ylab`: y-label; `col`: color. Pass `freq=FALSE` for a relative frequency histogram.
`rnorm(n,mean,sd)`	generate `n` *random* values of standard normal distribution with the given `mean` and `sd`.
`runif(n, min, max)`	Generate `n` uniformly distributed *random* values between `min` (inclusive) and `max` (inclusive).
`rexp(n,r)`	Generate `n` exponentially distributed values with rate `r` at which events occurs on average
`qqnorm(x)`	plot a Q-Q plot of `x` against a standard normal distribution
`qqline(x)`	add a reference line to a Q-Q plot created by `qqnorm()` to indicate the theoretical Q-Q plot of a normal distribution.
`dotchart(x)`	create a dotplot for the 1-D data `x`
`stem(x)`	create a stem plot for the 1-D data `x`
`plot(x,y, type='p')`	plot the scatter plot of data sets (x,y). If `x` is a dataframe, and no `y` is provided, then plot each column of `x` against the dataframe index. type: `'p'` for points, `'l'` for line, and `'b'` for both.
`ts.plot(x)`	plot the time series x.
`pie(x, labels)`	plot the pie chart of `x` using `labels`
`pareto.chart(x)`	Create Pareto chart of the 1-D data `x` using package `qcc`
`cor.test(x,y)`	perform the correlation test between `x` and `y`. The function returns an object that contains three attributes: `estimate`: which is the correlation `r` value depending on a `method`: "pearson"(default),"kendall", or "spearman"; `p.value`: which is the test statistics `p`-value; `conf.int`: which is the confidence interval for the default `conf.level` 0.95. The `alternative` hypothesis is "two.sided" (default), "less", "greater".
`lm(y~x, data)`	perform the linear regression of `y~x`, where `y`,`x` are column names in the dataframe `data`.

r-function	Description
`abline(reg_model, col="red")`	add a regression line from the `reg_model` in `red`.
`abline(a,b)`	add a line with intercept `a` and slope `b`
`abline(h=y_value)`	add a horizontal line at `y=y_value`
`abline(v=x_value)`	add a vertical line at `x=x_value`

2.1 Frequency distribution shows the count or frequency of each unique value or category in a dataset, providing a clear picture of how data is distributed across different values or groups.

2.1.1 Frequency distributions

The R command `table()` will generate a frequency distribution for any data set. Let's analyze example test scores from a fictional math class. Notice the first row of the output is the data name, the second row is the actual data, and the third row contains the number of times each data value appears.

```
# Load test data into a variable names scores
scores = c(95, 90, 85, 85, 87, 74, 75, 64, 85, 84, 87, 15, 20, 75, 75, 90, 75)

# Create a frequency table for the scores data
table(scores)
```

```
scores
15 20 64 74 75 84 85 87 90 95
 1  1  1  1  4  1  3  2  2  1
```

2.1.2 Relative frequency distributions

Relative frequency distributions give similar information as a frequency distribution except they use percentages. Let's examine the same `scores` data set defined above. Notice in the output that the second row is the actual data and the third row contains the relative frequencies (rounded to two decimal places).

```
# Create a relative frequency table for the scores data
rftable <- table(scores)/length(scores)
round(rftable, digits = 2)
```

```
scores
  15   20   64   74   75   84   85   87   90   95
0.06 0.06 0.06 0.06 0.24 0.06 0.18 0.12 0.12 0.06
```

2.2 Histograms

A histogram is a bar chart that shows how often different values occur in a dataset.

2.2.1 Histogram

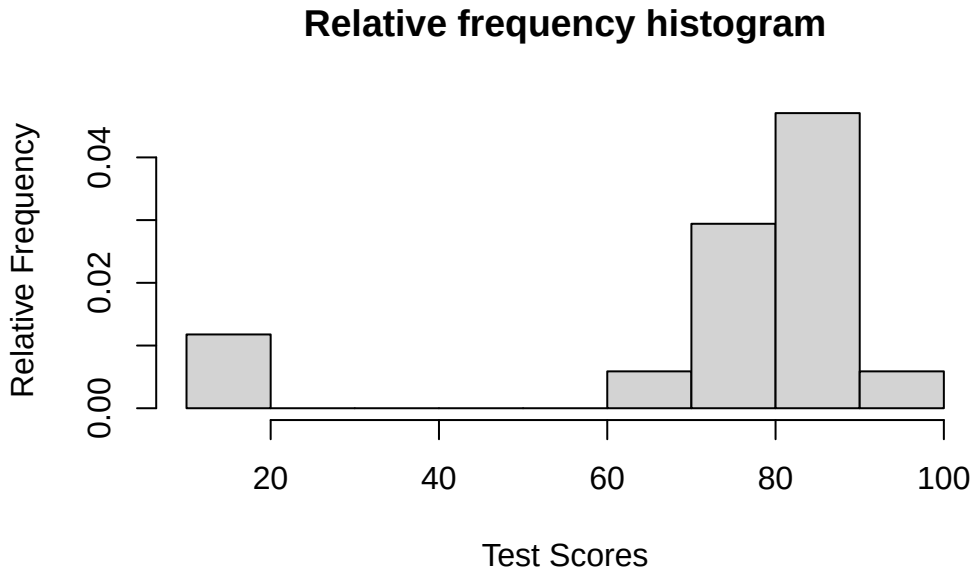
The command `hist()` will generate a histogram for any data. Here is an example using our `scores` data from above. Notice the x-axis represents the actual scores and the y-axis shows the frequency of the data points. We will use the following command options: 1) `main` allows the title to be specified, 2) `xlab` sets the x-axis label, and 3) `ylab` sets the y-axis label.

```
# Create a histogram and customize the axis labels and title
# main is the Plot title, xlab is the x-axis label, & ylab is the y-axis label
hist(scores, main = "Histogram for test scores", xlab = "Test Scores",
     ylab = "Frequency")
```

2.2.2 Relative frequency histogram

A relative histogram is a bar chart that displays the proportion or percentage of values in different bins within a dataset, providing a relative view of the data distribution.

```
# Using freq = FALSE in hist() will create a relative frequency histogram
hist(scores, freq = FALSE, main = "Relative frequency histogram",
     xlab = "Test Scores", ylab = "Relative Frequency")
```

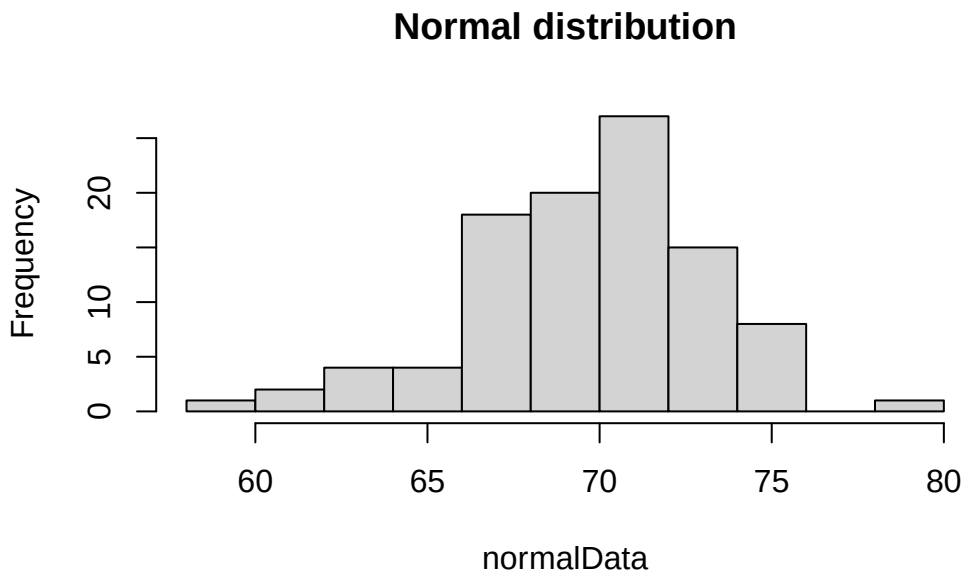


2.2.3 Common distributions

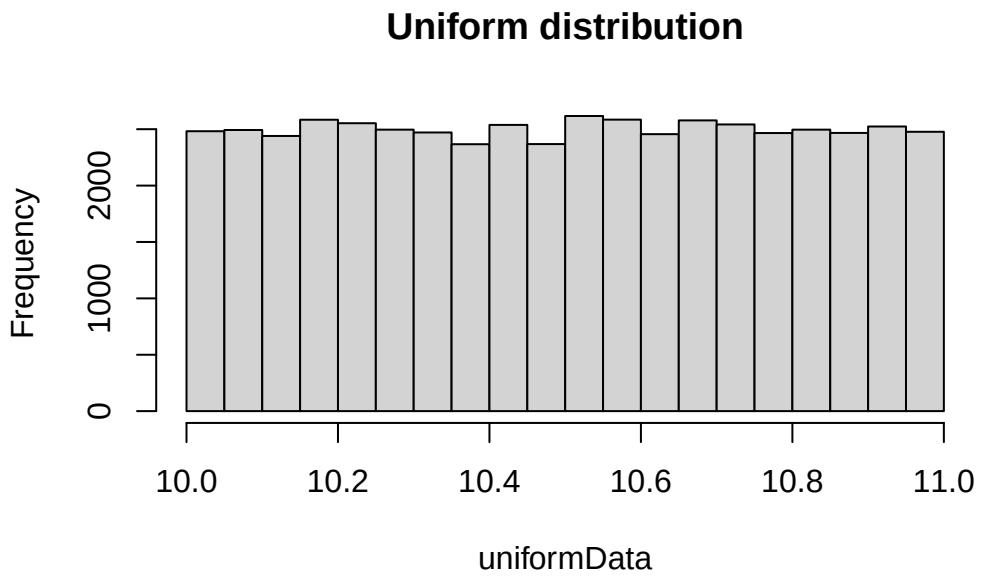
Normal distributions are bell-shaped and symmetrical, uniform distributions have constant probabilities across a range, skewed right distributions are characterized by a long tail on the right side, and skewed left distributions have a long tail on the left side, each exhibiting distinct patterns of data distribution. We will use the `hist()` command to explore each of these common distributions in the code below.

```
# Sample normal distribution
n <- 100
mean <- 69
sd <- 3.6
normalData <- rnorm(n, mean, sd)

# Sample uniform distribution using the command runif
```

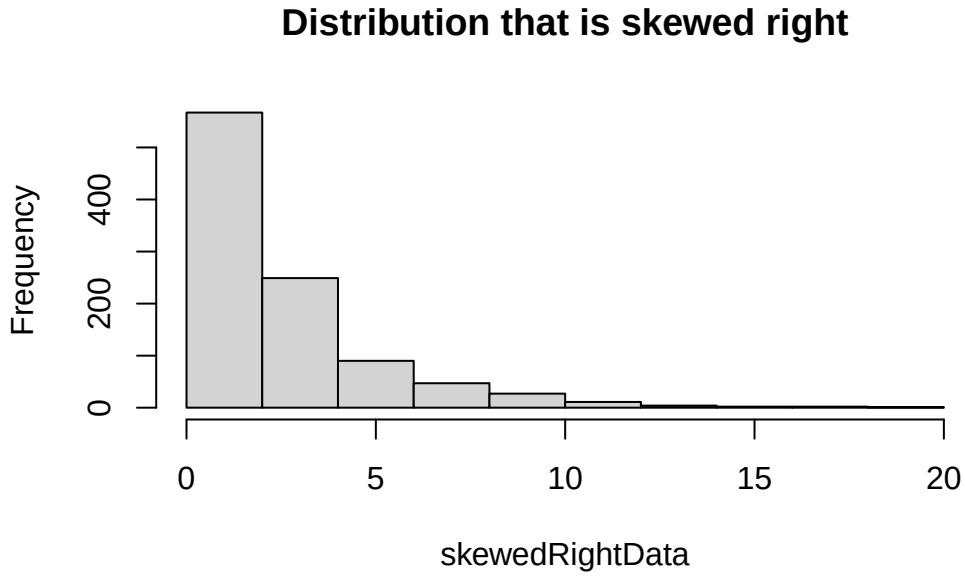
```r
uniformData <- runif(50000, min = 10, max = 11)

# Sample of a distribution that is skewed right
skewedRightData <- rexp(1000, 0.4)

# Sample of a distribution that is skewed left
skewedLeftData <- 1 - rexp(1000, 0.2)

# Create histogram of normal data
hist(normalData, main = "Normal distribution")
```



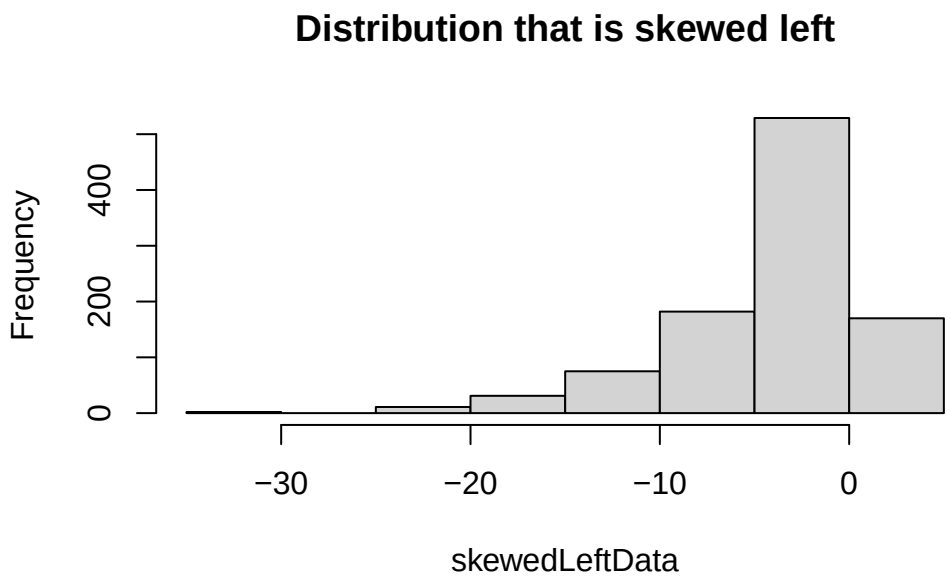

```r
# Create histogram of uniform data
hist(uniformData, main = "Uniform distribution")
```




```r
# Create histogram of skewed right data
hist(skewedRightData, main = "Distribution that is skewed right")
```



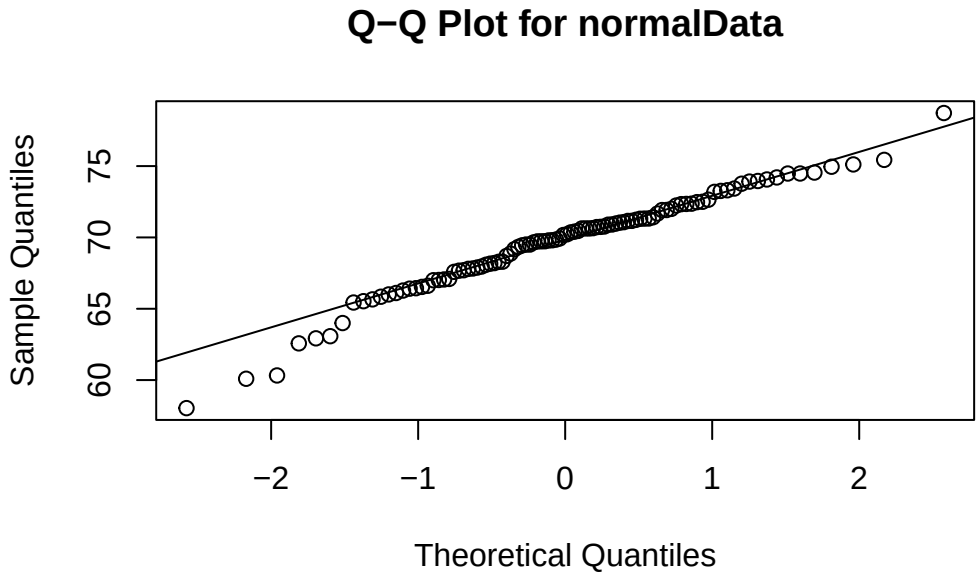

```r
# Create histogram of skewed left data
hist(skewedLeftData, main = "Distribution that is skewed left")
```



2.2.4 Normal quantile plots

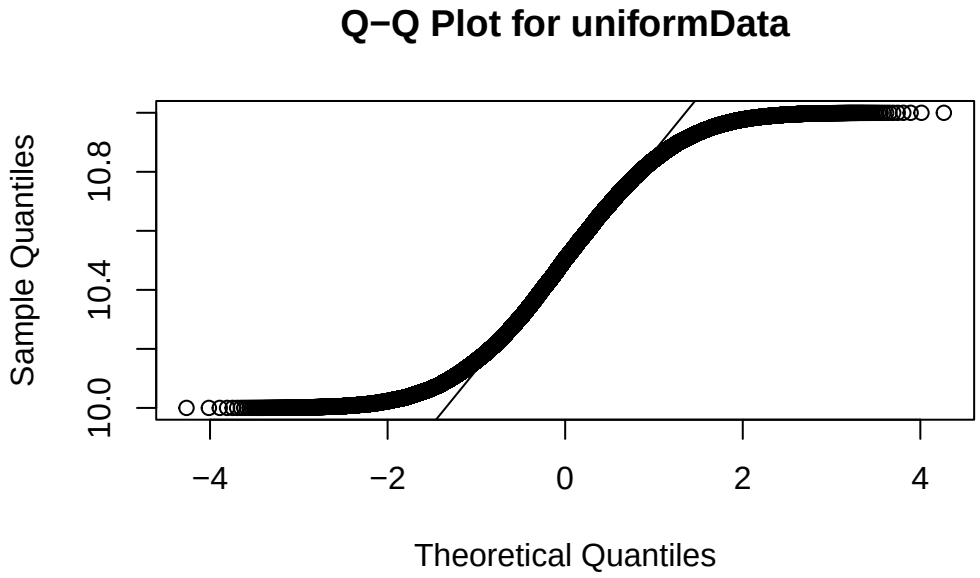
A normal quantile plot, also known as a Q-Q plot, is a graphical tool used to assess whether a dataset follows a normal distribution by comparing its quantiles (ordered values) to the quantiles of a theoretical normal distribution; if the points closely follow a straight line, the data is approximately normal. Let's use the commands `qqnorm()` and `qqline()` to visually test which data set is most likely a sample from a normal distribution.

```
# Test normalData from above
qqnorm(normalData, main = "Q-Q Plot for normalData")
qqline(normalData)
```



Notice that the `normalData` Q-Q plot shows the points close to the Q-Q line over the entire x-axis.

```
# Test uniformData from above
qqnorm(uniformData, main = "Q-Q Plot for uniformData")
qqline(uniformData)
```

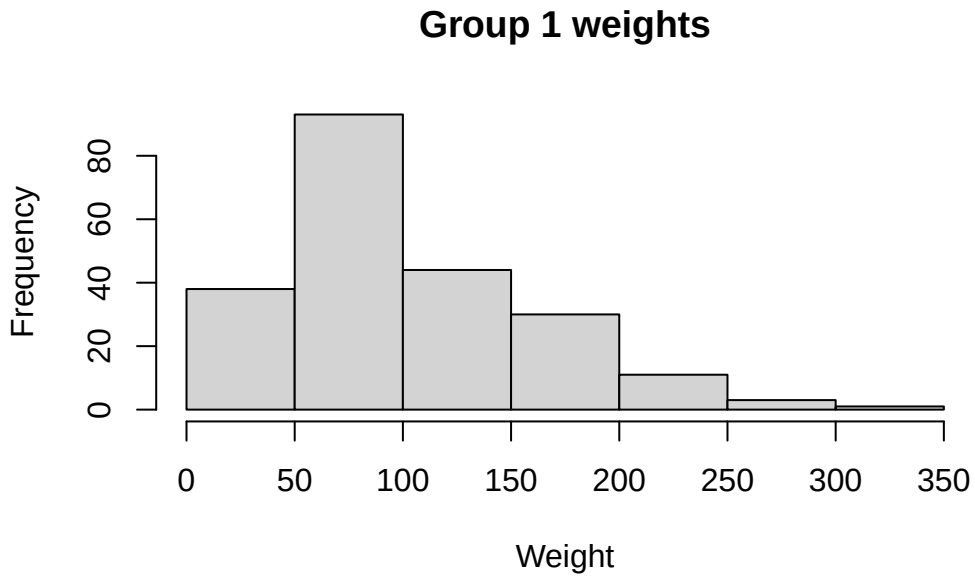


For the `uniformData` dataset, the Q-Q plot shows good agreement between points and line in the center (around 0) but not on either left or right of the x-axis.

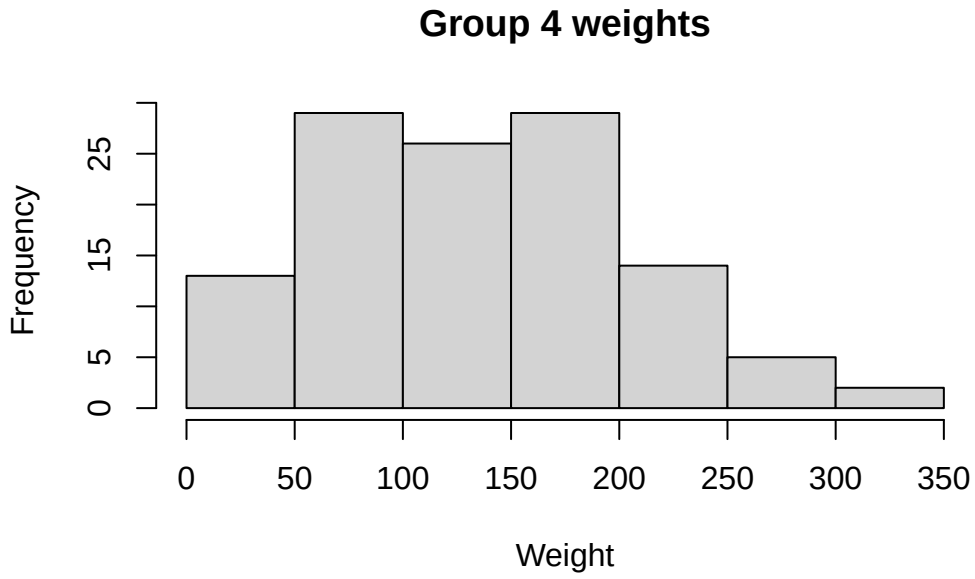
2.2.5 Let's put it all together!

In the built-in R dataset `ChickWeight`, weights are taken from several groups of chickens that were fed various diets. We are asked to use both histogram and Q-Q plots to determine if weights from group 1 and 4 are approximately normal, uniform, skewed left, or skewed right.

```r
# Load data from the built-in dataset into a variable named ChickWeight
data("ChickWeight")

# Extract all weights from group 1
group1Weights <- ChickWeight[ChickWeight$Diet == 1, 1]

# Extract all weights from group 4
group4Weights <- ChickWeight[ChickWeight$Diet == 4, 1]

# Create a histogram of weights from group 1
hist(group1Weights, main = "Group 1 weights", xlab = "Weight", ylab = "Frequency")
```

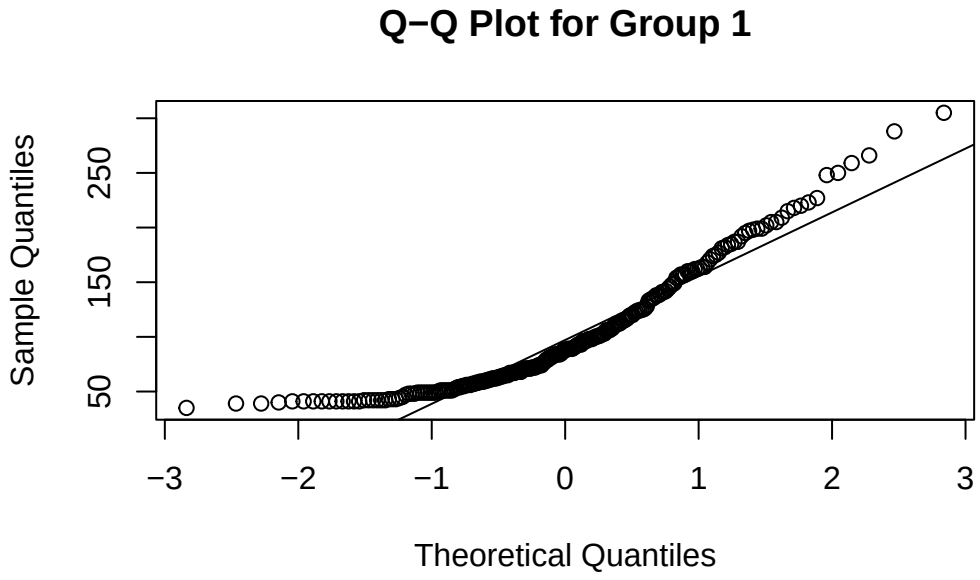



```r
# Create a histogram of weights from group 4
hist(group4Weights, main = "Group 4 weights", xlab = "Weight", ylab = "Frequency")
```

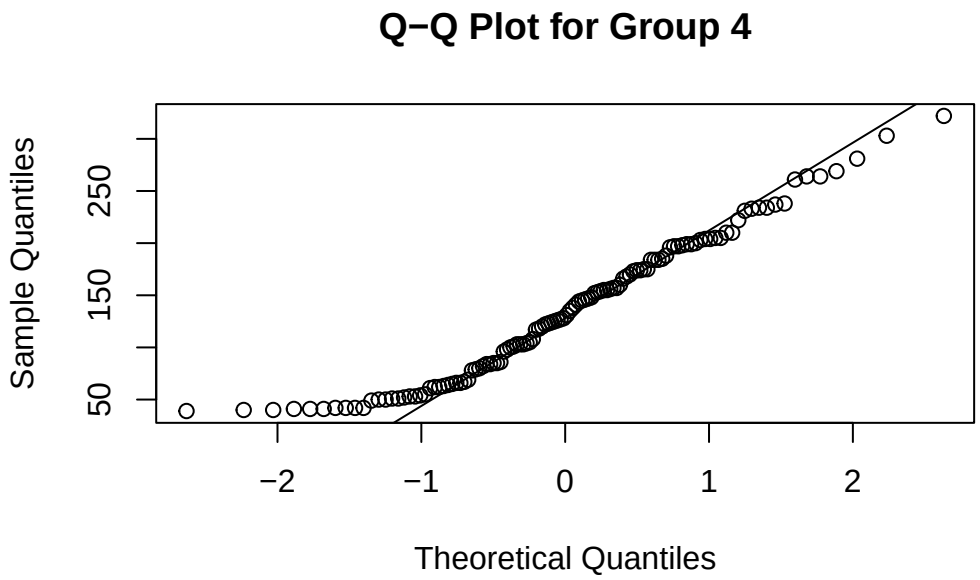


Is the group 1 distribution approximately normal or would a different distribution be a better fit? What about group 4? Now, let's confirm our results using Q-Q plots.

```
# Test group1Weights from above
qqnorm(group1Weights, main = "Q-Q Plot for Group 1")
qqline(group1Weights)
```



```r
# Test group4Weights from above
qqnorm(group4Weights, main = "Q-Q Plot for Group 4")
qqline(group4Weights)
```



Does the Q-Q plot confirm your guess from our visual inspection? Which group is closer to a normal distribution?

2.3 Graphs that enlighten and graphs that deceive

R has many commands to illustrate data revealing hidden patterns that could be otherwise missed. We will explore several of these commands using three different datasets:

(A) **Chicken Weights:** Same data used in Section 2.2: two different groups of chickens fed with different feed.

(B) **Airline Passengers:** A time series of the number of airline passengers in the US by month.

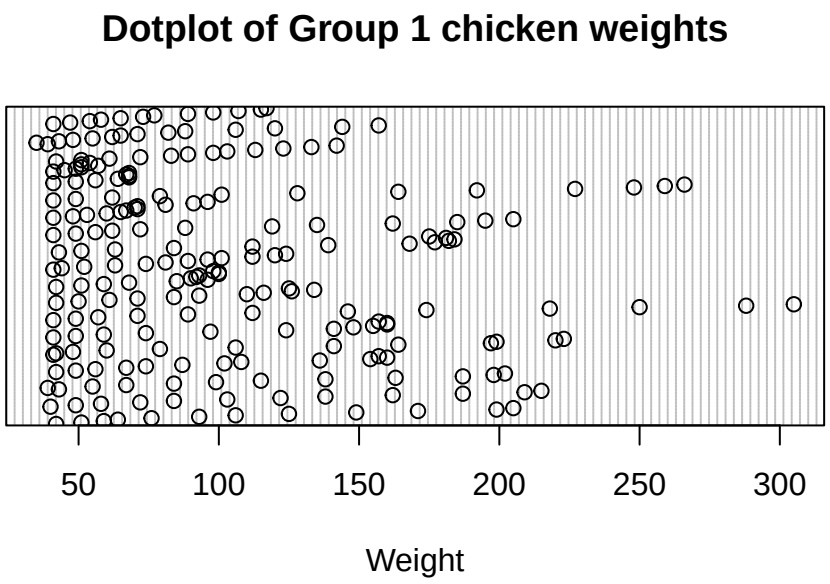
(C) **US Personal Expenditure** Average personal expenditures for adults in the US from 1960.

Below we will load these data sets when we need them.

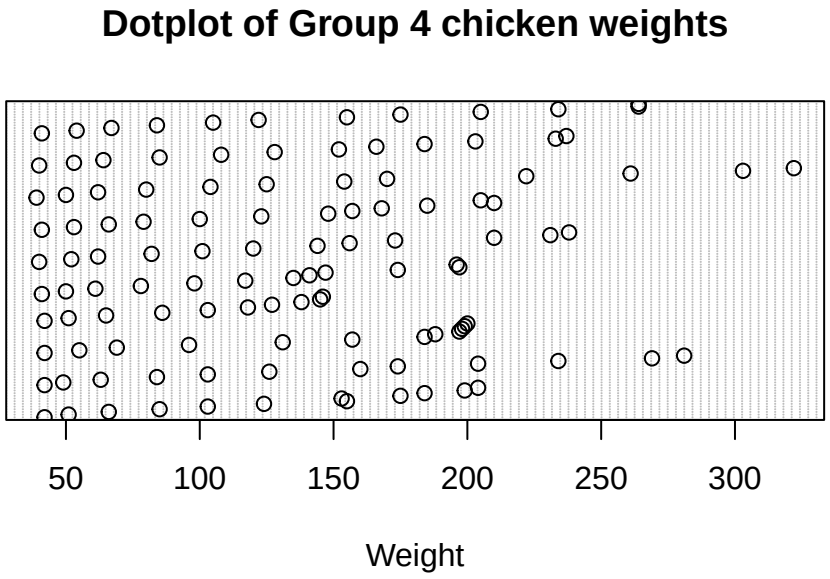
2.3.1 Dotplot

A dotplot is a simple graphical representation of data in which each data point is shown as a dot above its corresponding value on a number line, helping to visualize the distribution and identify patterns in a dataset. With our data previously loaded from the previous run, let's create a dotplot of the data. First for weights of both groups of chickens.

```
# Chicken weights:
# Load data from the built-in dataset into a variable named ChickWeight
data("ChickWeight")

# Extract all weights from group 1
group1Weights <- ChickWeight[ChickWeight$Diet == 1, 1]

# Extract all weights from group 4
group4Weights <- ChickWeight[ChickWeight$Diet == 4, 1]

# Dotplot for group 1 chickens
dotchart(group1Weights, main = "Dotplot of Group 1 chicken weights", xlab = "Weight")
```




```
# Dotplot for group 4 chickens
dotchart(group4Weights, main = "Dotplot of Group 4 chicken weights", xlab = "Weight")
```



2.3.2 Stem plot

A stem plot, also known as a stem-and-leaf plot (or just stemplot), is a graphical representation of data where each data point is split into a "stem" (the leading digit or digits) and "leaves" (the trailing digits) to display the individual values in a dataset while preserving their relative order, making it easier to see the distribution and identify key data points. Let's create a stemplot for our chicken weight data from above.

```
# Stemplot of group 1 weights
stem(group1Weights)
```

```
  The decimal point is 1 digit(s) to the right of the |

   2 | 599
   4 | 011111111112222223334578889999999901111112344556667788999
   6 | 00112223344555777788880111112223444679 9
   8 | 11234444578899990123336667888 9
  10 | 0011233666780222355679
  12 | 00234455683456889
  14 | 112468945777
  16 | 0002234481457
  18 | 124577257899
  20 | 255958
```

```
  22 | 037
  24 | 809
  26 | 6
  28 | 8
  30 | 5
```

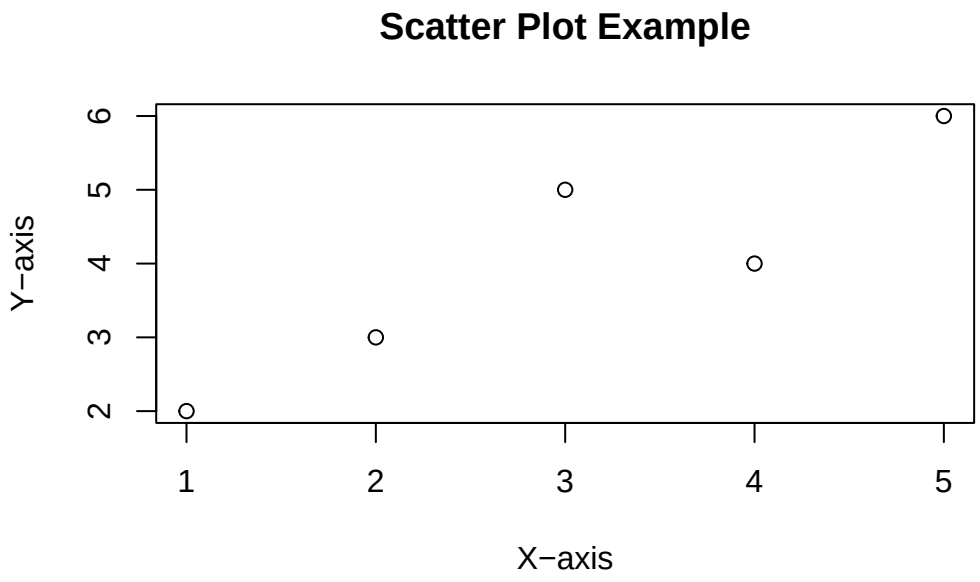
```r
# Stemplot of group 4 weights
stem(group4Weights)
```

```
  The decimal point is 1 digit(s) to the right of the |

   2 | 9
   4 | 0011122229001123345
   6 | 122345667989
   8 | 024455668
  10 | 0133345878
  12 | 02345678158
  14 | 14567823455677
  16 | 068034455
  18 | 44458677899
  20 | 03445500
  22 | 2134478
  24 |
  26 | 1449
  28 | 1
  30 | 3
  32 | 2
```

2.3.3 Scatter Plot

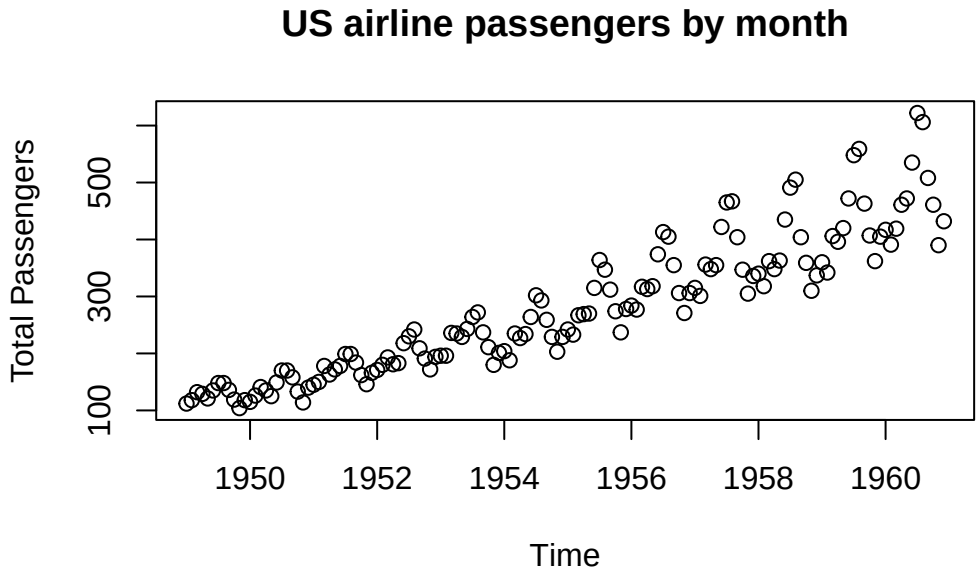
A scatter plot is a graphical representation that displays individual data points on a two-dimensional plane, with one variable on the x-axis and another on the y-axis, allowing you to visualize the relationship, pattern, or correlation between the two variables.

```r
# Sample data
x <- c(1, 2, 3, 4, 5)
y <- c(2, 3, 5, 4, 6)

# Create scatter plot
plot(x, y, main = "Scatter Plot Example", xlab = "X-axis", ylab = "Y-axis")
```



Real Data Example Let's create a scatter plot using the R command `plot()` for the US airline passengers by month using our data from above.

```
# Airline passengers:
# Load from the built-in dataset. This will create a variable named AirPassengers
# containing the time series.
data("AirPassengers")

# Plot each column against the row index (year). type="p" for points.
plot(AirPassengers, main = "US airline passengers by month", xlab = "Time",
     ylab = "Total Passengers", type = "p")
```

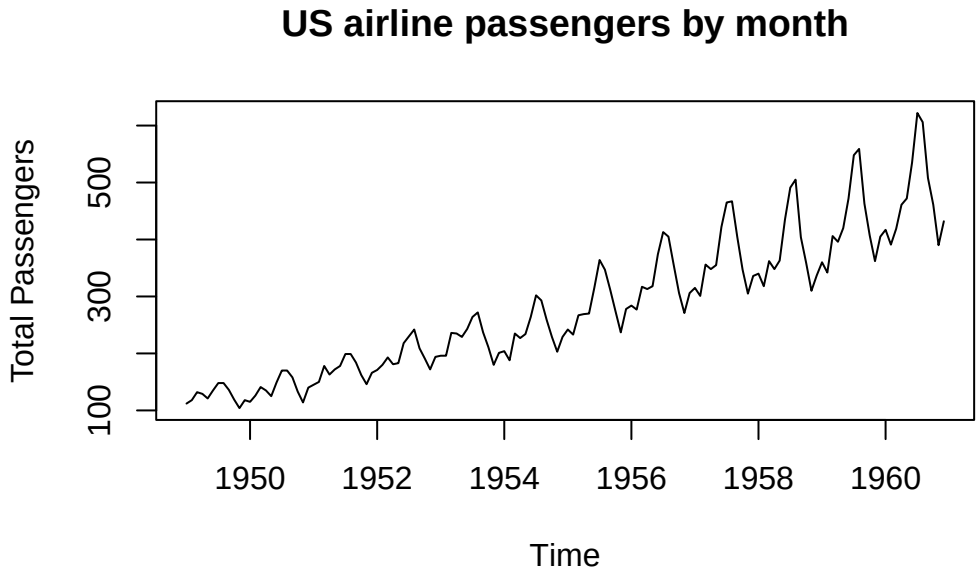


Notice the overall increasing trend of the data.

2.3.4 Time-series Graph

A time series is a sequence of data points collected or recorded at successive points in time, typically at evenly spaced intervals, and a time series graph visually represents this data over time, allowing us to observe trends, patterns, and changes in the data's behavior. Let's use the R command `ts.plot()` to plot the total US airline passengers by month using our data from above.

```
# Time series plot of AirPassengers
ts.plot(AirPassengers, main = "US airline passengers by month", xlab = "Time",
        ylab = "Total Passengers")
```

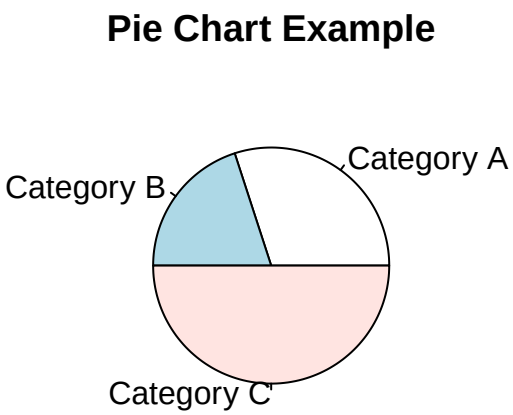


The time series graph shows several interesting phenomena: 1) airline travel is seasonal with the same basic pattern repeated each year and 2) the overall trend is increasing.

2.3.5 Pie Chart

A pie chart is a circular graph that visually represents data as slices, with each slice showing the proportion or percentage of different categories in the whole dataset.
A pie chart can be easily created as in the followng example:

```
# Creating sample data
data <- c(30, 20, 50) # Example data for the pie chart
labels <- c("Category A", "Category B", "Category C") # Labels for each category

# Creating a pie chart
pie(data, labels = labels, main = "Pie Chart Example")
```

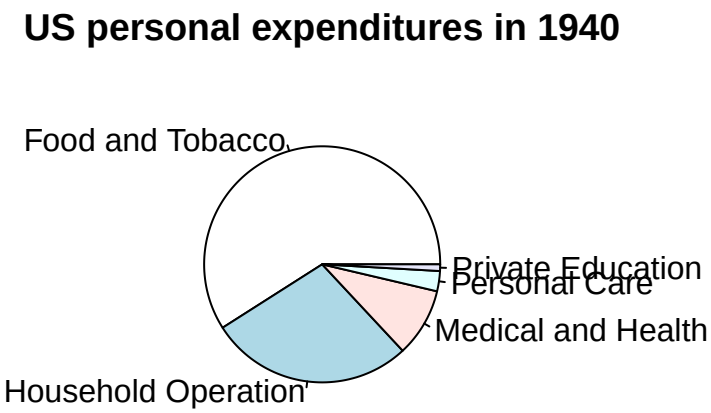


Real Data Example

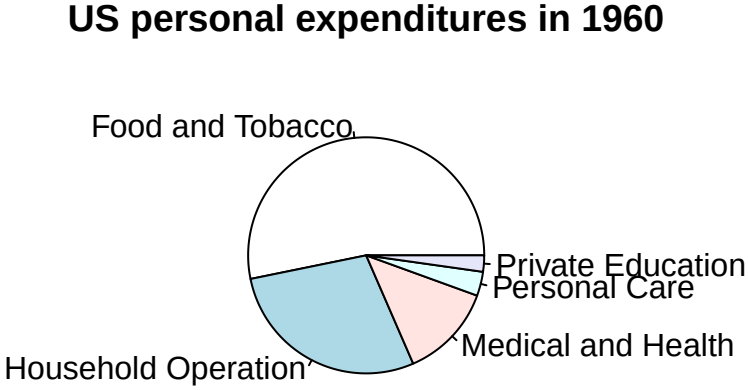
Let's use a pie chart to visualize the difference between average personal expenditure in the US in 1940 vs 1960 using `USPeronalExpenditure` defined above.

```
# Personal expenditure:
# Load from the built-in dataset.  This will create a variable named
# USPersonalExpenditure containing the data.
data("USPersonalExpenditure")

# We now extract only information from 1940
expenditures1940 <- USPersonalExpenditure[1:5]

# We now extract only information from 1960
expenditures1960 <- USPersonalExpenditure[21:25]

# Define categories for expenditure data
cats <- c("Food and Tobacco", "Household Operation", "Medical and Health",
          "Personal Care", "Private Education")

# Define category names from cats above
names(expenditures1940) <- cats
names(expenditures1960) <- cats

# Pie chart of 1940 expenditures: labels allows us to name the categories as
# defined in cats above
```

```r
pie(expenditures1940, main = "US personal expenditures in 1940")
```




```r
# Pie chart of 1960 expenditures: labels allows us to name the categories as
# defined in cats above
pie(expenditures1960, main = "US personal expenditures in 1960")
```



2.3.6 Pareto Chart

A Pareto chart is a specialized bar chart that displays data in descending order of frequency or importance, highlighting the most significant factors or categories, making it a visual tool for prioritization and decision-making. Let's use the `expenditures1940` and `expenditures1960` data from above to illustrate the usefulness of a Pareto chart.

The first time you run this code, you will need to install the following package. After this initial run, you can skip running this code:

```
# Installs the package 'qcc'.  ONLY RUN THIS CODE ONCE!
install.packages('qcc')
```

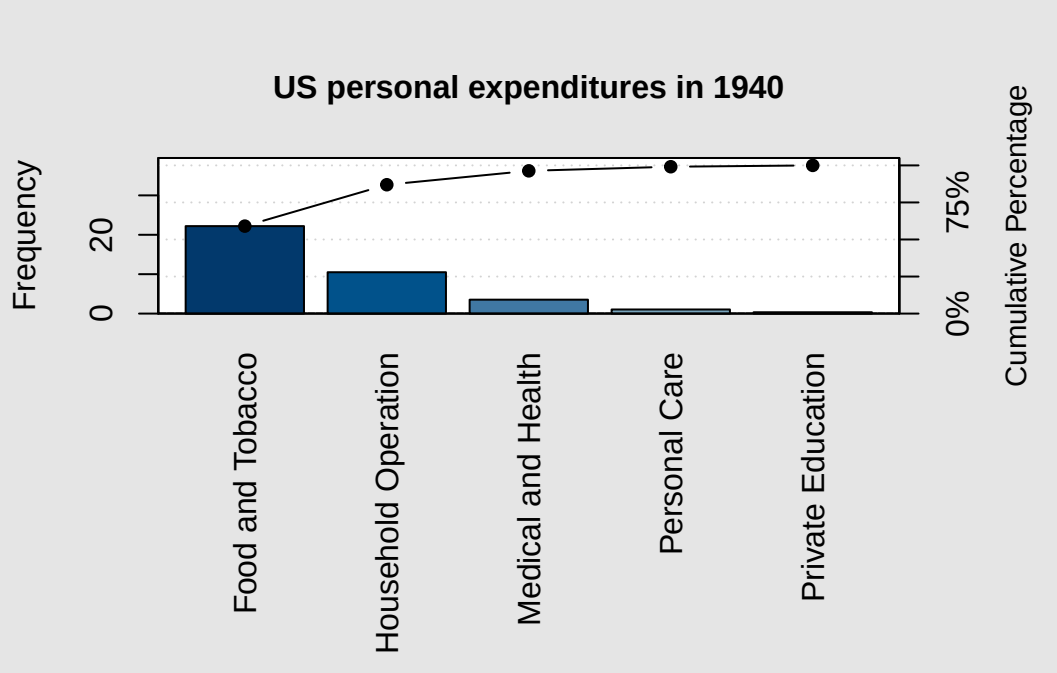
Now, let's create Pareto charts for the 1940 and 1960 expenditure data.

```
# Load 'qcc' package
library(qcc)
```

```
Package 'qcc' version 2.7
```

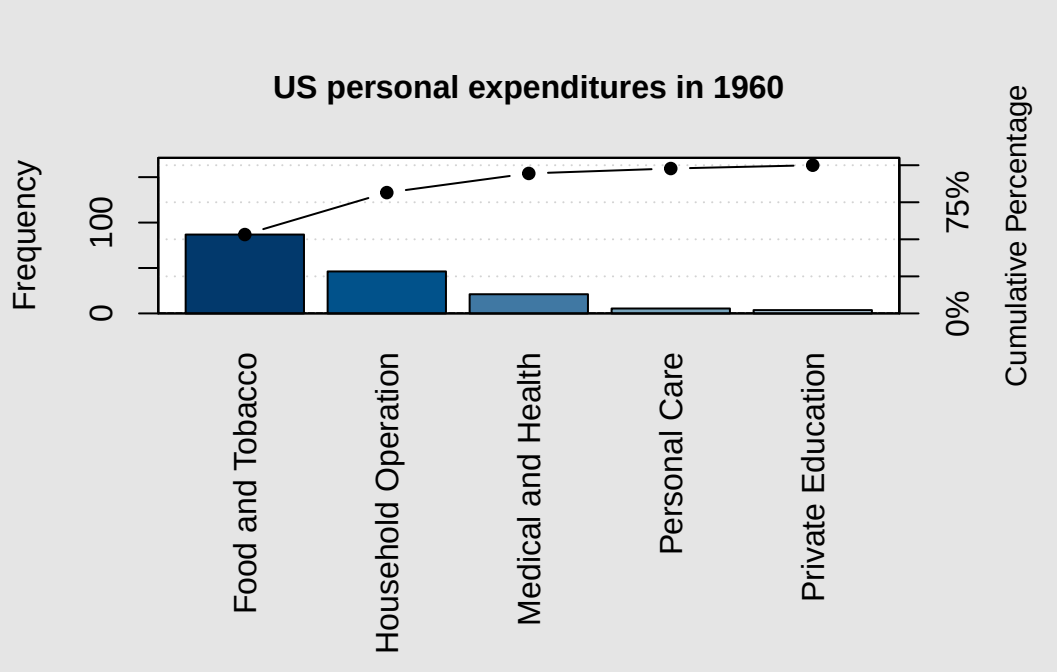
```
Type 'citation("qcc")' for citing this R package in publications.
```

```
# Create the Pareto chart for 1940 data
pareto.chart(expenditures1940, xlab = "", ylab="Frequency",
             main = "US personal expenditures in 1940")
```



```
Pareto chart analysis for expenditures1940
                      Frequency   Cum.Freq.  Percentage Cum.Percent.
  Food and Tobacco     22.2000000 22.2000000 59.0252852   59.0252852
  Household Operation  10.5000000 32.7000000 27.9173646   86.9426498
  Medical and Health    3.5300000 36.2300000  9.3855521   96.3282019
  Personal Care         1.0400000 37.2700000  2.7651485   99.0933503
  Private Education     0.3410000 37.6110000  0.9066497  100.0000000
```

```
# Create the Pareto chart for 1960 data
pareto.chart(expenditures1960, xlab = "", ylab="Frequency",
             main = "US personal expenditures in 1960")
```




```
Pareto chart analysis for expenditures1960
                      Frequency  Cum.Freq. Percentage Cum.Percent.
  Food and Tobacco     86.800000  86.800000  53.205835    53.205835
  Household Operation  46.200000 133.000000  28.319235    81.525070
  Medical and Health   21.100000 154.100000  12.933677    94.458747
  Personal Care         5.400000 159.500000   3.310040    97.768788
  Private Education     3.640000 163.140000   2.231212   100.000000
```

2.3.7 Let's put it all together!

Using the built-in dataset for quarterly profits of the company Johnson & Johnson, load the data and view it using this code.

```
# Johnson & Johnson Profits:
# Load data from the built-in dataset into a variable named JohnsonJohnson
data("JohnsonJohnson")

JohnsonJohnson
```

```
      Qtr1  Qtr2  Qtr3  Qtr4
1960  0.71  0.63  0.85  0.44
1961  0.61  0.69  0.92  0.55
1962  0.72  0.77  0.92  0.60
1963  0.83  0.80  1.00  0.77
1964  0.92  1.00  1.24  1.00
1965  1.16  1.30  1.45  1.25
1966  1.26  1.38  1.86  1.56
1967  1.53  1.59  1.83  1.86
1968  1.53  2.07  2.34  2.25
1969  2.16  2.43  2.70  2.25
1970  2.79  3.42  3.69  3.60
1971  3.60  4.32  4.32  4.05
1972  4.86  5.04  5.04  4.41
1973  5.58  5.85  6.57  5.31
1974  6.03  6.39  6.93  5.85
1975  6.93  7.74  7.83  6.12
1976  7.74  8.91  8.28  6.84
1977  9.54 10.26  9.54  8.73
1978 11.88 12.06 12.15  8.91
1979 14.04 12.96 14.85  9.99
1980 16.20 14.67 16.02 11.61
```

Now, select the best plot from those illustrated above and plot this data. Hint: this looks like a time series to me…

2.4 Scatter plots, correlation, and regression

Correlation quantifies the strength and direction of the relationship between two variables, helping assess how they move together (or in opposite directions). Any potential such relationship can be visualized using a scatter plot as introduced in Section 2.3.

2.4.1 Linear correlation

Linear correlation measures the strength and direction of the linear relationship between two variables, often represented by the correlation coefficient (r). The p-value associated with this coefficient assesses the statistical significance of the correlation, helping determine whether the observed relationship is likely due to chance or represents a real association. Let' consider the built-in dataset `mtcars` which contains several aspects and performance of several 1973 - 1974 model cars. This code loads the dataset and displays several of its entries.

```
# mtcars:
# Load data from the built-in dataset into a variable named mtcars
data("mtcars")

mtcars
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2

```
Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

Let's see if there is a linear relationship between miles per gallon (MPG) and the engine horse powerr (HP) using the R command `cor.test()` and storing the **linear correlation coefficient** (`r`) and **P-value** in the variable `mpgvshp`. Notice that `mtcars$mpg` extracts just the column of MPG from the dataset and similarly for `mtcars$hp`. The r-value can be found by calling `mpgvshp$estimate`, whereas, the P-value can be found by calling `mpgvshp$p.value`. Finally, the confidence interval for the estimated r is found using the `mpgvshp$conf.int` command.

```r
# Calculate the correlation between MPG and HP
mpgvshp <- cor.test(mtcars$mpg, mtcars$hp)
mpgvshp
```

```
    Pearson's product-moment correlation

data:  mtcars$mpg and mtcars$hp
t = -6.7424, df = 30, p-value = 1.788e-07
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.8852686 -0.5860994
sample estimates:
       cor
-0.7761684
```

```r
# Let's view the r- and P-values and critical r-value range
cat("r:", mpgvshp$estimate, "\n")
```

```
r: -0.7761684
```

```r
cat("P-value:", mpgvshp$p.value, "\n")
```

```
P-value: 1.787835e-07
```

```r
cat("Confidence interval for r: (", mpgvshp$conf.int[1], ", ", mpgvshp$conf.int[2], ")")
```

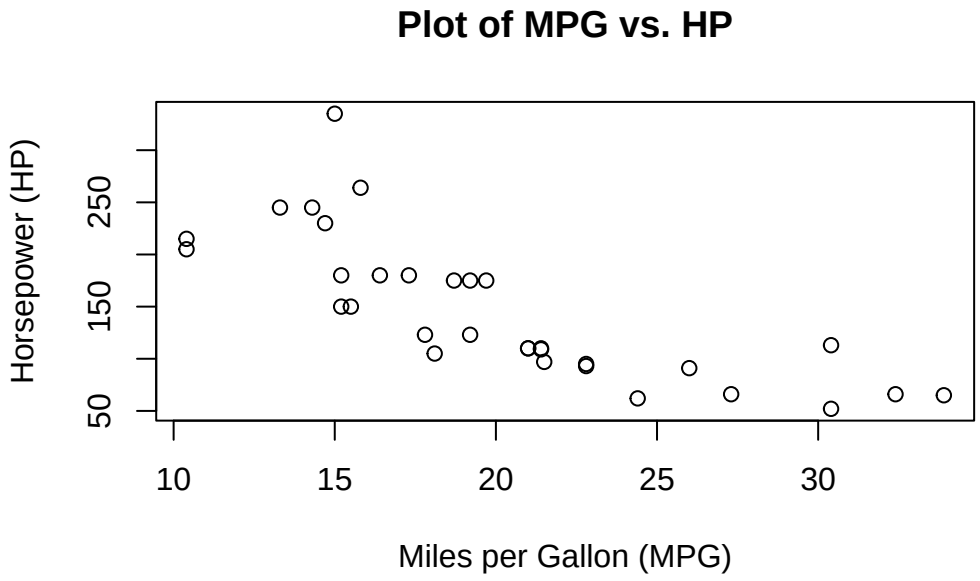
```
Confidence interval for r: ( -0.8852686 ,  -0.5860994 )
```

A negative r-value indicates that if a linear relationship is present then the relationship is negative, i.e., increasing the MPG decreases the HP. Having the absolute value of the r-value close to one indicates a linear relationship. Notice that the confidence interval for r is away from zero, supporting the conclusion that a *negative* linear relationship is present.

A P-value of less than **0.05** suggests that the sample results are *not* likely to occur merely by chance when there is no linear correlation. Thus, a small P-value such as the one we received here supports a conclusion that there is a linear correlation between MPG and HP.

Now, let's use a scatter plot to visualize the relationship.

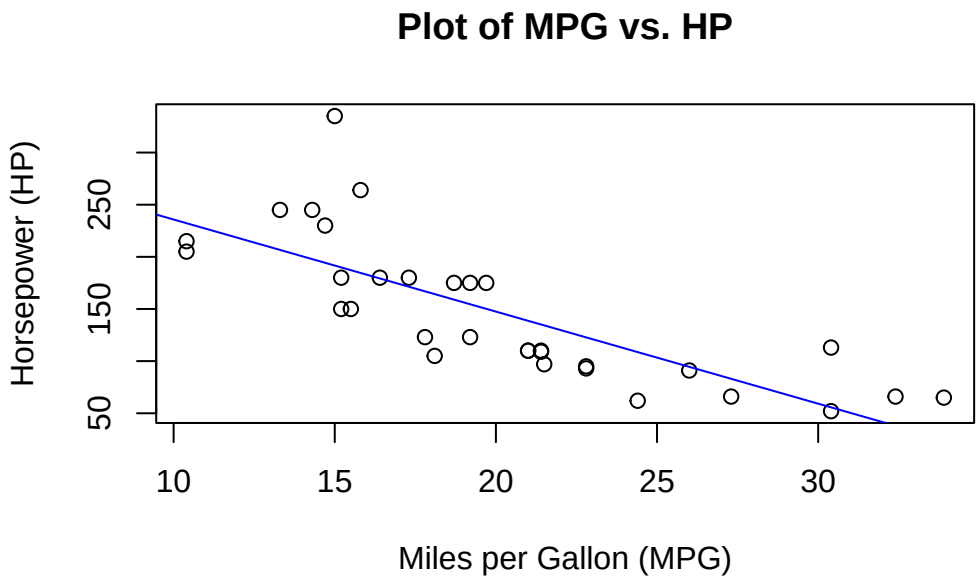
```
# Create a scatter plot to visualize the relationship
plot(mtcars$mpg, mtcars$hp, xlab = "Miles per Gallon (MPG)", ylab = "Horsepower (HP)",
     main = "Plot of MPG vs. HP")
```



2.4.2 Regression line

Regression analyzes and models the relationship between variables, allowing us to predict one variable based on the values of others. Let's return to our MPG vs HP example. We will use the R command `lm()` to create a linear model (or linear regression) for this data. We then use our scatter plot created previously to plot the model prediction alongside the actual data points. In this case, the R command `abline()` adds the regression line stored in `model` with the color being specified by the attribute `col`.

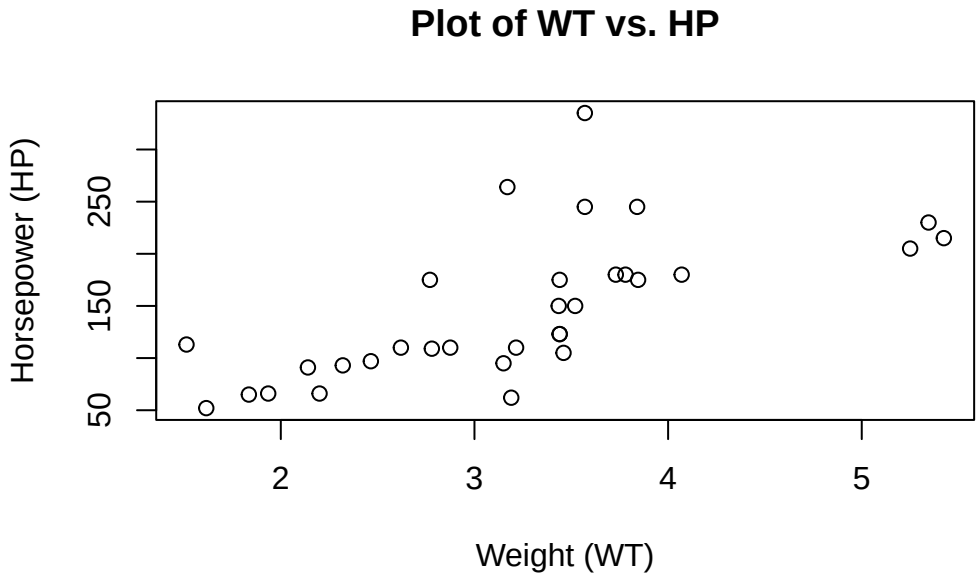
```r
# Create a linear regression model
model <- lm(hp ~ mpg, data = mtcars)

# Create a scatter plot to visualize the relationship
plot(mtcars$mpg, mtcars$hp, xlab = "Miles per Gallon (MPG)", ylab = "Horsepower (HP)",
      main = "Plot of MPG vs. HP")

# Add the regression line to the plot
abline(model, col = "blue")
```



2.4.3 Let's put it all together!

Using the same `mtcars` dataset, use what you have learned above to determine if there is a linear correlation between the weight of a car in the set versus the engine's horse power. The following code will walk you through the process. We begin with a visualization of the data using a scatter plot.

```r
# Create a scatter plot to visualize the relationship
plot(mtcars$wt, mtcars$hp, xlab = "Weight (WT)", ylab = "Horsepower (HP)",
     main = "Plot of WT vs. HP")
```



Now, let's determine if there is a linear relationship between car weight `mtcars$wt` and engine horsepower `mtcars$hp`.

```
# Calculate the correlation between MPG and HP
wtvshp <- cor.test(mtcars$wt, mtcars$hp)

wtvshp
```

```
    Pearson's product-moment correlation

data:  mtcars$wt and mtcars$hp
t = 4.7957, df = 30, p-value = 4.146e-05
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.4025113 0.8192573
sample estimates:
      cor
0.6587479
```

```
# Let's view the r- and P-values and critical r-value range
cat("r:", wtvshp$estimate, "\n")
```

```
r: 0.6587479
```

```r
cat("P-value:", wtvshp$p.value, "\n")
```

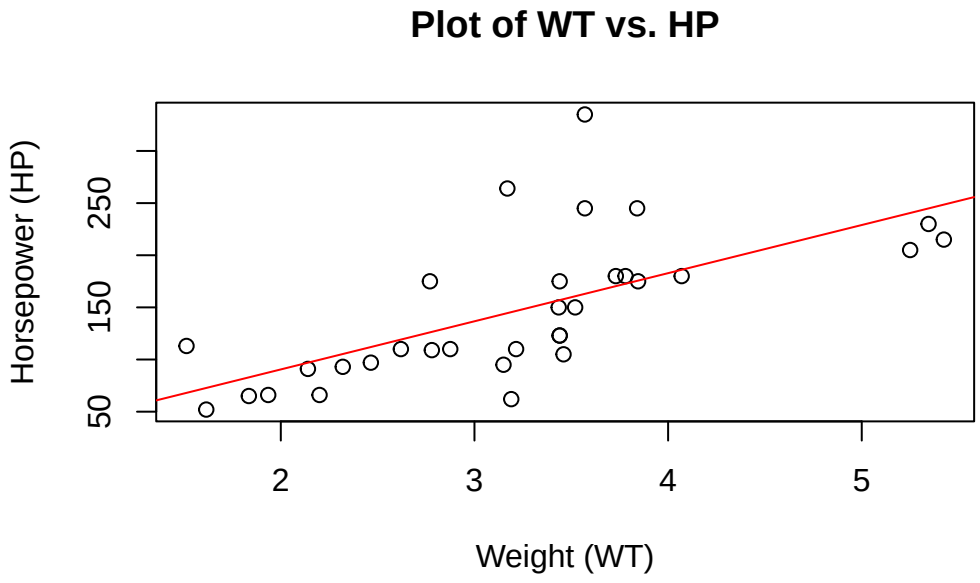
```
P-value: 4.145827e-05
```

```r
cat("Confidence interval for r: (", wtvshp$conf.int[1], ", ", wtvshp$conf.int[2], ")")
```

```
Confidence interval for r: ( 0.4025113 ,  0.8192573 )
```

What can we conclude about a possible linear relationship between car weight and horsepower? Is this relationship supported? Finally, let's visualize the regression line and data together.

```r
# Create a linear regression model
model2 <- lm(hp ~ wt, data = mtcars)

# Create a scatter plot to visualize the relationship
plot(mtcars$wt, mtcars$hp, xlab = "Weight (WT)", ylab = "Horsepower (HP)",
     main = "Plot of WT vs. HP")

# Add the regression line to the plot
abline(model2, col = "red")
```



What about causation? Does having a heavier car make it have higher or lower horsepower?

3 Describing, Exploring, and Comparing Data

r-function	Description
`mean(x)`	Calculate the mean of vector `x`
`median(x)`	Calculate the median of vector `x`
`Mode(x)`	Calculate the mode of vector `x`. Need to install the package `DescTools`
`max(x)`	Calculate the maximum of vector `x`
`min(x)`	Calculate the minimum of vector `x`
`range(x)`	Calculate the range of vector `x`. `range(x)` returns a vector `c(min(x),max(x))`
`sd`, `var`	calculate the sample standard deviation, use denominator `n-1`. To remove a `NA` value, pass the argument `na.rm=TRUE`. To calculate a population variance $\sigma^2(x)$, use the formula $\sigma^2(x) = var(x) * (length(x) - 1)/length(x)$, and $\sigma = \sqrt{\sigma^2}$.
`scale(x)`	convert data vector `x` into z-scores.
`quantile(x,probs)`	Calculate the percentiles at the indicated probability values `probs`.
`summary(x)`	Give a 5-number (min, Q1, median, mean, Q3, max) summary of the data vector `x`
`boxplot(x)`	Plot the boxplot of the data vector `x`
`hist(x)`	plot histogram of the 1-D data `x`. Optional arguments: `main`: main title; `xlab`: x-label; `ylab`: y-label; `col`: color. Pass `freq=FALSE` for a relative frequency histogram.
`rnorm(n,mean,sd)`	generate `n` *random* values of standard normal distribution with the given `mean` and `sd`.
`runif(n, min, max)`	Generate `n` uniformly distributed *random* values between min (inclusive) and max (inclusive).
`rexp(n,r)`	Generate `n` exponentially distributed *random* values with rate `r` at which events occurs on average

3.1 Measures of center

Measures of center, such as the mean and median, provide a central value that summarizes a dataset, helping to understand its typical or central tendency, which is crucial for making data-driven decisions and drawing inferences.

3.1.1 Mean

The mean, also known as the average, is a measure of center in a dataset that calculates the sum of all values divided by the total number of values, providing a representative value for the dataset. We will employ the R command `mean()` to calculate the mean of several datasets.

```
# Load test data into a variable names scores
scores <- c(95, 90, 85, 85, 87, 74, 75, 64, 85, 84, 87, 15, 20, 75, 75, 90, 75)
# Calculate mean of scores and then store it in the variable meanScore
meanScore <- mean(scores)

# print out the answer
cat("Mean test score is: ", meanScore, "\n")
```

```
Mean test score is:  74.17647
```

The mean is very sensitive to outliers. Let's see what happens when we take the same `scores` list and add some really low grades to the list.

```
# Previous test scores with a several much lower scores added
scores2 <- c(95, 90, 85, 85, 87, 74, 75, 64, 85, 84, 87, 15, 20, 75, 75, 90, 75,
             2, 1, 5, 3)

# Calculate mean of scores2 and then store it in the variable meanScore2
meanScore2 <- mean(scores2)

# print out the answer
cat("Mean test score is from original is: ", meanScore, ", while from scores2 is: ",
    meanScore2)
```

```
Mean test score is from original is:  74.17647 , while from scores2 is:  60.57143
```

This sensitivity to outliers is the notion of resistance. The mean is not a resistant measure of middle.

3.1.2 Median

The median is a measure of center in a dataset that represents the middle value when all values are ordered, and it is resistant to extreme outliers, making it a robust statistic for summarizing data. Let's return to the scores data and see the difference between mean and median of the two datasets `scores` and `scores2` using the R commands `median()`.

```r
# Calculate median of scores and then store it in the variable medianScore
medianScore <- median(scores)

# Calculate median of scores2 and then store it in the variable medianScore2
medianScore2 <- median(scores2)

# print out the answer
cat("Mean test score from original is: ", meanScore, ", while from scores2 is: ",
    meanScore2, "\n\n")
```

```
Mean test score from original is:  74.17647 , while from scores2 is:  60.57143
```

```r
cat("Median test score from original is: ", medianScore, ", while from scores2 is: ", median
```

```
Median test score from original is:  84 , while from scores2 is:  75
```

3.1.3 Mode

The mode is a statistical measure that represents the value or values that occur most frequently in a dataset, making it a useful indicator of the most common observation(s); however, it is not necessarily resistant to outliers, meaning extreme values can heavily influence the mode. There is no bulit-in R command for mode, so we will have to employ the package `DescTools`.

The first time you run this code, you will need to install the following package. After this initial run, you can skip running this code:

```r
# Installs the package 'DescTools'.  ONLY RUN THIS CODE ONCE!
install.packages('DescTools')
```

Once this package is installed, then we can load the library `DescTools` in order to use the R command `Mode()`.

```r
# Load the DescTools package
library(DescTools)

# Calculate the mode of both scores and scores2 using the Mode() method

# Calculate Mode of scores and then store it in the variable modeScore
modeScore <- Mode(scores)

# Calculate median of scores2 and then store it in the variable modeScore2
```

```r
modeScore2 <- Mode(scores2)

# print out the answer
cat("Mode test score from original is: ", modeScore, ", while from scores2 is: ",
    modeScore2, "\n")
```

```
Mode test score from original is:  75 , while from scores2 is:  75
```

3.1.4 Midrange

The midrange is a measure of center in a dataset that represents the arithmetic mean of the maximum and minimum values, and it is not resistant to extreme outliers, making it sensitive to extreme values. There is no built-in R command for midrange, thus we will use the following code to calculate the midrange of our `scores` and `scores2` data.

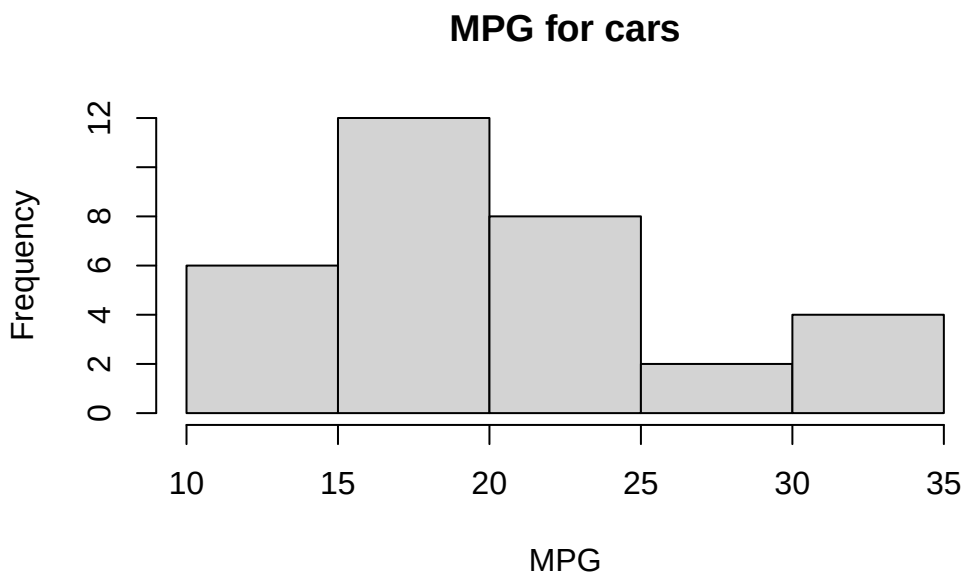
```r
# Calculate miderange of scores and then store it in the variable midrangeScore
midrangeScore <- (max(scores) - min(scores)) / 2

# Calculate midrange of scores2 and then store it in the variable midrangeScore2
midrangeScore2 <- (max(scores2) - min(scores2)) / 2

# print out the answer
cat("Midrange test score from original is: ", midrangeScore, ",
    while from scores2 is: ", midrangeScore2, "\n")
```

```
Midrange test score from original is:  40 ,
    while from scores2 is:  47
```

3.1.5 Let's put it all togeher!

Consider the built-in dataset `mtcars` which contains several aspects and performance of several 1973 - 1974 model cars which we studied in Section 2.4. We will calculate mean, meidan, mode, and midrange of the miles per gallon of the cars in the dataset. using the R commands illustrated in the previous sections, as well as compute the so-called 5-number summary using the R command `summary()`. First, let's plot a histogram of the data.

```r
# Extract the MPG data and store it into the variable carsMPG
carsMPG <- mtcars$mpg

# Generate a histogram of the MPG data from mtcars
hist(carsMPG, main = "MPG for cars", xlab = "MPG")
```




```
# Calculate mean of MPG data and then store it in the variable meanMPG
meanMPG <- round(mean(carsMPG), digits = 2)

# Calculate median of MPG data and then store it in the variable medianMPG
medianMPG <- median(carsMPG)

# Calculate Mode of scores and then store it in the variable modeMPG
modeMPG <- Mode(carsMPG)

# Calculate miderange of scores and then store it in the variable midrangeMPG
midrangeMPG <- (max(carsMPG) - min(carsMPG)) / 2

# print out the answer
cat("Mean \t Median \t \t \t  Mode \t \t \t Midrange \n")
```

```
Mean     Median                Mode           Midrange
```

```
cat(meanMPG, " \t ", medianMPG, " \t ", modeMPG, " \t ", midrangeMPG, "\n")
```

```
20.09     19.2        10.4 15.2 19.2 21 21.4 22.8 30.4        11.75
```

```
# Give the 5-number summary for MPG data
summary(carsMPG)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.40   15.43   19.20   20.09   22.80   33.90
```

Notice that there are 7 elements in the mode. That's because there are 7 most frequent elements, each appear twice. Which of these central measures best describes what you visually see as the "center" of data using the histogram? What does it "mean" that the mean and median are close to each other? Does the 5-number summary give us any additional information regarding the measure of "center" in the data?

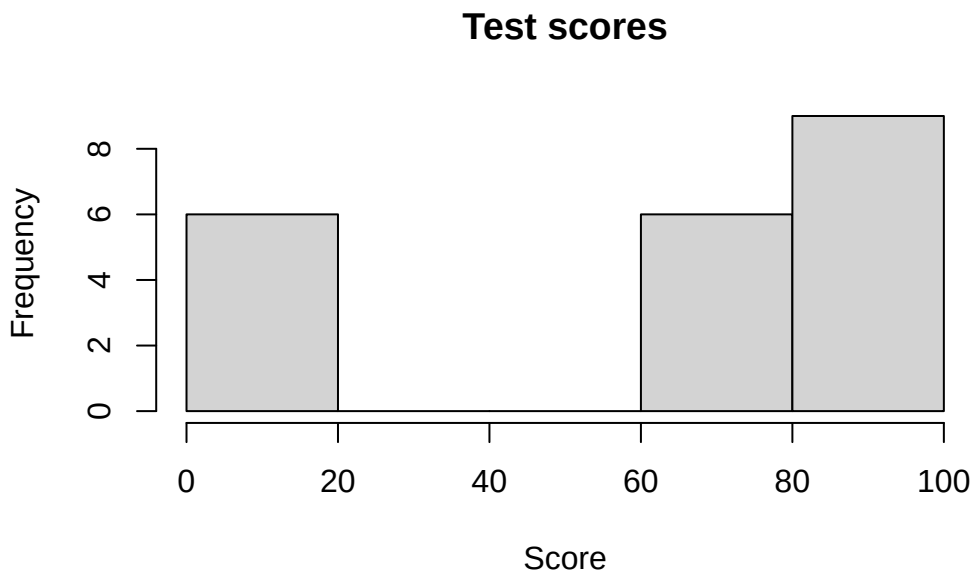
3.2 Measures of variation

Measures of variation, such as the range, variance, and standard deviation, provide insights into the spread or dispersion of data points within a dataset, helping us understand how much individual values deviate from the central tendency measures like the mean or median. These measures are essential because they quantify the degree of variability in data, allowing us to assess data quality, make more accurate predictions, and draw meaningful conclusions in statistical analysis.

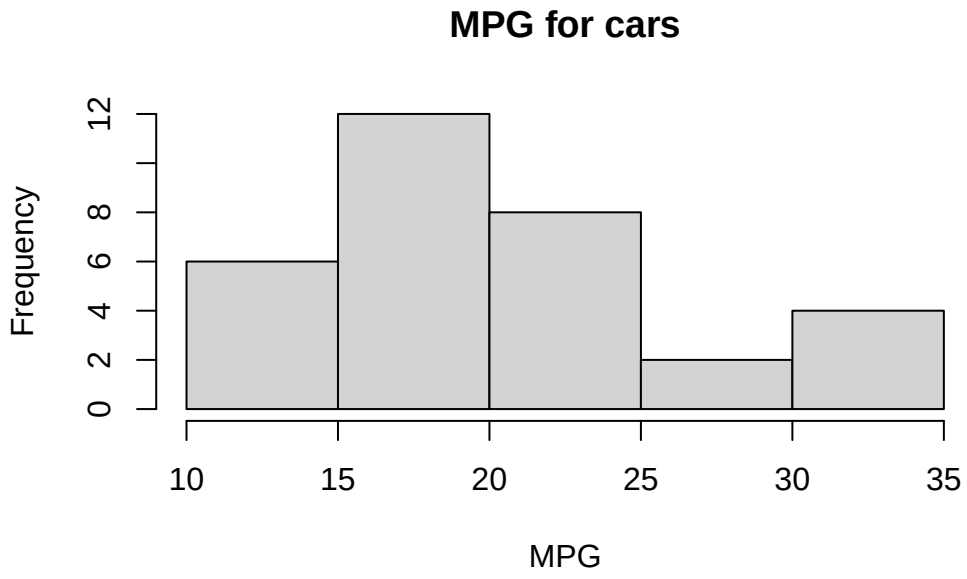
3.2.1 Visualizing variation

Histograms can visually represent the variation in a dataset by displaying the distribution of values across different bins or intervals, highlighting the frequency and pattern of data points, and revealing the shape and spread of the distribution. Let's compare histograms for our `scores2` and `carsMPG` datasets.

```
# Generate a histogram of the MPG data from scores2
hist(scores2, main = "Test scores", xlab = "Score")
```




```
# Generate a histogram of the MPG data from mtcars
hist(carsMPG, main = "MPG for cars", xlab = "MPG")
```



3.2.2 Range

The range is a measure of variation that represents the difference between the maximum and minimum values in a dataset, but it is not resistant to outliers, meaning extreme values can substantially affect the range. Let's compare the ranges of our `carsMPG` and `scores2` datasets using the R command `range()`.

```
# Calculate range of scores2 and then store it in the variable rangeScore2
rangeScore2 <- range(scores2)

# Calculate range of carsMPG and then store it in the variable rangeMPG
rangeMPG <- range(carsMPG)

# print out the answer
cat("Range for test scores from scores2 is: (", rangeScore2[1], ", ",
    rangeScore2[2], ") \n")
```

```
Range for test scores from scores2 is: ( 1 ,  95 )
```

```
cat("Range for MPG from carsMPG is: (", rangeMPG[1], ", ", rangeMPG[2], ") \n")
```

```
Range for MPG from carsMPG is: ( 10.4 ,  33.9 )
```

3.2.3 Standard deviation

Standard deviation is a measure of the dispersion or spread of data points in a dataset, with a higher value indicating greater variability, and it's calculated differently for **populations** (σ) and **samples** (s), where the **sample** standard deviation (s) is often used for practical data analysis. However, standard deviation is not resistant to extreme outliers, making it sensitive to the influence of extreme values on its magnitude. There is a built-in R command for **sample** standard deviation, but no such command for **population** standard deviation. Recall our test scores dataset `scores2`. Since this data represents the entire population (every student in the class), we will calculate **population** standard deviation for that dataset. However, the MPG data in `carsMPG` is only a sample of all the cars on the market in 1973 - 1974. Thus, we will employ the R command `sd()` to calculate **sample** standard deviation.

```
# Calculate population SD of scores2 and then store it in the variable popSDScore2
popSDScore2 <- sqrt(var(scores2) * (length(scores2) - 1) / length(scores2))

# Calculate sample SD of carsMPG and then store it in the variable samSDMPG
samSDMPG <- sd(carsMPG)
```

```r
# Print out the answer
cat("Population standard deviation for test scores from scores2 is: ", popSDScore2, "\n\n")
```

```
Population standard deviation for test scores from scores2 is:  34.35331
```

```r
cat("Sample standard deviation for MPG from carsMPG is: ", samSDMPG, " \n")
```

```
Sample standard deviation for MPG from carsMPG is:  6.026948
```

3.2.4 Variance

Variance measures the average of the squared differences between each data point and the mean of a dataset, providing a measure of data dispersion, but it is not resistant to extreme outliers, making it sensitive to the influence of extreme values on its magnitude. Variance is calculated differently for **populations** (σ^2) and **samples** (s^2), with the **sample** variance (s^2) being used for practical data analysis to account for bias when working with a subset of a larger population. Let's compare **population** variance for our `scores2` dataset and **sample** variance for our `carsMPG` dataset. As with standard deviation, although there is a built-in R command for **sample** variance, there is not a built-in command for **population** variance, so we will have to improvise.

```r
# Calculate population variance of scores2 and then store it in the variable
# popVarScore2
popVarScore2 <- var(scores2) * (length(scores2) - 1) / length(scores2)

# Calculate sample variance of carsMPG and then store it in the variable samSDMPG
samVarMPG <- var(carsMPG)

# Print out the answer
cat("Population variance for test scores from scores2 is: ", popVarScore2, "\n\n")
```

```
Population variance for test scores from scores2 is:  1180.15
```

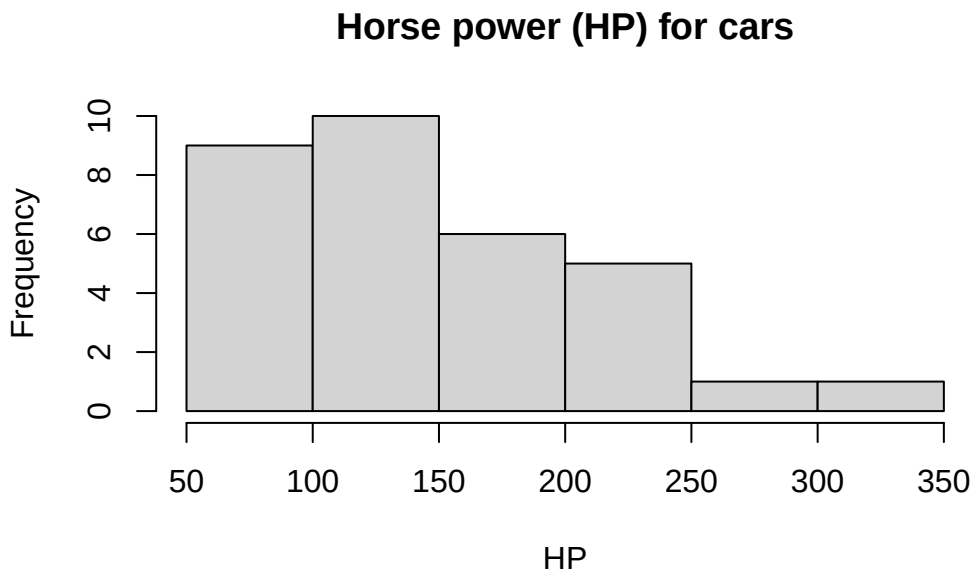
```r
cat("Sample variance for MPG from carsMPG is: ", samVarMPG, " \n")
```

```
Sample variance for MPG from carsMPG is:  36.3241
```

3.2.5 Let's put it all togeher!

Consider the built-in dataset `mtcars` which contains several aspects and performance of several 1973 - 1974 model cars which we studied in Section 2.4. We will first calculate mean and median of the horse power (HP) of the cars in the dataset. To calculate measures of variation, we note that since this is just a **sample** of all possible cars on the market during 1973 - 1974, we will employ **sample** variance and standard deviation using the R commands illustrated in the previous sections, along with a histogram to visually explore the data.

```
# Extract the HP data and store it into the variable carsHP
carsHP <- mtcars$hp

# Generate a histogram of the HP data from mtcars
hist(carsHP, main = "Horse power (HP) for cars", xlab = "HP")
```




```
# Calculate mean of HP data and then store it in the variable meanHP
meanHP <- round(mean(carsHP), digits = 2)

# Calculate median of HP data and then store it in the variable medianHP
medianHP <- median(carsHP)

# Calculate variance of HP data and then store it in the variable varHP
varHP <- var(carsHP)

# Calculate standard deviation of HP and then store it in the variable midrangeHP
```

```r
sdHP <- sd(carsHP)

# print out the answer
cat("Mean \t Median \t  variance \t Standard Deviation \n")
```

```
Mean     Median       variance   Standard Deviation
```

```r
cat(meanHP, " \t ", medianHP, " \t ", varHP, " \t ", sdHP, "\n")
```

```
146.69        123     4700.867        68.56287
```

Now, compare the MPG and HP data from the `mtcars` dataset. For MPG, we calculated a standard deviation around *36* and for HP of around *69*. Does this mean that the MPG data is less spread out than the HP data? Is your answer to this question consistent with the histograms we produced? Can we compare standard deviations from two totally different datasets in a meaningful way?

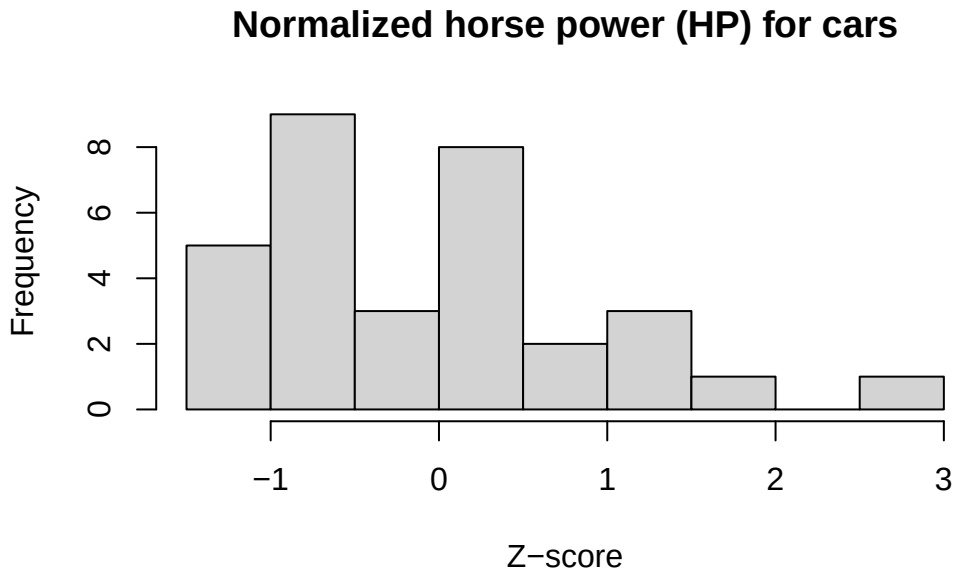
3.3 Measures of relative standing and boxplots

Measures of relative standing, such as percentiles and quartiles, provide information about where specific data points fall within a dataset, offering insights into the relative position of values. Boxplots are graphical representations that display the distribution of data, highlighting the median, quartiles, and potential outliers, making them valuable tools for comparing different datasets by visually assessing their central tendency, spread, and skewness.

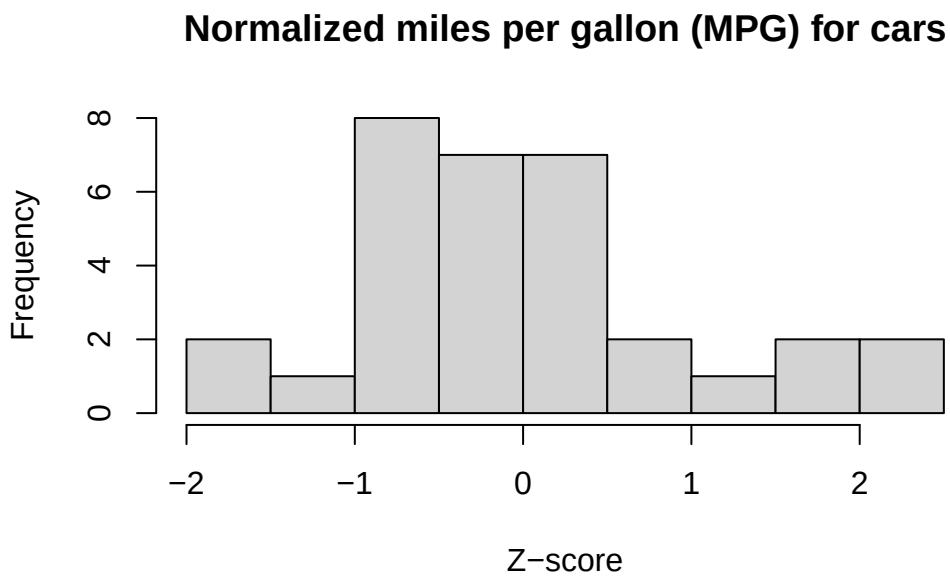
3.3.1 z-Scores

Z-scores, also known as standard scores, standardize individual data points by expressing how many standard deviations they are from the mean, enabling meaningful comparisons and assessments of data points' relative positions within a distribution, regardless of the original scale of the data. Z-scores are valuable for identifying outliers, understanding data distributions, and making statistical inferences, as they provide a common framework for measuring deviations from the mean across different datasets. Let's explore z-scores using the built-in dataset `mtcars` which contains several aspects and performance of several 1973 - 1974 model cars which we studied in the last section. Particularly, let's employ the built-in R command `scale()` to convert our dataset to z-scores which can be plotted in a histogram. Once the two datasets (MPG and HP) are normalized, we will be able to get a better picture of their spread away from the respective means.

```r
# Transform the MPG data to z-scores and store the new data in zcarsHP
zcarsHP <- scale(carsHP)

# Transform the MPG data to z-scores and store the new data in zcarsHP
zcarsMPG <- scale(carsMPG)

# Generate a histogram of the transformed HP data from mtcars
hist(zcarsHP, main = "Normalized horse power (HP) for cars", xlab = "Z-score")
```




```r
# Generate a histogram of the transformed MPG data from mtcars
hist(zcarsMPG, main = "Normalized miles per gallon (MPG) for cars", xlab = "Z-score")
```



Visually, the normalized MPG data is more concentrated around the transformed mean of 0, while the HP data is much more spread out.

Any data point that has a z-score of less than -2 or higher than 2 is considered to be significantly lower or higher, respectively. Let's view our transformed data sets MPG and HP to identify data points that are significantly higher.

```
# Find MPG data points with z-scores higher than 2
outliersMPG <- carsMPG[zcarsMPG > 2]

# Find HP data points with z-scores higher than 2
outliersHP <- carsHP[zcarsHP > 2]

# Print the data points with z-scores higher than 2
cat("MPG Data with z-scores higher than 2:", outliersMPG, "\n")
```

```
MPG Data with z-scores higher than 2: 32.4 33.9
```

```
cat("HP Data with z-scores higher than 2:", outliersHP, "\n")
```

```
HP Data with z-scores higher than 2: 335
```

3.3.2 Percentiles

Percentiles are statistical measures that divide a dataset into 100 equal parts, helping identify values below which a certain percentage of the data falls and enabling comparisons of data points in a ranked order. Let's use the built-in R command `quantile()`with the MPG data from the previous example to compute the 10th, 50th, and 90th percentiles for that dataset.

```
# Compute 10th, 50th, and 90th percentiles for the MPG dataset
percentiles <- c(0.1, 0.5, 0.9)
percentilesMPG <- quantile(carsMPG, probs = percentiles)

# Print the data points with z-scores higher than 2
percentilesMPG
```

```
  10%   50%   90%
14.34 19.20 30.09
```

Notice that both of our significantly larger MPG values (i.e., 32.4 and 33.9) both fall above the 90th percentile of the dataset.

3.3.3 Quartiles & the 5-number summary

Quartiles are statistical measures that divide a dataset into four equal parts, with three quartiles (Q1, Q2, Q3) providing insights into the data's spread and central tendencies; they are resistant to outliers, making them robust tools for summarizing data. The 5-number summary is a set of five statistics (minimum, Q1, median, Q3, maximum) that provide a concise description of a dataset's central tendencies and spread. Keeping with our MPG dataset, we will employ the R command `summary()` to give the 5-number summary (which will include Q1, Q2 (also known as the median), & Q3).

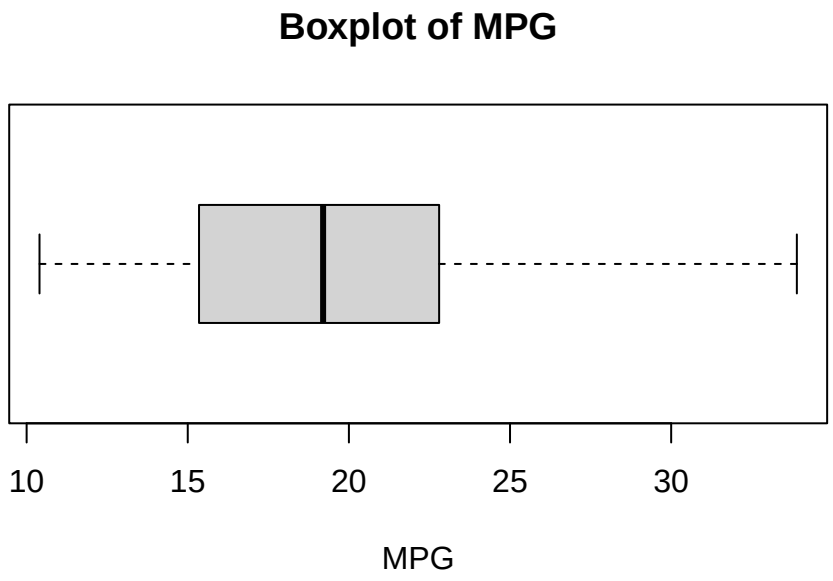
```
# Compute 5-number summary for MPG data and store it in fiveMPG
fiveMPG <- summary(carsMPG)

# Display the 5-number summary
fiveMPG
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.40   15.43   19.20   20.09   22.80   33.90
```

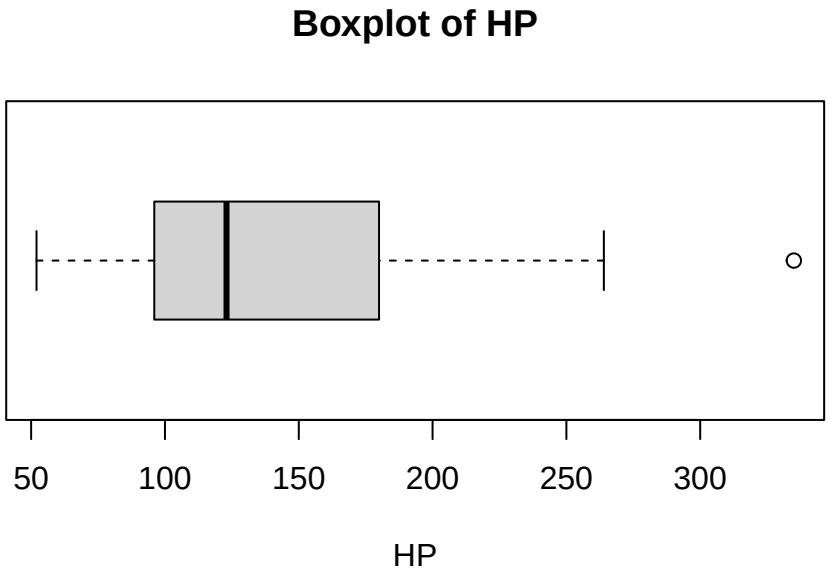
3.3.4 Boxplot

A boxplot, also known as a box-and-whisker plot, is a graphical representation of the five-number summary, displaying the median, quartiles, and potential outliers in a dataset, making it a valuable tool for visualizing the distribution and spread of data. We will employ the R command `boxplot()` to compare the MPG and HP datasets from previous examples. This R command actually creates a modified boxplot by default. Recall the only difference between a regular boxplot and a modified box plot is that data which falls outside of the interquartile range is denoted as an outlier and plotted as an individual point on the graph.

```
# Generate boxplot for MPG
boxplot(carsMPG, main = "Boxplot of MPG", horizontal = TRUE, xlab = "MPG")
```

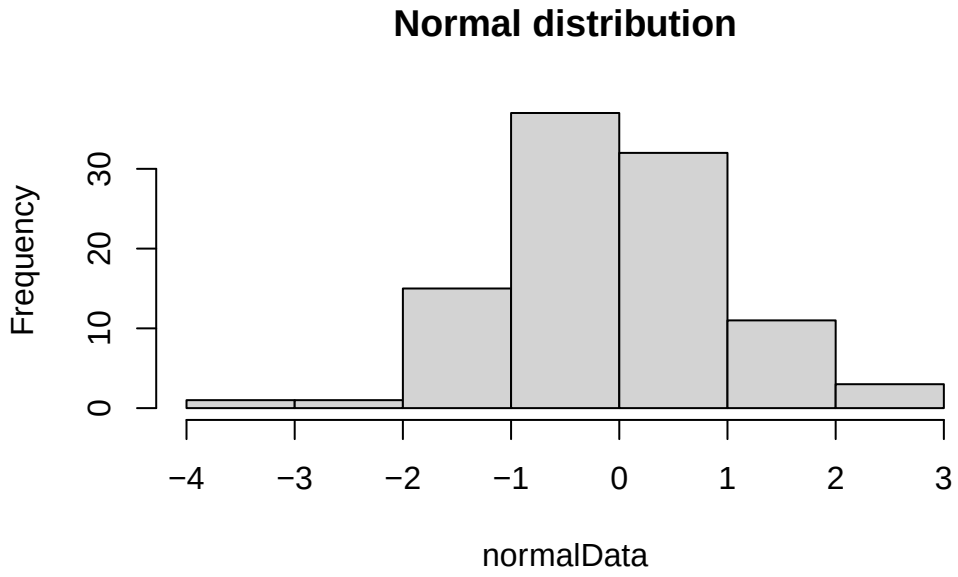



```
# Generate boxplot for HP
boxplot(carsHP, main = "Boxplot of HP", horizontal = TRUE, xlab = "HP")
```

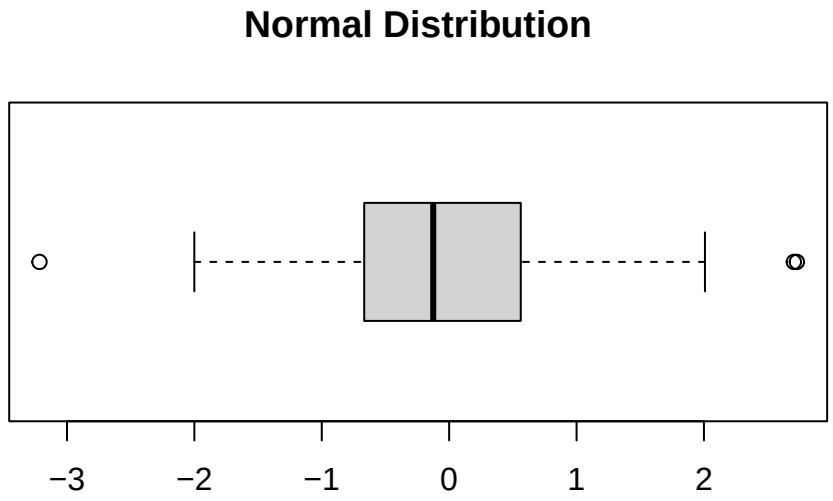


Let's also compare the boxplots of each of the four datasets for which we explored normal, skewed right, skewed left, and uniform distributions.

```
# Create histogram/boxplot of normal data
# Sample normal distribution
normalData <- rnorm(100)
hist(normalData, main = "Normal distribution")
```

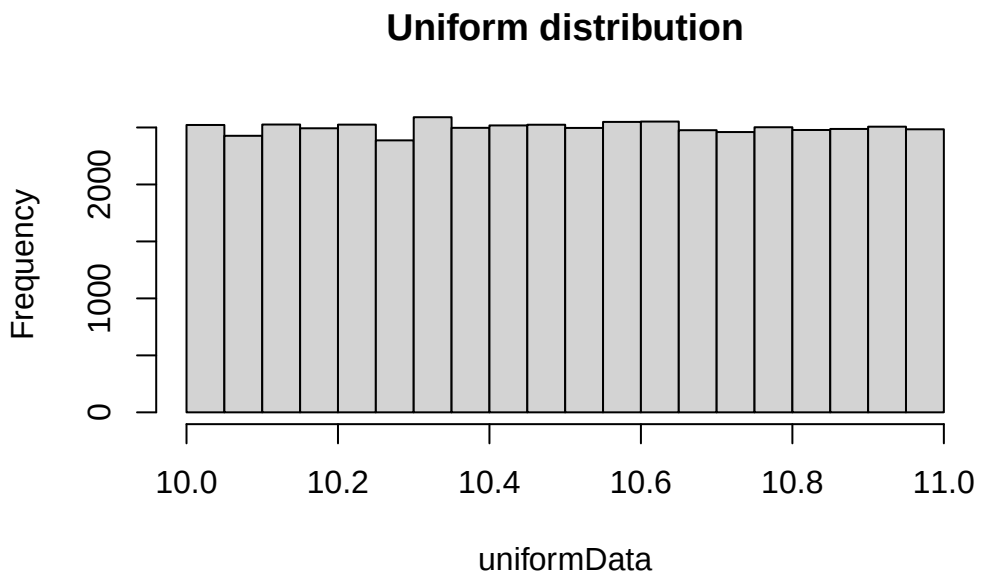



```
boxplot(normalData, main = "Normal Distribution", horizontal = TRUE)
```

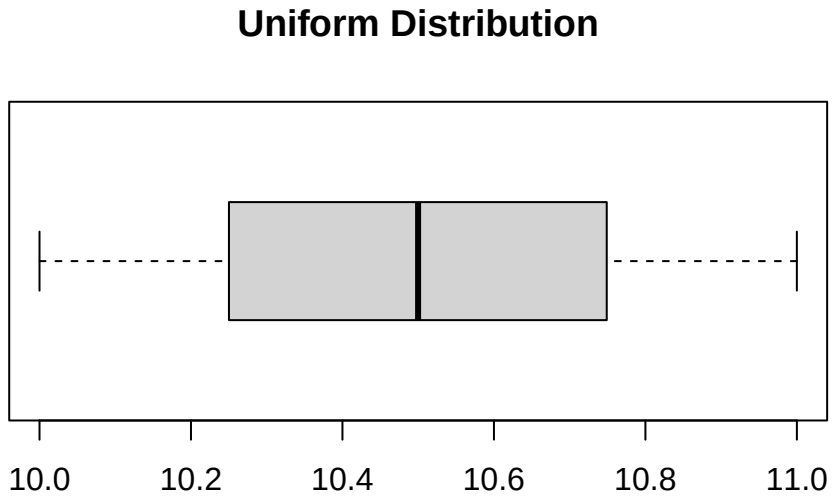



```
# Create histogram/boxplot of uniform data
# Sample uniform distribution using the command runif
```

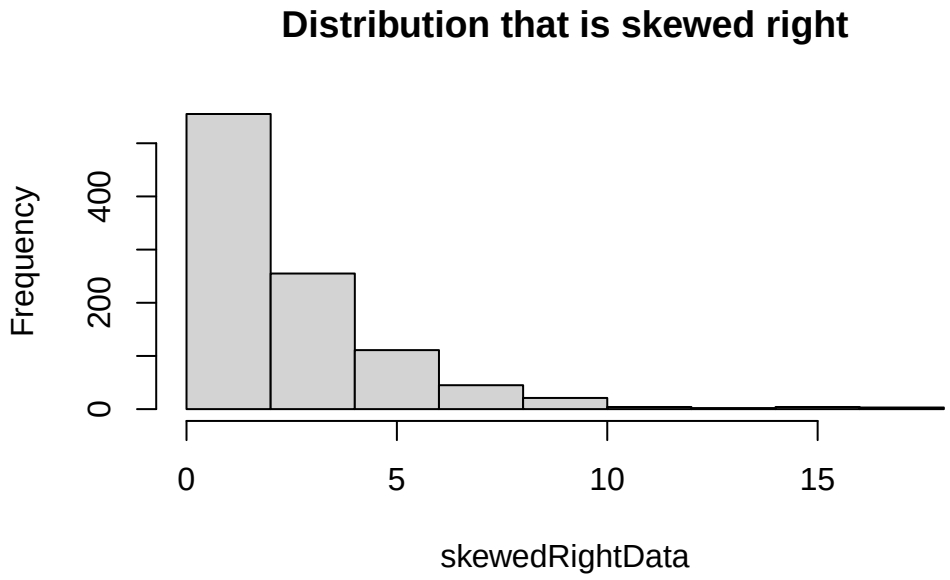
```r
uniformData <- runif(50000, min = 10, max = 11)
hist(uniformData, main = "Uniform distribution")
```



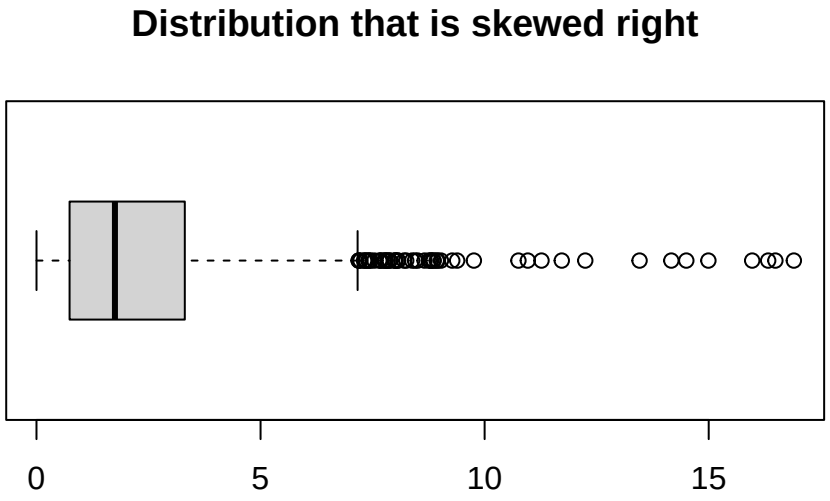

```r
boxplot(uniformData, main = "Uniform Distribution", horizontal = TRUE)
```



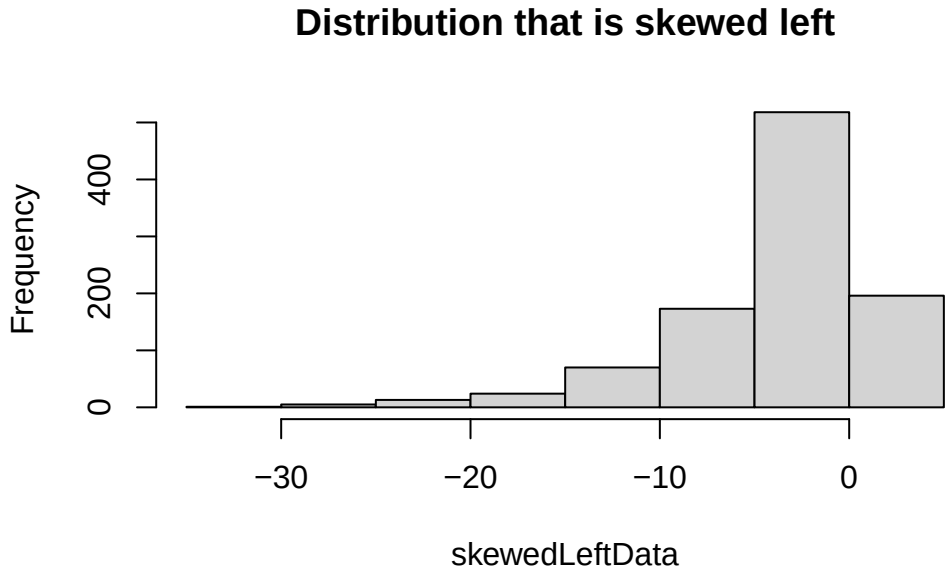
```r
# Sample of a distribution that is skewed right
skewedRightData <- rexp(1000, 0.4)
# Create histogram/boxplot of skewed right data
hist(skewedRightData, main = "Distribution that is skewed right")
```



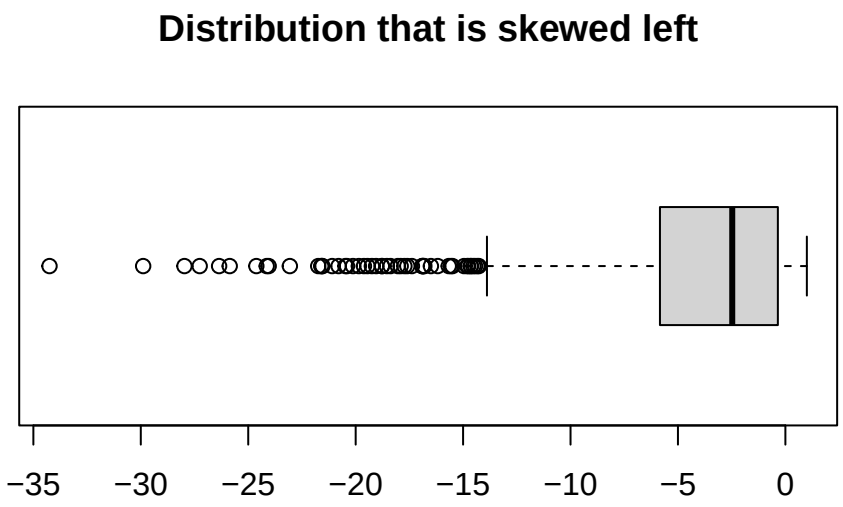

```r
boxplot(skewedRightData, main = "Distribution that is skewed right", horizontal = TRUE)
```




```r
# Sample of a distribution that is skewed left
skewedLeftData <- 1 - rexp(1000, 0.2)
# Create histogram/boxplot of skewed left data
hist(skewedLeftData, main = "Distribution that is skewed left")
```



```r
boxplot(skewedLeftData, main = "Distribution that is skewed left", horizontal = TRUE)
```

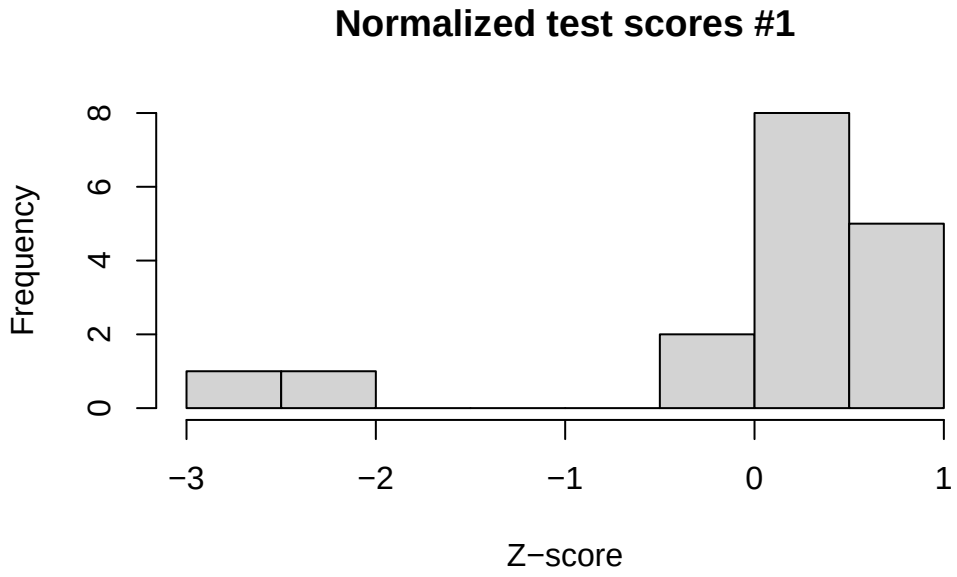


Notice there are a lot of outliers shown on the skewed left & right data. These points are what is causing the long tails on both histograms.

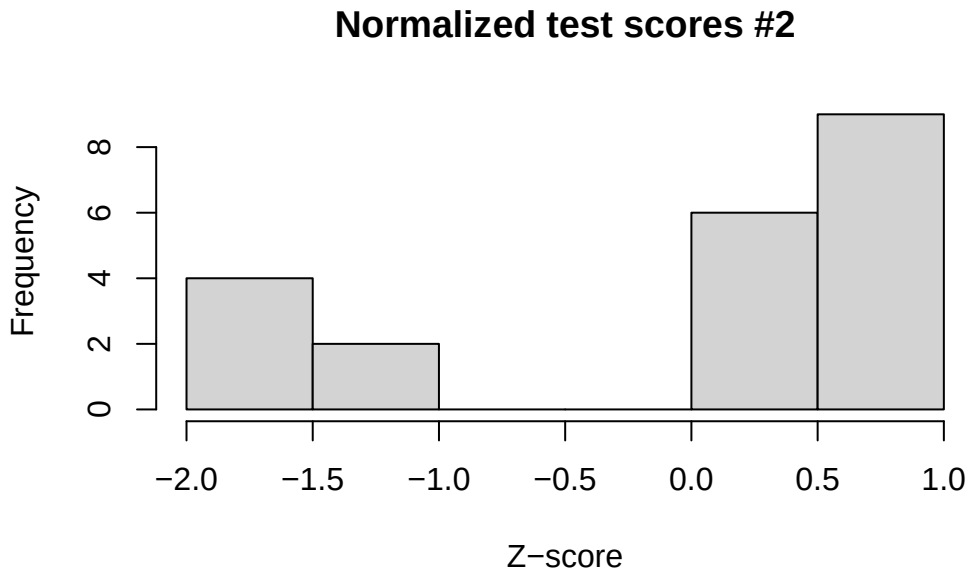
3.3.5 Let's put it all together!

We will use everything we have learned so far in this section to explore the differences between our two test score datasets, i.e., `scores` and `scores2`. These are fictional collections of test scores with `scores2` containing several more extremely low test scores than `scores`. Our first task is to transform the datasets to z-scores and visualize the scaled datasets with a histrogram.

```r
# Transform the scores data to z-scores and store the new data in zscores
zscores <- scale(scores)

# Transform the scorres2 data to z-scores and store the new data in zscores2
zscores2 <- scale(scores2)

# Generate a histogram of the transformed from scores
hist(zscores, main = "Normalized test scores #1", xlab = "Z-score")
```




```r
# Generate a histogram of the transformed from scores2
hist(zscores2, main = "Normalized test scores #2", xlab = "Z-score")
```



Out of the two fictional classes, are there any test scores that are significantly high or low? What can we conclude about those scores? Now, let's compute the 5-number summary for each group of test scores.

```r
# Compute 5-number summary for scores
summary(scores)
```

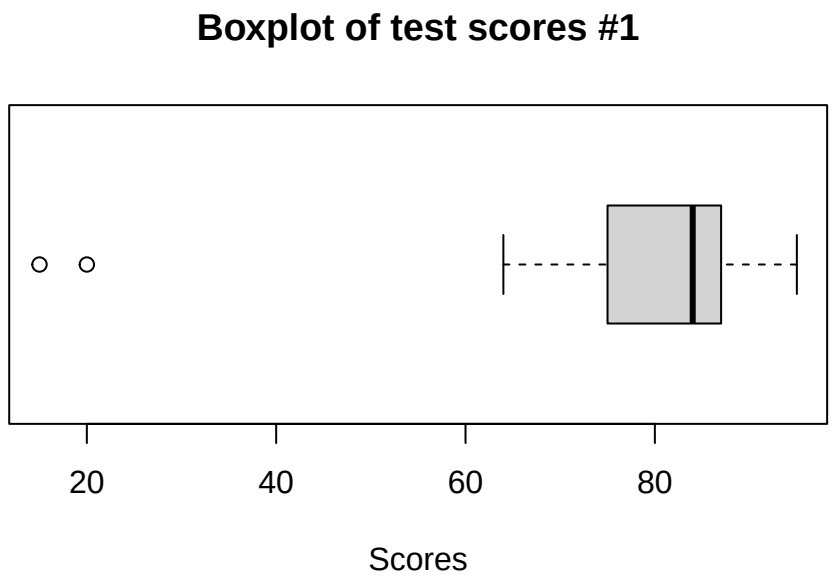
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  15.00   75.00   84.00   74.18   87.00   95.00
```

```r
# Compute 5-number summary for scores2
summary(scores2)
```

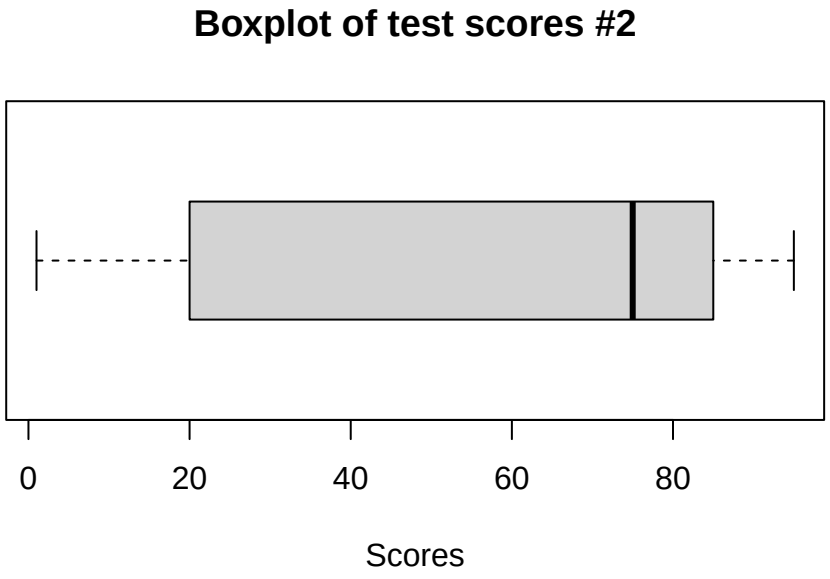
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00   20.00   75.00   60.57   85.00   95.00
```

Finally, let's create boxplots for both datasets and show them on the same plot window for comparison.

```r
# Generate boxplot for both
boxplot(scores, main = "Boxplot of test scores #1", horizontal = TRUE, xlab = "Scores")
```




```r
boxplot(scores2, main = "Boxplot of test scores #2", horizontal = TRUE, xlab = "Scores")
```



What conclusions can we draw regarding the two datasets? If these were two real classes, how would the boxplots help the teacher understand grade performance for the entire class?

4 PROBABILITY

r-function	Description
`length`	Compute the length of a vector
`mean`	compute the mean
`sum`	Compute the sum
`cat`	concatenate strings and variable values for formatted print
`sample(x, size, replace=FALSE), prob=NULL)`	Randomly sample `x` uniformly with the number of samples = `size`. If `prob` is provided, length(prob)=length(x), and `prob` should sum to 1.
`table`	tabulate the frequency counts of distinct values.
`barplot(x)`	plot barplot of the 1-D data x: arguments: `main`: main title; `xlab`: x-label; `ylab`: y-label; `col`: color
`factorial(n)`	n!
`permutations(n,m), combinations(n,m)`	list all permutations (combinations) of n objects taken m at a time (n>m). Accepts an optional argument `v` for indicating the set of permuted `n` elements.
`nrow`	find the number of rows in a matrix, dataframe or other rectangular data structure

4.1 Basic Concepts of Probability

In this code:

- We calculate the probability of drawing a Heart from a deck of four suits (*sample space*).
- We simulate random events such as a coin toss and rolling a six-sided die.
- We simulate multiple die rolls and visualize the resulting probability distribution.
- We calculate the probability of a specific outcome (such as, rolling a 3).

```
# Set a seed for reproducibility
set.seed(42)

# Define a sample space (e.g., a deck of cards)
sample_space <- c("Hearts", "Diamonds", "Clubs", "Spades")
```

```r
# Calculate the probability of drawing a Heart from the sample space
probability_heart <- sum(sample_space == "Hearts") / length(sample_space)

cat("Probability of drawing a Heart:", probability_heart, "\n")
```

```
Probability of drawing a Heart: 0.25
```

```r
# Simulate a random event (e.g., coin toss)
coin_toss <- sample(c("Heads", "Tails"), size = 1)

cat("Result of a random coin toss:", coin_toss, "\n")
```

```
Result of a random coin toss: Heads
```

```r
# Simulate rolling a six-sided die
die_roll <- sample(1:6, size = 1)

cat("Result of rolling a die:", die_roll, "\n")
```

```
Result of rolling a die: 5
```

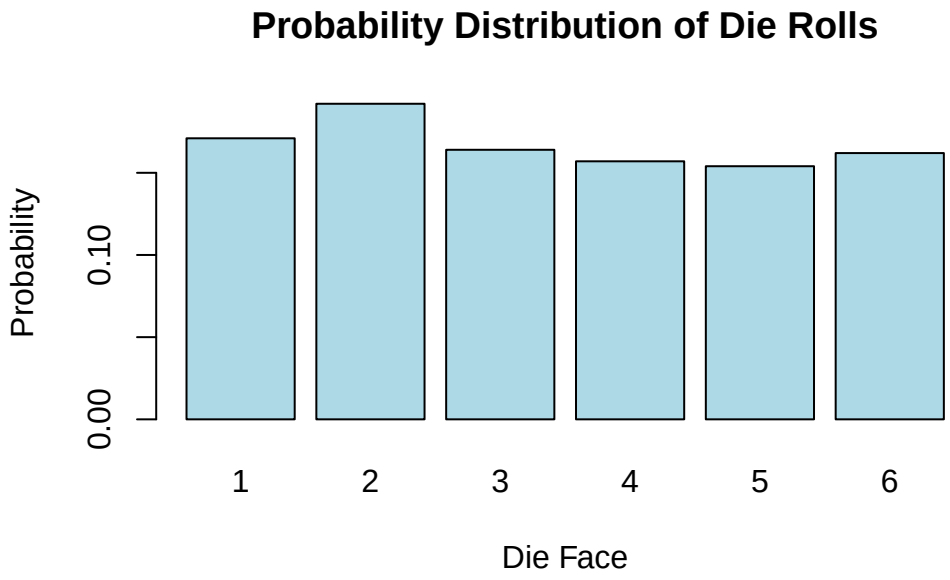
```r
# Simulate multiple die rolls and visualize the probability distribution
num_rolls <- 1000
die_rolls <- sample(1:6, size = num_rolls, replace = TRUE)

# Calculate the relative frequencies for each outcome
relative_frequencies <- table(die_rolls) / num_rolls
relative_frequencies
```

```
die_rolls
    1     2     3     4     5     6
0.171 0.192 0.164 0.157 0.154 0.162
```

```r
# Calculate the probability of rolling a 3
probability_roll_3 <- relative_frequencies[3]

cat("Probability of rolling a 3:", probability_roll_3, "\n")
```

```
Probability of rolling a 3: 0.164
```

```r
# Visualize the probability distribution with a bar plot
barplot(relative_frequencies, main = "Probability Distribution of Die Rolls",
        xlab = "Die Face", ylab = "Probability", col = "lightblue")
```



4.2 Addition rule and multiplication rule

4.3 Complements, conditional probability, and Bayes' theorem

4.4 Counting

4.4.1 Calculate factorial $n!$

R provides a built-in function to calculate factorial. You can use the `factorial()` function in R to compute the factorial of a number.

```r
n <- 5
factorial_result <- factorial(n)
cat("Factorial of", n, "is", factorial_result, "\n")
```

```
Factorial of 5 is 120
```

Replace the value of `n` with the number for which you want to calculate the factorial, and the `factorial()` function will return the result.

4.4.2 Find all permutations and the number of all permutations

To do this, we can use the `permutations` function from the **gtools** package. For any list of size `n`, this function computes all the different permutations $P(n, r)$ we can get when we select `r` items. Here are all the ways we can choose two numbers from a list consisting of `1,2,3`:

```
library(gtools)
permutations(3, 2)
```

```
     [,1] [,2]
[1,]    1    2
[2,]    1    3
[3,]    2    1
[4,]    2    3
[5,]    3    1
[6,]    3    2
```

Notice that the order matters here: `3,1` is different than `1,3`. Also, note that `(1,1)`, `(2,2)`, and `(3,3)` do not appear because once we pick a number, it can't appear again.

To get the actual number of permutations, one can use the R-function `nrow()` to find the total number of rows in the output of `permutations`:

```
library(gtools)
nrow(permutations(3,2))
```

```
[1] 6
```

Alternatively, we can add a vector `v` to indicate the objects that a permutation is performed on. If you want to see five random seven digit phone numbers out of all possible phone numbers (without repeats), you can type:

```
all_phone_numbers <- permutations(10, 7, v = 0:9) # Use digits 0, 1, ..., 9
n <- nrow(all_phone_numbers)
cat("total number of phone numbers n = ", n, "\n")
```

```
total number of phone numbers n =  604800
```

```
# Randomly sample 5 phone numbers
index <- sample(n, 5)
all_phone_numbers[index,]
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    8    9    5    1    6    0    3
[2,]    4    0    2    3    5    7    8
[3,]    0    4    6    1    3    2    7
[4,]    5    8    2    6    4    0    3
[5,]    7    5    1    0    9    2    6
```

The code `all_phone_numbers[index,]` extract the matrix `all_phone_numbers` with rows indexed by `index`, and all columns because no specific column index is given after `,`. Instead of using the numbers 1 through 10, the default, it uses what we provided through `v`: the digits 0 through 9.

4.4.3 Find all combinations and the number of all combinations

How about if the order doesn't matter? For example, in Blackjack if you get an Ace and a face card in the first draw, it is called a *Natural 21* and you win automatically. If we wanted to compute the probability of this happening, we would enumerate the *combinations*, not the permutations, since the order of drawn cards does not matter.

```
combinations(3,2)
```

```
     [,1] [,2]
[1,]    1    2
[2,]    1    3
[3,]    2    3
```

In the second line, the outcome does not include (2,1) because (1,2) already was enumerated. The same applies to (3,1) and (3,2).

To get the actual number of combinations, one can do

```
nrow(combinations(3,2))
```

```
[1] 3
```

(**optional**) Of course, one can define a R-function to calculate a permutation number.

```r
# Function to calculate permutation (nPr)
nPr <- function(n, r) {
  if (n < r) {
    return(0)
  } else {
    return(factorial(n) / factorial(n - r))
  }
}
nPr(3,2)
```

```
[1] 6
```

```r
# Function to calculate combination (nCr)
nCr <- function(n, r) {
  if (n < r) {
    return(0)
  } else {
    return(factorial(n) / (factorial(r) * factorial(n - r)))
  }
}
nCr(3,2)
```

```
[1] 3
```

5 Discrete Probability Distribution

r-function	Description
`mean`	calculate the mean
`sd`, `var`	calculate the sample standard deviation, use denominator `n-1`. To remove a `NA` value, pass the argument `na.rm=TRUE`.
`weighted.mean(x,wt)`	Calculate a weighted mean (expectation)
`median(x)`	calculate the median
`rbinom(s,size=n, prob=p)`	Generate `s` random binomial-distributed *random* values with `n` trials and success probability `p`
`dbinom(x,size=n, prob=p)`	Calculate the density (probability) of a binomial distribution with `x` successes in `n` trials given the success probability `p`
`pbinom(q,size=n, prob=p)`	Calculate the cumulative probability of a binomial distribution less than or equal to `q` successes in `n` trials given the success probability `p`. To calculate the probability greater than `q`, pass an argument `lower.tail=FALSE`.
`hist(x)`	plot histogram of the 1-D data x: arguments: `main`: main title; `xlab`: x-label; `ylab`: y-label; `col`: color. Pass `freq=FALSE` for a relative frequency histogram.
`rpois(n, lambda)`	Generate `n` random Poisson-distributed values with mean rate of events `lambda`
`dpois(x, lambda)`	Calculate the density (probability) of getting exactly `x` events given a Poisson-distribution with mean rate of events `lambda`
`ppois(x, lambda)`	Calculate the cumulative probability of less than or equal to `x` given a Poisson-distribution with mean rate of events `lambda`

5.1 Probability Distribution

5.1.1 Calculate sample mean, standard deviation and variance with equal probability

You can use R to calculate the **sample** mean, standard deviation, and variance of a given data set using built-in functions like `mean()`, `sd()`, and `var()`. Here's some sample R code to do that:

```r
# Sample data set
data_set <- c(12, 15, 18, 21, 24, 27, 30, 33, 36, 39)

# Calculate the mean
mean_value <- mean(data_set)
cat("Mean:", mean_value, "\n")
```

```
Mean: 25.5
```

```r
# Calculate the sample standard deviation
std_deviation <- sd(data_set)
cat("Standard Deviation:", std_deviation, "\n")
```

```
Standard Deviation: 9.082951
```

```r
# Calculate the sample variance
variance <- var(data_set)
cat("Variance:", variance, "\n")
```

```
Variance: 82.5
```

Just replace the data_set vector with your actual data, and this code will compute and print the mean, standard deviation, and variance for your data set. Note the results calculated by `mean()`, `sd()` and `var()` assumes each data points occurs with the equal probability $1/n$, where n is the number of data points.

5.1.2 Expectation and standard deviation with a given probability distribution

Calculation by definition:

```r
# Define the possible values and their corresponding probabilities
values <- c(1, 2, 3, 4, 5)
probabilities <- c(0.1, 0.2, 0.3, 0.2, 0.2)

# Calculate the mean (expected value)
mean_value <- sum(values * probabilities)

# Print the result
cat("Mean (Expected Value) =", mean_value, "\n")
```

```
Mean (Expected Value) = 3.2
```

Or one can use the following built-in function:

```
wt <- c(5,  5,  4,  1)/15
x <- c(3.7,3.3,3.5,2.8)
xm <- weighted.mean(x, wt)
xm
```

```
[1] 3.453333
```

To calculate the variance of a probability distribution in R, you can use the following codes.

```
# Define the values of the random variable (x_i)
values <- c(1, 2, 3, 4, 5)

# Define the probabilities (P(x_i))
probabilities <- c(0.2, 0.3, 0.1, 0.2, 0.2)

# Calculate the mean (expected value) of the random variable
mean_x <- sum(values * probabilities)

# Calculate the variance using the formula
variance <- sum((values - mean_x)^2 * probabilities)

# Print the variance
cat("Variance:", variance, "\n")
```

```
Variance: 2.09
```

5.1.3 Median

```
# Create a sample vector
data_vector <- c(12, 45, 23, 67, 8, 34, 19)

# Calculate the median
median_value <- median(data_vector)

# Print the median
cat("Median:", median_value, "\n")
```

```
Median: 23
```

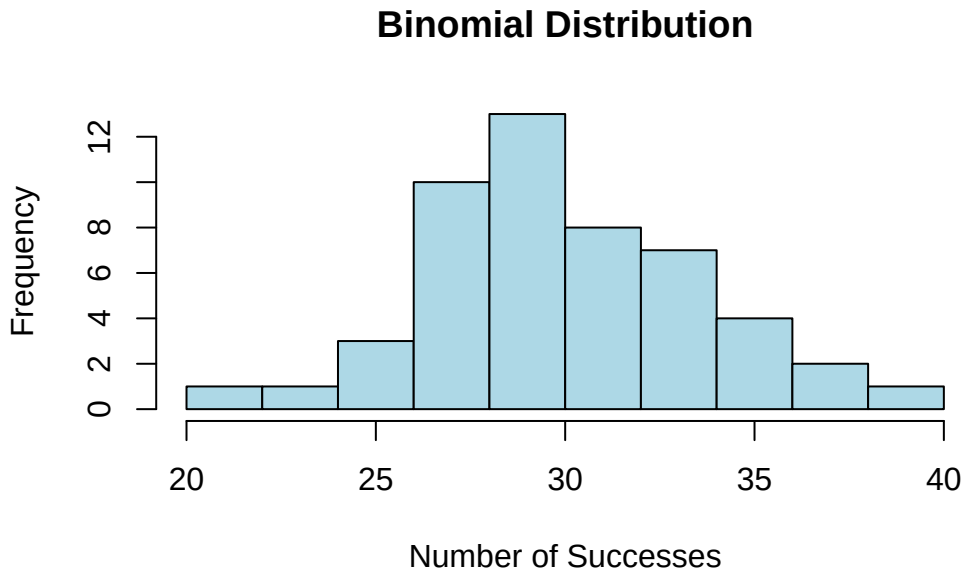
5.2 Binomial probability distributions

You can generate a data set with a binomial distribution in R using the `rbinom()` function. This function simulates random numbers following a binomial distribution. Here's an example code to generate a data set with a binomial distribution:

```
# Set the parameters for the binomial distribution
n <- 100    # Number of trials
p <- 0.3    # Probability of success in each trial

# Generate a dataset with a binomial distribution
binomial_data <- rbinom(50, size = n, prob = p)

# Print the generated dataset
print(binomial_data)
```

```
 [1] 32 28 28 34 31 28 30 29 32 33 29 29 39 31 35 29 26 30 33 26 32 27 27 29 29
[26] 36 28 34 28 28 36 38 29 30 32 28 33 30 20 34 26 28 23 30 38 34 31 32 30 36
```

```
# Create a histogram to visualize the data
hist(binomial_data, main = "Binomial Distribution", xlab = "Number of Successes",
     ylab = "Frequency", col = "lightblue", border = "black")
```



```r
# verify the mean =np, and var=npq
# Sample mean
mean(binomial_data)
```

```
[1] 30.56
```

```r
# Theoretical mean
n*p
```

```
[1] 30
```

```r
# Sample variance
var(binomial_data)
```

```
[1] 14.1698
```

```r
# Theoretical variance
n*p*(1-p)
```

```
[1] 21
```

You can calculate the probability of specific outcomes in a binomial distribution in R using the `dbinom()` function, which calculates the *probability mass function* (PMF) of the binomial distribution. Here's how to use it:

```r
# Set the parameters for the binomial distribution
x <- 2     # Number of successes (the outcome you want to calculate the
           # probability for)
n <- 10    # Number of trials
p <- 0.3   # Probability of success in each trial

# Calculate the probability of getting 'x' successes in 'n' trials
probability <- dbinom(x, size = n, prob = p)

# Print the calculated probability
cat("Probability of", x, "successes in", n, "trials:", probability, "\n")
```

```
Probability of 2 successes in 10 trials: 0.2334744
```

The `pbinom()` function in R is used to calculate cumulative probabilities for a binomial distribution. Specifically, it calculates the cumulative probability that a random variable following a binomial distribution is less than or equal to a specified value. In other words, it gives you the *cumulative distribution function* (CDF) for a binomial distribution.

Here's the basic syntax of the `pbinom()` function:

```
pbinom(q, size, prob, lower.tail = TRUE)
```

`q`: The value for which you want to calculate the cumulative probability.

`size`: The number of trials or events in the binomial distribution.

`prob`: The probability of success in each trial.

`lower.tail`: A logical parameter that determines whether you want the cumulative probability for values less than or equal to `q` (`TRUE`) or greater than `q` (`FALSE`). By default, it is set to `TRUE`.

The `pbinom()` function returns the cumulative probability for the specified value `q` based on the given parameters.

Here's an example of how to use `pbinom()`:

```
# Calculate the cumulative probability that X is less than or equal to 3
cumulative_prob <- pbinom(3, size = 10, prob = 0.3)

# Print the cumulative probability
cat("Cumulative Probability:", cumulative_prob, "\n")
```

```
Cumulative Probability: 0.6496107
```

In this example, we're calculating the cumulative probability that a random variable following a binomial distribution with parameters `size = 10` and `prob = 0.3` is less than or equal to `3`. The result is stored in the cumulative_prob variable and printed to the console.

You can use the `pbinom()` function to answer questions like "What is the probability of getting at most 3 successes in 10 trials with a success probability of 0.3?" by specifying the appropriate values for `q`, `size`, and `prob`.

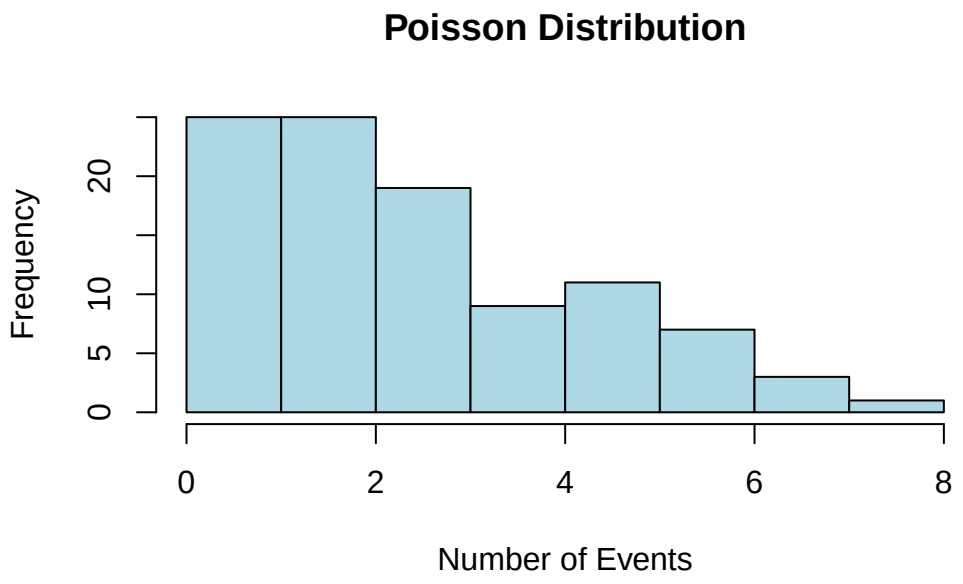
5.3 Poisson probability distributions (Optional)

To generate a data set with a Poisson distribution in R, you can use the `rpois()` function. The Poisson distribution is often used to model the number of events occurring in a fixed interval of time or space when the events happen with a known constant mean rate. Here's how you can use `rpois()`:

```
# Set the parameters for the Poisson distribution
lambda <- 3  # Mean (average) rate of events

# Generate a dataset with a Poisson distribution
poisson_data <- rpois(n = 100, lambda = lambda)

# Print the generated dataset
print(poisson_data)
```

```
  [1] 3 2 4 2 4 0 2 3 2 1 1 1 3 5 2 5 1 7 1 4 5 1 2 3 2 2 1 1 3 3 5 3 3 4 7 1 0
 [38] 3 1 2 3 3 0 2 2 4 2 4 6 6 5 1 2 3 4 2 2 8 1 6 3 3 4 2 2 2 6 5 3 5 0 6 2 2
 [75] 5 5 3 4 2 2 1 5 3 0 1 1 6 6 1 2 2 3 7 1 2 1 5 1 3 1
```

```
# Create a histogram to visualize the data
hist(poisson_data, main = "Poisson Distribution", xlab = "Number of Events",
     ylab = "Frequency", col = "lightblue", border = "black")
```



```r
# Sample mean
mean(poisson_data)
```

```
[1] 2.89
```

```r
#Theoretical mean = lambda

# Sample Variance
var(poisson_data)
```

```
[1] 3.41202
```

```r
#Theoretical variance = lambda
```

To calculate the probability of a specific value occurring in a Poisson distribution in R, you can use the `dpois()` function. This function calculates the *probability mass function* (PMF) of the Poisson distribution. Here's how to use it.

```r
# Set the parameters for the Poisson distribution
x <- 2     # The specific value for which you want to calculate the probability
lambda <- 3  # Mean (average) rate of events

# Calculate the probability of getting exactly 'x' events
probability <- dpois(x, lambda)

# Print the calculated probability
cat("Probability of", x, "events:", probability, "\n")
```

```
Probability of 2 events: 0.2240418
```

To calculate the *cumulative distribution function* (CDF) for a Poisson distribution in R, you can use the `ppois()` function. This function calculates the cumulative probability that a Poisson random variable is less than or equal to a specified value. Here's how to use it:

```r
# Set the parameters for the Poisson distribution
x <- 2  # The specific value to calculate the cumulative probability
lambda <- 3  # Mean (average) rate of events

# Calculate the cumulative probability of getting less than or equal to 'x' events
cumulative_prob <- ppois(x, lambda)

# Print the calculated cumulative probability
cat("Cumulative Probability of less than or equal to", x, "events:",
    cumulative_prob, "\n")
```

```
Cumulative Probability of less than or equal to 2 events: 0.4231901
```

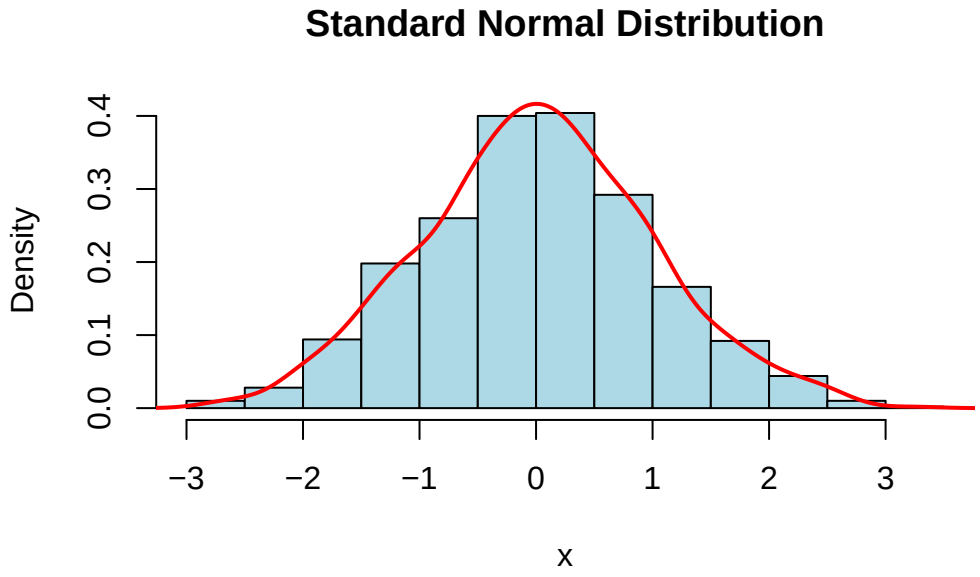
6 NORMAL PROBABILITY DISTRIBUTION

r-function	Description
`rnorm(n,mean,sd)`	generate `n` random values of standard normal distribution with the given `mean` and `sd`.
`hist(x)`	Plot the histogram of the data vector `x`, pass `probability=TRUE` to use density estimate. Pass `breaks` argument to specify edges of bins. Eg.: `breaks = seq(0,1, by=0.1)`. `breaks="FD"` is a method based on data variability.
`seq(start, end, by=step)`	Generate a sequence.
`density(x)`	Estimate the density of the data vector `x`
`lines(x,y)`	Add a line to an existing plot. `y` may be omitted depending on `x`
`pnorm(q,mean=0, sd=1)`	Calculate the cumulative probability $P(X \leq q)$ for a normal distributed random variable X with a given `mean` and `sd`.
`diff(x)`	Calculate the first difference of a vector `x`.
`qnorm(p, mean=0, sd=1)`	Calculate the quantile of the normal distribution corresponding to the probability `p` (from left-tail).
`scale(x,center, scale)`	Scale data `x` to z-score using a given mean (`center`) and standard deviation (`scale`). E.g.: `scale(x, center=5, scale=2)`
`rbinom(s,size=n, prob=p)`	Generate `s` random binomial-distributed values with `n` trials and success probability `p`
`replicate(n, expr)`	Perform the Monte-Carlo simulation by replicating the experiment given by the expression `expr` `n` times.

6.1 THE standard normal distribution

6.1.1 Normal distribution graph (Optional)

```r
set.seed(123)                          # Set the seed for reproducibility
x <- rnorm(1000, mean = 0, sd = 1)  # Generate data for a standard normal distribution

# Plot the data with density curve
hist(x, probability = TRUE, col = "lightblue", main = "Standard Normal Distribution")
lines(density(x), col = "red", lwd = 2)
```



6.1.2 Find the probability (area) when z scores are given

```
# Find the area under the curve to the left of a certain value: P(z<1)
pnorm(1, mean = 0, sd = 1)
```

```
[1] 0.8413447
```

```
# Find the area under the curve to the right of a certain value: P(z>1)
1-pnorm(1, mean = 0, sd = 1)
```

```
[1] 0.1586553
```

```
# Find the area under the curve between two values: P(-1<z<1)
diff(pnorm(c(-1, 1), mean = 0, sd = 1))
```

```
[1] 0.6826895
```

6.1.3 Find z scores when the area is given

```r
# Find the value with a certain area under the curve to its left: critical value
alpha <- 0.05
qnorm(1-alpha, mean = 0, sd = 1) # find the critical Z score.
```

```
[1] 1.644854
```

6.2 REAL application of normal distribution

6.2.1 Convert an individual x value to a z-score

```r
x <- 80  # the individual value
mu <- 75  # the mean of the distribution
sigma <- 10  # the standard deviation of the distribution

# Calculate z-scores for the individual value using scale()
z_scores <- scale(x, center = mu, scale = sigma)
cat("Z-score:", z_scores, "\n") # print the z-score
```

```
Z-score: 0.5
```

```r
z <- (x - mu) / sigma  # find the z-score by using the formula
cat("Z =", z, "\n") # print the z-score
```

```
Z = 0.5
```

6.2.2 Find the probability when x value is given (page 269 Pulse Rates Question)

```r
x1 <- 60
x2 <- 80
mu <- 69.6
sigma <- 11.3
# Find the probability that X is less than 60: P(X<60)
pnorm(x1, mean = mu, sd = sigma)
```

```
[1] 0.1977856
```

```r
# Find the probability that X is great than 80: P(X>80)
1-pnorm(x2, mean = mu, sd = sigma)
```

```
[1] 0.1786939
```

```r
# Find the probability between two values: P(60<X<80)
diff(pnorm(c(x1, x2), mean = mu, sd = sigma))
```

```
[1] 0.6235205
```

6.2.3 Convert a z-score back to x value

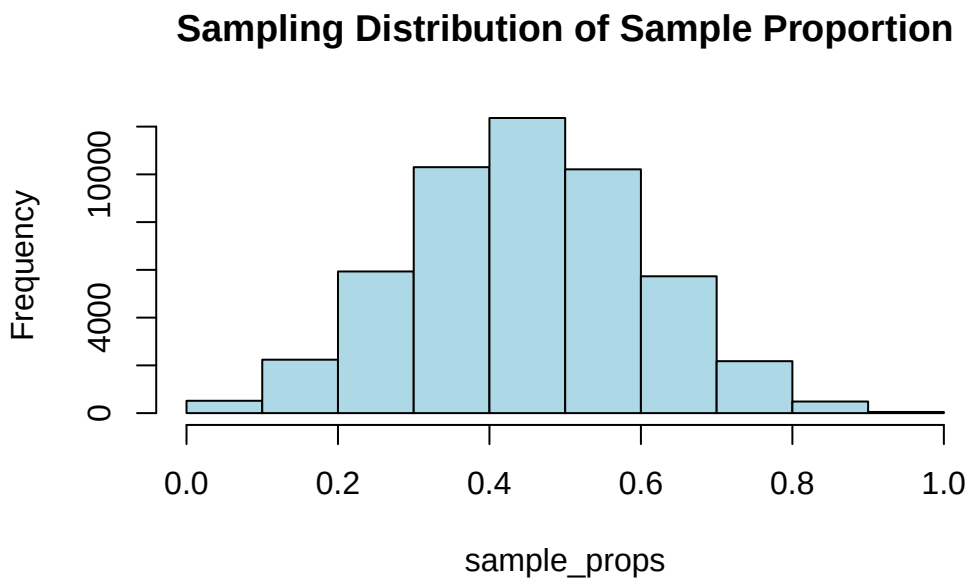
```r
z <- 1.96  # the z-score
mu <- 100  # the mean of the distribution
sigma <- 15  # the standard deviation of the distribution
x <- z * sigma + mu  # convert the z score to individual x value using formula
cat("X =", x, "\n")  # print the individual x value
```

```
X = 129.4
```

6.3 SAMPLING distributions and estimators (Optional)

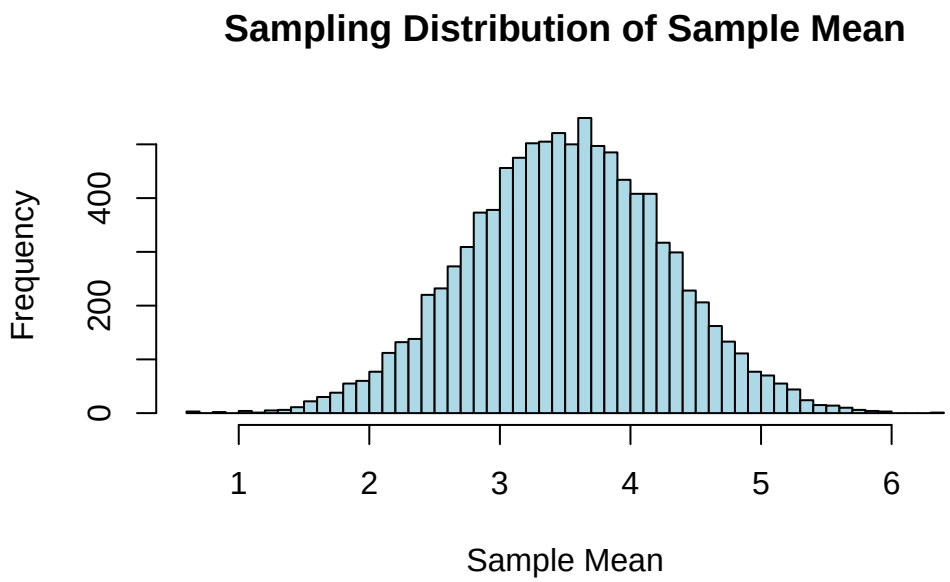
6.3.1 General behavior of sampling distribution of sample proportions

```r
# Set the seed for reproducibility
set.seed (123)
# Generate data
n <- 10  # sample size
p <- 0.5  # population proportion
samples <- replicate(50000, rbinom(1, size = n, prob = p))

# Calculate sample proportions of successes
sample_props <- samples / n

# Plot the histogram

hist(sample_props, breaks = seq( 0, 1, by = 0.1 ), col = "lightblue",
     main = "Sampling Distribution of Sample Proportion")
```



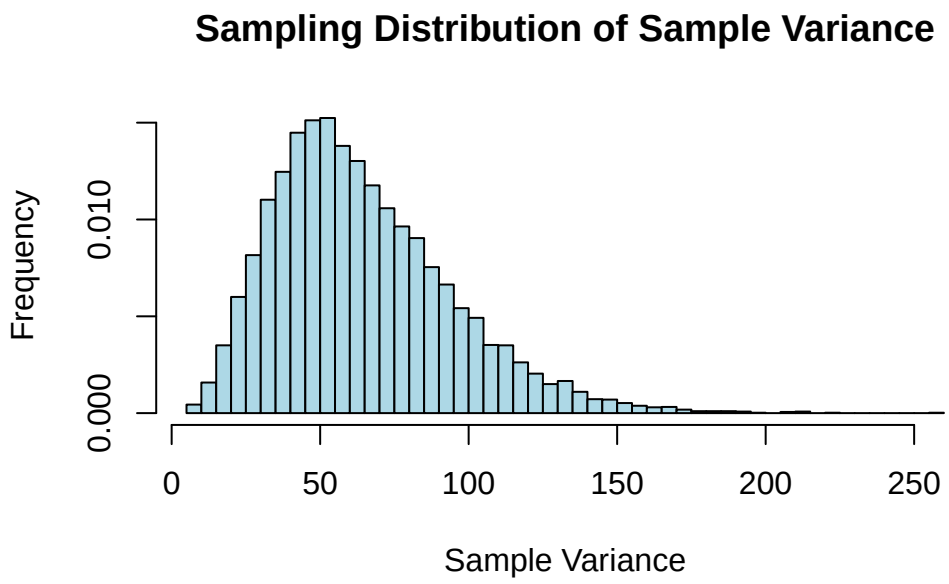
6.3.2 general behavior of sampling distribution of sample means

```
#input the parameter values
mu <- 3.5
sigma <- 1.7
n <- 5
# Simulate sampling distribution
sample_means <- replicate(10000, mean(rnorm(n, mu, sigma)))

# Create a histogram of the sampling distribution of the sample mean
hist(sample_means, breaks ="FD",  main = "Sampling Distribution of Sample Mean",
     xlab = "Sample Mean", ylab = "Frequency", col = "lightblue", border = "black")
```



6.3.3 general behavior of sampling distribution of sample variances

```r
mu <- 4    # True population mean
sigma <- 8       # Population standard deviation
sample_size <- 10         # Sample size
num_samples <- 10000       # Number of samples
# Function to calculate sample variance
sample_variance <- function(sample) {
  n <- length(sample)
  mean_sample <- mean(sample)
  sum_squared_deviations <- sum((sample - mean_sample)^2)
  return(sum_squared_deviations / (n - 1))
}
# Simulate sampling distribution
sample_variances <- replicate(num_samples, sample_variance(rnorm(sample_size,
                                                                 mu, sigma)))

# Create a histogram of the sampling distribution of sample variance
hist(sample_variances, breaks = "FD", freq = FALSE,
     main = "Sampling Distribution of Sample Variance",
     xlab = "Sample Variance", ylab = "Frequency", col = "lightblue", border = "black")
```



6.4 THE central limit theorem

6.4.1 Find the probability when individual value is used (Page 292 Ejection Seat Question)

```
mu <- 171 # population mean
sigma <- 46 # population standard deviation
n <- 25 # sample size
x_lower <- 140
x_upper <- 211

# Find the probability between two X values
probability_range <- diff(pnorm(c(x_lower, x_upper), mean = mu, sd = sigma))
probability_range
```

```
[1] 0.5575477
```

6.4.2 Find the probability when sample mean is used (Page 292 Ejection Seat Question)

```r
# Find the probability between two mean values $x/bar$ (CLT)
standard_error <- sigma / sqrt(n) # Calculate the standard error of the sample mean
probability_range <- diff(pnorm(c(x_lower, x_upper), mean = mu,
                                sd = standard_error))# Find the probability
probability_range
```

```
[1] 0.9996167
```

7 ESTIMATING PARAMETERS AND DETERMINGING SAMPLE SIZES

r-function	Description
`prop.test(successes, n, conf.level=0.95)`	Perform a hypothesis test for a single proportion. Pass a hypothesized (H_0) proportion `p` if it's not 0.5. Eg. `p=0.6` for H_0. Pass a parameter `alternative` for alternative hypothesis: "two.sided"(default) ,"less", "greater". Note the `conf.int` given by the test uses Wilson's method than the Wald method used in the book.
`t.test(x, conf.level=0.95)`	Perform a t-test for a population mean. Accepts an additional `alternative` argument for H_1. The default hypothesized mean is `mu=0`. Otherwise, pass a hypothesized mean value.
`qt(p, df)`	calculate the quantile for the probability `p` of t-distribution with degree of freedom equal to `df`
`qchisq(p, df)`	calculate the quantile for the probability `p` of χ^2-distribution with degree of freedom equal to `df`

7.1 ESTIMATING a population proportion (Page 313 Online Course Example)

7.1.1 Getting the CI directly

```r
p_hat <- 0.53 # 0.53 for 53% sample proportion
n <- 950 # sample size
success <- n*p_hat # number of success

# Calculate a 95% confidence interval for the population proportion
result <- prop.test(success, n, conf.level = 0.95)

# Extract the confidence interval
conf_interval <- result$conf.int
# Print the confidence interval (calculated by the Wilson method)
cat("Confidence Interval:", conf_interval[1], "to", conf_interval[2], "\n")
```

```
Confidence Interval: 0.4976792 to 0.5620751
```

7.1.2 Getting the CI step by step using the textbook's Wald's Method (slightly different result than the result given above)

1.Critical value

```
# Confidence level (e.g., 0.95 for 95% confidence)
confidence_level <- 0.95
# get alpha value
alpha <- 1-confidence_level

# Find the critical Z-value using qnorm()
critical_z <- qnorm (1 - alpha/2)
# Print the result
cat("Critical Z =", critical_z, "\n")
```

```
Critical Z = 1.959964
```

2. Margin of error

```
# Calculate the standard error
standard_error <- sqrt((p_hat * (1 - p_hat)) / n)
# Calculate the margin of error
margin_of_error <- critical_z * standard_error
# Print the result
cat("E=", margin_of_error, "\n")
```

```
E= 0.03173753
```

3. Confidence interval

```
# Calculate the confidence interval
confidence_interval <- c (p_hat - margin_of_error,
                          p_hat + margin_of_error)

# Print the confidence interval
cat("Confidence Interval:", confidence_interval[1], "to", confidence_interval[2], "\n")
```

```
Confidence Interval: 0.4982625 to 0.5617375
```

7.2 ESTIMATING a population mean

7.2.1 Get the CI directly with sample data values given. (Page 343 Mercury question)

In this case, the population σ is unknown.

```
# Calculate a 98% confidence interval for the population mean
#Sample data
mercury <- c(0.56, 0.75, 0.10, 0.95, 1.25, 0.54, 0.88)
result <- t.test(mercury,conf.level = 0.98)

# Extract the confidence interval
conf_interval <- result$conf.int

# Print the confidence interval
cat("Confidence Interval:", conf_interval[1], "to", conf_interval[2], "\n")
```

```
Confidence Interval: 0.2841145 to 1.153028
```

7.2.2 Get the CI step by step with given mean and standard deviation (Page 341 Hershey kisses question)

1. Critical value

```
confidence_level <- 0.99  # Confidence level (e.g., 0.99 for 99% confidence)
alpha <- 1- confidence_level
n <- 32         # Sample size

# Calculate the degrees of freedom
degrees_of_freedom <- n - 1

# Find the critical t-value using qt()
critical_t <- qt(1 - alpha/ 2, df = degrees_of_freedom)

# Print the result
cat("Critical t-value for degrees of freedom =", degrees_of_freedom,
    "and confidence level =", confidence_level, ":", critical_t, "\n")
```

```
Critical t-value for degrees of freedom = 31 and confidence level = 0.99 : 2.744042
```

2. Margin of error

```r
# Given sample standard deviation (this is s value)
sample_standard_deviation <- 0.1077

# Calculate the standard error
standard_error <- sample_standard_deviation / sqrt(n)

# Calculate the margin of error
margin_of_error <- critical_t * standard_error

# Print the result
cat("Margin of Error for confidence level =", confidence_level,
    "and sample size =", n, ":", margin_of_error, "\n")
```

```
Margin of Error for confidence level = 0.99 and sample size = 32 : 0.0522434
```

3. Confidence interval

```r
x_bar<- 4.5210       # Sample mean

# Calculate the lower and upper bounds of the confidence interval
lower_bound <- x_bar - margin_of_error
upper_bound <- x_bar + margin_of_error

# Print the result
cat("Confidence Interval:", lower_bound, "to", upper_bound, "\n")
```

```
Confidence Interval: 4.468757 to 4.573243
```

7.3 ESTIMATING a population Deviation or Variance (body temperature example page 353)

7.3.1 Critical values

```r
confidence_level <- 0.95  # Confidence level ( 0.95 for 95% confidence)
alpha <- 1- confidence_level
sample_size <- 106          # Sample size
degrees_of_freedom <- sample_size - 1  # df for the chi-squared distribution

# Find the critical values using the chi-squared distribution
lower_critical_value <- qchisq(1-alpha/2, df = degrees_of_freedom)
```

```r
upper_critical_value <- qchisq(alpha/2, df = degrees_of_freedom)

# Print the results
cat("Lower Critical Value:", lower_critical_value, "\n")
```

```
Lower Critical Value: 135.247
```

```r
cat("Upper Critical Value:", upper_critical_value, "\n")
```

```
Upper Critical Value: 78.5364
```

7.3.2 Confidence interval

```r
sample_standard_deviation <- 0.62                  # sample standard deviation s
sample_variance <- sample_standard_deviation^2     # Sample variance

# Calculate the confidence interval for variance
confidence_interval <- c(((sample_size - 1) * sample_variance) / lower_critical_value,
                         ((sample_size - 1) * sample_variance) / upper_critical_value)

# Print the confidence interval
confidence_interval
```

```
[1] 0.2984318 0.5139273
```

8 Hypothesis Testing

r-function	Description
`prop.test(successes, n, conf.level=0.95)`	Perform a hypothesis test for a single proportion. Pass a hypothesized (H_0) proportion `p` if it's not 0.5. Eg. `p=0.6` for H_0. Pass a parameter `alternative` for alternative hypothesis: "two.sided"(default) ,"less", "greater". Note the `conf.int` given by the test uses Wilson's method than the Wald method used in the book. The test returns three values: `$p.value` (for p-value), `$statistic` (for test-statistic), and `$conf.int` (for confidence interval).
`t.test(x, conf.level=0.95)`	Perform a t-test for a population mean. Accepts an additional `alternative` argument for H_1. The default hypothesized mean is `mu=0`. Otherwise, pass a hypothesized mean value.
`z.test(x, sigma.x=sigma, conf.level=0.95)`	Perform a z-test for a population mean with known `sigma.x=sigma`. Accepts an additional `alternative` argument for H_1. The default hypothesized mean is `mu=0`. Otherwise, pass a hypothesized mean value.
`qnorm(p, mean=0, sd=1)`	Calculate the quantile of the normal distribution corresponding to the probability `p` (from left-tail).
`qt(p, df)`	calculate the quantile for the probability `p` of t-distribution with degree of freedom equal to `df`
`qchisq(p, df)`	calculate the quantile for the probability `p` of χ^2-distribution with degree of freedom equal to `df`
`attach(df)`	add a data frame `df` to the search path, which allows you to access the variables within the data frame `df` directly by their names instead of using a normal way such as `df$var`.
`table`	tabulate the frequency counts of distinct values.
`prop.table(table)`	compute the proportions of a table or data. Pass an argument `margin` for the direction: `1` for rows, `2` for columns, or `NULL` for the entire table (default).

8.1 Basic of Hypothesis Testing

We will use the following functions to perform hypothesis tests.

```
library(BSDA)
# prop.test(x, n, p = NULL,
#           alternative = c("two.sided", "less", "greater"),
#           conf.level = 0.95, correct = TRUE)

# t.test(x, y = NULL,
#       alternative = c("two.sided", "less", "greater"),
#       mu = 0, paired = FALSE, var.equal = FALSE,
#       conf.level = 0.95, ...)

# z.test(
#   x, y = NULL,
#   alternative = "two.sided",
#   mu = 0, sigma.x = NULL, sigma.y = NULL,
#   conf.level = 0.95)
```

We use `qnorm()` and `qt()` functions to calculate critical values. For example, we can obtain $z_{0.95}$ using the `qnorm(0.95)` for a normal distribution, and the critical value $t_{0.05,5}$ using `qt(0.95, 5)` for a t-distribution with 5 degree of freedom with $\alpha = 0.05$ as below.

```
qnorm(0.95)
```

```
[1] 1.644854
```

```
qt(0.95, 5)
```

```
[1] 2.015048
```

8.2 Testing a Claim About a Proportion

`mtcars` dataset has data for 32 automobiles in 1973-1974 with 11 variables. Among these variable, we are interested to check if the proportion of V-shaped engine (`vs = 0`) is 0.5. That is, $H_0 : p = 0.5$.

```
data(mtcars)
attach(mtcars)
table(vs)
```

```
vs
 0  1
18 14
```

```r
prop.table(table(vs))
```

```
vs
     0      1
0.5625 0.4375
```

8.2.1 Two-sided proportion test using the z-test (method in the textbook)

We first check if we can use a normal approximation to perform a proportion test. With a sample size of $n = 32$ and a proportion of interest $p = 0.5$, both the expected number of successes and failures are $np = n(1-p) = 32 \cdot 0.5 = 16$. Since they are greater than 5, we can apply the proportion test using a normal approximation. In our sample, the number of success (`vs=0`) is 18 and the sample proportion is 0.56.

```r
# Example data
successes <- 18  # Number of successes
trials <- 32    # Total number of trials
null_prob <- 0.5  # Hypothesized population proportion under the null hypothesis

# Calculate the sample proportion
sample_proportion <- successes / trials

# Perform the z-test
z_stat <- (sample_proportion - null_prob) / sqrt(null_prob * (1 - null_prob) / trials)

# Calculate the p-value
p_value <- 2 * (1 - pnorm(abs(z_stat)))

# Calculate the critical value
alpha <- 0.05
critical_value <- c(qnorm(alpha),qnorm(1-alpha))

# Print the results
cat("Z-statistic:", z_stat, "\n")
```

```
Z-statistic: 0.7071068
```

```r
cat("p-value:", p_value, "\n")
```

```
p-value: 0.4795001
```

```r
cat("Critical values:", critical_value, "\n")
```

```
Critical values: -1.644854 1.644854
```

We are ready to make a decision using the following method:

- p-value method: The p-value 0.4795001 is greater than the *significance level* $\alpha = 0.05$, therefore we *fail* to reject the Null hypothesis $H_0 : p = 0.5$.
- **critical value** method: The test statistics 0.7071068 is not as extreme as the two critical values, therefore we *fail* to reject the Null Hypothesis.

8.2.2 Two-sided Proportion Test using the built-in function `prop.test`

Next we will use the R built-in `prop.test()` function to perform one sample proportion test. The syntax is below.

```r
# prop.test(x, n, p = p_0, conf.level=0.95, alternative=c("two.sided", "less",
# "greater"))
```

Depending on the alternative hypothesis H_1, we can choose one among `two.sided`, `less`, and `greater`:

1. $H_1 : p \neq p_0$: `alternative = "two.sided"`
2. $H_1 : p < p_0$:`alternative = "less"`
3. $H_1 : p > p_0$: `alternative = "greater"`

It is remarkable that the built-in `prop.test` uses the Pearson χ^2 distributed test statistic which is different than the $z-$test used by the textbook.

$$H_0 : p = 0.5 \quad \text{vs} \quad H_1 : p \neq 0.5$$

```r
res <- prop.test(x=18, n=32, p = 0.50, alternative = "two.sided", conf.level = 0.95)
res
```

```
    1-sample proportions test with continuity correction

data:  18 out of 32, null probability 0.5
X-squared = 0.28125, df = 1, p-value = 0.5959
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.3788033 0.7316489
```

```
sample estimates:
     p
0.5625
```

```
cat("The p-value is given by ", res$p.value, "\n")
```

```
The p-value is given by  0.5958831
```

```
cat("The $chi^2$ test statistic  is given by ", res$statistic, "\n")
```

```
The $chi^2$ test statistic  is given by  0.28125
```

```
cat("The confidence interval is given by (", res$conf.int[1], "," ,res$conf.int[2], ")\n")
```

```
The confidence interval is given by ( 0.3788033 , 0.7316489 )
```

Decision:

- **P-Value**: we fail to reject the Null Hypothesis since p-value 0.596 is greater than $\alpha = 0.05$.
- **Critical Value**: the χ^2 test statistic 0.281 is not as extreme as the critical values which can be found as below. Thus, we fail to reject the Null Hypothesis.

```
# the critical value can be calculated by the following code.
c(qchisq(0.025, 1), qchisq(0.975,1))
```

```
[1] 0.0009820691 5.0238861873
```

- **Confidence Interval**: the claimed proportion 0.5 falls within the confidence interval of (0.379, 0.732). Thus we fail to reject the null hypothesis.

8.2.3 One-sided Proportion Test

$$H_0 : p = 0.5 \quad \text{vs} \quad H_1 : p > 0.5$$

```
res <- prop.test(x=18, n=32, p = 0.50, alternative = "greater", conf.level = 0.95)
res
```

```
    1-sample proportions test with continuity correction

data:  18 out of 32, null probability 0.5
X-squared = 0.28125, df = 1, p-value = 0.2979
alternative hypothesis: true p is greater than 0.5
95 percent confidence interval:
 0.4041836 1.0000000
sample estimates:
     p
0.5625
```

Decision:

- **P-Value**: we fail to reject the null hypothesis since p-value 0.298 is greater than $\alpha = 0.05$.
- **Critical Value**: the χ^2 test statistic 0.281 does not fall in the critical region which is greater than 3.8414588 or smaller than 0.0039321. Thus, we fail to reject the null hypothesis. The critical value can be found by

```
# the critical value can be calculated by the following code.
c(qchisq(0.05,1), qchisq(0.95,1))
```

```
[1] 0.00393214 3.84145882
```

- **Confidence Interval**: the claimed proportion 0.5 falls within the confidence interval of (0.404, 1). Thus we fail to reject the null hypothesis.

8.3 Testing a Claim About a Mean

8.3.1 Unknown σ with Normality Assumption

We use one sample t-test with `t.test()` function when we assume normality for population or the sample size is large enough. The syntax is as below if we want to test with a sample vector (variable) `x` for $H_0 : \mu = m$ with a given confidence level `conf.level`, for example, `conf.level=0.95`.

```
# t.test(x, mu= m, conf.level=0.95, alternative=c("two.sided", "less", "greater"))
```

Depending on the alternative hypothesis H_1, we can choose one among `two.sided`, `less`, and `greater`.

1. $H_1 : \mu \neq m$: `alternative = "two.sided"`
2. $H_1 : \mu < m$:`alternative = "less"`
3. $H_1 : \mu > m$: `alternative = "greater"`

As an example, we test for `mpg` with $H_0 : \mu = 22$. That is, we test if the population mean of `mpg` is equal to 22. `mtcars` cars have 32 samples and the sample size is large enough to use t-test with $\alpha = 0.05$.

8.3.1.1 Two-sided t-test

$$H_0 : \mu = 22 \quad \text{vs} \quad H_1 : \mu \neq 22$$

```
res <- t.test(mpg, mu=22, alternative = "two.sided", conf.level = 0.95)
res
```

```
    One Sample t-test

data:  mpg
t = -1.7921, df = 31, p-value = 0.08288
alternative hypothesis: true mean is not equal to 22
95 percent confidence interval:
 17.91768 22.26357
sample estimates:
mean of x
 20.09062
```

```
cat("The p-value is given by ", res$p.value, "\n")
```

```
The p-value is given by  0.08287848
```

```
cat("The test statistic is given by ", res$statistic, "\n")
```

```
The test statistic is given by  -1.792127
```

```r
cat("The confidence interval is given by (", res$conf.int[1], "," ,res$conf.int[2], ")\n")
```

```
The confidence interval is given by ( 17.91768 , 22.26357 )
```

Decision:

- **P-Value**: we fail to reject the null hypothesis since p-value 0.083 is greater than $\alpha = 0.05$.
- **Critical Value**: the test statistic t = -1.792 is closer to 0 than the critical values which can be found as below. Thus, we fail to reject the null hypothesis.

```r
# the critical value can be calculated by the following code.
c(qt(0.025, df=31), qt(0.975, df=31))
```

```
[1] -2.039513  2.039513
```

- **Confidence Interval**: the claimed mean 22 falls within the confidence interval of (17.918, 22.264). Thus we fail to reject the null hypothesis.

8.3.1.2 One-sided t-test $H_1 : \mu < m$

$$H_0 : \mu = 22 \quad \text{vs} \quad H_1 : \mu < 22$$

```r
res <- t.test(mpg, mu=22, alternative = "less", conf.level = 0.95)
res
```

```
    One Sample t-test

data:  mpg
t = -1.7921, df = 31, p-value = 0.04144
alternative hypothesis: true mean is less than 22
95 percent confidence interval:
     -Inf 21.89707
sample estimates:
mean of x
 20.09062
```

Decision:

- **P-Value**: we reject the null hypothesis since p-value 0.041 is less than $\alpha = 0.05$.

- **Critical Value**: the test statistic t = -1.792 falls in the critical region which is less than $t_{0.05,31}$ = -1.696. Thus, we reject the null hypothesis.

```
# the critical value can be calculated by the following code.
qt(0.05, df=31)
```

```
[1] -1.695519
```

- **Confidence Interval**: the claimed mean $\mu = 22$ does not fall within the confidence interval of $(-\infty, 21.897)$. Thus we reject the null hypothesis.

8.3.2 Known σ with Normality Assumption

We use one sample z-test or normal test with `z.test()` function when we assume normality for population with known population standard deviation σ. The syntax is as below if we want to test with a sample vector (variable) `x` for $H_0 : \mu = m$ with $\alpha = 0.05$ and a known `sigma`.

```
#library(BSDA)
# z.test(x, mu = m, sigma.x = sigma, conf.level = 0.95,
# alternative = c("two.sided", "less", "greater"))
```

Depending on the alternative hypothesis H_1, we can choose one among `two.sided`, `less`, and `greater`.

1. $H_1 : \mu \neq m$: `alternative = "two.sided"`
2. $H_1 : \mu < m$:`alternative = "less"`
3. $H_1 : \mu > m$: `alternative = "greater"`

For example, we test for `mpg` with $H_0 : \mu = 22$. Assume `mpg` follows a normal distribution with $\sigma = 6$, then we can use z-test with $\alpha = 0.05$.

8.3.2.1 Two-sided z-test

$$H_0 : \mu = 22 \quad \text{vs} \quad H_1 : \mu \neq 22$$

```
library(BSDA)
res <- z.test(mpg, mu=22, sigma.x = 6, alternative = "two.sided", conf.level = 0.95)
res
```

```
    One-sample z-Test

data:  mpg
z = -1.8002, p-value = 0.07183
alternative hypothesis: true mean is not equal to 22
95 percent confidence interval:
 18.01177 22.16948
sample estimates:
mean of x
 20.09062
```

```
cat("The p-value is given by ", res$p.value, "\n")
```

```
The p-value is given by  0.07183285
```

```
cat("The test statistic is given by ", res$statistic, "\n")
```

```
The test statistic is given by  -1.800176
```

```
cat("The confidence interval is given by (", res$conf.int[1], "," ,res$conf.int[2], ")\n")
```

```
The confidence interval is given by ( 18.01177 , 22.16948 )
```

Decision:

- **P-Value**: we fail to reject the null hypothesis since p-value 0.072 is greater than $\alpha = 0.05$.
- **Critical Value**: the test statistic z = -1.8 is closer to 0 than the critical values as found below. Thus, we fail to reject the null hypothesis.

```
# the critical value can be calculated by the following code.
c(qnorm(0.025), qnorm(0.975))
```

```
[1] -1.959964  1.959964
```

- **Confidence Interval**: the claimed mean 22 falls within the confidence interval of (18.012, 22.169). Thus we fail to reject the null hypothesis.

8.3.2.2 One-sided z-test $H_1 : \mu < m$

$$H_0 : \mu_{mpg} = 22 \quad \text{vs} \quad H_1 : \mu_{mpg} < 22$$

```r
res <- z.test(mpg, mu=22, sigma.x = 6, alternative = "less", conf.level = 0.95)
res
```

```
    One-sample z-Test

data:  mpg
z = -1.8002, p-value = 0.03592
alternative hypothesis: true mean is less than 22
95 percent confidence interval:
       NA 21.83526
sample estimates:
mean of x
 20.09062
```

Decision:

- **P-Value**: we reject the null hypothesis since p-value 0.036 is less than $\alpha = 0.05$.
- **Critical Value**: the test statistic z = -1.8 falls in the critical region which is less than $z_{0.05}$ = -1.645. Thus, we reject the null hypothesis.

```r
# the critical value can be calculated by the following code.
qnorm(0.05)
```

```
[1] -1.644854
```

- **Confidence Interval**: the claimed mean does not fall within the confidence interval of $(-\infty, 21.835)$. Thus we reject the null hypothesis.

9 INFERENCE FROM TWO SAMPLES

There is no content for this chapter.

10 Correlation and Regression

r-function	Description
`data('dataset_name')`	Load a R built-in dataset named by 'dataset_name'
`names(x)`	retrieve or set the names of elements in `x`
`attach(df)`	add a data frame `df` to the search path, which allows you to access the variables within the data frame `df` directly by their names instead of using a normal way such as `df$var`.
`cor(x,y)`	Find the correlation of two vectors `x` and `y`
`qplot(x,y,data)`	create a quick plot data `(x,y)` in the dataframe `data`
`geom_text(aes(x,y, label))`	Add a `label` at the coordinate `(x,y)` in the current plot.
`lm(y~x, data)`	perform the linear regression of `y~x`, where `y`,`x` are column names in the dataframe `data`.
`summary(lm_model)`	summarize the linear model `lm_model` obtained by the R-function `lm`.

10.1 Correlation

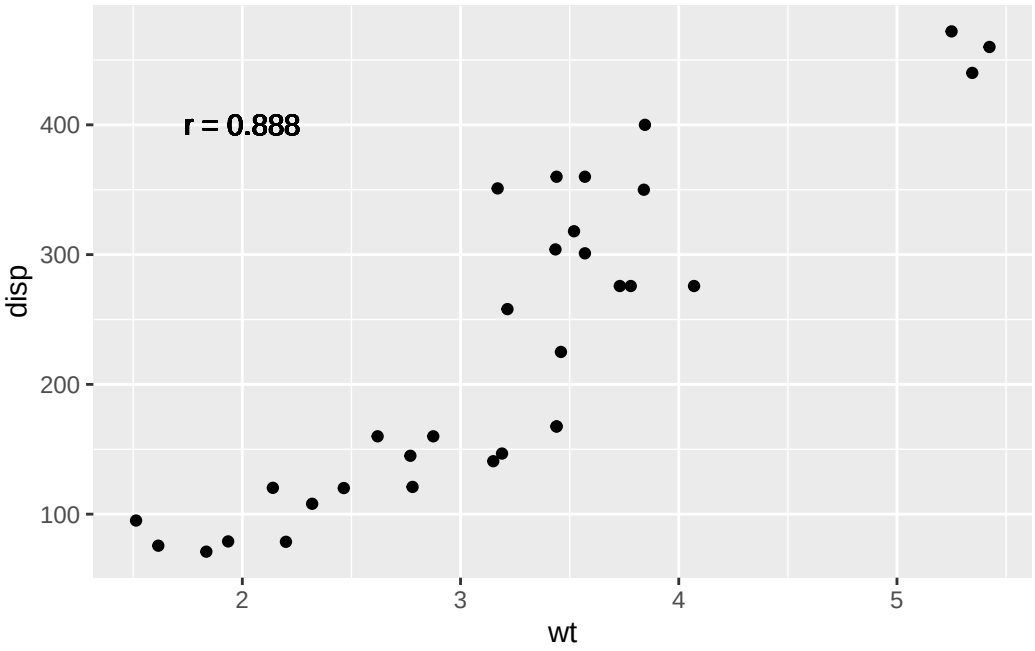
We check if a linear correlation exists between two variables using `cor()` function.

```
# We can calculate the correlation coefficient between x and y with the following code.
# cor(x, y)
```

```
library(tidyverse)
library(patchwork)
data("mtcars")
names(mtcars)
```

```
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
[11] "carb"
```

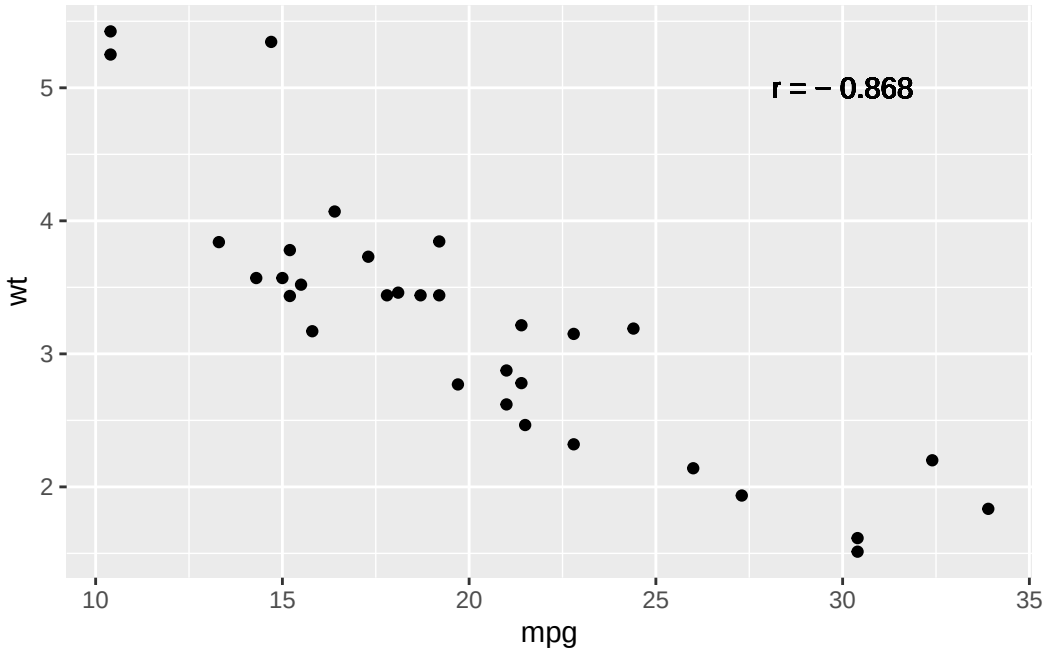
```
attach(mtcars)
# positive correlation
qplot(wt, disp, data = mtcars) +
  geom_text(aes(x=2, y=400, label="r = 0.888"))
```




```
cor(wt, disp)
```

```
[1] 0.8879799
```

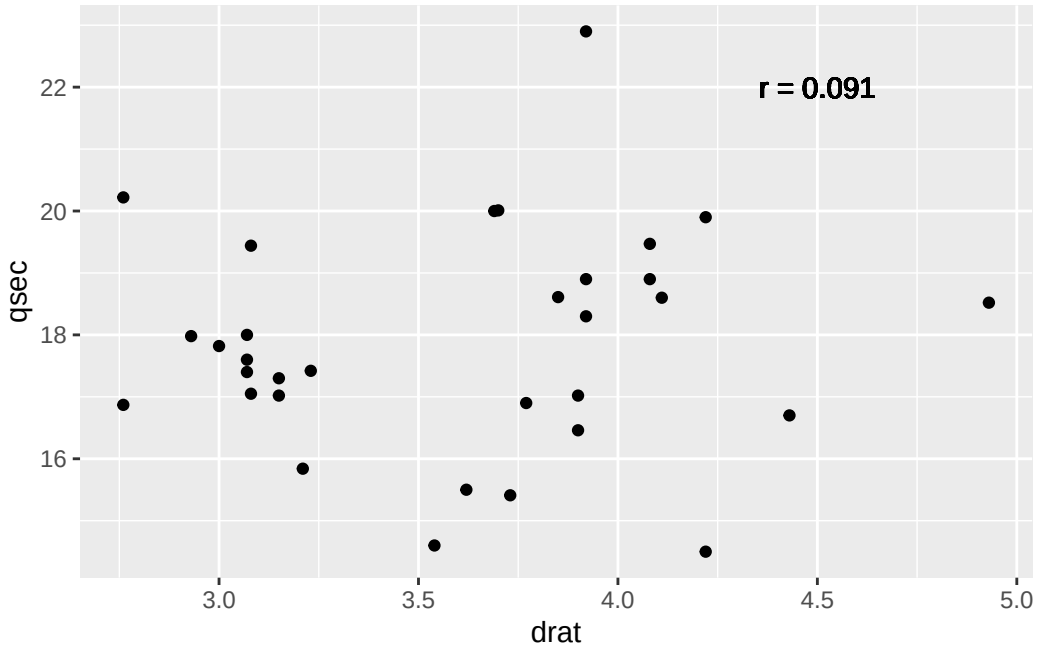
```
# negative correlation
qplot(mpg, wt, data = mtcars)  +
  geom_text(aes(x=30, y=5, label="r = - 0.868"))
```




```
cor(mpg, wt)
```

```
[1] -0.8676594
```

```
# no correlation
qplot(drat, qsec, data = mtcars)  +
  geom_text(aes(x=4.5, y=22, label="r = 0.091"))
```




```
cor(drat, qsec)
```

```
[1] 0.09120476
```

- `wt` and `disp` have a positive correlation with r =0.888.
- `wt` and `disp` have a negative correlation with r = -0.868.
- `wt` and `disp` does not have a significant correlation with r = -0.175.

10.2 Linear Regression

Assume we have a data set `data` with `x` and `y` variables and we model their relationship by linear regression. We can find the slope and the intercept of the estimated regression line using the following code.

```
# res <- lm(y ~ x, data)
# summary(res)
```

For example, we can find the regression line equation between `disp` (x-variable, predictor) and `wt` (y-variable, response) as below.

```
data("mtcars")
res <- lm(wt ~ disp, mtcars)
summary(res)
```

```
Call:
lm(formula = wt ~ disp, data = mtcars)

Residuals:
     Min       1Q   Median       3Q      Max
-0.89044 -0.29775 -0.00684  0.33428  0.66525

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.5998146  0.1729964   9.248 2.74e-10 ***
disp        0.0070103  0.0006629  10.576 1.22e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4574 on 30 degrees of freedom
Multiple R-squared:  0.7885,    Adjusted R-squared:  0.7815
F-statistic: 111.8 on 1 and 30 DF,  p-value: 1.222e-11
```

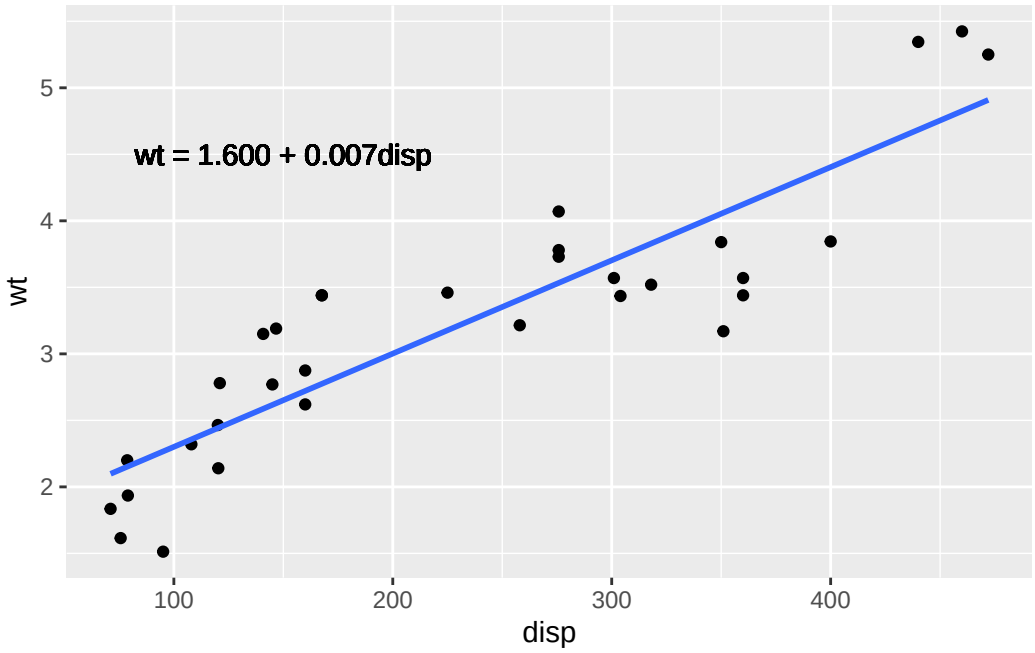
The estimated regression line is `wt = 1.600 + 0.007 disp` since the intercept is 1.6 and the slope is 0.007. Both of them are significantly different from 0 with a significance level $\alpha = 0.05$ because their p-values are almost `0`. The linear relation means that one inch increase in `disp` (displacement) makes 7 lbs increase in `wt` (weight). On average, if a car has a one-inch longer displacement, it is 7 pounds heavier.

If a car has 200 inches displacement, then its estimated weight can be calculated as

$$1.600 + 0.007 \cdot 200 = 3000 \text{ lbs}$$

We next use the `R` package `ggplot` to visualize the data set and the regression line.

```
ggplot(mtcars, aes(x=disp, y=wt)) +  # define x and y
  geom_point()+                      # scatter  plot
  geom_smooth(method=lm, se=FALSE) + # add a regression line
  geom_text(aes(x = 150, y = 4.5, label = "wt = 1.600 + 0.007disp")) #add a label
```


wt = 1.600 + 0.007disp
wt
disp
5
4
3
2
100
200
300
400

References

Triola, Mario F. 2022. *Elementary Statistics*. USA: Pearson.