3.1 READING DATA FRAMES

Before proceeding with any data analysis and visualization it is important to understand how data is organized and also the different type of data that exist in a dataset. A dataset is a set of values of the variables defined for the individuals/subjects in a selected sample that is under study. The variables can be qualitative or quantitative e.g. age, gender, shopping habits etc.

3.1.1 Data frames

A data frame is mostly organized as a *data matrix* where each row of the matrix contains values for one subject. Consider the following data frame:

Table 1: employee.csv

No	Age	Gender	Salary	Monthly Expenditure	Occupation	Healthy Lifestyle
1	34	0	35760	34908	Teacher	Moderate
2	21	0	23500	20950	Analyst	Good
3	59	1	21090	12080	Graphic Designer	Bad
4	45	1	45090	34090	Software Engineer	Good
5	37	0	67050	45780	Manager	Moderate

Each row in the data set contains values captures for one individual. All the rows are of the same length and each column represents a variable. For instance *age* is a variable representing the age of an individual.

There are different types of data that are stored in a data set which are categorized into two groups: *Numeric data* and *Qualitative data (non-numeric data)*

- **Numeric data -** there are two types of numeric data namely:
  - **Discrete data:** the variable can take only integer values e.g. 1, 2
  - **Concrete data:** any real-numbered values e.g. age, salary and monthly expenditure

- **Qualitative** (**non-numeric, categorical**) **data** consists of
  - **Nominal data:** data that is unordered e.g. Occupation
  - **Ordinal or ordered data:** data that is ordered e.g. good- moderate- bad

3.1.2 Reading data from comma-separated-values(csv) files

The command *read.csv* is used to read data from csv files. Assuming that you have a csv file named customer, data from the file can be extracted as follows:

```
# Read data from customer.csv from same directory
> customer <- read.csv("customer.csv")

# Read data from customer.csv by specifying the path (absolute path)
> customer <- read.csv("c:/tutorial/data/customer.csv")

# Read data from customer.csv found in a subdirectory (relative path)
> customer <- read.csv("data/customer.csv")
```

The data is extracted from the csv file and placed in the data frame *customer*.

3.1.3 Reading data from spreadsheets

The *read.table* command is used to extract data from excel spreadsheet as follows:

```
# Read data from customer.xls (header is present) delimited by Tab
> customer <- read.table("customer.xls", header-TRUE)

# Read data from customer.xls (header is present), delimited by “;”
> customer <- read.table("customer.xls", header-TRUE, sep=“;”)
```

3.2 MANIPULATING DATA

3.2.1 Variables in data frames

Contents in the data frame can be viewed as follows:

```
# View first few rows of the dataframe

> head(customers)
  FirstName  LastName      Company                  Address               City
1 Essie      Vaill      Litronic Industries     14225 Hancock Dr        Anchorage
2 Cruz       Roudabush  Meridian Products       2202 S Central Ave       Phoenix
3 Billie     Tinnes     D & M Plywood Inc       28 W 27th St             New York
4 Zackary    Mockus     Metropolitan Elevator Co  286 State St Perth      Amboy
5 Rosemarie  Fifield    Technology Services     3131 N Nimitz Hwy  #-105  Honolulu
6 Bernard    Laboy      Century 21 Keewaydin Prop 22661 S Frontage Rd     Channahon

  County     State  ZIP     Phone        Fax
1 Anchorage   AK    99515   907-345-0962  907-345-1215
2 Maricopa    AZ    85004   602-252-4827  602-252-4009
3 New York    NY    10001   212-889-5775  212-889-5764
4 Middlesex   NJ    8861    732-442-0638  732-442-5218
5 Honolulu    HI    96819   808-836-8966  808-836-6008
6 Will        IL    60410   815-467-0487  815-467-1244

  Email                    Web
1 essie@vaill.com        http://www.essievaill.com
2 cruz@roudabush.com     http://www.cruzroudabush.com
3 billie@tinnes.com      http://www.billietinnes.com
4 zackary@mockus.com     http://www.zackarymockus.com
5 rosemarie@fifield.co   http://www.rosemariefifield.com
```

```
6 bernard@laboy.com      http://www.bernardlaboy.com

# View the type and content of each variable
> str(customers)
'data.frame':           500 obs. of  12 variables:
 $ FirstName: Factor w/ 448 levels "Abel","Adela",..: 135 83 40 447 369 31 396 426 240 280 ...
 $ LastName : Factor w/ 500 levels "Achorn","Agresta",..: 462 380 447 315 142 256 186 352 195 123 ...
 $ Company  : Factor w/ 500 levels "A B C Lock & Key",..: 261 287 124 289 449 88 240 406 246 13 ...
 $ Address  : Factor w/ 490 levels "1 First Federal Plz",..: 94 181 232 237 266 192 464 418 434 387 ...
 $ City     : Factor w/ 332 levels "Abilene","Ada",..: 13 244 212 241 134 56 244 305 318 228 ...
 $ County   : Factor w/ 216 levels "Adams","Alameda",..: 8 118 136 124 86 208 118 106 110 144 ...
 $ State    : Factor w/ 48 levels "AK","AL","AR",..: 1 4 34 31 12 15 4 38 5 10 ...
 $ ZIP      : int  99515 85004 10001 8861 96819 60410 85051 18087 91790 32809 ...
 $ Phone    : Factor w/ 500 levels "201-224-7741",..: 451 250 36 343 379 395 253 269 286 152 ...
 $ Fax      : Factor w/ 500 levels "201-224-7282",..: 451 250 36 344 379 395 253 269 286 152 ...
 $ Email    : Factor w/ 500 levels "abel@tuter.com",..: 147 89 42 499 415 32 444 475 270 316 ...
 $ Web      : Factor w/ 500 levels "http://www.abeltuter.com",..: 147 89 42 499 415 32 444 474 270 315 ...

# Extracting individual objects using $
> firstname <- customers$FirstName
> head(firstname)
[1] Essie     Cruz      Billie    Zackary   Rosemarie Bernard
448 Levels: Abel Adela Aileen Alejandra Alejandro Alene Alfonso Alissa Allan Allie Allyson Alva Alyssa ... Zane
```

3.2.2 Manipulating data frames

There are various manipulations that can be done to the structure and the data of the data frames. Column names can be altered or deleted. Manipulation of the data depends on the type of data in the data frame.

- **Changing column name**
  To change the names of columns in a data frame, they first have to be copied to a vector using the *names function.*

```
# Read the column names
> names(customers)
 [1] "FirstName" "LastName"  "Company" "Address" "City" "County" "State" "ZIP" "Phone"
[10] "Fax"    "Email"    "Web"

# Change LastName to LName
> names(customers)[2] <- "LName"
> names(customers)
 [1] "FirstName" "LName""Company"  "Address"  "City"  "County" "State" "ZIP" "Phone"
[10] "Fax"      "Email"     "Web"
```

- **Deleting columns**

To remove a column a new data frame, without the column to be removed, is created. For instance, if the column *LastName* has be to be removed then a new data frame *customers2* is created from *customers* without the second column '*LastName'*

```
# Delete column 'LastName' - Create data frame customers2 without column 'LastName'
> customers2 <- customers[,-2]
> names(customers2)
[1] "FirstName" "Company" "Address"  "City"  "County"  "State"  "ZIP" "Phone"   "Fax"
[10] "Email"    "Web"
```

3.2.3 Missing data in dataframes

Missing values are represented using *NA* abbreviation which stands for *Not Available.* This abbreviation should not be used to represent other values like Not Applicable or Not Allowed. When importing from source files, NA is automatically to fields containing missing values. Some functions for missing values are presented:

```
# Is a value missing? (TRUE or FALSE)
> is.na(customer)

    FirstName LastName Company Address  City County State   ZIP Phone   Fax Email   Web
 [1,]     FALSE    FALSE   FALSE   FALSE FALSE  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [2,]     FALSE    FALSE   FALSE   FALSE FALSE  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [3,]     FALSE    FALSE   FALSE   FALSE FALSE  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [4,]     FALSE    FALSE   FALSE   FALSE FALSE  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [5,]     FALSE    FALSE   FALSE   FALSE FALSE  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [6,]     FALSE    FALSE   FALSE   FALSE FALSE  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [7,]     FALSE    FALSE   FALSE   FALSE FALSE  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [8,]     FALSE    FALSE   FALSE   FALSE FALSE  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [9,]     FALSE    FALSE   FALSE   FALSE FALSE  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[10,]     FALSE    FALSE   FALSE   FALSE FALSE  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

3.2.4 Subsetting data frames

With the *subset* function, rows and columns that meet a certain criterion can be selected.

```
# Rows where city is New York and Zip code is 10022 are selected
> subset(customer, City == "New York" & ZIP == "10022")

    FirstName LastName       Company      Address     City   County   State
97      Mitzi  Ihenyen Helm, Norman O   979 3rd Ave New York New York     NY
392    Elaine    Renzi Delvel Chem Co 221 E 59th St New York New York     NY

402    Doreen  Sakurai      Airlifter 211 E 50th St New York New York     NY

    ZIP        Phone          Fax           Email       Web
97  10022 212-838-8303 212-838-3221  mitzi@ihenyen.com  http://www.mitziihenyen.com
392 10022 212-826-7966 212-826-2043  elaine@renzi.com   http://www.elainerenzi.com
40  10022 212-759-4757 212-759-7548 doreen@sakurai.com http://www.doreensakurai.com
```

3.2.5 Trained and Test Data

Data in data frames can be split into trained and test data so that they can be used to develop models for prediction. Assuming we want to split 70% of data selected randomly into a training set and the remaining is assigned to the test set. This can be done as follows:

```
# Extract 70% of the row numbers
> dt <-sort(sample(nrow(customers),nrow(customers)*.7))

# dt contains 350 row numbers (70% of 500 rows number found in customers)
> str(dt)
 int [1:350] 2 3 7 9 11 12 13 14 16 17 ...
```

```
# Assign 70% of the data in customers to train data frame
> train <-customers[dt,]

# Assign 30% of the data in customers to test data frame
> test <-customers[-dt,]
```

Subsetting can also be done if there is no need to split the data frame randomly.

3.3 EXPORTING DATA

The *write.csv* function is used to write a dataframe to a csv file as follows:

```
# Export test dataframe as customerTest.csv
> write.csv(test,"customerTest.csv", row.names=FALSE)
```

The write.table function is used to export to text file as follows:

```
# Export test dataframe as customerTest.txt
> write.table(test,"customerTest.csv", sep=";")
```

3.4 DESCRIPTIVE STATISTIC MEASURES

Descriptive statistics is used to summarize variables of a dataset. The measures that are usually of interest are *mean*, *median*, *variance* and *standard deviation*. We shall have a look at these descriptive measures and how they are represented in R.

To proceed, first save table1 as a csv file named *employee.csv* and load in R using the following codes.

```
# Read data from employee.csv
> employee <- read.csv("employee.csv")
```

It is good to check whether there are any missing data before doing any analysis. In this case we see that there are missing values for row 5.

```
> is.na(employee)
     No   Age  Gender  Salary  Monthly.Expenditure  Occupation Healthy.Lifestyle
[1,] FALSE FALSE  FALSE  FALSE     FALSE            FALSE        FALSE
[2,] FALSE FALSE  FALSE  FALSE     FALSE            FALSE        FALSE
[3,] FALSE FALSE  FALSE  FALSE     FALSE            FALSE        FALSE
[4,] FALSE FALSE  FALSE  FALSE     FALSE            FALSE        FALSE
[5,] TRUE  TRUE   TRUE   TRUE      TRUE             FALSE        FALSE
[6,] FALSE FALSE  FALSE  FALSE     FALSE            FALSE        FALSE
```

It is important to note that if there are missing values it will impact on the analysis of the data. For instance, salary is row 5 is missing. The output for the mean salary will be *NA*. Hence missing values need to be filled or removed. This can be done by using the subset function as follows:

```
# Subset of data where Salary is not missing:

employee1 <- subset(employee, !is.na(Salary))

Remove all rows from a dataset where ANY variable
# has a missing value:
employee_nona <- employee[complete.cases(employee),]
```

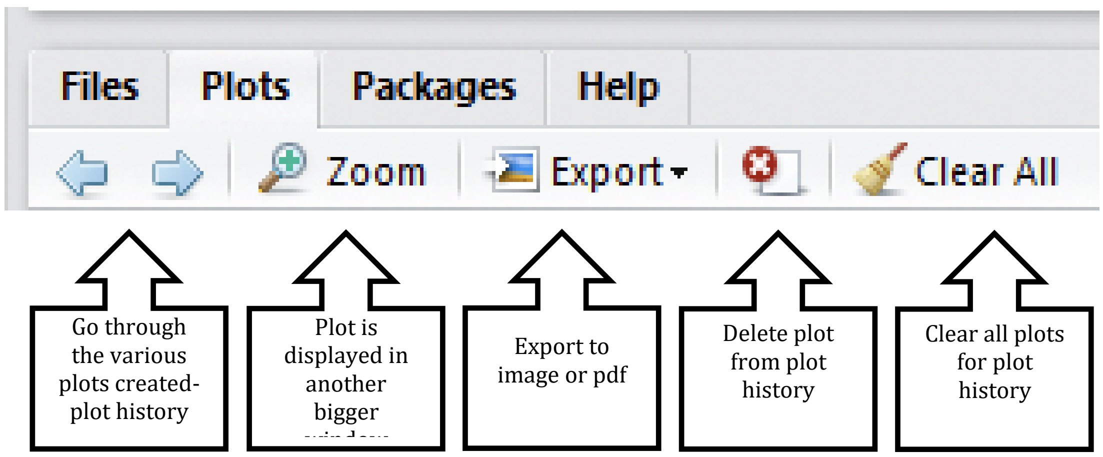
```
# Extract the salary
> salary <- employee1$Salary

# Find the number of observations
> length(salary)
[1] 5
```

Calculate the average (mean) salary, median, variance and standard deviation as follows:

```
# Calculate mean
> mean(salary)
[1] 38498

# Calculate median
> median(salary)
[1] 35760

# Calculate variance
> var(salary)
[1] 348536570

# Calculate standard deviation
> sd(salary)
[1] 18669.13
```

3.5 DATA VISUALIZATION IN R

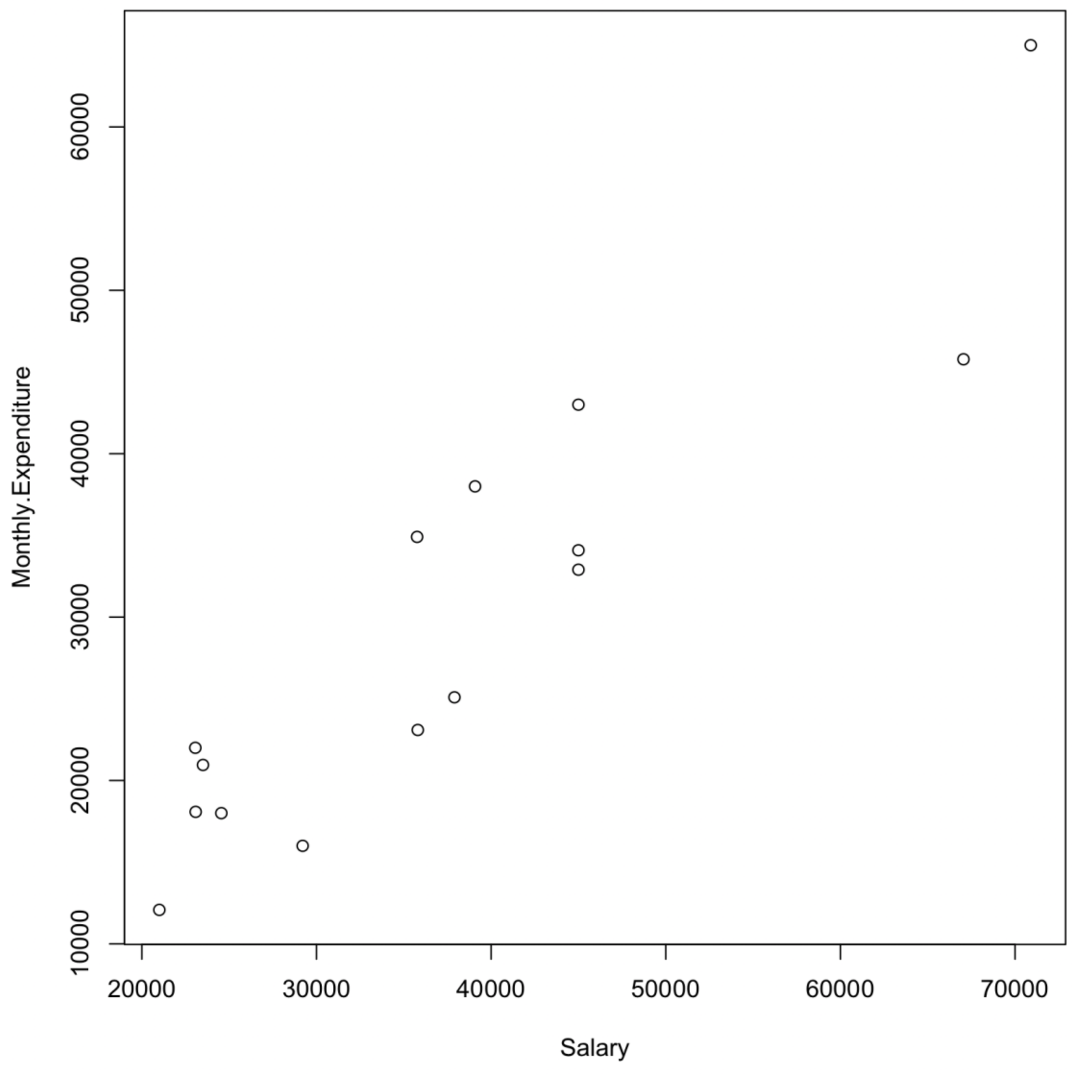
By default, when plots are generated in RStudio, they are displayed in the built-in plotting window (normally, the bottom-right).


Files
Plots
Packages
Help
Zoom
Export
Clear All
Go through the various plots created- plot history
Plot is displayed in another bigger
Export to image or pdf
Delete plot from plot history
Clear all plots for plot history

3.5.1 Scatter Plots

Scatter plots can be drawn as follows:

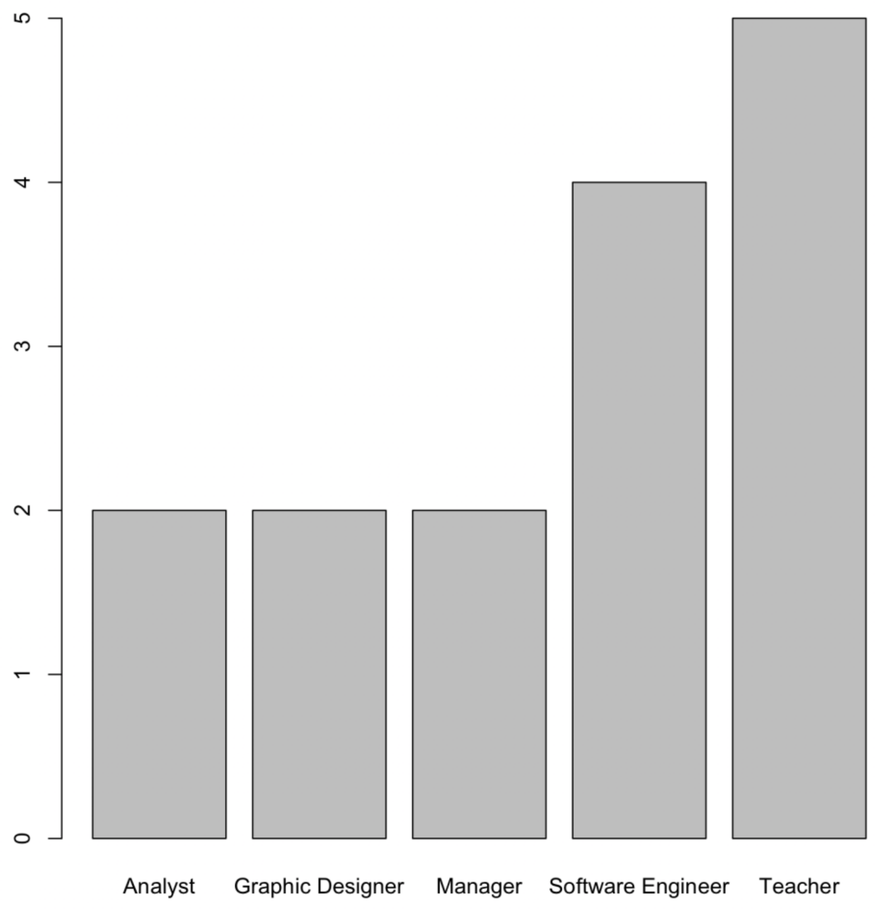
```
# Plot Salary and Monthly.Expenditure
> with(employee, plot(Salary, Monthly.Expenditure)
```



3.5.2 Bar plots

The *barplot* function is used to draw bar plot. The following is an example of bar plot for occupations.

```
# Extract the occupation
> occupation <- employee1$Occupation


# Draw barplot
> barplot(table(occupation))
```

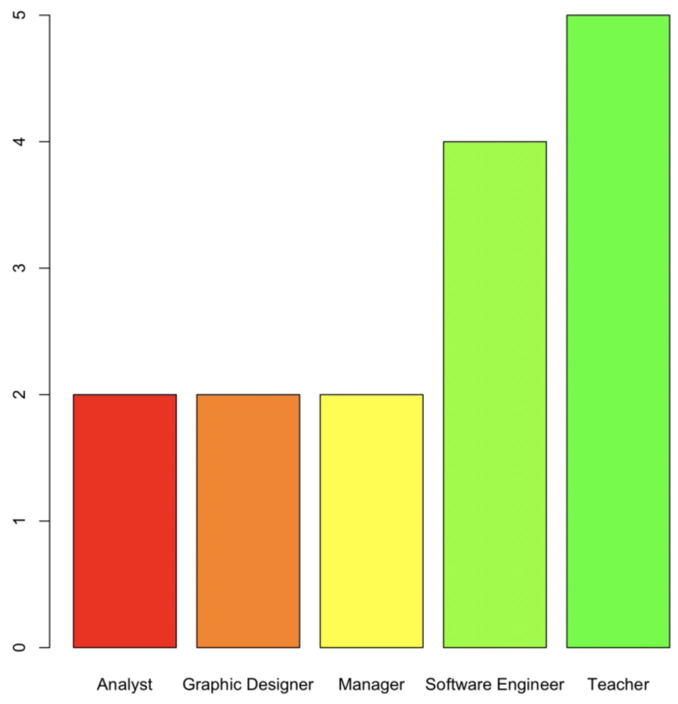


The *table* function calculates the number of individuals for each category

```
# Calculate number of individuals per occupation
> table(occupation)
Analyst   Graphic Designer      Manager  Software Engineer       Teacher
2              2              2         4              5
```

The colors of the barplot can be altered using the *col* parameter

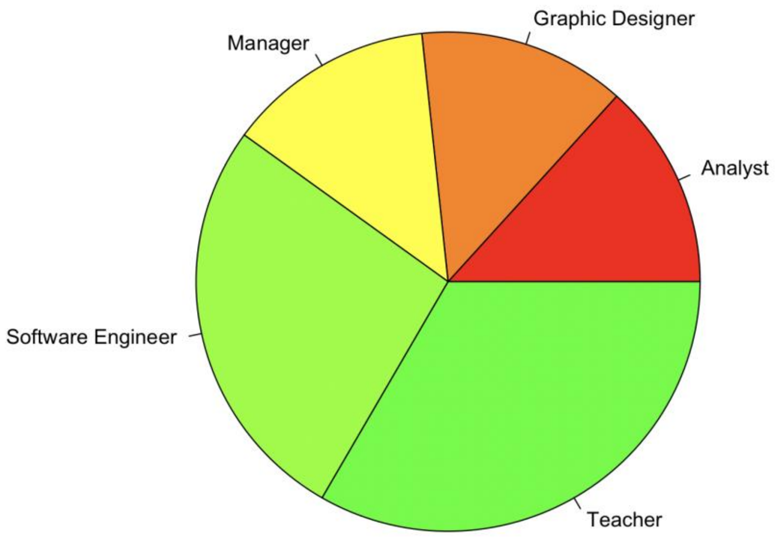
```
# Draw barplot
> barplot(table(occupation), col=rainbow(12))
```



3.5.3 Pie Charts

Pie charts are drawn using the pie function

```
# Draw pie
> pie(table(occupation), col=rainbow(12))
```


Graphic Designer
Manager
Analyst
Software Engineer
Teacher