Question 1: Consider the dataset german credit.xlsx which has 1000 observations on 20 different attributes of credit worthiness for two different types of customers viz. good credit customers and bad credit customers. Consider 800 random observations from the given dataset as training data and keep the remaining as test data. Learn a 2 class classification hypothesis on the training data, which predicts the customer type based on the given 20 attributes of credit worthiness using the following supervised learning techniques:

A. Logistic Regression.
B. Artificial Neural Network.
C. k-Nearest Neighbors

For each of the above techniques perform the following tasks:

(i) Obtain the estimated classes for all the observations in the training data. Construct the observed vs. predicted classification table and calculate the class wise and overall percentage of misclassification, for the training data.

(ii) Obtain the estimated classes for all the observations in the test data. Construct the observed vs. predicted classification table and calculate the class wise and overall percentage of misclassification, for the test data.

Solution:

Logistic Regression

Logistic regression is a method for fitting a regression curve, y = f(x), when y is a categorical variable. The typical use of this model is predicting y given a set of predictors x. The predictors can be continuous, categorical or a mix of both.

The categorical variable y, in general, can assume different values. In the simplest case scenario y is binary meaning that it can assume either the value 1 or 0. A classical example used in machine learning is email classification: given a set of attributes for each email such as number of words, links and pictures, the algorithm should decide whether the email is spam (1) or not (0). In this case we call the model "binomial logistic regression", since the variable to predict is binary, however, logistic regression can also be used to predict a dependent variable which can assume more than 2 values. In this second case we call the model "multinomial logistic regression". A typical example for instance, would be classifying films between "Entertaining", "borderline" or "boring".

Logistic regression implementation in R

R makes it very easy to fit a logistic regression model. The function to be called is **glm()** and the fitting process is not so different from the one used in linear regression. In this case we are going to fit a binary logistic regression model and explain each step.

The dataset

We will be working on "german credit" dataset. It is available to download from:

Stalog (German Credit Data) UCI Machine Learning Repository.

The data cleaning process

When working with a real dataset we need to take into account the fact that some data might be missing or corrupted, therefore we need to prepare the dataset for our analysis. As a first step we load the csv data using the **read.csv()** function.

Then we will check the structure of the dataset by using **str()** function.

Now we need to check for missing values and look how many unique values there are for each variable using the **apply()** function which applies the function passed as argument to each column of the data frame.

```
> german <- read.csv(file.choose())

> str(german)

'data.frame':  1000 obs. of  21 variables:
 $ Creditability                  : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Account.Balance                : int  1 1 2 1 1 1 1 1 4 2 ...
 $ Duration.of.Credit..month.     : int  18 9 12 12 12 10 8 6 18 24 ...
 $ Payment.Status.of.Previous.Credit: int  4 4 2 4 4 4 4 4 4 2 ...
 $ Purpose                        : int  2 0 9 0 0 0 0 0 3 3 ...
 $ Credit.Amount                  : int  1049 2799 841 2122 2171 2241 3398...
 $ Value.Savings.Stocks           : int  1 1 2 1 1 1 1 1 1 3 ...
 $ Length.of.current.employment   : int  2 3 4 3 3 2 4 2 1 1 ...
 $ Instalment.per.cent            : int  4 2 2 3 4 1 1 2 4 1 ...
 $ Sex...Marital.Status           : int  2 3 2 3 3 3 3 3 2 2 ...
 $ Guarantors                     : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Duration.in.Current.address    : int  4 2 4 2 4 3 4 4 4 4 ...
 $ Most.valuable.available.asset  : int  2 1 1 1 2 1 1 1 3 4 ...
 $ Age..years.                    : int  21 36 23 39 38 48 39 40 65 23 ...
 $ Concurrent.Credits             : int  3 3 3 3 1 3 3 3 3 3 ...
 $ Type.of.apartment              : int  1 1 1 1 2 1 2 2 2 1 ...
 $ No.of.Credits.at.this.Bank     : int  1 2 1 2 2 2 2 1 2 1 ...
 $ Occupation                     : int  3 3 2 2 2 2 2 2 1 1 ...
 $ No.of.dependents               : int  1 2 1 2 1 2 1 2 1 1 ...
 $ Telephone                      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Foreign.Worker                 : int  1 1 1 2 2 2 2 2 1 1 ...


> apply(german,2,function(x) sum(is.na(x)))

                    Creditability                   Account.Balance
                                0                                 0
      Duration.of.Credit..month. Payment.Status.of.Previous.Credit
                                0                                 0
                          Purpose                     Credit.Amount
                                0                                 0
            Value.Savings.Stocks      Length.of.current.employment
                                0                                 0
             Instalment.per.cent              Sex...Marital.Status
                                0                                 0
                      Guarantors       Duration.in.Current.address
                                0                                 0
   Most.valuable.available.asset                       Age..years.
                                0                                 0
              Concurrent.Credits                 Type.of.apartment
                                0                                 0
      No.of.Credits.at.this.Bank                        Occupation
                                0                                 0
                No.of.dependents                         Telephone
                                0                                 0
                  Foreign.Worker
                                0
```

Conclusion: Now we have seen the structure of our dataset and there are no missing values, hence we will now proceed to modelling part.

Model fitting

In order to produce reproducible research we will set seed value. We will divide the dataset into 2 parts i.e. training and testing, containing 800 and 200 observations respectively. We are going to build a predictive model based on training data and will test its predictive power based on test data.

Now, let's fit the model. We will specify the parameter **family**=binomial in the **glm()** function.

By using function **summary()** we obtain the results of our model:

```
> set.seed(123)
> n <- dim(german)[1]
> samp <- sample(1:n, 0.8*n)
> train <- german[samp, ]
> test <- german[-samp, ]
> model <- glm(Creditability ~., family = binomial, data=train)
> summary(model)

Call:
glm(formula = Creditability ~ ., family = binomial, data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.6619  -0.7252   0.4214   0.7230   1.8173

Coefficients:
                                 Estimate Std. Error z value Pr(>|z|)
(Intercept)                    -4.368e+00  1.127e+00  -3.876 0.000106 ***
Account.Balance                 6.093e-01  8.025e-02   7.592 3.14e-14 ***
Duration.of.Credit..month.     -1.744e-02  1.062e-02  -1.641 0.100698
Payment.Status.of.Previous.Credit  4.023e-01  9.727e-02   4.136 3.54e-05 ***
Purpose                         4.343e-02  3.440e-02   1.263 0.206691
Credit.Amount                  -1.229e-04  4.821e-05  -2.550 0.010781 *
Value.Savings.Stocks            2.794e-01  6.841e-02   4.084 4.42e-05 ***
Length.of.current.employment    8.799e-02  8.059e-02   1.092 0.274858
Instalment.per.cent            -2.765e-01  9.387e-02  -2.946 0.003222 **
Sex...Marital.Status            2.906e-01  1.321e-01   2.200 0.027805 *
Guarantors                      3.115e-01  2.065e-01   1.509 0.131395
Duration.in.Current.address    -1.344e-02  8.800e-02  -0.153 0.878620
Most.valuable.available.asset  -2.151e-01  1.040e-01  -2.068 0.038615 *
Age..years.                     1.478e-02  9.376e-03   1.576 0.114946
Concurrent.Credits              2.661e-01  1.243e-01   2.141 0.032247 *
Type.of.apartment               3.752e-01  1.923e-01   1.951 0.051008 .
No.of.Credits.at.this.Bank     -2.156e-01  1.855e-01  -1.162 0.245177
Occupation                     -2.494e-02  1.570e-01  -0.159 0.873752
No.of.dependents               -1.750e-01  2.671e-01  -0.655 0.512266
Telephone                       3.351e-01  2.126e-01   1.576 0.115008
Foreign.Worker                  1.103e+00  6.348e-01   1.738 0.082194 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 963.44  on 799  degrees of freedom
Residual deviance: 741.40  on 779  degrees of freedom
AIC: 783.4

Number of Fisher Scoring iterations: 5
```

Interpreting the results of our logistic regression model

```
> sig.var<- summary(model)$coeff[-1,4] < 0.05
> names(sig.var)[sig.var == T]

[1] "Account.Balance"                    "Payment.Status.of.Previous.Credit"
[3] "Credit.Amount"                      "Value.Savings.Stocks"
[5] "Instalment.per.cent"                "Sex...Marital.Status"
[7] "Most.valuable.available.asset"      "Concurrent.Credits"
```

Conclusion: Now we can analyze the fitting and interpret what the model is telling us. We can see that only 8 variable shown above in the output are coming out to be significant. General procedure is to keep the significant variables and drop rest of the variable but our aim here is to build a predictive model and not to build a parsimony model hence we will make the prediction based on all the variables.

Assessing the predictive ability of the model

In the steps above, we briefly evaluated the fitting of the model, now we would like to see how the model is doing when predicting y on a new set of data. By setting the parameter type='response', R will output probabilities in the form of P(y=1|X). Our decision boundary will be 0.5. If P(y=1|X) > 0.5 then y = 1 otherwise y=0. Note that for some applications different thresholds could be a better option.

```
> # Training Data:
> pred1<- predict.glm(model, newdata = train, type = "response")
> result1<- table(train[, 1], floor(pred1+0.5))
> result1

     0   1
  0 111 121
  1  59 509

> error1<- sum(result1[1,2], result1[2,1])/sum(result1)
> error1
[1] 0.225
```

Conclusion: We have applied our model on the training data and we have obtain the classification table shown above. We see 121 customers are misclassified as good credit customers out of 232 and 59 customers are misclassified as bad credit customers out of 568. Over all we have **22.5%** misclassification.

```
> # Test Data:
> pred2<- predict.glm(model, newdata = test, type = "response")
> result2<- table(test[, 1], floor(pred2+0.5))
> result2

     0   1
  0  31  37
  1  15 117

> error2<- sum(result2[1,2], result2[2,1])/sum(result2)
> error2
[1] 0.26
```

Conclusion: We have applied our model on the test data and we have obtain the classification table shown above. We see 37 customers are misclassified as good credit customers out of 68 and 15 customers are misclassified as bad credit customers out of 132. Over all we have **26%** misclassification.

Artificial Neural Network

Credit scoring is the practice of analysing a person background and credit application in order to assess the creditworthiness of the person. One can take numerous approaches on analysing this creditworthiness. In the end it basically comes down to first selecting the correct independent variables (e.g. income, age, gender) that lead to a given level of creditworthiness. In other words: creditworthiness = f(income, age, gender, …). A credit scoring system can be represented by linear regression, logistic regression, machine learning or a combination of these. Neural networks are situated in the domain of machine learning. The actual procedure of building a credit scoring system is much more complex and the resulting model will most likely not consist of solely or even a neural network.

Artificial Neural Network implementation in R

R makes it very easy to make an artificial neural network. The function to be called is **neuralnet()** and this function is available in "**neuralnet**" package which is available on CRAN website.

The dataset

We will be working on "german credit" dataset. It is available to download from:

Stalog (German Credit Data) UCI Machine Learning Repository.

The data cleaning process is already done in the previous part and structure of the data is now known.

Model fitting

In order to produce reproducible research we will set seed value. We will divide the dataset into 2 parts i.e. training and testing, containing 800 and 200 observations respectively. We are going to build a predictive model based on training data and will test its predictive power based on test data.

Before fitting a neural network, some preparation need to be done. Neural networks are not that easy to train and tune. We are going to address data pre-processing. It is good practice to normalize your data before training a neural network. It cannot be emphasized enough how important this step is, depending on your dataset, avoiding normalization may lead to useless results or to a very difficult training process (most of the times the algorithm will not converge before the number of maximum iterations allowed). We choose to use the min-max method and scale the data in the interval [0, 1].

```
> set.seed(123)
> maxs <- apply(german, 2, max)
> mins <- apply(german, 2, min)
> scaled <- as.data.frame(scale(german, center = mins, scale = maxs - mins))
> n <- dim(german)[1]
> samp <- sample(1:n, 0.8*n)
> train <- scaled[samp, ]
> test <- scaled[-samp, ]
```

Now we will fit the model:

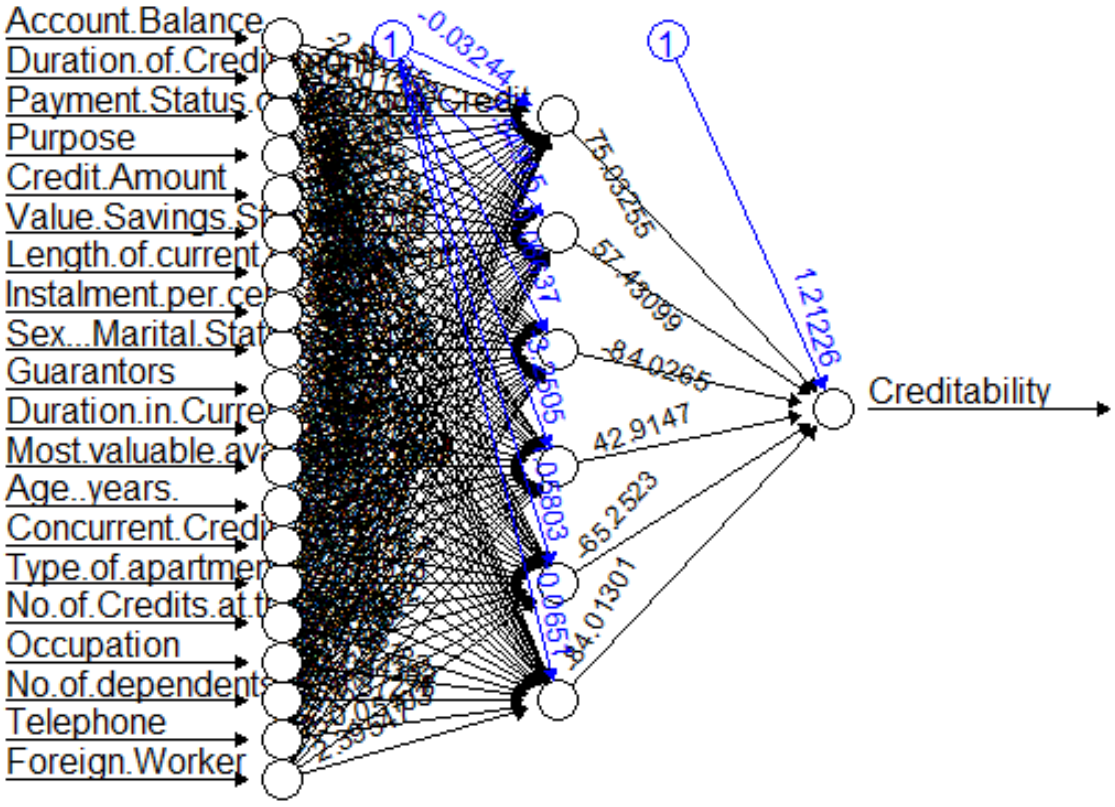
There is no fixed rule as to how many layers and neurons to use although there are several more or less accepted rules of thumb. Usually, if at all necessary, one hidden layer is enough for a vast numbers of applications. As far as the number of neurons is concerned, it should be between the input layer size and the output layer size, usually 2/3 of the input size. At least in our brief experience testing again and again is the best solution since there is no guarantee that any of these rules will fit our model best.

For some reason the formula y ~. is not accepted in the **neuralnet()** function. We need to first write the formula and then pass it as an argument in the fitting function.

The hidden argument accepts a vector with the number of neurons for each hidden layer, while the argument linear.output is used to specify whether we want to do regression linear.output=TRUE or classification linear.output = FALSE.

```
> library(neuralnet)
> name <- names(train)
> f <- as.formula(paste("Creditability ~", paste(name[-1], collapse = " + ")))
> creditnet <- neuralnet(f , data=train, hidden = 6, linear.output = FALSE)
> plot(creditnet)
```

Graphical representation of our neural network is:



The black lines show the connections between each layer and the weights on each connection while the blue lines show the bias term added in each step. The bias can be thought as the intercept of a linear model. The net is essentially a black box so we cannot say that much about the fitting, the weights and the model. Suffice to say that the training algorithm has converged and therefore the model is ready to be used.

Assessing the predictive ability of the model

In the steps above, we briefly evaluated the fitting of the model, now we would like to see how the model is doing when predicting y on a new set of data.

```
> # Training Data:
> res1 <- compute(creditnet, train[, -1])
> result1 <- data.frame(actual=train[, 1], prediction=round(res1$net.result))
> x <- table(result1)
> x
      prediction
actual   0   1
     0 200  32
     1  13 555

> error1<- sum(x[1,2], x[2,1])/sum(x)
> error1
[1] 0.05625
```

Conclusion: We have applied our model on the training data and we have obtain the classification table shown above. We see 32 customers are misclassified as good credit customers out of 232 and 13 customers are misclassified as bad credit customers out of 568. Over all we have **5.625%** misclassification.

```
# Test Data:
> res2 <- compute(creditnet, test[, -1])
> result2 <- data.frame(actual=test[, 1], prediction=round(res2$net.result))
> y <- table(result2)
> y
      prediction
actual   0   1
     0  33  35
     1  28 104

> error2<- sum(y[1,2], y[2,1])/sum(y)
> error2
[1] 0.315
```

Conclusion: We have applied our model on the test data and we have obtain the classification table shown above. We see 35 customers are misclassified as good credit customers out of 68 and 28 customers are misclassified as bad credit customers out of 132. Over all we have **31.5%** misclassification.

k-Nearest Neighbors

The kNN algorithm is a non-parametric algorithm that can be used for either classification or regression. Non-parametric means that it makes no assumption about the underlying data or its distribution. It is one of the simplest Machine Learning algorithms, and has applications in a variety of fields, ranging from the healthcare industry, to the finance industry.

For each data point, the algorithm finds the k closest observations, and then classifies the data point to the majority. Usually, the k closest observations are defined as the ones with the smallest Euclidean distance to the data point under consideration. For example, if k = 3, and the three nearest observations to a specific data point belong to the classes A, B, and A respectively, the algorithm will classify the data point into class A. If k is even, there might be ties. To avoid this, usually weights are given to the observations, so that nearer observations are more influential in determining which class the data point belongs to. An example of this system is giving a weight of 1/d to each of the observations, where d is distance to the data point. If there is still a tie, then the class is chosen randomly.

K-Nearest Neighbors implementation in R

R makes it very easy to implement K-Nearest Neighbors. The function to be called is **knn()** and this function is available in "class" package available on CRAN website.

The dataset

We will be working on "german credit" dataset. It is available to download from:

Stalog (German Credit Data) UCI Machine Learning Repository.

The data cleaning process is already done in the previous part and structure of the data is now known.

Model fitting

In order to produce reproducible research we will set seed value. We will divide the dataset into 2 parts i.e. training and testing, containing 800 and 200 observations respectively. We are going to build a predictive model based on training data and will test its predictive power based on test data.

```
> german <- read.csv(file.choose())
> set.seed(123)
> n <- dim(german)[1]
> samp <- sample(1:n, 0.8*n)
> train <- german[samp, ]
> test <- german[-samp, ]
```

If we look at the help file for knn using ?knn, we will see that we have to provide the testing set, training set, and the classification vector all at the same time. For most other prediction algorithms, we build the prediction model on the training set in the first step, and then use the model to test our predictions on the test set in the second step. However, the kNN function does both in a single step.

Assessing the predictive ability of the model

We will fit the model on training dataset and test its prediction power on the training dataset at the same time:

```
> library(class)
> prediction1 <- knn(train, train, train[, 1], k=12)
> result1 <- table(train[, 1], prediction1)
> result1
   prediction1
      0   1
  0  49 183
  1  30 538

> error1<- sum(result1[1,2], result1[2,1])/sum(result1)
> error1
[1] 0.26625
```

Conclusion: We have applied our model on the training data and we have obtain the classification table shown above. We see 183 customers are misclassified as good credit customers out of 232 and 30 customers are misclassified as bad credit customers out of 568. Over all we have **26.65%** misclassification.

Now we will fit the model on training dataset and test its prediction power on the test dataset:

```
> prediction2 <- knn(train, test, train[, 1], k=12)
> result2 <- table(test[, 1], prediction2)
> result2
   prediction2
      0   1
  0   8  60
  1  11 121

> error2<- sum(result2[1,2], result2[2,1])/sum(result2)
> error2
[1] 0.355
```

Conclusion: We have applied our model on the test data and we have obtain the classification table shown above. We see 60 customers are misclassified as good credit customers out of 68 and 11 customers are misclassified as bad credit customers out of 132. Over all we have **35.5%** misclassification.

Comparing 3 methods of supervised learning

Now that we have predicted the customer type based on the given 20 attributes of credit worthiness using the three supervised learning techniques, now we will compare each of them on the basis of misclassification they have made:

		Logistic Regression	Neural Network	K-Nearest Neighbors
	False Negative	52.10%	13.79%	78.87%
	False Positive	10.38%	2.28%	5.28%
Training	**Overall**	22.50%	5.60%	26.60%
	False Negative	54.41%	51.47%	88.23%
	False Positive	11.36%	21.21%	8.33%
Test	**Overall**	26%	31.50%	35.50%

Conclusion:

False Negative or Type 2 error is when a bad credit customer is considered to be a good credit customer while False Positive or Type 1 error is when a good credit customer is considered to be a bad credit customer.

We see K-Nearest Neighbors approach is producing large number of false negative, clearly it's not a good choice when it comes to predicting credit worthiness of a customer.

Now comparing neural network and logistic regression approach we see former is working well on the training dataset but not good on the test dataset, while later is producing less false positive and performing equally well on training and test dataset.

Now taking into account for computational aspect of the model, interpretability and complexity we may go for logistic regression for predicting the credit worthiness of a customer in our "german credit" dataset.

Question 2: A manufacturer wants to manufacture t-shirts for males and females in 4 different sizes respectively. Consider the dataset heights and weights.xlsx which has data on the average height and weight for healthy males and females. Use this dataset to suggest the measurements (in height and weight) to the manufacturer for both males and females for all the different size types. Explain the possibility of refinement in no. of available t-shirt sizes for females.

Solution:

K-Means Cluster

K Means Clustering is an unsupervised learning algorithm that tries to cluster data based on their similarity. Unsupervised learning means that there is no outcome to be predicted, and the algorithm just tries to find patterns in the data. In k means clustering, we have the specific the number of clusters we want the data to be grouped into. The algorithm randomly assigns each observation to a cluster, and finds the centroid of each cluster. Then, the algorithm iterates through two steps:

1. Reassign data points to the cluster whose centroid is closest.
2. Calculate new centroid of each cluster.

These two steps are repeated till the within cluster variation cannot be reduced any further. The within cluster variation is calculated as the sum of the Euclidean distance between the data points and their respective cluster centroids.

K-Mean Cluster implementation in R

R makes it very easy to make K-Mean Cluster. The function to be called is **kmeans()**.

The dataset

We will be working on “Height and weight” dataset. It has information on the average height and weight for healthy males and females.

The data cleaning process

When working with a real dataset we need to take into account the fact that some data might be missing or corrupted, therefore we need to prepare the dataset for our analysis. As a first step we load the csv data using the **read.csv()** function.

Then we will check the structure of the dataset by using **str()** function.

Now we need to check for missing values and look how many unique values there are for each variable using the **apply()** function which applies the function passed as argument to each column of the data frame.

```
> htwt <- read.csv(file.choose())
> apply(htwt,2,function(x) sum(is.na(x)))
  Male_height   Male_weight Female_height Female_weight
            0             0             0             0

> str(htwt)
'data.frame':  45 obs. of  4 variables:
 $ Male_height  : int  158 158 158 160 160 160 163 163 163 165 ...
 $ Male_weight  : int  58 59 63 59 60 64 60 61 64 61 ...
 $ Female_height: int  147 147 147 150 150 150 153 153 153 155 ...
 $ Female_weight: int  46 49 54 47 50 55 47 51 56 48 ...
```

Conclusion: We observe we don't have the problem of missing data.

Model fitting

We now use k-means cluster approach to find out the ideal size of clothes for male and female separately.

For Males:

```
> set.seed(123)
> library(ggplot2)
> model1 <- kmeans(htwt[,1:2], 4)
> model1
K-means clustering with 4 clusters of sizes 12, 11, 13, 9

Cluster means:
  Male_height Male_weight
1    161.5000    61.33333
2    182.2727    70.18182
3    172.0000    65.46154
4    190.4444    76.55556

Clustering vector:
 [1] 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 4 2 4 4
[40] 4 4 4 4 4 4

Within cluster sum of squares by cluster:
[1] 143.6667 175.8182 199.2308 138.4444
 (between_SS / total_SS =  90.6 %)

Available components:
[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
> size <- as.factor(unlist(model1[1]))
> size <- factor(size, levels=c(4,3,2,1), labels=c("XL","Medium","Large","Small"))
> p <- ggplot(htwt, aes(Male_height, Male_weight, color = size))
> p <- p +  geom_point(alpha = 0.8, size = 1.5) + ggtitle("Male Size")
> p
```

Conclusion: Based on the clusters formed, now we know what size would fit an individual with an average height and weight given below:

Male_height	Male_weight	Size
161.5	61.33333	Small
172	65.46154	Medium
182.2727	70.18182	Large
190.4444	76.55556	XL

Graphical representation of the clusters is shown towards the end.

For Females:

```
> set.seed(123)
> library(ggplot2)
> model2 <- kmeans(htwt[, 3:4], 4)
> model2
K-means clustering with 4 clusters of sizes 13, 11, 12, 9

Cluster means:
  Female_height Female_weight
1      151.7692      50.84615
2      172.0909      58.63636
3      162.3333      55.16667
4      180.0000      64.44444

Clustering vector:
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 4 2 4 4
[40] 4 4 4 4 4 4

Within cluster sum of squares by cluster:
[1] 314.0000 209.4545 254.3333 190.2222
 (between_SS / total_SS =  86.1 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
> size <- as.factor(unlist(model2[1]))
> size <- factor(size, levels=c(4,3,2,1), labels=c("XL","Medium","Large","Small"))
> q <- ggplot(htwt, aes(Female_height, Female_weight, color = size))
> q <- q +  geom_point(alpha = 0.8, size = 1.5) + ggtitle("Female Size")
> q
```

Conclusion: Based on the clusters formed, now we know what size would fit an individual with an average height and weight given below:

Female_height	Female_weight	Size
151.7692	50.84615	Small
162.3333	55.16667	Medium
172.0909	58.63636	Large
180	64.44444	XL

Graphical representation of the clusters is shown towards the end.

One possibility of refinement in no. of available t-shirt sizes for females can be that instead of having 4 sizes, we can have 5 sizes, which would make the clusters more homogenous.

Hence making 5 sizes in female clothing we get

For Females with 5 sizes:

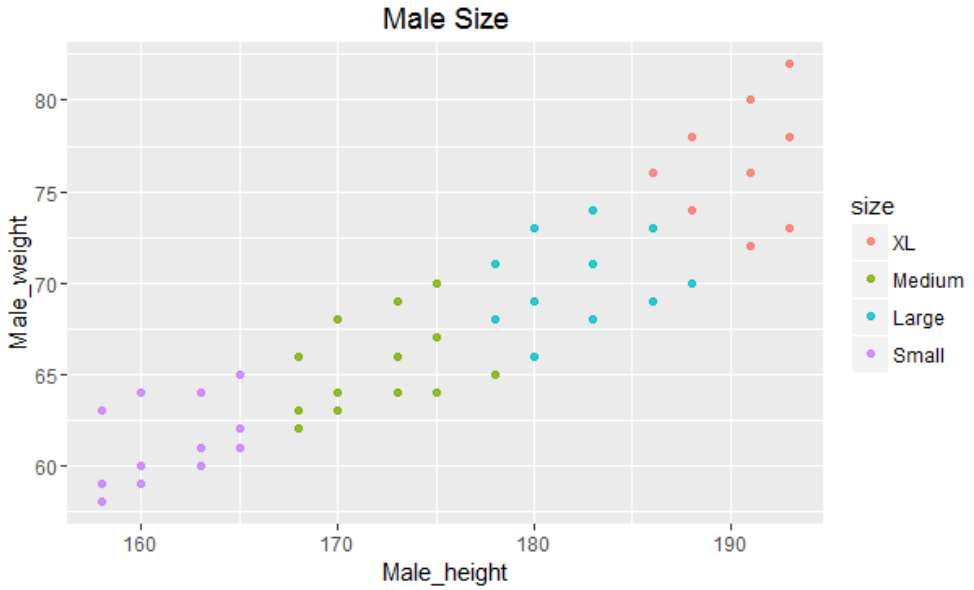
```
> set.seed(123)
> library(ggplot2)
> model3 <- kmeans(htwt[, 3:4], 5)
> model3
K-means clustering with 5 clusters of sizes 11, 9, 10, 8, 7

Cluster means:
  Female_height Female_weight
1      150.9091      50.45455
2      167.4444      58.22222
3      160.0000      53.80000
4      175.6250      58.75000
5      180.2857      65.85714

Clustering vector:
 [1] 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 2 3 2 2 2 2 2 2 2 2 4 4 4 4 4 5 4 4 5
[40] 4 5 5 5 5 5

Within cluster sum of squares by cluster:
[1] 213.6364 151.7778 185.6000 111.3750 118.2857
 (between_SS / total_SS =  88.8 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
> size <- as.factor(unlist(model3[1]))
> size <- factor(size, levels = c(5,2,4,3,1), labels=c("XL","Medium","Large","Small"
,"XS"))
> r <- ggplot(htwt, aes(Female_height, Female_weight, color = size))
> r <- r +  geom_point(alpha = 0.8, size = 1.5) + ggtitle("Female Size")
> r
```

Conclusion: Based on the clusters formed, now we know what size would fit an individual with an average height and weight given below:

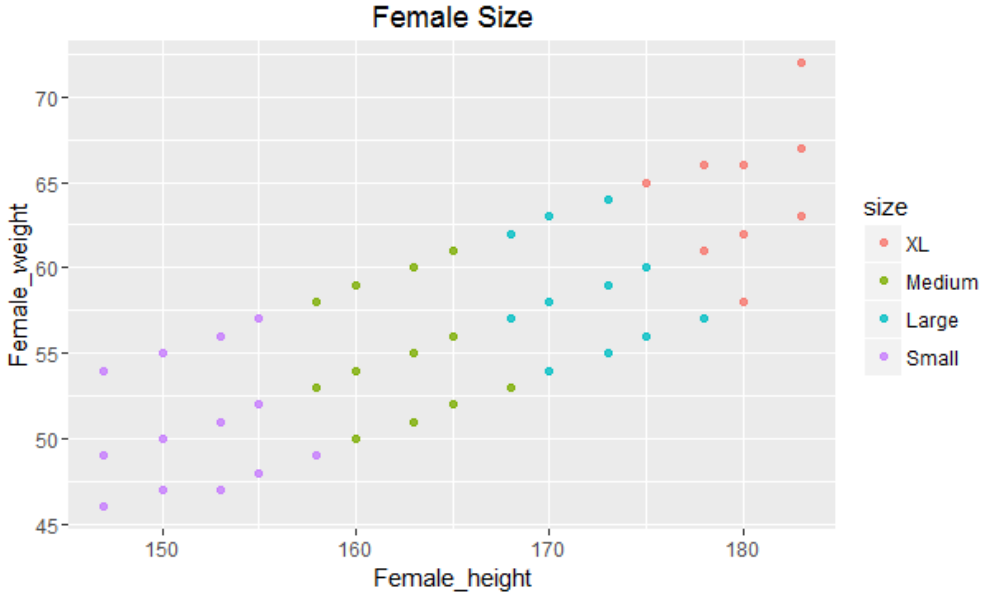
Female_height	Female_weight	Size
150.9091	50.45455	XS
160	53.8	Small
167.4444	58.22222	Medium
175.625	58.75	Large
180.2857	65.85714	XL

Graphical representation of the clusters is shown towards the end.

Plot showing 4 sizes available for males:



Plot showing 4 sizes available for females:



Plot showing 5 sizes available for females, as a refinement over 4 sizes: