IDS Assignment 2

Manan Bhatia

2022-09-29

DECLARATION:

The following declaration must be included in a clearly visible and readable place on the first page of the report. ———————————————————————————— By including this statement, we the authors of this work, verify that: • I hereby certify that no part of this assignment/product has been copied from any other student’s work or from any other source except where due acknowledgement is made in the assignment. • No part of this assignment/product has been written/produced for us by another person except where such collaboration has been authorised by the subject lecturer/tutor concerned. • I am aware that this work may be reproduced and submitted to plagiarism detection software programs for the purpose of detecting possible plagiarism (which may retain a copy on its database for future plagiarism checking). • I hereby certify that we have read and understand what the School of Computer, Data and Mathematical Sciences defines as minor and substantial breaches of misconduct as outlined in the learning guide for this unit. ——————————————————————————————

Question 01 – Data Visualisation:

Use appropriate data visualization techniques and comment on the association of the price of King County houses with other characteristics of the house.

```
house2=read.csv("kc_house2.csv", header = TRUE)
attach(house2)
dim(house2)

## [1] 340  10

names(house2)

##  [1] "id"             "bedrooms"       "bathrooms"      "sqft_living"
##  [5] "sqft_lot"       "floors"         "waterfront"     "sqft_living15"
##  [9] "sqft_lot15"     "price_cat"

View(house2)
head(house2)

##           id bedrooms bathrooms sqft_living sqft_lot floors waterfront
## 1 7922800400        5      3.25        3.25   14.342      2          0
## 2 1516000055        3      2.25        2.15   21.235      1          0
## 3 2123039032        1      0.75        0.76   10.079      1          1
## 4 9297300045        3      2.00        1.97    4.166      2          0
## 5 1860600135        5      2.50        3.65    9.050      2          0
## 6 1560930070        4      3.50        2.84   40.139      1           0
##   sqft_living15 sqft_lot15 price_cat
## 1          2.96     11.044       low
## 2          2.57     18.900       low
## 3          1.23     14.267       low
## 4          2.39      4.166       low
## 5          2.88      5.400      high
## 6          3.18     36.852       low
```

```
str(house2)
```

```
## 'data.frame':    340 obs. of  10 variables:
##  $ id           : num  7.92e+09 1.52e+09 2.12e+09 9.30e+09 1.86e+09 ...
##  $ bedrooms     : int  5 3 1 3 5 4 3 3 3 2 ...
##  $ bathrooms    : num  3.25 2.25 0.75 2 2.5 3.5 3 2.5 3 2.25 ...
##  $ sqft_living  : num  3.25 2.15 0.76 1.97 3.65 ...
##  $ sqft_lot     : num  14.34 21.23 10.08 4.17 9.05 ...
##  $ floors       : num  2 1 1 2 2 1 2 2 2 1.5 ...
##  $ waterfront   : int  0 0 1 0 0 0 1 1 0 0 ...
##  $ sqft_living15: num  2.96 2.57 1.23 2.39 2.88 3.18 2.28 3.98 3.08 2.13
...
##  $ sqft_lot15   : num  11.04 18.9 14.27 4.17 5.4 ...
##  $ price_cat    : chr  "low" "low" "low" "low" ...
```

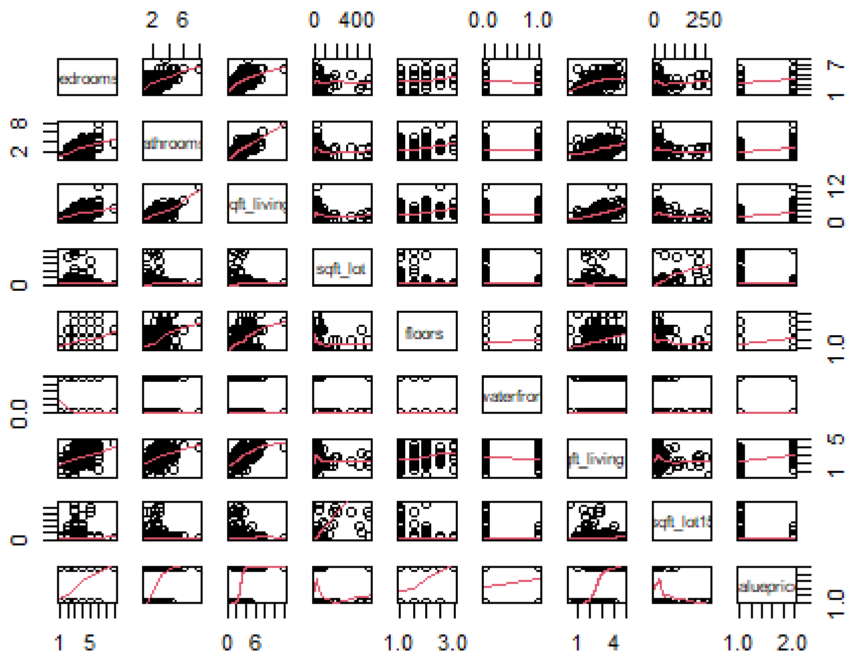
```
valueprice= as.character(house2$price_cat)
valueprice[valueprice == "high"] = 1
valueprice[valueprice == "low"] = 0
valueprice= as.numeric(valueprice)
head(valueprice)
```

```
## [1] 0 0 0 0 1 0
```

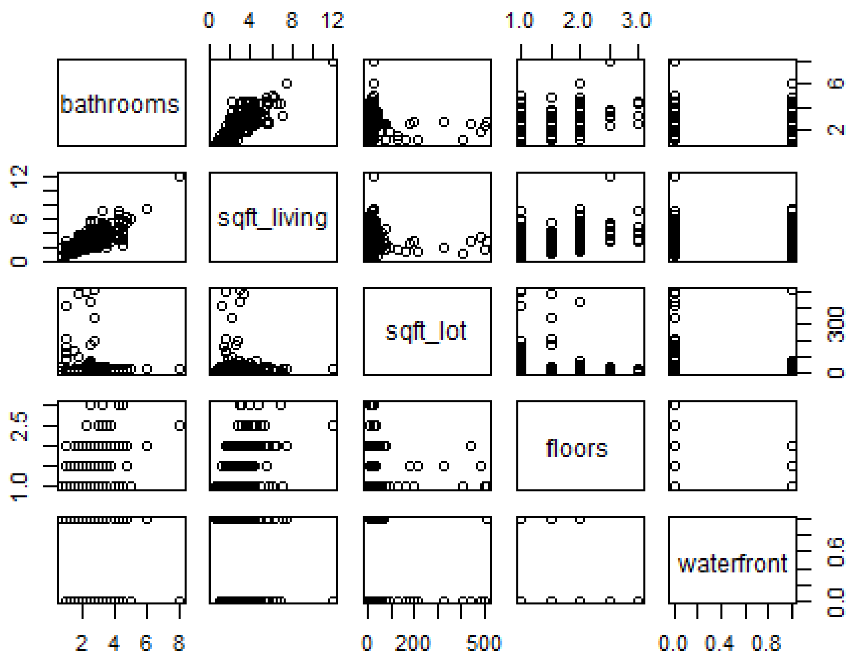
```
newhouse2 = data.frame(house2[,c(-1, -10)], valueprice =
as.factor(valueprice))
head(newhouse2)
```

```
##   bedrooms bathrooms sqft_living sqft_lot floors waterfront sqft_living15
## 1        5      3.25        3.25   14.342      2          0          2.96
## 2        3      2.25        2.15   21.235      1          0          2.57
## 3        1      0.75        0.76   10.079      1          1          1.23
## 4        3      2.00        1.97    4.166      2          0          2.39
## 5        5      2.50        3.65    9.050      2          0          2.88
## 6        4      3.50        2.84   40.139      1          0          3.18
##   sqft_lot15 valueprice
## 1     11.044          0
## 2     18.900          0
## 3     14.267          0
## 4      4.166          0
## 5      5.400          1
## 6     36.852          0
```

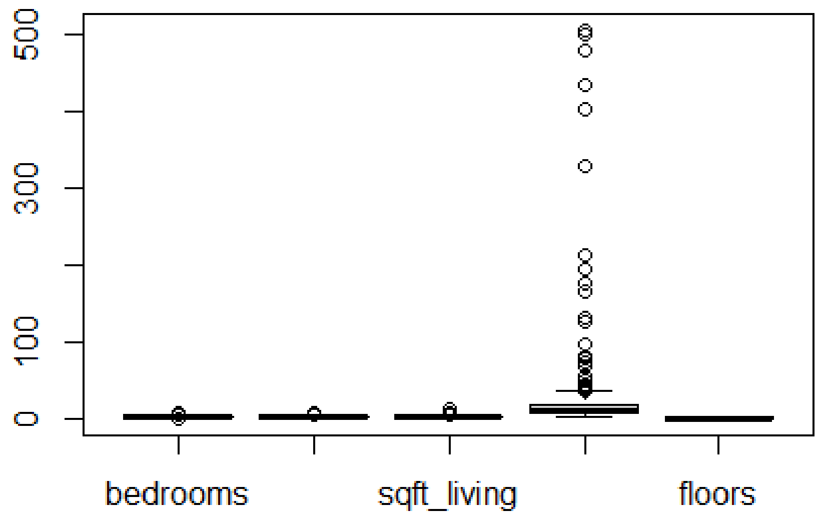
```
pairs(newhouse2, panel=panel.smooth)
```



```
pairs(newhouse2 [,2:6])
```



```
boxplot(house2 [,2:6])
```



For the matrix plot, the price_cat sub data which is part of house2 data, is a categorical variable. The categorical variable is then changed into a numerical variable. So, from changing “low” and “high” to “0” and “1” respectively. A box plot is also plotted for comparison between it and the matrix plot.

Looking at both plots, I believe that the matrix plot is appropriate because the box plot has very low outcomes and the matrix plot is more accurate and more readable.

The price_cat associating with other variables in the house2 dataset shows that the matrix correlation goes up with some association, while the other association of the matrix correlation goes down.

Question 02 – Principal Component Analysis:

a) Calculate the mean and the variance for each appropriate variable and discuss if scaling is necessary and justify your findings.

```
sapply(house2[,2:6], mean)

##    bedrooms   bathrooms sqft_living    sqft_lot      floors
##    3.614706    2.541912    2.976085   24.737594    1.444118
```

It can be seen that the variables have different means. We can see that the bathrooms and sqft_living and bedrooms have quite similar variable, whereas sqft_lot has a big mean and the floors has a small mean variable.

```
sapply(house2[,2:6], var)

##     bedrooms    bathrooms  sqft_living     sqft_lot       floors
##    1.0929984    0.8564609    1.5279319 3779.5763196    0.2638296
```

It can be seen that the variables have different variances. We can see that sqft_lot has a large variance variable whereas bedrooms, bathrooms, sqft_living and floors have small variances.

Between the mean and the variance, there is a significant difference in the ranges, therefore this data requires scaling.

b) Perform a Principal Component Analysis and give the principal component loadings.

```
pr.out = prcomp(house2[,2:6], scale. = TRUE)

pr.out$center

##    bedrooms   bathrooms sqft_living    sqft_lot      floors
##    3.614706    2.541912    2.976085   24.737594    1.444118
```

This gives the means of the original variables

```
pr.out$scale

##    bedrooms   bathrooms sqft_living    sqft_lot      floors
##   1.0454657   0.9254517   1.2360954  61.4782589   0.5136435
```

This gives the standard deviances of the original variables.

```
pr.out$rotation

##                     PC1         PC2         PC3         PC4          PC5
## bedrooms    -0.47121144 -0.13758239  0.45156575 -0.74496598 -0.012038809
## bathrooms   -0.57146349 -0.04476012  0.02963463  0.37593987  0.727473034
## sqft_living -0.56225205 -0.05880938  0.11760910  0.44881200 -0.682019041
## sqft_lot     0.08949624 -0.98269541 -0.15972133  0.02803196  0.001860162
## floors      -0.35672443  0.09959320 -0.86940632 -0.31855440 -0.074058213
```

This gives the principal component loading vectors. Each column contains corresponding principal component loading vector. It can be seen that there are four principal components. There are almost equal contributions between sqft_living and bathrooms to PC1. Major contribution for PC2 given by sqft_lot.

```
head(pr.out$x)

##             PC1         PC2        PC3        PC4         PC5
## [1,] -1.5874066  0.04436819 -0.2668099 -0.9495104  0.30906129
## [2,]  1.1364086  0.10419027  0.4073703  0.2933352  0.29733709
## [3,]  3.5801093  0.68439116 -0.6077871  0.5993580 -0.09214379
## [4,]  0.6533117  0.59157946 -1.2660419 -0.5015452  0.05543562
## [5,] -1.3139323  0.14620142 -0.2390193 -1.1113553 -0.50135500
## [6,] -0.3725166 -0.42286376  0.8958631  0.3476961  0.88827622
```

c) Explain the proportion of variance explained by each principal component using a graph.

PC1 = -0.47121144*bedrooms - 0.57146349*bathrooms - 0.56225205*sqft_living + 0.08949624*sqft_lot - 0.35672443*floors

PC2 = -0.13758239*bedrooms - 0.04476012*bathrooms - 0.05880938*sqft_living - 0.98269541*sqft_lot + 0.09959320*floors

PC3 = 0.45156575*bedrooms + 0.02963463*bathrooms + 0.11760910*sqft_living - 0.15972133*sqft_lot - 0.86940632*floors

PC4 = -0.7449698*bedrooms + 0.37593987*bathrooms + 0.44881200*sqft_living + 0.02803196*sqft_lot - 0.31855440*floors

PC5 = -0.012038809*bedrooms + 0.727473034*bathrooms - 0.682019041*sqft_living + 0.001860162*sqft_lot - 0.074058213*floors

```
summary(pr.out)

## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5
## Standard deviation     1.5814 0.9963 0.9083 0.68136 0.46606
## Proportion of Variance 0.5002 0.1985 0.1650 0.09285 0.04344
## Cumulative Proportion  0.5002 0.6987 0.8637 0.95656 1.00000
```

It can be seen that the first principal component explains about 50% of the variation in the data, the next principal component explains about 20% of the variation in the dataset, the next principal component explains about 17% of the variation in the dataset, the next one explains about 9.3% of the variation in the dataset and the last principal component explains about 4% of the variation in the dataset.

```
pr.out$sdev

## [1] 1.5814229 0.9962994 0.9083072 0.6813603 0.4660636
```

This gives the standard deviation of each principal component.

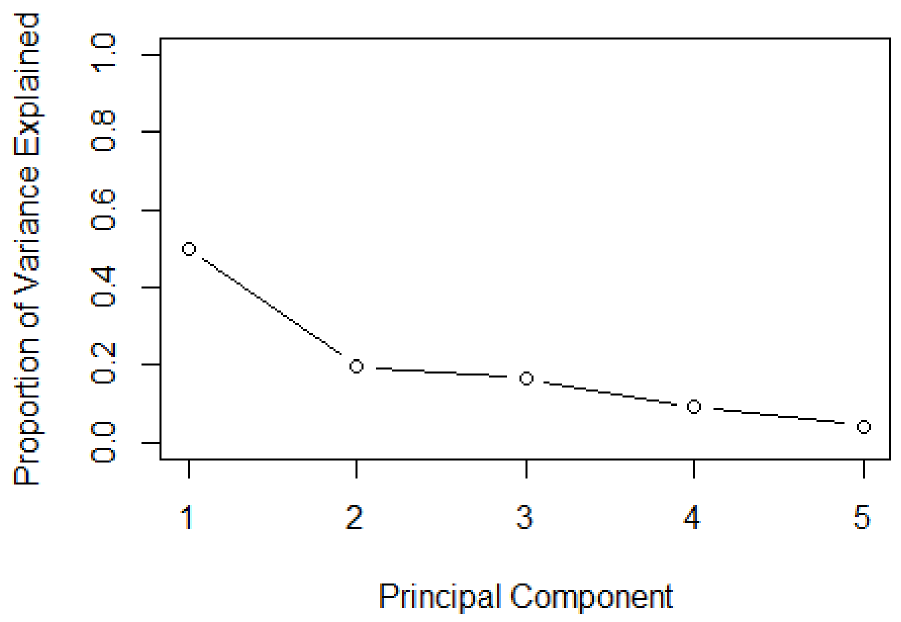
```
pr.var = pr.out$sdev^2
pr.var

## [1] 2.5008983 0.9926125 0.8250220 0.4642519 0.2172153
```

This gives the variance of each principal component.

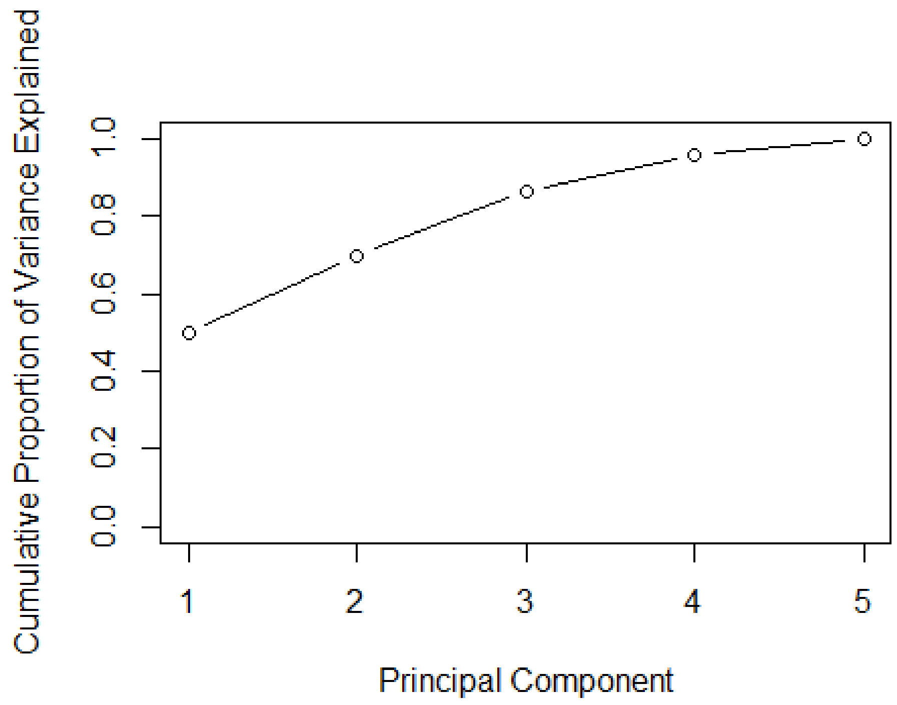
```
pve = pr.var/sum(pr.var)
pve

## [1] 0.50017966 0.19852250 0.16500440 0.09285038 0.04344306
```

This gives the propotion of variance explained by each principal component.

```
plot(pve,xlab="Principal Component",ylab="Proportion of Variance Explained",
ylim = c(0,1), type = 'b')
```



```
plot(cumsum(pve),xlab="Principal Component",ylab="Cumulative Proportion of
Variance Explained",
ylim = c(0,1), type = 'b')
```



d) Write the first two principal components in terms of the original variables in the given dataset.

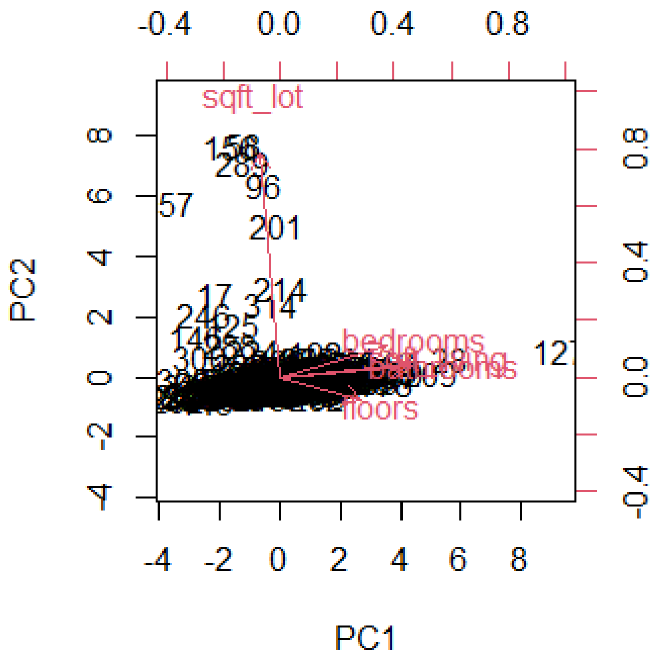
```
pr.out

## Standard deviations (1, .., p=5):
## [1] 1.5814229 0.9962994 0.9083072 0.6813603 0.4660636
##
## Rotation (n x k) = (5 x 5):
##                     PC1         PC2         PC3         PC4          PC5
## bedrooms    -0.47121144 -0.13758239  0.45156575 -0.74496598 -0.012038809
## bathrooms   -0.57146349 -0.04476012  0.02963463  0.37593987  0.727473034
## sqft_living -0.56225205 -0.05880938  0.11760910  0.44881200 -0.682019041
## sqft_lot     0.08949624 -0.98269541 -0.15972133  0.02803196  0.001860162
## floors      -0.35672443  0.09959320 -0.86940632 -0.31855440 -0.074058213

head(pr.out$x)

##             PC1         PC2        PC3        PC4         PC5
## [1,] -1.5874066  0.04436819 -0.2668099 -0.9495104  0.30906129
## [2,]  1.1364086  0.10419027  0.4073703  0.2933352  0.29733709
## [3,]  3.5801093  0.68439116 -0.6077871  0.5993580 -0.09214379
## [4,]  0.6533117  0.59157946 -1.2660419 -0.5015452  0.05543562
## [5,] -1.3139323  0.14620142 -0.2390193 -1.1113553 -0.50135500
## [6,] -0.3725166 -0.42286376  0.8958631  0.3476961  0.88827622
```

PC1 = 50%

PC2 = 19.8%

e) Construct the Biplot and interpret it

```
pr.out$rotation = -pr.out$rotation
pr.out$x = -pr.out$x
biplot(pr.out, scale = 0)
```



Question 03 - Clustering

a) Cluster the 340 houses in King County in the given dataset into 2 groups using k-means clustering

```
km.out = kmeans(house2[,2:6], 2, nstart=20)
km.out$cluster

##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [149] 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

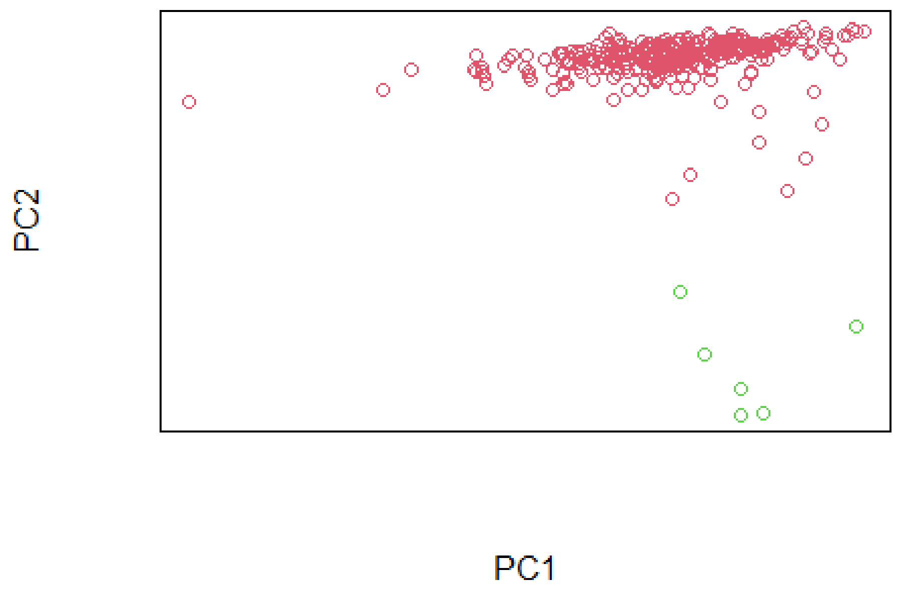
```
1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
1 1 1
## [297] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [334] 1 1 1 1 1 1 1
```

```
km.out
```

```
## K-means clustering with 2 clusters of sizes 334, 6
##
## Cluster means:
##   bedrooms bathrooms sqft_living  sqft_lot   floors
## 1 3.619760  2.548653    2.988380  17.24479 1.446108
## 2 3.333333  2.166667    2.291667 441.83683 1.333333
##
## Clustering vector:
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [149] 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
1 1 1
## [297] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [334] 1 1 1 1 1 1 1
##
## Within cluster sum of squares by cluster:
## [1] 197213.60  22744.66
##  (between_SS / total_SS =  82.8 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
"tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

b) Visually display the clusters using the first two principal components (PCs).

```
pp = prcomp(house2[,2:6], scale =TRUE)
plot(pp$x[,1:2], col=fitted(km.out, "classes")+1, xaxt="n", yaxt="n")
```



c) Cluster the 340 houses in King County in the given dataset into 2 groups using hierarchical clustering. Consider Euclidean distance as the dissimilarity measure and the closest distance between two clusters as the maximum distance between them.

```
hc.complete = hclust(dist(house2[,2:6]), method = "complete")
hc.complete

##
## Call:
## hclust(d = dist(house2[, 2:6]), method = "complete")
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 340
```

d) Cluster the 340 houses in King County in the given dataset into 2 groups using hierarchical clustering. Consider Euclidean distance as the dissimilarity measure and the closest distance between two clusters as the average distance between them.

```
hc.average = hclust(dist(house2[,2:6]), method = "average")
hc.average

##
## Call:
## hclust(d = dist(house2[, 2:6]), method = "average")
##
```

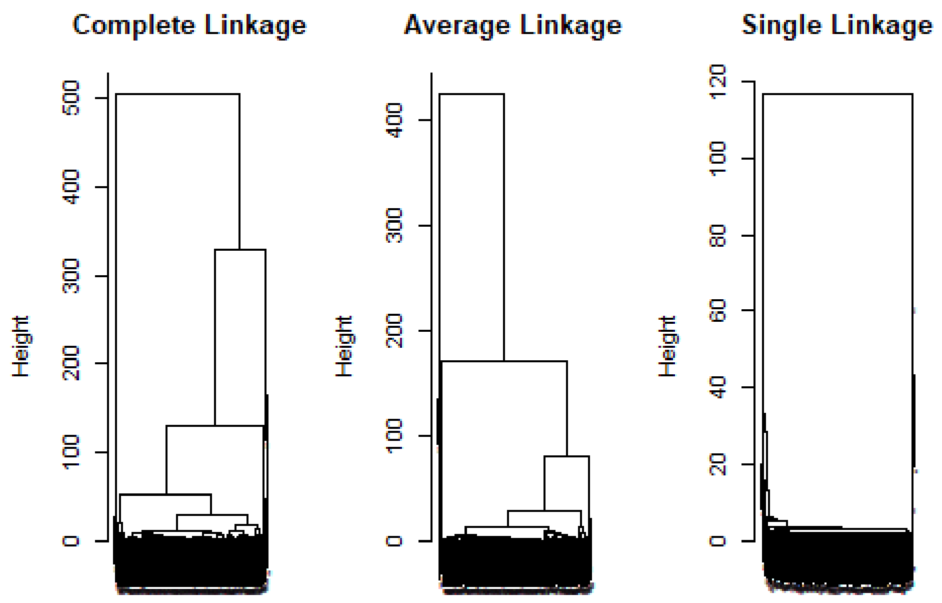
```
## Cluster method   : average
## Distance         : euclidean
## Number of objects: 340
```

e) Cluster the 340 houses in King County in the given dataset into 2 groups using hierarchical clustering. Consider Euclidean distance as the dissimilarity measure and the closest distance between two clusters as the minimum distance between them.

```
hc.single = hclust(dist(house2[,2:6]), method = "single")
hc.single

##
## Call:
## hclust(d = dist(house2[, 2:6]), method = "single")
##
## Cluster method   : single
## Distance         : euclidean
## Number of objects: 340
```

f) Visually display the clusters obtained in part c, d and e using the first two principal components (PCs).

```
par(mfrow = c(1,3))
plot(hc.complete, main = "Complete Linkage", xlab = "", sub = "", cex = 0.05)
plot(hc.average, main = "Average Linkage", xlab = "", sub = "", cex = 0.05)
plot(hc.single, main = "Single Linkage", xlab = "", sub = "", cex = 0.05)
```



```
cutree(hc.complete, 2)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [149] 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
1 1 1
## [297] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [334] 1 1 1 1 1 1 1
```

```
cutree(hc.average, 2)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [149] 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
1 1 1
## [297] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [334] 1 1 1 1 1 1 1
```

```
cutree(hc.single, 2)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
1 1 1
## [149] 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
1 1 1
## [297] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [334] 1 1 1 1 1 1 1
```

g) Compare the results in part b, f and comment on your findings.

In terms of k-means, the first 6 observations have a mean shift relative to the next 334 observations which also shows that the correlation pushes towards PC1. For the hierarchical data, all three features of complete, average and single linkage generally separate the observations into their correct groups as this proves that PC1 is more favoured than PC2. Therefore, looking at both the plots respectively, we see that the clustering is pushed more towards PC1 and that is why there is more 1s in the dendograms than PC2.

Question 04 – Support Vector machines

a) Divide the dataset into two sets namely training set and test set by assigning 75% of the observations to training set and the rest of the observations to the test set

```
set.seed(1)
trid <- sample(1:nrow(house2), nrow(house2)*0.75)
train <- house2[trid,]
test <- house2[-trid,]

dim(test)

## [1] 85 10

dim(train)

## [1] 255  10
```

b) Fit a support vector classifier, in order to classify whether a house has high or low price.

```
library(e1071)
svm_fit = svm(formula = valueprice ~ ., data= newhouse2, kernel = "linear",
cost = 1, scale = TRUE)
summary(svm_fit)

##
## Call:
## svm(formula = valueprice ~ ., data = newhouse2, kernel = "linear",
##     cost = 1, scale = TRUE)
```

```
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  135
##
##  ( 68 67 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
set.seed(1)
tune_out <- tune(svm, valueprice ~ ., data = newhouse2, kernel = "linear",
ranges = list(cost = c(0.001, 0.01, 0.1, 1, 10, 100, 1000)), scale= TRUE)
summary(tune_out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.1764706
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.2588235 0.05849582
## 2 1e-02 0.1794118 0.05955428
## 3 1e-01 0.1764706 0.07069708
## 4 1e+00 0.1823529 0.06011651
## 5 1e+01 0.1794118 0.05623384
## 6 1e+02 0.1794118 0.05623384
## 7 1e+03 0.1794118 0.05623384
```

```
best_model = tune_out$best.model
summary(best_model)
```

```
##
## Call:
## best.tune(method = svm, train.x = valueprice ~ ., data = newhouse2,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 10, 100, 1000)),
##     kernel = "linear", scale = TRUE)
```

```
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##
## Number of Support Vectors:  157
##
##  ( 78 79 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

c) Fit a support vector machine with polynomial basis kernels, in order to classify whether a house has high or low price.

```
set.seed(1)
polytune_out = tune(svm, valueprice ~ ., data=newhouse2, kernel =
"polynomial", ranges = list(cost = c(0.001, 0.01, 0.1, 1, 10, 100, 1000), d =
c(2:5)), scale =TRUE)
summary(polytune_out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost d
##    10 3
##
## - best performance: 0.1911765
##
## - Detailed performance results:
##     cost d     error dispersion
## 1  1e-03 2 0.5676471 0.07076503
## 2  1e-02 2 0.5558824 0.07395303
## 3  1e-01 2 0.5264706 0.08255112
## 4  1e+00 2 0.3470588 0.06620370
## 5  1e+01 2 0.2823529 0.04842780
## 6  1e+02 2 0.2911765 0.06569359
## 7  1e+03 2 0.2911765 0.06855739
## 8  1e-03 3 0.5617647 0.07264170
## 9  1e-02 3 0.4382353 0.16778747
## 10 1e-01 3 0.2794118 0.06827641
## 11 1e+00 3 0.2147059 0.07343131
## 12 1e+01 3 0.1911765 0.08231792
## 13 1e+02 3 0.2029412 0.04892147
## 14 1e+03 3 0.2352941 0.08877829
## 15 1e-03 4 0.5352941 0.09383126
## 16 1e-02 4 0.5500000 0.06208290
## 17 1e-01 4 0.5205882 0.05890517
## 18 1e+00 4 0.3529412 0.11764706
## 19 1e+01 4 0.3323529 0.12011304
## 20 1e+02 4 0.3058824 0.07492146
## 21 1e+03 4 0.3000000 0.08964023
## 22 1e-03 5 0.5088235 0.10921586
## 23 1e-02 5 0.5264706 0.08018865
## 24 1e-01 5 0.4000000 0.10207384
## 25 1e+00 5 0.3205882 0.08484783
## 26 1e+01 5 0.2794118 0.07994856
## 27 1e+02 5 0.2647059 0.05545936
## 28 1e+03 5 0.2205882 0.07498558
```

```
polybest_model = polytune_out$best.model
summary(polybest_model)
```

```
##
## Call:
## best.tune(method = svm, train.x = valueprice ~ ., data = newhouse2,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 10, 100, 1000), d =
c(2:5)),
##     kernel = "polynomial", scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  168
##
##  ( 83 85 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

d) Fit a support vector machine with radial basis kernels, in order to classify whether a house has high or low price.

```
set.seed(1)
radtune_out = tune(svm, valueprice ~., data=newhouse2, kernel="radial",
ranges = list(cost = c(0.001, 0.01, 0.1, 1, 10, 100, 1000), gamma = c(0.5, 1,
2, 3, 4)), scale= TRUE)
summary(radtune_out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.1970588
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1  1e-03   0.5 0.5323529 0.14033655
## 2  1e-02   0.5 0.5323529 0.14033655
## 3  1e-01   0.5 0.2558824 0.05197003
## 4  1e+00   0.5 0.1970588 0.05554594
## 5  1e+01   0.5 0.2294118 0.06473558
## 6  1e+02   0.5 0.2500000 0.05923062
## 7  1e+03   0.5 0.2500000 0.07498558
## 8  1e-03   1.0 0.5264706 0.15089761
## 9  1e-02   1.0 0.5264706 0.15089761
## 10 1e-01   1.0 0.3117647 0.05580490
## 11 1e+00   1.0 0.2117647 0.06323339
## 12 1e+01   1.0 0.2294118 0.05150559
## 13 1e+02   1.0 0.2529412 0.06960094
## 14 1e+03   1.0 0.2588235 0.07310334
## 15 1e-03   2.0 0.5470588 0.11280902
## 16 1e-02   2.0 0.5470588 0.11280902
## 17 1e-01   2.0 0.4705882 0.11091871
## 18 1e+00   2.0 0.2470588 0.05750148
## 19 1e+01   2.0 0.2852941 0.07725839
## 20 1e+02   2.0 0.2823529 0.06960094
## 21 1e+03   2.0 0.2823529 0.06960094
## 22 1e-03   3.0 0.5500000 0.10921586
## 23 1e-02   3.0 0.5500000 0.10921586
## 24 1e-01   3.0 0.5411765 0.10846523
## 25 1e+00   3.0 0.3000000 0.05333910
## 26 1e+01   3.0 0.3235294 0.07719616
## 27 1e+02   3.0 0.3235294 0.07719616
## 28 1e+03   3.0 0.3235294 0.07719616
## 29 1e-03   4.0 0.5558824 0.09847940
## 30 1e-02   4.0 0.5558824 0.09847940
## 31 1e-01   4.0 0.5558824 0.09847940
## 32 1e+00   4.0 0.3411765 0.04205414
## 33 1e+01   4.0 0.3441176 0.05890517
## 34 1e+02   4.0 0.3441176 0.05890517
## 35 1e+03   4.0 0.3441176 0.05890517
```

```
radbest_model = radtune_out$best.model
summary(radbest_model)
```

```
##
## Call:
## best.tune(method = svm, train.x = valueprice ~ ., data = newhouse2,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 10, 100, 1000), gamma =
c(0.5,
##         1, 2, 3, 4)), kernel = "radial", scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  228
##
##  ( 108 120 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

e) Comment on the prediction accuracy of the model in part b, c and d. Hence suggest the best model with clear justification.

Comment on Linear Kernel Basis:

The linear model fitted with the cost of 1 and there were 157 support vectors. (78 support vectors are from one class and 79 support vectors are from the other class.)

From the tune_out output, it can be clearly seen that the optimal value of cost when using 10- fold cross validation is 1 since the error (0.176) is minimum when the cost is 1.

Thus, the optimal model using a linear kernel function can be obtained when cost is 1. The best model thus obtained consists of 157 support vectors with 78 support vectors from one class and 79 support vectors from the other class.

Comment on Polynomial Kernel Basis:

The polynomial model fitted with the cost of 0.1 and there were 211 support vectors. (106 support vectors are from one class and 105 support vectors are from the other class.)

From the polytune_out output, it can be clearly seen that the optimal value of cost when using 10- fold cross validation is 0.1 since the error (0.191) is minimum when the cost is 0.1.

Thus, the optimal model using a polynomial kernel function can be obtained when cost is 0.1. The best model thus obtained consists of 211 support vectors with 106 support vectors from one class and 105 support vectors from the other class.

Comment on Radial Kernel Basis:

The radial model fitted with the cost of 1 and there were 228 support vectors. (108 support vectors are from one class and 120 support vectors from the other class.)

From the radtune_out output, it can be clearly seen than the optimal valuse when using 10-fold cross validation is 1 since the error (0.197) is minimum when the cost is 1.

Thus, the optimal model using a radial kernel function can be obtained when cost is 1. The best model thus obtained consists of 228 support vectors with 108 support vectors from one class and 120 support vectors from the other class.