CSCI-B 565 DATA MINING
Homework II
Morning Class
Computer Science Core
Spring
Indiana University,
Bloomington, IN

Shanmukh Sista
ssista@indiana.edu

01/21/2015

All the work herein is solely mine.

Question 1

1. The sample space for the given problem is a set of all possible outcomes for the event i.e. throwing a six faced pair of dice. In our case , the sample space would be a set of values for the two pair of dice which would give us a sum of the values on individual die. These values are [1,1, 1,2,1,3.....1,6, 2, 1....2,6, ...6,1 ,6,2 ....6,6] (A total of 36 possible values for a throw).
2. A random variable for the given problem is a set of all possible values for the throw of the dice.
3. A function of the random variable for the given problem can be defined as :

$$P(X \to d_1 + d_2) = \begin{cases} X = 9 & \text{Friend Buys 3 cups coffee} \\ X \neq 9 & \text{I buy cookie} \end{cases}$$

4. Probability mass function for the throw of dice can be defined as the probability of the sum of the values equal to 9

$$\text{Probability mass function} = P(X = x)$$

5. Mass function over g(X)
6. Expected Value of X is :

$$\sum_{1}^{n} xp(x) = 2 * \tfrac{2}{36} + 3 * \tfrac{2}{36} + 4 * \tfrac{4}{36} + 5 * \tfrac{4}{36} + 6 * \tfrac{6}{36} + 7 * \tfrac{6}{36} + 8 * 636 + 9 * \tfrac{4}{36} + 10 * \tfrac{4}{36} + 11 * \tfrac{2}{36} + 12 * \tfrac{2}{36}$$
$$= 7$$

7. Expected value of g(X) =

$$(3.3 * 3 * \tfrac{4}{36}) - (1.0 * \tfrac{32}{36}) = .22$$

8. The winner is most likely to have cookies as the probability that the sum of two dice is equal to 9 is $\frac{4}{36}$.

Question 2

(a) Uses of Linear Regression According to the paper[1], following are the potential uses of regression.

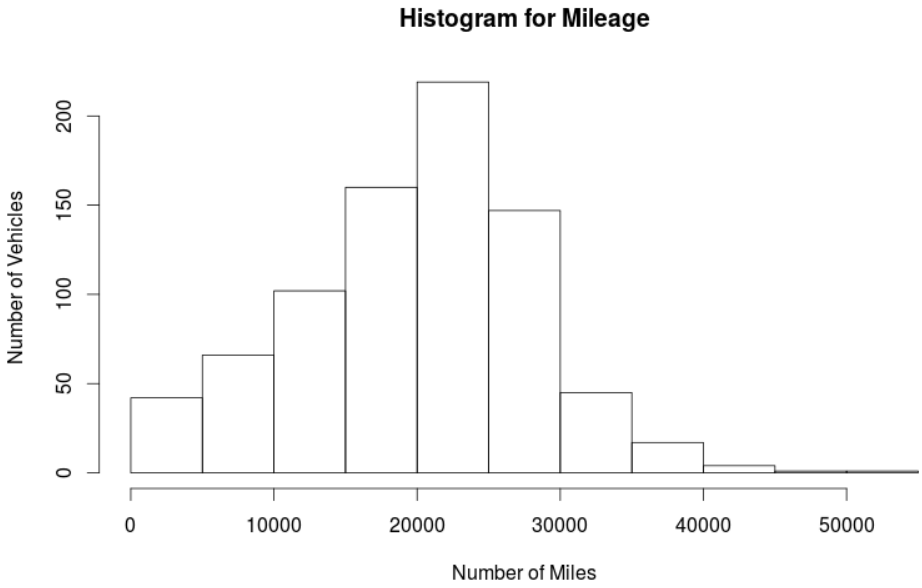
- Pure Description
  As the name suggests, pure description means the use of regression to describe our response variable based on the input attributes. In this case, we can consider all the attributes or decide to drop a few parameters to get an estimate of the response variable. This can be further decided based on the R2 values or the percentage reduction of the standard deviation.
- Prediction and estimation An important use of linear regression is the prediction and estimation of the regression parameters. In this case we would be required to control the variance in the data and make sure that even though there is some inherent variance in the existing system, our estimation of the linear regression paraemeters don't cause any additional variance as a result of their estimation.
- Extrapolation The regression model being developed can be used to extrapolate the response variable based on our input values.
- Estimation of Parameters.
- Control Control can be understood as the variation in the ouptut if the input is varied. A good linear regression model should have a good estimate of the regression coefficients to obtain a good control.
- Model Binding. This is an important use of Linear regression. A model can be built for Linear regression based on the input attributes. Subsetting, backward elimination techniques can be used to obtain a very good fit for the model.

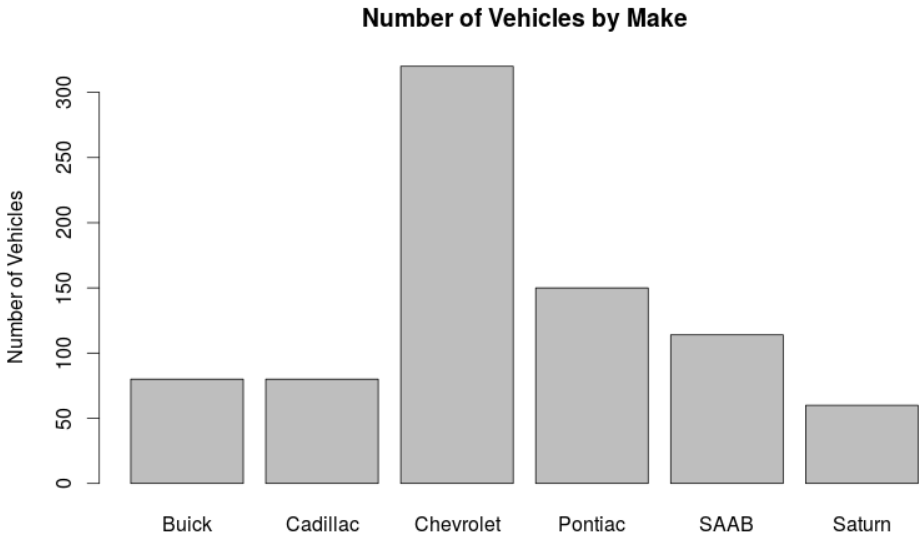
Ridge regression[2] deals mostly with multi-collinearity in the data. This is the case when two or more regression variables have a high value of correlation between them. Thus, for a small modification in any one of the attributes can result in erratic results for the output response estimation. Ridge regression accounts for this bias by introducing a constant to the equation and estimating the output based on this constant.

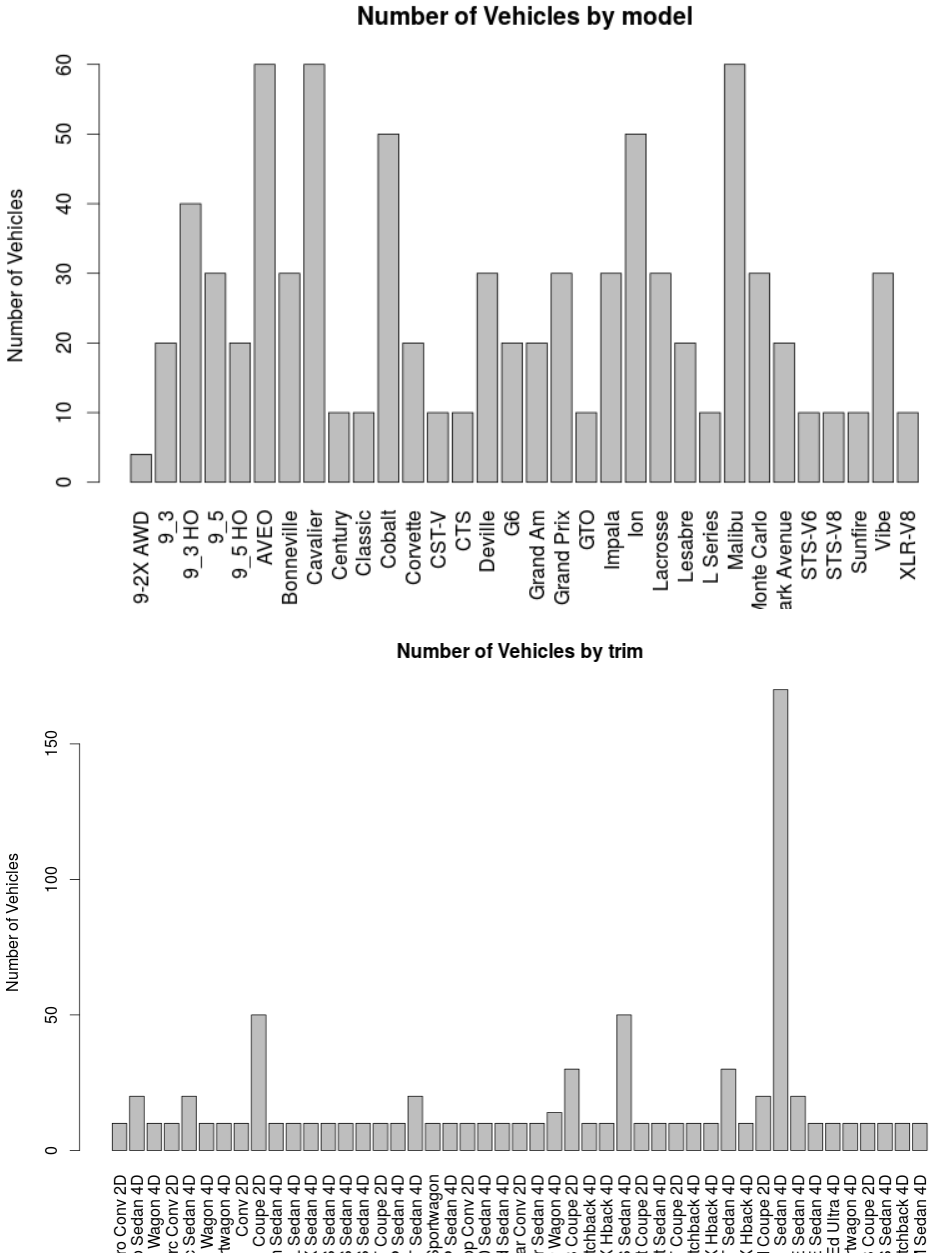
(b) Mysql Table for Vehicle Data

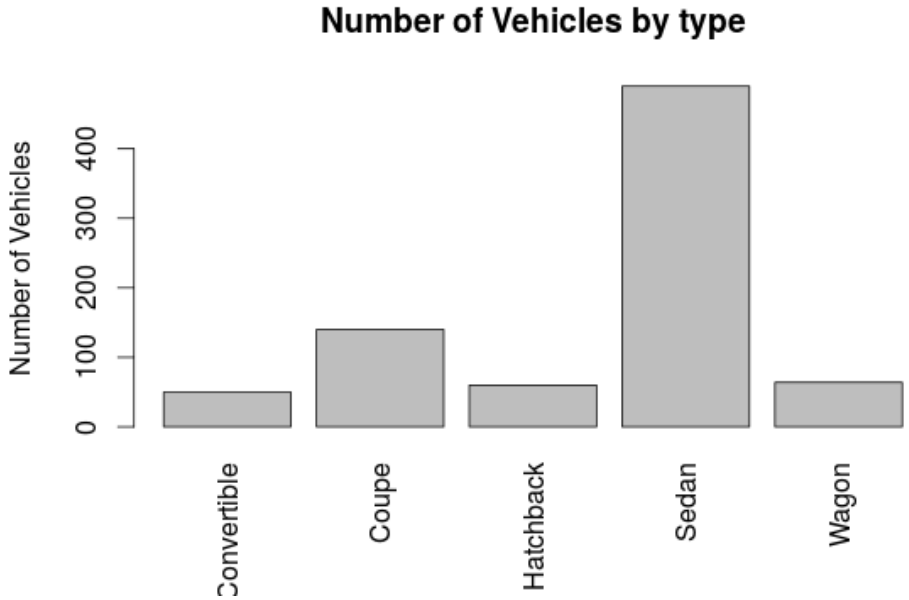
```
CREATE TABLE veh_pri(
    price decimal(8,2)  NULL DEFAULT NULL,
    mileage  mediumint(8) unsigned  NULL DEFAULT NULL,
    make varchar(20)  NULL DEFAULT NULL,
    model varchar(20) NULL DEFAULT NULL,
    trim varchar(20)  NULL DEFAULT NULL,
    type varchar(20)  NULL DEFAULT NULL,
    cylinders tinyint(3) unsigned NULL default null ,
    liters decimal(3,1) null default null,
    doors tinyint(3) unsigned null default null,
    cruise bit(1) NULL default null ,
    sound bit(1) null default null,
    leather bit(1) null default null
)
```

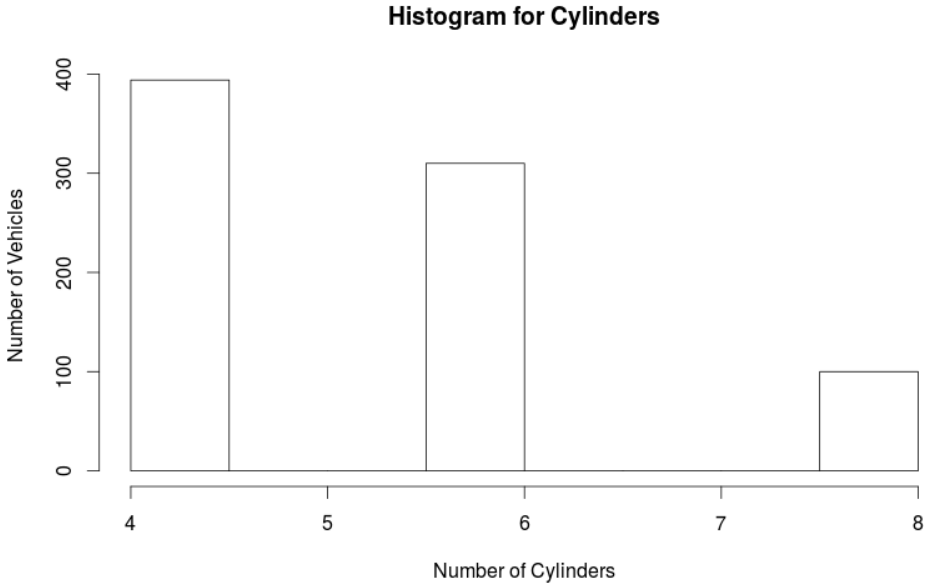
(c) Histogram

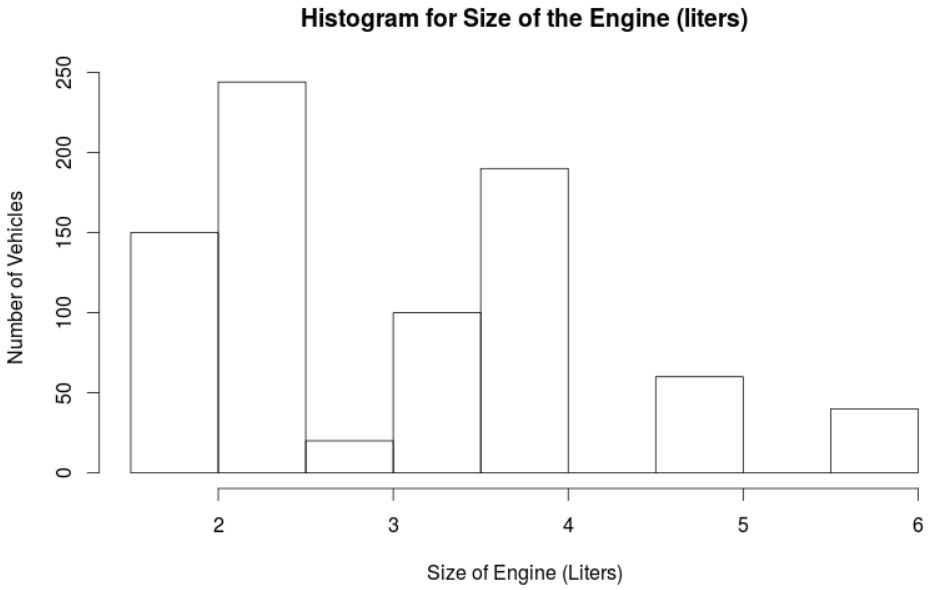

Histogram for Mileage
Number of Vehicles
0
50
100
150
200
0
10000
20000
30000
40000
50000
Number of Miles

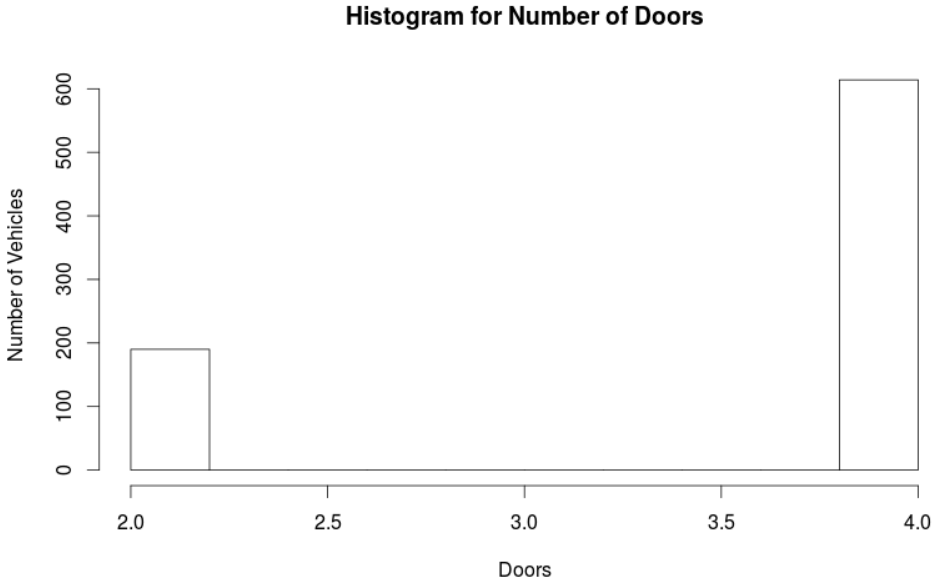

Number of Vehicles by Make
Number of Vehicles
0
50
100
150
200
250
300
Buick
Cadillac
Chevrolet
Pontiac
SAAB
Saturn

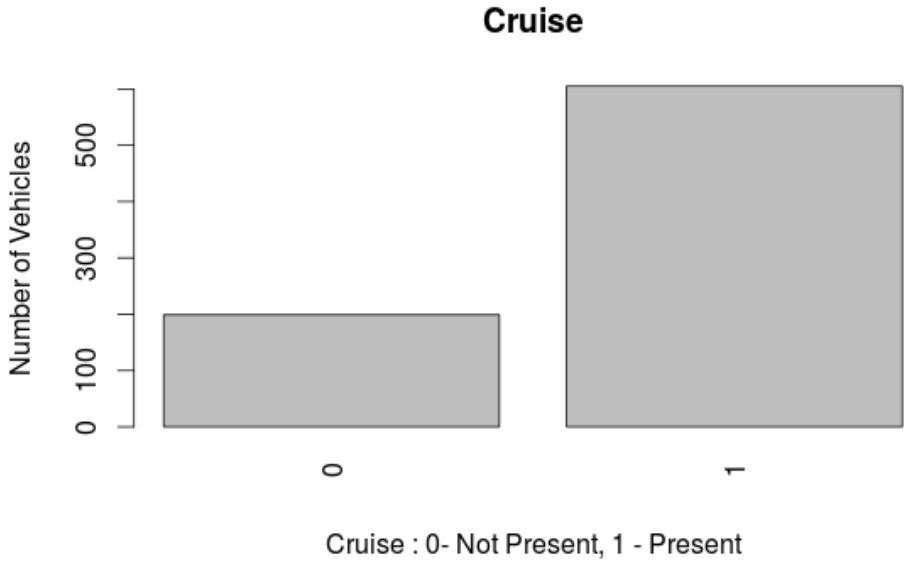

Number of Vehicles by model
Number of Vehicles
0
10
20
30
40
50
60
9-2X AWD
9_3
9_3 HO
9_5
9_5 HO
AVEO
Bonneville
Cavalier
Century
Classic
Cobalt
Corvette
CST-V
CTS
Deville
G6
Grand Am
Grand Prix
GTO
Impala
Ion
Lacrosse
Lesabre
L Series
Malibu
Monte Carlo
ark Avenue
STS-V6
STS-V8
Sunfire
Vibe
XLR-V8
Number of Vehicles by trim
Number of Vehicles
0
50
100
150

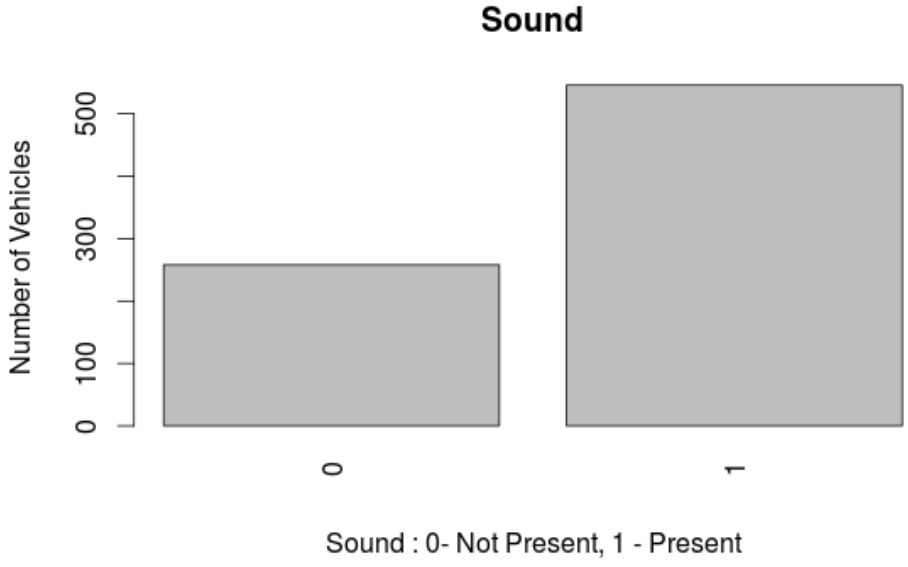

Number of Vehicles by type
Number of Vehicles
0
100
200
300
400
Convertible
Coupe
Hatchback
Sedan
Wagon

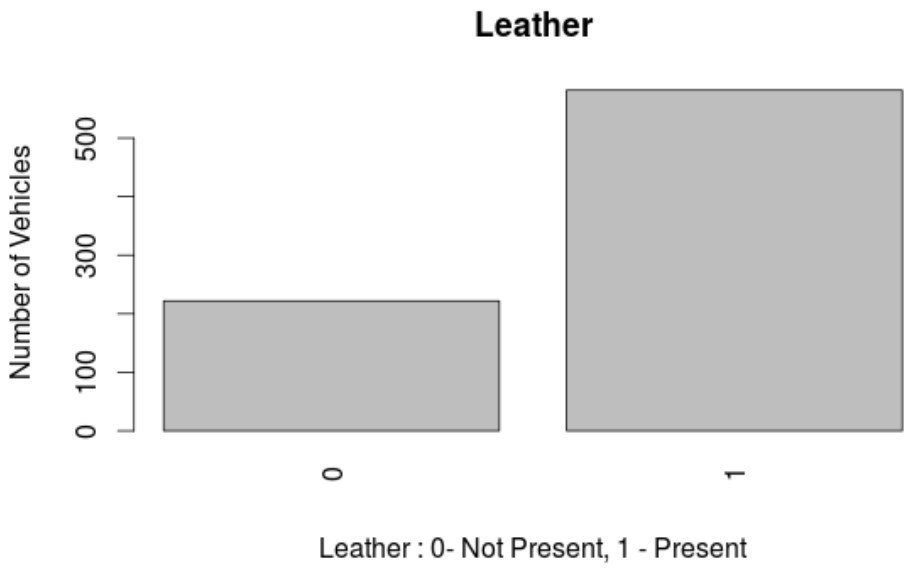

Histogram for Cylinders
Number of Vehicles
0
100
200
300
400
4
5
6
7
8
Number of Cylinders


Histogram for Size of the Engine (liters)
Number of Vehicles
0
50
100
150
200
250
2
3
4
5
6
Size of Engine (Liters)

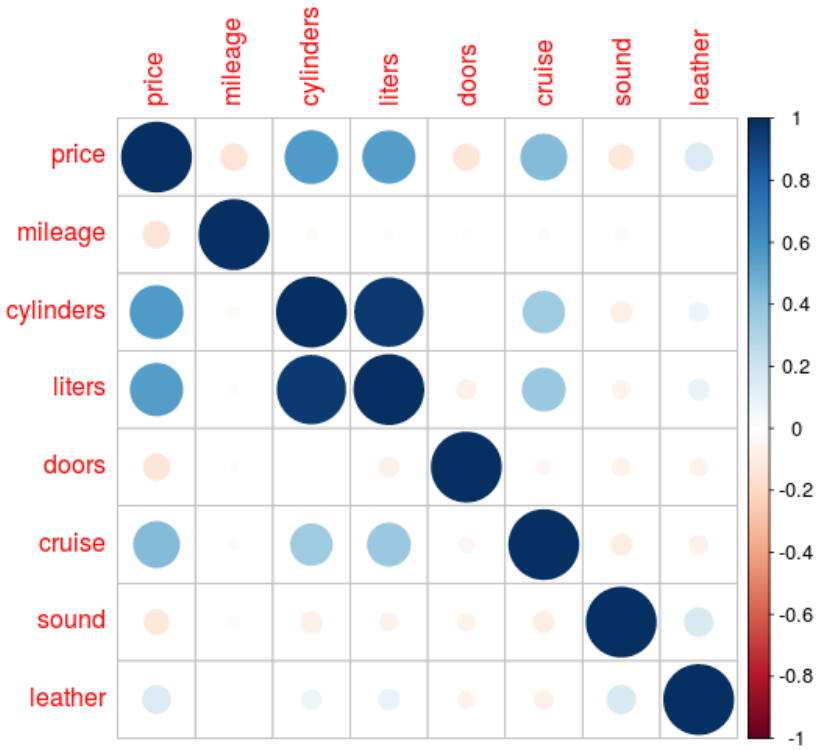

Histogram for Number of Doors
Number of Vehicles
0
100
200
300
400
500
600
2.0
2.5
3.0
3.5
4.0
Doors


Cruise
Number of Vehicles
0
100
300
500
0
1
Cruise : 0- Not Present, 1 - Present


Sound
Number of Vehicles
0
100
300
500
0
1
Sound : 0- Not Present, 1 - Present



(d) Correlation The correlation between various attributes is shown below.



The above correlation matrix has been plotted using the corrplot package and gives a very good overview of the correlation beween the various attributes and the price of the vehicle as well. It is obvious from the above figure that cylinders and engine capacity for a vehicle are directly correlated with a value of 1.0. So , higher the number of cylinders, higher is the engine capacity in liters.

Apart from the correlation between attributes, it can be observed that a few of them affect the price considerably and have to be included as a part of our model .

- Cylinders
- Liters
- Cruise

(e) Linear Models in Data

Output	Input	Summary
price	mileage, liters, cruise	Residual standard error: 7748 on 800 degrees of freedom Multiple R-squared:,0.3879 Adjusted R-squared:,0.3856, F-statistic:,169 on 3 and 800 DF, p-value: <2.2e-16
price	mileage , liters, cruise, make	Residual standard error: 4000 on 796 degrees of freedom Multiple R-squared:,0.8377,„Adjusted R-squared:,0.8362, F-statistic: 586.7 on 7 and 796 DF,„p-value: <2.2e-16
price	make	Residual standard error: 5911 on 798 degrees of freedom Multiple R-squared:,0.6447,„ Adjusted R-squared:,0.6425, F-statistic: 289.6 on 5 and 798 DF,„p-value: <2.2e-16
price	liters	Residual standard error: 8207 on 802 degrees of freedom Multiple R-squared:,0.3115,„Adjusted R-squared:,0.3107, F-statistic: 362.9 on 1 and 802 DF,„p-value: <2.2e-16
price	doors	Multiple R-squared:,0.01925,„Adjusted R-squared:,0.01803, F-statistic: 15.74 on 1 and 802 DF,„p-value: 7.906e-05
price	sound	Multiple R-squared:,0.01546,„Adjusted R-squared:,0.01423, F-statistic:,12.6 on 1 and 802 DF,„p-value: 0.0004092
price	leather	Multiple R-squared:,0.02471,„Adjusted R-squared:,0.02349, F-statistic: 20.32 on 1 and 802 DF,„p-value: 7.526e-06
price	model	Multiple R-squared:,0.949,„Adjusted R-squared:,0.9469, F-statistic: 463.1 on 31 and 772 DF,„p-value: <2.2e-16
price	mileage , liters, cruise, make, model	Multiple R-squared:,0.9496,„Adjusted R-squared:,0.9474, F-statistic: 439.4 on 33 and 770 DF,„p-value: <2.2e-16
price	type	Residual standard error: 8249 on 799 degrees of freedom Multiple R-squared:,0.3071,„ Adjusted R-squared:,0.3036, F-statistic: 88.51 on 4 and 799 DF,„p-value: <2.2e-16
price	trim	Multiple R-squared:,0.6332,„Adjusted R-squared:,0.6109, F-statistic: 28.41 on 46 and 757 DF,„p-value: <2.2e-16
price	mileage , liters, cruise, make, model, trim	Multiple R-squared:,0.9706,„ Adjusted R-squared:,0.9677, F-statistic: 345.1 on 70 and 733 DF,„p-value: <2.2e-16

This model is a good fit based on the given data. It can be seen from this model that make affects the price to a great extent. We can also infer this fact from the first model without the make. Although the price is affected by liters and cruise, we can see that the R2 value for the attributes is quite low.

It can be inferred from these three individual models that the attributes doors, sound and leather have a very little impact on the overall price of the vehicle.

(f) Analysis of variance for different models.

```
 fit.f0 = lm(price ~  liters + cruise + mileage, data = vehdata)
fit.f1 = lm ( price ~ make + price + liters + cruise , data = vehdata)
fit.f2 = lm ( price ~ make , data = vehdata)
summary(fit)
fit.f3 = lm ( price ~ liters , data = vehdata)
summary(fit)
fit.f4 = lm ( price ~ doors , data = vehdata)
summary(fit)
fit.f5 = lm ( price ~ sound , data = vehdata)
summary(fit)
fit.f6 = lm ( price ~ leather , data = vehdata)
summary(fit)
fit.f7 = lm ( price ~ model , data = vehdata)
summary(fit)
fit.f8 = lm ( price ~ make + model + price + liters + cruise , data = vehdata)
summary(fit)
fit.f9 = lm ( price ~ make + model + price + liters + cruise , data = vehdata)
  summary(fit)
fit.f10 = lm ( price ~ trim , data = vehdata)
summary(fit)
fit.f11 = lm ( price ~ type , data = vehdata)
summary(fit)
fit.f12 = lm ( price ~ make + model + price + liters + cruise + trim , data = vehdata)
summary(fit)

#Anova to compare individual models.
anova( fit.f12, fit.f4)
Analysis of Variance Table

Model 1: price ~ make + model + price + liters + cruise + trim
Model 2: price ~ doors
  Res.Df        RSS  Df  Sum of Sq      F    Pr(>F)
1    733 2.3105e+09
2    802 7.6951e+10 -69 -7.464e+10 343.18 < 2.2e-16 ***

anova( fit.f12, fit.f5)
Analysis of Variance Table

Model 1: price ~ make + model + price + liters + cruise + trim
Model 2: price ~ sound
  Res.Df        RSS  Df   Sum of Sq      F    Pr(>F)
1    733 2.3105e+09
2    802 7.7248e+10 -69 -7.4938e+10 344.54 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> anova( fit.f12, fit.f6)
Analysis of Variance Table

Model 1: price ~ make + model + price + liters + cruise + trim
Model 2: price ~ leather
  Res.Df        RSS  Df   Sum of Sq      F    Pr(>F)
```

```
1    733 2.3105e+09
2    802 7.6523e+10 -69 -7.4212e+10 341.21 < 2.2e-16 ***
---

> anova( fit.f12, fit.f7)
Analysis of Variance Table

Model 1: price ~ make + model + price + liters + cruise + trim
Model 2: price ~ model
  Res.Df        RSS  Df   Sum of Sq      F    Pr(>F)
1    733 2310529729
2    772 4004070935 -39 -1693541206 13.776 < 2.2e-16 ***

> anova( fit.f12, fit.f8)
Analysis of Variance Table

Model 1: price ~ make + model + price + liters + cruise + trim
Model 2: price ~ make + model + price + liters + cruise
  Res.Df        RSS  Df   Sum of Sq      F    Pr(>F)
1    733 2310529729
2    770 3956651243 -37 -1646121513 14.114 < 2.2e-16 ***
> anova( fit.f12, fit.f9)
Analysis of Variance Table

Model 1: price ~ make + model + price + liters + cruise + trim
Model 2: price ~ make + model + price + liters + cruise
  Res.Df        RSS  Df   Sum of Sq      F    Pr(>F)
1    733 2310529729
2    770 3956651243 -37 -1646121513 14.114 < 2.2e-16 ***

> anova( fit.f12, fit.f10)
Analysis of Variance Table

Model 1: price ~ make + model + price + liters + cruise + trim
Model 2: price ~ trim
  Res.Df        RSS  Df   Sum of Sq      F    Pr(>F)
1    733 2.3105e+09
2    757 2.8778e+10 -24 -2.6467e+10 349.86 < 2.2e-16 ***

> anova(fit.f12)
Analysis of Variance Table

Response: price
           Df     Sum Sq    Mean Sq   F value    Pr(>F)
make        5 5.0584e+10 1.0117e+10 3209.4554 < 2.2e-16 ***
model      26 2.3874e+10 9.1822e+08  291.3002 < 2.2e-16 ***
liters      1 4.7114e+07 4.7114e+07   14.9467 0.0001204 ***
cruise      1 3.0533e+05 3.0533e+05    0.0969 0.7557149
trim       37 1.6461e+09 4.4490e+07   14.1141 < 2.2e-16 ***
Residuals 733 2.3105e+09 3.1522e+06
```

(g) The p-value is mainly useful to test the significance of a statistical data for a hypothesis[3]. For a given Null

Hypothesis H, the P value can give the confidence with which we can accept the hypothesis.
R^2 can be defined as the measure of the closeness of a model with the actual data in linear regression. It is also known as the coefficient of determination. R^2 values can explain the variance in the data based on the defined model. A higher R^2 value indicates that our model fits the data better than a model with a lower R^2 values.
Based on the given data and histograms, I have selected the best possible models to analyse them. This has in turn resulted in selecting the best fit for the given data.

As it can be seen from the table below, Individual parameters such as doors, sound and leather have a very low value for R^2 and p. Thus we can safely assume that they are not the major contributors in the pricing for a car.

But if we were to consider a few combinations based on the histograms above, we find that the major contributors to price are mileage , liters and cruise. So by just looking at the correlation matrix and fitting a model I find that just by using these three attributes the R^2 value is not that bad. A .38 value can explain about 40 percent of the variance. So the next question is how can I improve this model ? Well, if the histograms are analyzed, it can be seen that the make, model and trim are important predictors of the price. The results can be clearly seen in the table. At the end the value of R^2 is close to .98 and a p value close to zero; which is an excellent fit for the given data. For the table given in the question, if we have a very large R value and a very low p value, then that model is a good estimate of the attributes.

If the R value is less and p value is also less, then it means that we have a lot of unexplained variance and we can be sure that our model does not explain this.

If the R value is less and p value is higher, this means that there is a variation unexplained in our data and we can't be very certain about it .

Question 3

(a) There are not many errors in the database. Some of the movidid's are missing. By adding a primary key for both movies and users, I have ensured that there are unique users and movies in the database. The ratings table has some inconsistent columns. First, the zipcode values have distinct values are in different values. Accepted range may be 5 or 9 characters seperated by a hiphen. But here we have incorrect values as well. Please refer the appendix to see a list of queries I have performed to check the data for consistency. The data cleaning process does affect my algorithm. My algorithm is based on the userid's to cluster the data. I have written a generic code but to avoid computational overhead and after analyzing the given data, it is safe to consider only unique userid's for the algorithm.

(b) Rating Comparision for men and women. For solving this question, I have made the following assumptions.

- Computer scientists fall under the occupation id's 4,12,15,17.
- To get the estimate of the similarity of the ratings by men and women, I have calculated the mean ratings for both men and women and variance for both the cases. If both these parameters are similar, then it can be inferred that men and women computer scientists give similar ratings to movies.

The following SQL will give us the details for male and female computer scientists.

```
select  t1.gender, avg(rating) as 'mean', std(rating) 'standard deviation' ,
variance(rating) as 'variance'
from (select movieid, rating, r.userid, gender, occupation
from ratings r inner join users u on u.userid = r.userid
and u.occupation in ( 4,12,15,17))
t1 group  by t1.gender
```

Output For the query :
'F', '3.6003', '1.1350', '1.2883'
'M', '3.5907', '1.1166', '1.2467'

It can be seen from the above output that the mean, standard deviation and ratings for both the movies rated by men and women are very similar. Hence it can be inferred that men and women computer scientists do rate movies similarly.

(c) From the data, men have rated more movies than women. The SQL for getting the result is

```
select gender, count(u.gender) from ratings r
inner join users u on u.userid = r.userid
group by u.gender
```

Result : gender, count(u.gender)

'F', '246440'

'M', '753769'

(d) The comedy genre has the highest number of ratings. To get this result , I have performed the following steps (SQL code has been given in the Appendix)

- Get all the rows containing genre values from the movies table.
- Output these rows into a text file on my linux file system (/tmp/genre.txt) with new line seperator same as the genre field sepeartor. This way when I import the data into the database table again, i will get every genre seperated by | as a new row.
- Now, Import the data into mysql table by using a row seperator as the | character.
- Create another table to hold all the genres and assign integer labels to each genre for the clustering.
- Join the tables movies, ratings and genres to get the result.

(e) Java code for clustering on Age and Profession, refer to Appendix

(f) Java code for clustering on Genre and Ratings, refer Appendix

(g) I have used the Euclidean distance function for the given samples.
For the first clustering, both the attributes are numeric, so i directly used the formula for euclidean distance for this algorithm.
For clustering based on ratings and genre, I first got all the distinct values for genres and then joined them with the ratings table to get values for rating, genre and users. After this, I have used the Euclidean distance for these two attributes.

(h) The quality of clustering can be judged by two methods.

(a) The closeness of items in the cluster
Each of the points inside a cluster should have a high degree of similarity.

(b) The dissimilarity between clusters.
There should be a high degree of dissimilarity between different clusters.

If these two conditions are met, we can say that we have a very good clustering for the given data.

Question 2

1. Table Creation Scripts

```
CREATE TABLE veh_pri(
price decimal(8,2)  NULL DEFAULT NULL,
    mileage  mediumint(8) unsigned  NULL DEFAULT NULL,
    make varchar(20)  NULL DEFAULT NULL,
    model varchar(20) NULL DEFAULT NULL,
    trim varchar(20)  NULL DEFAULT NULL,
    type varchar(20)  NULL DEFAULT NULL,
```

```
    cylinders tinyint(3) unsigned NULL default null ,
    liters decimal(3,1) null default null,
    doors tinyint(3) unsigned null default null,
    cruise bit(1) NULL default null ,
sound bit(1) null default null,
    leather bit(1) null default null
)
```

2. Data Import from file to Mysql using R

```
importVehicleData <- function(){
  library("DBI")
  library(RMySQL)
  #Connect to the mysql database
  db = dbConnect(RMySQL::MySQL(), user='root', password='root', dbname='csci_b565')
  #Read csv data from the file
  data <- read.table("/home/shanmukh/Desktop/data.csv", header=FALSE, sep=";")
  res <-  dbGetQuery(db, "TRUNCATE TABLE veh_pri")
  #for each row in the data frame, insert it into the veh_pri table.
  apply(data, 1 , function(row){
    #format the data as per mysql table definition and Generate SQL Statements to erowport t
    query <- paste("INSERT INTO veh_pri VALUES(",round(as.double(row["V1"]),digits=2), ", ",
                   as.numeric(row["V2"]), ", '",
                   row["V3"], "', '",
                   row["V4"], "', '",
                   row["V5"], "', '",
                   row["V6"], "', ",
                   row["V7"], ", ",
                   row["V8"], ", ",
                   row["V9"], ", ",
                   row["V10"], ", ",
                   row["V11"], ", ",
                   row["V12"], ")", sep="")
    # Execute this sql query
    dbGetQuery(db, query)
  })
  # Close the current mysql connection.
  dbDisconnect(db)
}
```

3. Data Import to R from mysql

```
library("DBI")
library(RMySQL)
#Connect to the mysql database
db = dbConnect(RMySQL::MySQL(), user='root', password='Rfja_2208', dbname='csci_b565')
# Read data from the veh_pri table.
vehdata <- dbGetQuery(db, 'SELECT `veh_pri`.`price`, `veh_pri`.`mileage`,
    `veh_pri`.`make`,
    `veh_pri`.`model`,
    `veh_pri`.`trim`,
    `veh_pri`.`type`,
    `veh_pri`.`cylinders`,
    `veh_pri`.`liters`,
```

```
    ‘veh_pri‘.‘doors‘,
    case  ‘veh_pri‘.‘cruise‘
    when 1 then 1
    when 0 then 0
    end as cruise,
    case ‘veh_pri‘.‘sound‘
    when 1 then 1
    when 0 then 0
    end as sound,
   case  ‘veh_pri‘.‘leather‘
     when 1 then 1
    when 0 then 0
    end as leather
FROM ‘csci_b565‘.‘veh_pri‘;
’)
dbDisconnect(db)
```

Question 3

1. Table Creation scripts

```
CREATE TABLE movies ( movieid int PRIMARY KEY  ,
moviename text ,
                        genre text )

CREATE TABLE users ( userid int primary key ,
gender char(1),
                    age smallint,
                    occupation smallint,
                    zipcode varchar(10)
)
DROP Table ratings
CREATE TABLE ratings ( userid int,
movieid int,
                        rating smallint,
                        timestamp bigint
                        )
```

2. Data Import for Movies, Users and Ratings.

```
library("DBI")
library(RMySQL)
shouldImportMovies = TRUE
shouldImportUsers = TRUE
shouldImportRatings = TRUE
Sys.setlocale('LC_ALL','C')
#Connect to the mysql database
db = dbConnect(RMySQL::MySQL(), user='root',
               password='root', dbname='csci_b565')
if ( shouldImportMovies){
  res <-  dbGetQuery(db, "TRUNCATE TABLE movies")
  #for each row in the data frame, insert it into the veh_pri table.
  print("Importing Movie Data")
  data <- readLines("/home/shanmukh/Desktop/ml-1m/movies.dat")
  lapply(data,function(row){
```

```
    #for each row seperate columns and build an sql string.
    splitRow <- strsplit(row, "::")
    r <- unlist(splitRow,c)
    movieid = dbEscapeStrings(db,r[1])
    moviename= dbEscapeStrings(db,r[2])
    moviegenres = dbEscapeStrings(db,r[3])
    #insert these values to sql table
    insertQuery = paste("INSERT INTO movies VALUES
                        ( " , movieid, ",'" , moviename,
                        "','", moviegenres, "')", sep="")
    result <- dbGetQuery(db, insertQuery)
  })
  print("Movies Data import complete")
}
if ( shouldImportUsers){
  #Import Users Data Now
  print("Importing Users Data.")
  res <-  dbGetQuery(db, "TRUNCATE TABLE users")
  #for each row in the data frame, insert it into the veh_pri table.
  data <- readLines("/home/shanmukh/Desktop/ml-1m/users.dat")
  lapply(data,function(row){
    #for each row seperate columns and build an sql string.
    splitRow <- strsplit(row, "::")
    r <- unlist(splitRow,c)
    userid = dbEscapeStrings(db,r[1])
    gender = dbEscapeStrings(db,r[2])
    age = r[3]
    occupation = r[4]
    zipcode = r[5]
    #insert these values to sql table
    insertQuery = paste("INSERT INTO users
                        VALUES
                        ( " , userid , ",'"
                        ,gender , "',", age,"," ,
                        occupation ,",'" , zipcode, "')", sep="")
    result <- dbGetQuery(db, insertQuery)
  })
}
if (shouldImportRatings){
  #Import ratings Data Now
  print("Importing ratings Data.")
  res <-  dbGetQuery(db, "TRUNCATE TABLE ratings")
  data <- readLines("/home/shanmukh/Desktop/ml-1m/ratings.dat")
  dataLenght <<- length(data)

  stepCount <<- 3000
  for ( i in seq(1, length(data), stepCount)){
  mainInsertQuery = paste0("INSERT INTO ratings values ")
  end = i + stepCount - 1
  if ( end <= length(data)){
    lapply(data[(i:end)],function(row){
      splitRow <- strsplit(row, "::")

      r <- unlist(splitRow,c)
```

```
      userid = r[1]
      movieid = r[2]
      rating = r[3]
      timestamp = r[4]
      #insert these values to sql table

      insertQuery = paste0("( " , userid , "," ,movieid ,
                           "," ,rating , " , " , timestamp , ")")
      mainInsertQuery <<- paste(mainInsertQuery, insertQuery, sep=",")
    })
  }
  else{
    end = length(data)
    lapply(data[(i:end)],function(row){
      splitRow <- strsplit(row, "::")
      r <- unlist(splitRow,c)
      userid = r[1]
      movieid = r[2]
      rating = r[3]
      timestamp = r[4]
      #insert these values to sql table

      insertQuery = paste0("( " , userid , "," ,movieid ,
                           "," ,rating , " , " , timestamp , ")")
      mainInsertQuery <<- paste(mainInsertQuery, insertQuery, sep=",")
    })
  }
  print(paste("Processed ", i - 1 , "records"))
  #Get the substring
  mainInsertQuery = sub(",", " ", mainInsertQuery)
  dbGetQuery(db,mainInsertQuery)
}
print("Ratings Import complete.")
}
# Close the current mysql connection.
dbDisconnect(db)
```

3. Data Cleaning SQL Scripts

```
#check if there are any duplicate movies in the database
select count(distinct movieName) from movies

#get the total number of ratings and total number of distinct movieid and user id
#if this number does not match, then one user has rated a movie more than once.
select count(*) FROM ratings
select count(distinct movieid, userid) from ratings

#check if all the zip codes are unique.
select distinct( LENGTH(zipcode)) as ZipLength from users

#select distinct gender values from the table.
select distinct(gender) from users

#since we have only male and female values, we can proceed with the next value.
```

```
#Get all the distinct professions. 4,12,15,17 - may fall as computer scientists.
(select * from ratings r inner join users u on u.userid = r.userid
and u.occupation in ( 4,12,15,17)) t1

#who has rated more movies men / women

select gender, count(u.gender) from ratings r inner join
users u on u.userid = r.userid group by u.gender

#Question 3.4

#export data to text file
select distinct genre into outfile '/tmp/genre.txt'
fields terminated by '|' escaped by ''
 lines terminated by '|'
from movies

#read from the file as comma seperated values
create  table genres ( genrename varchar(100))

create table genre_categories (
genreid int primary key auto_increament,
genrename varchar(100));

load data infile '/tmp/genre.txt'
ignore into table genres
lines terminated by '|';

insert into genre_categories ( genrename )
(select distinct genrename from genres)

#count for all genre ratings
select r.userid, m.movieid , rating, gc.genrename from ratings r inner join movies m on
r.movieid =m.movieid inner join genre_categories gc where locate( gc.genrename, m.genre) > 0
genre_categories group by  r.userid, m.movieid , rating, gc.genrename order by rand() limit


#max genre count
select  max(ratingcount)
from (select gc.genrename as 'genrecategory', count(r.movieid)
as 'ratingcount' from ratings r inner join movies m
on r.movieid =m.movieid
inner join genre_categories gc
where locate( gc.genrename, m.genre) > 0
group by gc.genrename) temp
```

4. Agglomerative Clustering for Ratings and Genre. Main.java

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Set;
import java.util.Stack;
import java.util.TreeMap;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

import org.jgrapht.graph.DefaultWeightedEdge;
import org.jgrapht.traverse.DepthFirstIterator;
public class Main {
 //Agglomerative heirarchial clustering java
 public static void main(String[] args) throws InterruptedException {
  Connection conn = null;
  Statement s = null;
  PreparedStatement ps = null;
  ResultSet rs = null;
  //Load jdbc driver
  try {
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    conn = DriverManager
              .getConnection("jdbc:mysql://localhost/?user=root&password=root");
    s = conn.createStatement();
       s.executeUpdate("use csci_b565;");
       System.out.println("Fetching data...");
    //rs = s.executeQuery("  limit 5");
    rs = s.executeQuery("select r.userid, m.movieid , rating, gc.genreid "
      + "from ratings r inner join movies m on  r.movieid =m.movieid inner "
      + "join genre_categories gc where locate( gc.genrename, m.genre) > 0"
      + " group by  r.userid, m.movieid , rating, gc.genrename "
      + "order by rand() limit 1000");
    long startTime= System.nanoTime();
    List<List<RatingsNode>> data= new ArrayList<List<RatingsNode>>();
    RatingsNode n ;
    while ( rs.next()){
     //Create a new node
     List<RatingsNode> ln = new ArrayList<RatingsNode>();
     n = new RatingsNode(rs.getInt(1), rs.getInt(2) , rs.getInt(3), rs.getInt(4));
     ln.add(n);
     data.add(ln);
     n = null;
    }
```

```
//Now we have all the age entries of database in this arraylist.
//now create a distance matrix

ConcurrentHashMap<String, Double> map = new
  ConcurrentHashMap<String, Double>(data.size()/10);
List<RatingsNode> outerNode ,innerNode;
double d ;
String key ="";
//For every item in treemap
System.out.println("Processing...");
while ( map.size() != 1){
 System.out.println("New Iteration Started");
 int datasize = data.size();
 //System.out.println("\n\n*****New Iteration*********\n\n");
 //Compute distance matrix for one iteration
 for ( int i = 0 ; i < datasize ;i++){
  outerNode = data.get(i);
  String outerKey = "";
  for ( RatingsNode on : outerNode){
   if (outerNode.size() > 1){
    if ( outerKey != ""){
     outerKey = outerKey + ";";
    }
    outerKey = outerKey + String.valueOf(on.getId());
   }
   else{
    outerKey = outerKey + String.valueOf(on.getId());
   }
  }
  //System.out.println("outer key is " + outerKey);
  for ( int j = i+1 ; j < data.size(); j++){
   innerNode = data.get(j);
   String innerKey = "";
   for ( RatingsNode in : innerNode){
    if (innerNode.size() > 1){
     if ( innerKey != ""){
      innerKey = innerKey + ";";
     }
     innerKey = innerKey + String.valueOf(in.getId());
    }
    else{
     innerKey = innerKey +  String.valueOf(in.getId());
    }
   }
   d = RatingsNode.CalculateDistance(outerNode, innerNode);
   key= outerKey + ";" + innerKey;
   map.put( key , d);

  }
 }
 //Hashmap is the distance matrix
 //Loop through all the values in the hashmap and find the minimum value for key
 double min = 0.0;
 String minkey = "";
```

```
int ic = 0;
Set<String> keys = map.keySet();
for ( String k : keys){
 if ( ic == 0 ){
  min = map.get(k);
  minkey = k ;
  ic++;
 }
 else{
  double lmin = map.get(k);
  if ( lmin < min){
   min = lmin;
   minkey = k;
  }
 }
}
//Add this minkey to our cluster adn remove the association from hashtable.
//Delete all the keys that have these vertices.
Stack<List<RatingsNode>> ds = new Stack<List<RatingsNode>>();
List<List<RatingsNode>> rm = new ArrayList<List<RatingsNode>> ();
HashMap<String , RatingsNode> hm = new HashMap<String , RatingsNode>();
for ( String v : minkey.split(";")){
 //create a new list
 //v is the key in the treenode as well.
 //loop through the list to find the items

  for (List<RatingsNode> n1 : data){
   for( RatingsNode in1 : n1 ){
   //check if the given node list is in the array s
   String in1id = String.valueOf(in1.getId());
   if (v.compareTo(in1id) == 0){
    //System.out.println("Removing node " + in1id);
    if ( !hm.containsKey(v)){
     hm.put(v, in1);
     rm.add(n1);
     break;
    }
    //data.remove(k);
    //Check if the elements are already present in the stack.
    //ds.push(n1);
    //System.out.println("pushing data");
    break;
   }
  }
 }

 //Add a node to data

 for ( String ikey : keys ){
  if ( ikey.contains(v)){
   map.remove(ikey);
   //Combine and remove from list

  }
```

```
    }
   }

   //System.out.println(minkey + " is minimum with a value  "+ min);


   List<RatingsNode> newNode = new ArrayList<RatingsNode>();

   for ( List<RatingsNode> rm1 : rm){
    //remove from list
    data.remove(rm1);
   }
   for ( String ks : hm.keySet()){
    newNode.add(hm.get(ks));
   }
   //add this list to the existing list
   data.add(newNode);
   //System.out.println("map after key deletion");
   /*for ( String keyss : map.keySet()){
    System.out.println("key value is "+ keyss +  " Val : " + map.get(keyss));
   }*/
   /*for ( List<AgeProfessionNode> ln  : data){
    System.out.println("Parent -> children");
    for ( AgeProfessionNode lns : ln){
     //System.out.println("CHild " + lns.getId());
     System.out.println(lns.Print());
    }
   }*/
   //This minimum will form a new cluster
   //update the treemap with the new points

  }
  //Write all the clusters to file.
  FileWriter write = new FileWriter( "cluster-"+ System.nanoTime()  + ".txt", false);
  PrintWriter printer = new PrintWriter(write);


 int level = 0;
 for ( List<RatingsNode> ln  : data){
  //System.out.println("Level " + level);
  for ( RatingsNode lns : ln){
   //System.out.println("CHild " + lns.getId());
   StringBuilder sb = new StringBuilder();
   sb.append(level);
   sb.append(lns.Print());
   System.out.print(sb.toString());
   printer.println(sb.toString());
  }
  level++;
 }
 printer.close();
  long endtime = System.nanoTime();
  System.out.println("Total time : " + (endtime - startTime)/1000000000.0);
} catch (ClassNotFoundException e) {
```

```
    // TODO Auto-generated catch block
    e.printStackTrace();
   }
    catch (SQLException e) {
     // TODO Auto-generated catch block
     e.printStackTrace();
    } catch (InstantiationException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
   } catch (IllegalAccessException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
   } catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
   }
 }

}
```

RatingsNode.java

```
import java.util.List;
import java.util.Set;

import org.jgrapht.event.GraphVertexChangeEvent;
import org.jgrapht.graph.DefaultWeightedEdge;


public class RatingsNode implements Comparable<RatingsNode>{
 private  int id ;
 private int rating, movieid;
 private int genre;
 private boolean processedNode ;
 public RatingsNode(int id, int movieid, int rating, int genre){
  this.id = id;
  this.movieid = movieid;
  this.rating = rating;
  this.genre = genre;
 }
 public int getId(){
  return this.id;
 }
 @Override
 public synchronized boolean equals(Object other) {
     if (!(other instanceof RatingsNode)) {
         return false;
     }
     RatingsNode otherobj = (RatingsNode) other;
     // Custom equality check here.
     return (this.id == (otherobj.getId()));
 }
 public int getGenre(){
  return this.genre;
 }
```

```
 public synchronized String Print(){
  return (";" + id  + ";" + this.movieid +";"+ rating + ";" + this.genre + "\n");
 }
 public synchronized void setProcessed ( boolean ans){
  this.processedNode = ans ;
 }
 public int getAge(){
  return this.rating;
 }
 public synchronized boolean isProcessed(){
  return processedNode;
 }
 public Double computeDistance(RatingsNode node){
  return Math.abs(Math.random());
 }
 public int GetMovieId(){
  return this.movieid;
 }
 public int GetRating(){
  return this.rating;
 }
 public static double CalculateDistance( List<RatingsNode> n1 , List<RatingsNode> n2){
  double d = 0.0;
  //For each of the clusters , find the optimal distance ( average link )
  double avgDist = 0.0;
  if ( n1.size() == 1 && n2.size() ==1 ){
   d = Math.sqrt(Math.pow((n2.get(0).GetRating() - n1.get(0).GetRating()),2) +
     Math.pow((n1.get(0).getGenre() -n2.get(0).getGenre()),2));
  }
  else{
   for ( RatingsNode nn1 : n1){
    double ld = 0.0;
    //Compute the distance between nn1 and nn2
    for ( RatingsNode nn2 : n2){
     ld = Math.sqrt(Math.pow((nn1.GetRating() - nn2.GetRating()),2) +
       Math.pow((nn1.getGenre() -nn2.getGenre()),2));
     d += ld ;
    }
   }
   //take the average
   d = d / ( n1.size() * n2.size());
  }
  return d;
 }
 @Override
 public int compareTo(RatingsNode o) {
  // TODO Auto-generated method stub
  //Calculate distances for these two nodes
  return 0;
 }
}
```

5. Clustering on Age and Profession

   Main.java

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Set;
import java.util.Stack;
import java.util.TreeMap;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

import org.jgrapht.graph.DefaultWeightedEdge;
import org.jgrapht.traverse.DepthFirstIterator;
public class Main {
 //Agglomerative heirarchial clustering java
 public static void main(String[] args) throws InterruptedException {
  AgeProfessionCluster c = new AgeProfessionCluster();
  Connection conn = null;
  Statement s = null;
  PreparedStatement ps = null;
  ResultSet rs = null;
  //Load jdbc driver
  try {
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    conn = DriverManager
             .getConnection("jdbc:mysql://localhost/?user=root&password=root");
    s = conn.createStatement();
       s.executeUpdate("use csci_b565;");
    //rs = s.executeQuery("  limit 5");
    rs = s.executeQuery("select userid, age, occupation from users LIMIT 20");
    long startTime= System.nanoTime();
    List<List<AgeProfessionNode>> data= new ArrayList<List<AgeProfessionNode>>();
    AgeProfessionNode n ;
    while ( rs.next()){
     //Create a new node
     List<AgeProfessionNode> ln = new ArrayList<AgeProfessionNode>();
     n = new AgeProfessionNode(rs.getInt(1), rs.getInt(2) , rs.getInt(3));
     ln.add(n);
     data.add(ln);
     n = null;
    }
```

```
//Now we have all the age entries of database in this arraylist.
//now create a distance matrix

ConcurrentHashMap<String, Double> map = new
  ConcurrentHashMap<String, Double>(data.size()/10);
List<AgeProfessionNode> outerNode ,innerNode;
double d ;
String key ="";
//For every item in treemap
System.out.println("Processing...");
while ( map.size() != 1){
 System.out.println("print loop");
 int datasize = data.size();
 //System.out.println("\n\n*****New Iteration*********\n\n");
 //Compute distance matrix for one iteration
 for ( int i = 0 ; i < datasize ;i++){
  outerNode = data.get(i);
  String outerKey = "";
  for ( AgeProfessionNode on : outerNode){
   if (outerNode.size() > 1){
    if ( outerKey != ""){
     outerKey = outerKey + ";";
    }
    outerKey = outerKey + String.valueOf(on.getId());
   }
   else{
    outerKey = outerKey + String.valueOf(on.getId());
   }
  }
  //System.out.println("outer key is " + outerKey);
  for ( int j = i+1 ; j < data.size(); j++){
   innerNode = data.get(j);
   String innerKey = "";
   for ( AgeProfessionNode in : innerNode){
    if (innerNode.size() > 1){
     if ( innerKey != ""){
      innerKey = innerKey + ";";
     }
     innerKey = innerKey + String.valueOf(in.getId());
    }
    else{
     innerKey = innerKey +  String.valueOf(in.getId());
    }
   }
   d = AgeProfessionNode.CalculateDistance(outerNode, innerNode);
   key= outerKey + ";" + innerKey;
   map.put( key , d);

  }
 }
 //Hashmap is the distance matrix
 //Loop through all the values in the hashmap
 //and find the minimum value for key
 double min = 0.0;
```

```
String minkey = "";
int ic = 0;
Set<String> keys = map.keySet();
for ( String k : keys){
 if ( ic == 0 ){
  min = map.get(k);
  minkey = k ;
  ic++;
 }
 else{
  double lmin = map.get(k);
  if ( lmin < min){
   min = lmin;
   minkey = k;
  }
 }
}
//Add this minkey to our cluster adn remove the association from hashtable.
//Delete all the keys that have these vertices.
Stack<List<AgeProfessionNode>> ds = new Stack<List<AgeProfessionNode>>();
List<List<AgeProfessionNode>> rm = new ArrayList<List<AgeProfessionNode>> ();
HashMap<String, AgeProfessionNode> hm = new HashMap<String, AgeProfessionNode>();
for ( String v : minkey.split(";")){
 //create a new list
 //v is the key in the treenode as well.
 //loop through the list to find the items

  for (List<AgeProfessionNode> n1 : data){
   for( AgeProfessionNode in1 : n1 ){
   //check if the given node list is in the array s
   String in1id = String.valueOf(in1.getId());
   if (v.compareTo(in1id) == 0){
    //System.out.println("Removing node " + in1id);
    if ( !hm.containsKey(v)){
     hm.put(v, in1);
     rm.add(n1);
     break;
    }
    //data.remove(k);
    //Check if the elements are already present in the stack.
    //ds.push(n1);
    //System.out.println("pushing data");
    break;
   }
  }
 }

 //Add a node to data

 for ( String ikey : keys ){
  if ( ikey.contains(v)){
   map.remove(ikey);
   //Combine and remove from list
```

```
    }
   }
  }

  //System.out.println(minkey + " is minimum with a value  "+ min);


  List<AgeProfessionNode> newNode = new ArrayList<AgeProfessionNode>();

  for ( List<AgeProfessionNode> rm1 : rm){
   //remove from list
   data.remove(rm1);
  }
  for ( String ks : hm.keySet()){
   newNode.add(hm.get(ks));
  }
  //add this list to the existing list
  data.add(newNode);
  //System.out.println("map after key deletion");
  /*for ( String keyss : map.keySet()){
   System.out.println("key value is "+ keyss +  " Val : " + map.get(keyss));
  }*/
  /*for ( List<AgeProfessionNode> ln  : data){
   System.out.println("Parent -> children");
   for ( AgeProfessionNode lns : ln){
    //System.out.println("CHild " + lns.getId());
    System.out.println(lns.Print());
   }
  }*/
  //This minimum will form a new cluster
  //update the treemap with the new points

 }
 //Write all the clusters to file.
 FileWriter write = new FileWriter( "cluster.txt" , false);
 PrintWriter printer = new PrintWriter(write);


int level = 0;
for ( List<AgeProfessionNode> ln  : data){
 //System.out.println("Level " + level);
 for ( AgeProfessionNode lns : ln){
  //System.out.println("CHild " + lns.getId());
  StringBuilder sb = new StringBuilder();
  sb.append(level);
  sb.append(lns.Print());
  System.out.print(sb.toString());
  printer.println(sb.toString());
 }
 level++;
}
printer.close();
 long endtime = System.nanoTime();
 System.out.println("Total time : " + (endtime - startTime)/1000000000.0);
```

```
  } catch (ClassNotFoundException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  }
   catch (SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
   } catch (InstantiationException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  } catch (IllegalAccessException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  } catch (IOException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  }
 }

}
```

AgeProfessionNode.java

```
import java.util.List;
public class AgeProfessionNode implements Comparable<AgeProfessionNode>{
 private  int id ;
 private int age, professionCode;
 private boolean processedNode ;
 public AgeProfessionNode(int id, int age, int professionCode){
  this.age = age;
  this.id = id;
  this.professionCode = professionCode;
 }
 public int getId(){
  return this.id;
 }
 @Override
 public synchronized boolean equals(Object other) {
     if (!(other instanceof AgeProfessionNode)) {
         return false;
     }
     AgeProfessionNode otherobj = (AgeProfessionNode) other;
     // Custom equality check here.
     return (this.id == (otherobj.getId()));
 }
 public synchronized String Print(){
  return (";" + id  + ";" + age + ";" + this.professionCode + "\n");
 }
 public synchronized void setProcessed ( boolean ans){
  this.processedNode = ans ;
 }
 public int getAge(){
  return this.age;
```

```
}
public synchronized boolean isProcessed(){
 return processedNode;
}
public Double computeDistance(AgeProfessionNode node){
 return Math.abs(Math.random());
}
public int GetProfession(){
 return this.professionCode;
}
public static double CalculateDistance( List<AgeProfessionNode> n1 , List<AgeProfessionNo
 double d = 0.0;
 //For each of the clusters , find the optimal distance ( average link )
 double avgDist = 0.0;
 if ( n1.size() == 1 && n2.size() ==1 ){
  d = Math.sqrt(Math.pow((n2.get(0).getAge() - n1.get(0).getAge()),2) +
    Math.pow((n1.get(0).GetProfession() -n2.get(0).GetProfession()),2));
 }
 else{
  for ( AgeProfessionNode nn1 : n1){
   double ld = 0.0;
   //Compute the distance between nn1 and nn2
   for ( AgeProfessionNode nn2 : n2){
    ld = Math.sqrt(Math.pow((nn1.getAge() - nn2.getAge()),2) +
      Math.pow((nn1.GetProfession() -nn2.GetProfession()),2));
    d += ld ;
   }
  }
  //take the average
  d = d / ( n1.size() * n2.size());
 }
 return d;
}
@Override
public int compareTo(AgeProfessionNode o) {
 // TODO Auto-generated method stub
 //Calculate distances for these two nodes
 return 0;
}
}
```

Below is the code

The contents... test[4]

References

[1] R. R. Hocking. A biometrics invited paper. the analysis and selection of variables in linear regression. *Biometrics*, 32(1):1–49, 1976.

[2] ncss pdf. Ridge regression. http://www.ncss.com/wp-content/themes/ncss/pdf/Procedures/NCSS/.

[3] Jim Frost. How to correctly interpret p values. http://blog.minitab.com/blog/adventures-in-statistics/how-to-correctly-interpret-p-values.

[4] Shonda Kuiper. Introduction to multiple regression: How much is your car worth? *Journal of Statistics Education*, 16(3), 2008.