FUNDAMENTAL OF DATA SCIENCE

HOMEWORK – 2

Name: GOUTHAM SELVAKUMAR

PROBLEM – 1

a. Visualize the distributions of the variables in this data. You can choose bar graphs, histograms and density plots. Make appropriate choices given each type of variables and be careful when selecting parameters like the number of bins for the histograms. Note there are some numerical variables and some categorical ones. The ones labeled as a 'bool' are Boolean variables, meaning they are only true or false and are thus a special type of categorical. Checking all the distributions with visualization and summary statistics is a typical step when beginning to work with new data.

Installing Packages:

```r
library(tidyr)
library(tidyverse)
library(caret)
library(dplyr)
library(GGally)
library(ggplot2)
library(e1071)
```

Importing and viewing the dataset:

```r
library(readxl)
setwd("C:/Users/admin/Desktop")
cpBankData<-read.csv("BankData.csv")
summary(cpBankData)
head(cpBankData)
```

```
       X             cont1            cont2            cont3           bool1              bool2              cont4            bool3              cont5        
 Min.   :  1.0   Min.   :13.75   Min.   : 0.000   Min.   : 0.000   Length:690         Length:690         Min.   : 0.0   Length:690         Min.   :   0  
 1st Qu.:173.2   1st Qu.:22.60   1st Qu.: 1.000   1st Qu.: 0.165   Class :character   Class :character   1st Qu.: 0.0   Class :character   1st Qu.:  75  
 Median :345.5   Median :28.46   Median : 2.750   Median : 1.000   Mode  :character   Mode  :character   Median : 0.0   Mode  :character   Median : 160  
 Mean   :345.5   Mean   :31.57   Mean   : 4.759   Mean   : 2.223                                         Mean   : 2.4                      Mean   : 184  
 3rd Qu.:517.8   3rd Qu.:38.23   3rd Qu.: 7.207   3rd Qu.: 2.625                                         3rd Qu.: 3.0                      3rd Qu.: 276  
 Max.   :690.0   Max.   :80.25   Max.   :28.000   Max.   :28.500                                         Max.   :67.0                      Max.   :2000  
                 NA's   :12                                                                                                                NA's   :13    
     cont6            approval          credit.score        ages      
 Min.   :     0.0   Length:690         Min.   :583.7   Min.   :11.00  
 1st Qu.:     0.0   Class :character   1st Qu.:666.7   1st Qu.:31.00  
 Median :     5.0   Mode  :character   Median :697.3   Median :38.00  
 Mean   :  1017.4                      Mean   :696.4   Mean   :39.67  
 3rd Qu.:   395.5                      3rd Qu.:726.4   3rd Qu.:48.00  
 Max.   :100000.0                      Max.   :806.0   Max.   :84.00  
```

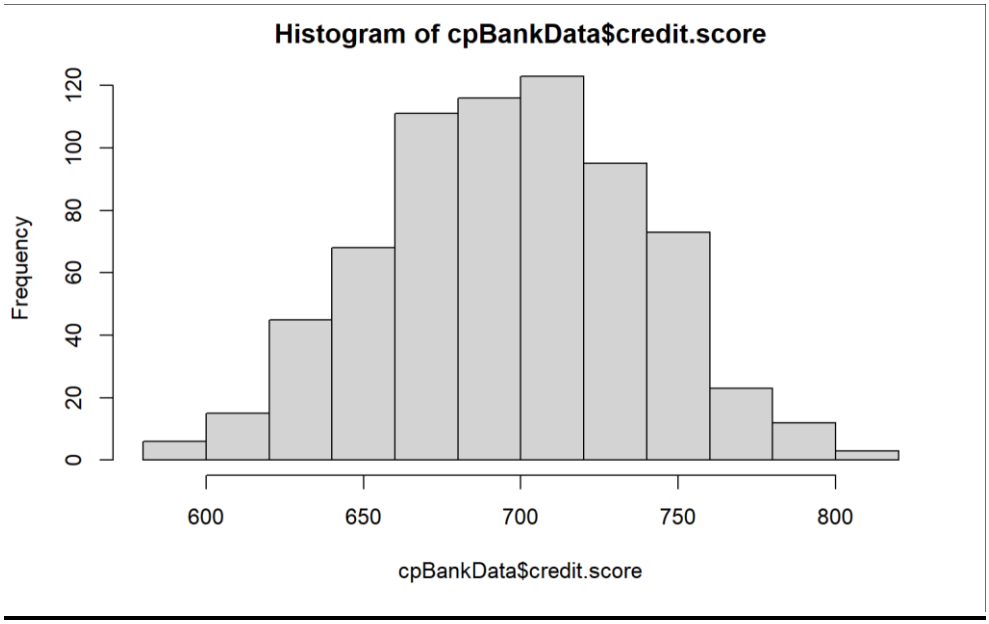
Description: df [6 x 13]

	X	cont1	cont2	cont3	bool1	bool2	cont4	bool3	cont5
	<int>	<dbl>	<dbl>	<dbl>	<chr>	<chr>	<int>	<chr>	<int>
1	1	30.83	0.000	1.25	t	t	1	f	202
2	2	58.67	4.460	3.04	t	t	6	f	43
3	3	24.50	0.500	1.50	t	f	0	f	280
4	4	27.83	1.540	3.75	t	t	5	t	100
5	5	20.17	5.625	1.71	t	f	0	f	120
6	6	32.08	4.000	2.50	t	f	0	t	360

6 rows | 1-10 of 13 columns

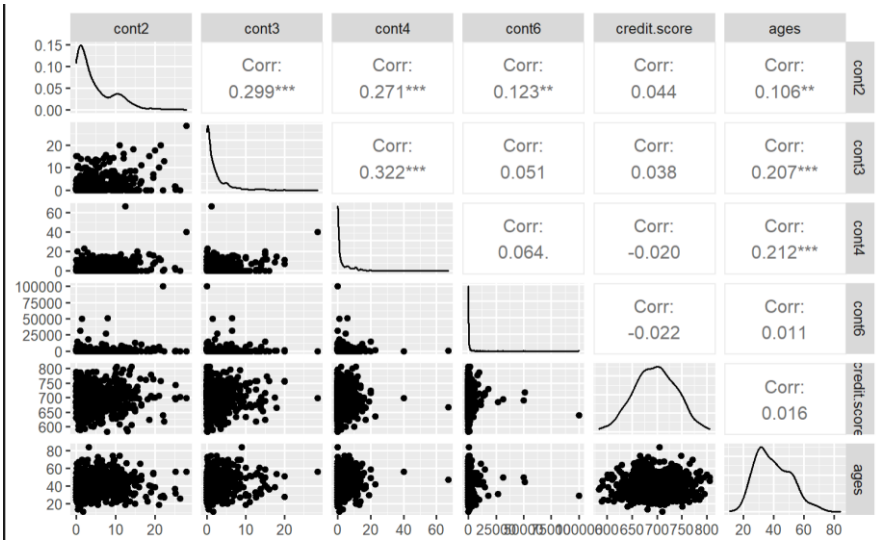
Visualizing credit scores by creating a histogram with 8 bins:

```r
```{r}
hist(cpBankData$credit.score, nclass = 8)
cpBankData %>% select(cont2, cont3, cont4, cont6, credit.score, ages) %>% ggpairs()
hist(cpBankData$ages, nclass = 12)
```
```



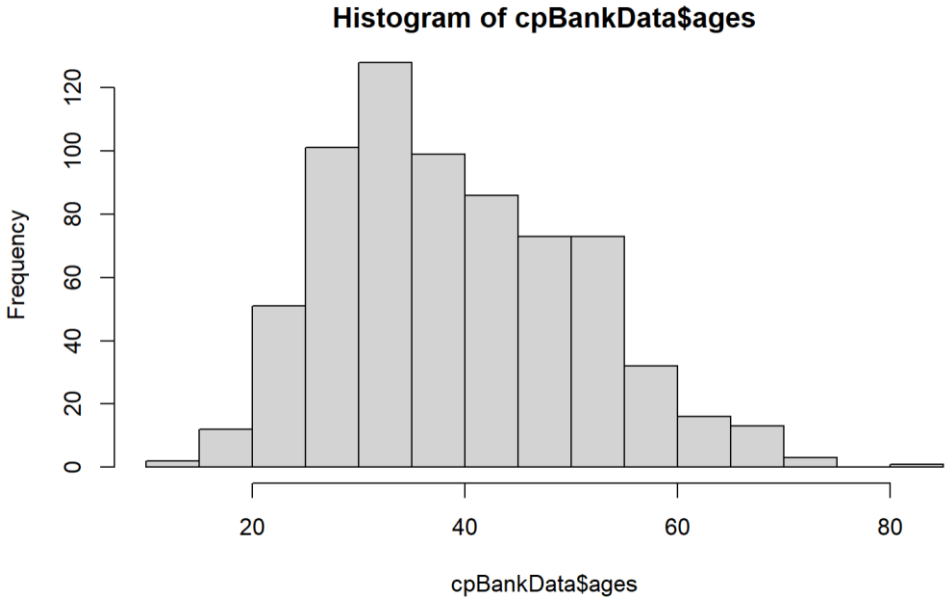
This shows us a clear distribution of the credit score of all loan applicants with this dataset.

Now, perhaps try to visualize and find the relationship between numericals using a scatterplot matrix.



Based on the correlation data from the scatterplot matrix, there may be a positive correlation between cont2 and cont3, cont4 and cont2, cont4 and cont3, and ages and cont3.

Now, perhaps take a look at the groups represented in this dataset.



The 30+ but under the 40 age group is considered the highest represented group in this dataset.

So now, we're somewhat familiar with the dataset, we can see that the dataset collected data on loan applicants to decide on their approval or rejection for a loan application. We can see there is a column specific for approvals with + and – for approval and rejection, respectively.

b. Now apply normalization to some of these numerical distributions. Specifically, choose to apply z-score to one, min-max to another, and decimal scaling to a third. Explain your choices of which normalization applies to which variable in terms of what the variable means, what distribution it starts with, and how the normalization will affect it.

In this part, I normalized cont6, ages, and credit.score. It is unclear what cont6 is, although it may be a 'continuous' variable that helped with predicting the approval rate for an applicant. Credit scores and ages are a clearer indicator of approval rates, however.

First, I used Z-Score normalization for credit scores, which is a fairly common method of normalizing data. Since, the credit score is a much larger number compared to other data in this dataset, normalizing it will help provide a normal distribution and reduce the redundancy, while maintaining the integrity of the credit score.

```
# Used Z-Score normalization for the credit.score data
# Z-Score equation
```{r}
zscore <- function(x)
{
  a=((x=mean(x))/sd(x))
}
cpBankData["credit.score"] <- (lapply(cpBankData["credit.score"], zscore))
```
```

Next, I used the min-max method to normalize the age's data. While I could've used this method on the credit scores, I saved this method for ages because I want to keep ages as a positive number. The credit score data, for example, could be normalized into negative numbers and ruled as 'low credit scores'. Keeping the ages positive is more logical in this case.

```
# Used Min-Max normalization for the ages data
# Min-Max equation
```{r}
min_max <- function(x,new_max=6,new_min=0)
{
  a= (((x-min(x))* (new_max-new_min))/(max(x)))+new_min
  return(a)
}
cpBankData["ages"] <- (lapply(cpBankData["ages"],min_max))
```
```

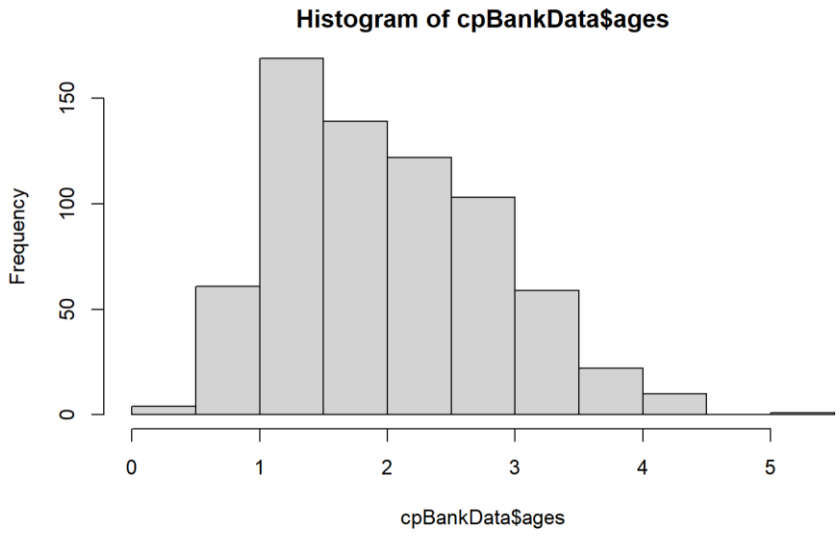
Now, I'm using the decimal scaling to normalize the cont6 column. Decimal scaling normalizes the data by moving the decimal point. I decided to use it on cont6 since there is a large variation in the numbers so I want it to be lowered to more reasonable variations.

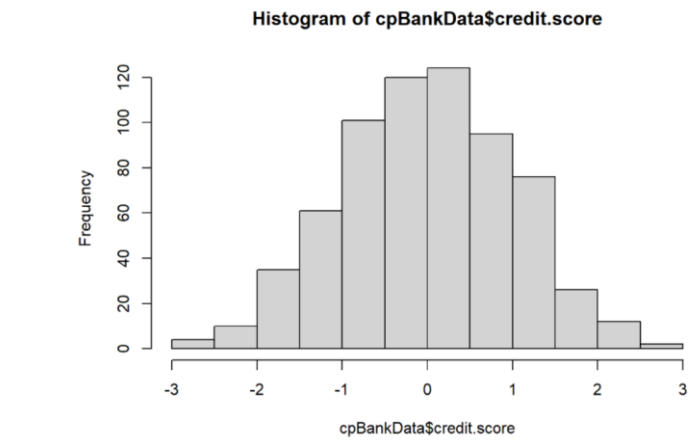
```
# Decimal Scaling Equation is
```{r}
decimal_scale <- function(x)
{
  a=x/100
  return(a)
}
cpBankData["cont6"] <- (lapply(cpBankData["cont6"],decimal_scale))
```
```

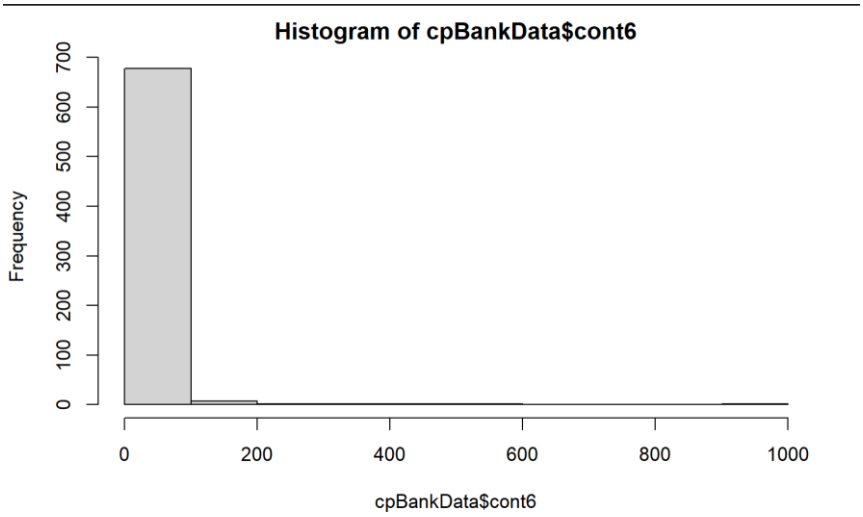
c. Visualize the new distributions for the variables that have been normalized. What has changed from the previous visualization?

Normalized Data:

```
# Visualizing the normalized ages
# Normalized Data, all cpBankData have been normalized
```{r}
hist(cpBankData$ages, nclass = 10)
hist(cpBankData$credit.score, nclass = 8)
hist(cpBankData$cont6, nclass = 8)
```
```

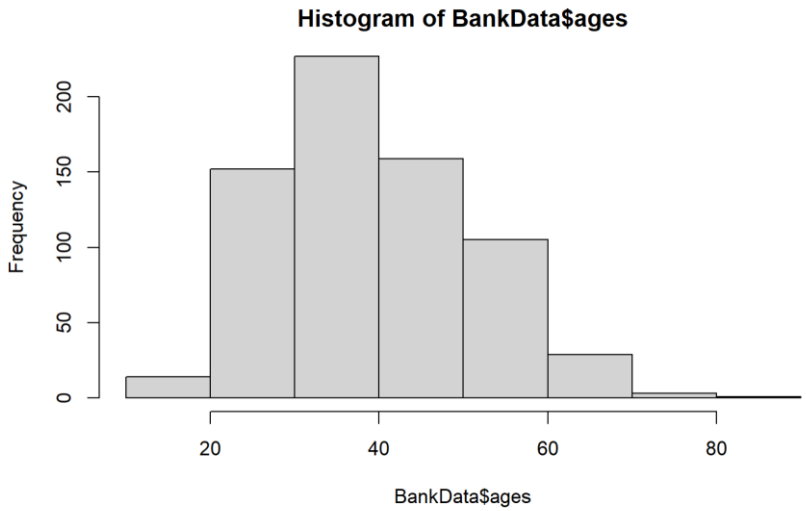


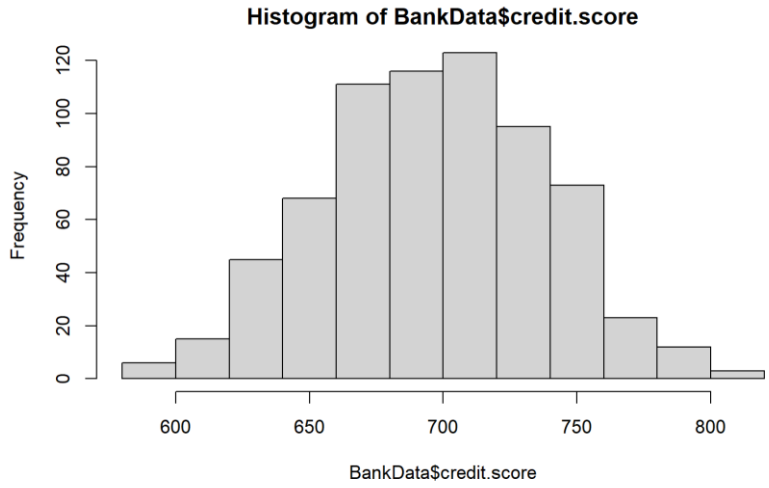


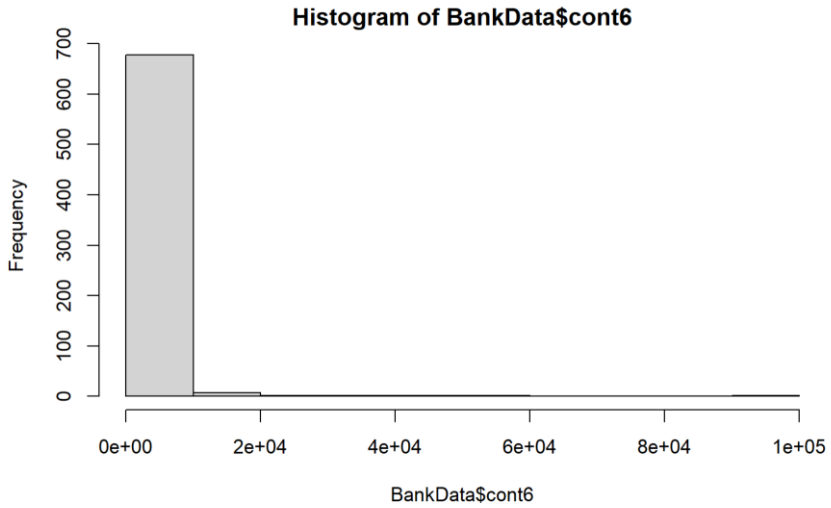


Not Normalized:

```
# Not Normalized Data
```{r}
setwd("C:/Users/admin/Desktop")
BankData<-read.csv("BankData.csv")
hist(BankData$ages, nclass = 10)
hist(BankData$credit.score, nclass = 8)
hist(BankData$cont6, nclass = 8)
```
```







Based on the visualizations above, it is mostly obvious that the credit score and ages datasets have a more normal and even distribution. You can especially notice this change in the ages dataset where the same number if bins resulted in a smoother bar graph. On the other hand, cont6 had such huge variations that even the normalized

version of the data is still quite large due to the many outliers. This would be hard to manipulate without understanding how the cont6 data affects approval, or understanding what the cont6 variable truly is.

d. Choose one of the numerical variables to work with for this problem. Let's call it v. Create a new variable called v_bins that is a binned version of that variable. This v_bins will have a new set of values like low, medium, high. Choose the actual new values (you don't need to use low, medium, high) and the ranges of v that they represent based on your understanding of v from your visualizations. You can use equal depth, equal width or custom ranges. Explain your choices: why did you choose to create that number of values and those particular ranges?

I decided to work with the raw values of credit scores in this question, since I'm already familiar with the expected results. I will now use the conventional credit score ratings from very poor to exceptional, with the lowest being 300 and highest being 850.

```
# Importing the data again with the original values
```{r}
setwd("C:/Users/admin/Desktop")
BankData<-read.csv("BankData.csv")
BankData$v_bins <- BankData$credit.score
# Create names for the bins from poor to exceptional
names <- c("verypoor", "fair", "good", "verygood", "exceptional")
# Created a vector of break points using the threshold credit scores
b <- c(-Inf, 500, 670, 740, 800, Inf)
# Bin the data into 5 bins
BankData$v_bins <- cut(BankData$v_bins, breaks = b, labels = names)
head(BankData)
```
```

Description: df [6 x 14]

	X <int>	cont1 <dbl>	cont2 <dbl>	cont3 <dbl>	bool1 <chr>	bool2 <chr>	cont4 <int>	bool3 <chr>	cont5 <int>
1	1	30.83	0.000	1.25	t	t	1	f	202
2	2	58.67	4.460	3.04	t	t	6	f	43
3	3	24.50	0.500	1.50	t	f	0	f	280
4	4	27.83	1.540	3.75	t	t	5	t	100
5	5	20.17	5.625	1.71	t	f	0	f	120
6	6	32.08	4.000	2.50	t	f	0	t	360

6 rows | 1-10 of 14 columns

e. Building on (d), use v_bins to create a smoothed version of v. Choose a smoothing strategy to create a numerical version of the binned variable and explain your choices.

I used the mean of each of the 5 bins, then replaced each bin with the bin's average which will allow us to smooth the data but hold on to the integrity of the values. I then binded the separate sets using the tidyverse function.

```r
verypoor <- BankData %>% filter(v_bins == 'verypoor') %>% mutate(credit.score = mean(credit.score, na.rm = T))
fair <- BankData %>% filter(v_bins == 'fair') %>% mutate(credit.score, na.rm = T)
good <- BankData %>% filter(v_bins == 'good') %>% mutate(credit.score, na.rm = T)
verygood <- BankData %>% filter(v_bins == 'verygood') %>% mutate(credit.score, na.rm = T)
exceptional <- BankData %>% filter(v_bins == 'exceptional') %>% mutate(credit.score = mean(credit.score, na.rm = T))
# Now that we're done smoothing the data and mutating it, we can bind them
bind_rows(list(verypoor, fair, good, verygood, exceptional))
head(BankData)
```

Description: df [690 x 15]

X	cont1	cont2	cont3	bool1	bool2	cont4	bool3	cont5	cont6
<int>	<dbl>	<dbl>	<dbl>	<chr>	<chr>	<int>	<chr>	<int>	<int>
1	30.83	0.000	1.250	t	t	1	f	202	0
3	24.50	0.500	1.500	t	f	0	f	280	824
4	27.83	1.540	3.750	t	t	5	t	100	3
13	38.25	6.000	1.000	t	f	0	t	0	0
16	36.67	4.415	0.250	t	t	10	t	320	0
19	21.83	0.250	0.665	t	f	0	t	0	0
21	25.00	11.250	2.500	t	t	17	f	200	1208
24	27.42	14.500	3.085	t	t	1	f	120	11
26	15.83	0.585	1.500	t	t	2	f	100	0
28	56.58	18.500	15.000	t	t	17	t	0	0

1-10 of 690 rows | 1-10 of 15 columns

Previous 1 2 3 4 5 6 ... 69 Next

Description: df [6 x 14]

	X	cont1	cont2	cont3	bool1	bool2	cont4	bool3	cont5
	<int>	<dbl>	<dbl>	<dbl>	<chr>	<chr>	<int>	<chr>	<int>
1	1	30.83	0.000	1.25	t	t	1	f	202
2	2	58.67	4.460	3.04	t	t	6	f	43
3	3	24.50	0.500	1.50	t	f	0	f	280
4	4	27.83	1.540	3.75	t	t	5	t	100
5	5	20.17	5.625	1.71	t	f	0	f	120
6	6	32.08	4.000	2.50	t	f	0	t	360

6 rows | 1-10 of 14 columns

PROBLEM – 2

a. Apply SVM to the data from Problem 1 to predict approval and report the accuracy using 10-fold cross validation.

```r
library(readxl)
setwd("C:/Users/admin/Desktop")
Bank_Data<-read.csv("BankData.csv")
# installing kernlab
library(kernlab)
# Let's fit the model first
svm1 <- train(approval ~., data = Bank_Data, method = "svmLinear")
svm1
```

```
Support Vector Machines with Linear Kernel

666 samples
 13 predictor
  2 classes: '-', '+'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 666, 666, 666, 666, 666, 666, ...
Resampling results:

  Accuracy   Kappa
  0.8481258  0.7002727

Tuning parameter 'C' was held constant at a value of 1
```

I noticed that the accuracy is at 0.84 here, which is decent although not close to perfect.

```
# Evaluation method parameter, using all the datapoints to train the model
train_control = trainControl(method = "cv", number = 10)
# Scaling Method
preproc = c("center", "scale")
svm2 <- train(approval ~., data = Bank_Data, method = "svmLinear", trControl = train_control, preProcess = preproc)
svm2
```

```
Support Vector Machines with Linear Kernel

666 samples
 13 predictor
  2 classes: '-', '+'

Pre-processing: centered (16), scaled (16)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 600, 599, 600, 599, 599, 600, ...
Resampling results:

  Accuracy   Kappa
  0.8636137  0.7290904

Tuning parameter 'C' was held constant at a value of 1
```

```
# Now, reporting the accuracy using 10-fold cross validation
train_control_cv = trainControl(method = "cv", number = 10)
train_control_l00cv = trainControl(method = "L00CV", number = 10)
svm3 <- train(approval ~., data = Bank_Data, method = "svmLinear", trControl = train_control_cv)
svm3
```

```
Support Vector Machines with Linear Kernel

666 samples
 13 predictor
  2 classes: '-', '+'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 601, 599, 599, 599, 599, 599, ...
Resampling results:

  Accuracy  Kappa
  0.863382  0.7282009

Tuning parameter 'C' was held constant at a value of 1
```

b. Next, use the grid search functionality when training to optimize the C parameter of the SVM. What parameter was chosen and what is the accuracy?

```
```{r}
grid <- expand.grid(C = 10^seq(-5,2,0.5))
svm_grid <- train(approval ~., data = Bank_Data, method = "svmLinear", trControl = train_control, tuneGrid = grid)
svm_grid
```
```

```
Support Vector Machines with Linear Kernel

666 samples
 13 predictor
  2 classes: '-', '+'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 599, 599, 600, 600, 599, 600, ...
Resampling results across tuning parameters:

  C             Accuracy   Kappa
  1.000000e-05  0.5510403  0.00000000
  3.162278e-05  0.5510403  0.00000000
  1.000000e-04  0.5510403  0.00000000
  3.162278e-04  0.5690638  0.04396078
  1.000000e-03  0.8393035  0.67351677
  3.162278e-03  0.8665084  0.73458228
  1.000000e-02  0.8649932  0.73183647
  3.162278e-02  0.8634781  0.72873998
  1.000000e-01  0.8634781  0.72873998
  3.162278e-01  0.8634781  0.72873998
  1.000000e+00  0.8634781  0.72873998
  3.162278e+00  0.8634781  0.72873998
  1.000000e+01  0.8634781  0.72873998
  3.162278e+01  0.8634781  0.72873998
  1.000000e+02  0.8634781  0.72873998

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was C = 0.003162278.
```

c. **Sometimes even if the grid of parameters in (b) includes the default value of C = 1 (used in (a)), the accuracy result will be different for this value of C. What could make that different?**

The value of C is dependent on how much error we're allowing in the dataset. Therefore, the higher the C value is, the less error we are permitting. On the other hand, the lower the C value is, the more error we're allowing. We want to set a larger margin, therefore, the smaller value of parameter C will provide me with a larger margin, which makes the value of C = 0.003162 better in this case than C=1.

PROBLEM – 3

We will take SVM further in this problem, showing how it often gets used even when the data are not suitable, by first engineering the numerical features we need. There is a Star Wars dataset in the dplyr library. Load that library and you will be able to see it (head (starwars)). There are some variables we will not use, so first remove films, vehicles, starships and name. Also remove rows with missing values.

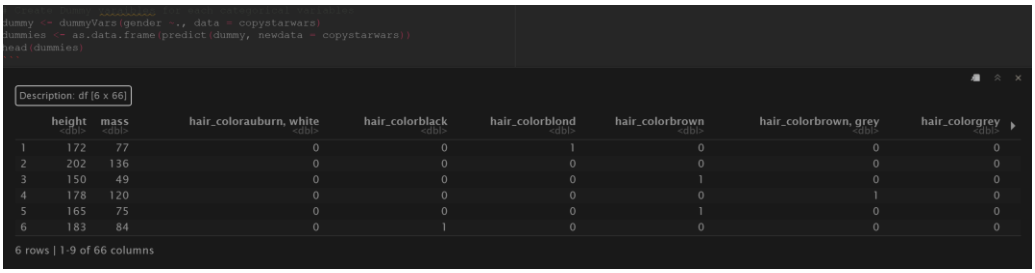
```r
# Make a copy of starwars
copystarwars <- starwars
# Remove unneeded columns
copystarwars <- copystarwars %>% select(-c("name", "films", "vehicles", "starships"))
# Completely remove NA rows
copystarwars <- na.omit(copystarwars)
```

a. **Several variables are categorical. We will use dummy variables to make it possible for SVM to use these. Leave the gender category out of the dummy variable conversion to use as a categorical for prediction. Show the resulting head.**

```r
# Find all categoricals
summary(copystarwars)
```

```
     height         mass         hair_color         skin_color
 Min.   : 88   Min.   : 20.00   Length:29          Length:29
 1st Qu.:170   1st Qu.: 75.00   Class :character   Class :character
 Median :180   Median : 79.00   Mode  :character   Mode  :character
 Mean   :178   Mean   : 77.77
 3rd Qu.:188   3rd Qu.: 83.00
 Max.   :228   Max.   :136.00
  eye_color          birth_year         sex
 Length:29          Min.   :  8.00   Length:29
 Class :character   1st Qu.: 31.00   Class :character
 Mode  :character   Median : 46.00   Mode  :character
                    Mean   : 51.29
                    3rd Qu.: 57.00
                    Max.   :200.00
    gender           homeworld           species
 Length:29          Length:29          Length:29
 Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character
```

Now, that we know that besides gender, these are our categorical variables = hair_color, skin_color, eye_color, homeworld, species, sex



Description: df [6 x 66]

	height <dbl>	mass <dbl>	hair_coloratburn, white <dbl>	hair_colorblack <dbl>	hair_colorblond <dbl>	hair_colorbrown <dbl>	hair_colorbrown, grey <dbl>	hair_colorgrey <dbl>
1	172	77	0	0	1	0	0	0
2	202	136	0	0	0	0	0	0
3	150	49	0	0	0	1	0	0
4	178	120	0	0	0	0	1	0
5	165	75	0	0	0	1	0	0
6	183	84	0	1	0	0	0	0

6 rows | 1-9 of 66 columns

b. Use SVM to predict gender and report the accuracy.

```r
# Copy the gender column to the dummies dataset
dummies$gender <- copystarwars$gender
# Run the Model
svm1 <- train(gender ~., data = dummies, method = "svmLinear")
svm1
```

```
Support Vector Machines with Linear Kernel

29 samples
66 predictors
 2 classes: 'feminine', 'masculine'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 29, 29, 29, 29, 29, 29, ...
Resampling results:

  Accuracy   Kappa
  0.8520695  0.5735759

Tuning parameter 'C' was held constant at a value of 1
```

c. **Given that we have so many variables, it makes sense to consider using PCA. Run PCA on the data and determine an appropriate number of components to use. Document how you made the decision, including any graphs you used. Create a reduced version of the data with that number of principle components. Note: make sure to remove gender from the data before running PCA because it would be cheating if PCA had access to the label you will use. Add it back in after reducing the data and show the result.**

```r
# Remove the gender
dummies$gender <- NULL
# Find and remove the near 0 variance predictors
nzv <- nearZeroVar(dummies)
# get PCA object
copystarwars.pca <- prcomp(dummies)
summary(copystarwars.pca)
```

```
Importance of components:
                          PC1     PC2      PC3     PC4     PC5     PC6     PC7     PC8     PC9    PC10    PC11    PC12    PC13    PC14    PC15    PC16
Standard deviation     41.2498 22.9472 10.69955 0.78075 0.73976 0.58724 0.57943 0.51580 0.4766 0.41954 0.38652 0.37506 0.37045 0.35266 0.34046 0.32506
Proportion of Variance  0.7252  0.2244  0.04879 0.00026 0.00023 0.00015 0.00014 0.00011 0.0001 0.00008 0.00006 0.00006 0.00006 0.00005 0.00005 0.00005
Cumulative Proportion   0.7252  0.9496  0.99838 0.99864 0.99887 0.99902 0.99916 0.99928 0.9994 0.99945 0.99951 0.99957 0.99963 0.99968 0.99973 0.99978
                          PC17    PC18    PC19    PC20    PC21    PC22    PC23    PC24    PC25    PC26    PC27    PC28      PC29
Standard deviation     0.31306 0.29384 0.27264 0.25462 0.24142 0.20136 0.18976 0.16411 0.14962 0.11579 0.0376 0.02181 3.822e-15
Proportion of Variance 0.00004 0.00004 0.00003 0.00003 0.00002 0.00002 0.00002 0.00001 0.00001 0.00001 0.0000 0.00000 0.000e+00
Cumulative Proportion  0.99982 0.99986 0.99989 0.99991 0.99994 0.99996 0.99997 0.99998 0.99999 1.00000 1.0000 1.00000 1.000e+00
```

```
# Use 4 PCs to model our data
target <- copystarwars %>% dplyr::select(gender)
# Create the components
preProc <- preProcess(dummies, method = "pca", pcaComp = 2)
copystarwars.pc <- predict(preProc, dummies)
# Put back the gender column in the dataset
copystarwars.pc$gender <- copystarwars$gender
head(copystarwars.pc)
```

Description: df [6 x 3]

	PC1 <dbl>	PC2 <dbl>	gender <chr>
1	-0.1842293	1.7888683	masculine
2	2.6287944	0.7247147	masculine
3	-3.5412999	0.3049210	feminine
4	0.4172034	1.7928384	masculine
5	-2.1186809	0.6821707	feminine
6	-0.7075658	1.4212027	masculine

6 rows

d. Use SVM to predict gender again, but this time use the data resulting from PCA. Evaluate the results with a confusion matrix and at least two partitioning methods, using grid search on the C parameter each time.

```
```{r}
# Create a new file for dummies before manipulating it
starwars_dummies <- dummies
# Move the gender column back into the dataset
starwars_dummies$gender <- copystarwars$gender
train_control = trainControl(method = "cv", number = 5)
svm_starwars <- train(gender ~., data = starwars_dummies, method = "svmLinear", trControl = train_control)
svm_starwars
```
```

```
Support Vector Machines with Linear Kernel

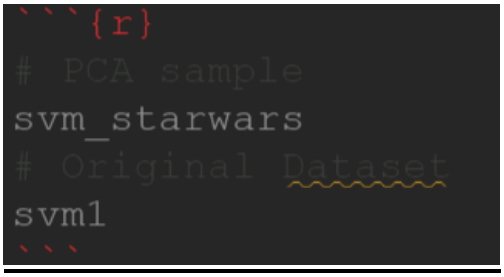
29 samples
66 predictors
 2 classes: 'feminine', 'masculine'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 24, 23, 23, 22, 24
Resampling results:

  Accuracy   Kappa
  0.8933333  0.56

Tuning parameter 'C' was held constant at a value of 1
```

e. Whether or not it has improved the accuracy, what has PCA done for the complexity of the model?




```
Support Vector Machines with Linear Kernel

29 samples
66 predictors
 2 classes: 'feminine', 'masculine'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 24, 23, 23, 22, 24
Resampling results:

  Accuracy   Kappa
  0.8933333  0.56

Tuning parameter 'C' was held constant at a value of 1
```

```
Support Vector Machines with Linear Kernel

29 samples
66 predictors
 2 classes: 'feminine', 'masculine'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 29, 29, 29, 29, 29, 29, ...
Resampling results:

  Accuracy   Kappa
  0.8520695  0.5735759

Tuning parameter 'C' was held constant at a value of 1
```

PROBLEM 4 (Bonus Problem 1)

a. Explore the variables to see if they have reasonable distributions and show your work. We will be predicting the type variable – does that mean we have a class imbalance?

```
library(caret)
data(Sacramento)
```

b. Use SVM to predict type and use grid search to get the best accuracy you can. The accuracy may be good, but look at the confusion matrix as well. Report what you find. Note that the kappa value provided with your SVM results can also help you see this. It is a measure of how well the classifier performed that takes into account the frequency of the classes.

```
> df_sac <- select(Sacramento, -c(city,zip))
> df_sac <- select(df_sac, -c('latitude','longitude'))
> svm4 <- train(type ~., data = df_sac, method = "svmLinear")
> svm4
Support Vector Machines with Linear Kernel

932 samples
  4 predictor
  3 classes: 'Condo', 'Multi_Family', 'Residential'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 932, 932, 932, 932, 932, 932, ...
Resampling results:

  Accuracy   Kappa
  0.9297427  0.1387877

Tuning parameter 'C' was held constant at a value of 1
```

c. Use SVM to predict type and use grid search to get the best accuracy you can. The accuracy may be good, but look at the confusion matrix as well. Report what you find. Note that the kappa value provided with your SVM results can also help you see this. It is a measure of how well the classifier performed that takes into account the frequency of the classes.

```
# Using the grid search
grid_sac <- expand.grid(C=10^seq(-5,2,0,0.5))
svm_grid_sac <- train(type ~., data = df_sac, method = "svmLinear", trControl = train_control, tuneGrid = grid)
svm_grid_sac
```

```
Support Vector Machines with Linear Kernel

932 samples
  4 predictor
  3 classes: 'Condo', 'Multi_Family', 'Residential'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 838, 839, 837, 840, 838, 840, ...
Resampling results across tuning parameters:

  C            Accuracy   Kappa
  1.000000e-05  0.9292432  0.0000000
  3.162278e-05  0.9292432  0.0000000
  1.000000e-04  0.9292432  0.0000000
  3.162278e-04  0.9292432  0.0000000
  1.000000e-03  0.9292432  0.0000000
  3.162278e-03  0.9292432  0.0000000
  1.000000e-02  0.9292432  0.0000000
  3.162278e-02  0.9292432  0.0000000
  1.000000e-01  0.9292432  0.0000000
  3.162278e-01  0.9292432  0.0175705
  1.000000e+00  0.9303185  0.1009121
  3.162278e+00  0.9303185  0.1009121
  1.000000e+01  0.9303185  0.1009121
  3.162278e+01  0.9303185  0.1009121
  1.000000e+02  0.9303185  0.1009121

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was C = 1.
```

d. Return to (b) and try at least one other way to try to improve the data before running SVM again, as in (c).

```
train_control = trainControl(method = "cv", number = 10)
preproc = c("center", "scale")
svm_fold <- train(type ~., data = df_sac, method = "svmLinear", trControl = train_control, preProcess = preproc)
svm_fold
```

```
Support Vector Machines with Linear Kernel

932 samples
  4 predictor
  3 classes: 'Condo', 'Multi_Family', 'Residential'

Pre-processing: centered (4), scaled (4)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 839, 838, 839, 839, 838, 839, ...
Resampling results:

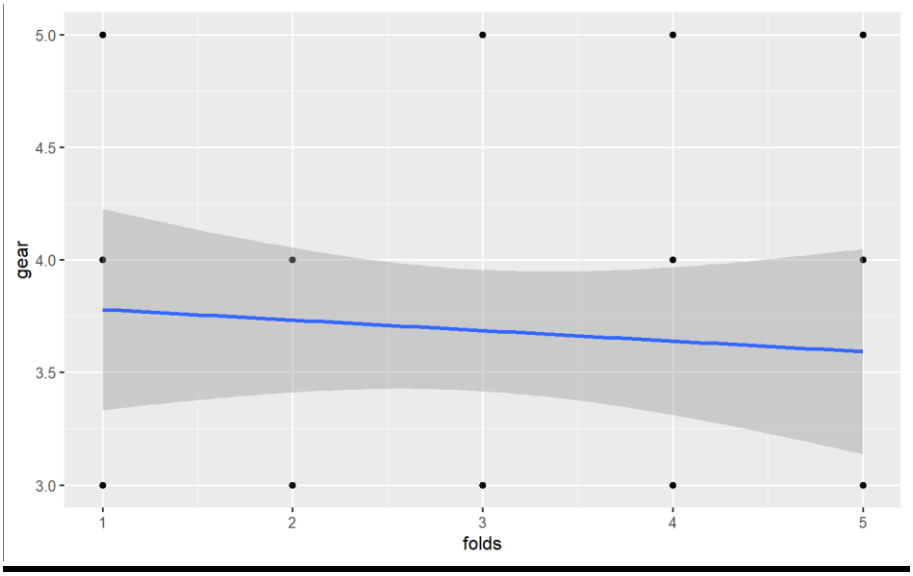
  Accuracy   Kappa
  0.9281385  0.02095555

Tuning parameter 'C' was held constant at a value of 1
```

PROBLEM 5 (Bonus Problem 2)

To understand just how much different subsets can differ, create a 5 fold partitioning of the cars data included in R (mtcars) and visualize the distribution of the gears variable across the folds. Rather than use the fancy trainControl methods for making the folds, create them directly so you actually can keep track of which data points are in which fold. This is not covered in the tutorial, but it is quick. Here is code to create 5 folds and a variable in the data frame that contains the fold index of each point. Use that resulting data frame to create your visualization.

```
```{r}
# Make a copy to modify
mycars <- mtcars
View(mycars)
# Initialize new variable to hold fold indices
mycars <- mtcars
mycars$folds = 0
# This loop sets all the rows in a given fold to have that fold's index in the folds variable.
# Take a look at the result and use it to create the visualization
flds = createFolds(1:nrow(mycars), k=5, list=TRUE)
for (i in 1:5)
{
  mycars$folds[flds[[i]]] = i
}
# Implementing a graph plot for 5 folds
ggplot(mycars,aes(folds, gear)) + geom_point() + geom_smooth(method = lm)
```
```



By the 5th fold, number of gears were 3.5 and the regression line was mean of point in each fold and grey area is confidence band.