DASC 5420 - Assignment 2

Thai Pham - T00727094

2024-04-06

```
# install and call required library
pkg_list = c("dplyr","tidyverse","ISLR","ISLR2", "caret","ModelMetrics","corrplot", 'ggpubr', 'glmnet',
'GGally', 'class', 'boot', 'pROC')
# Install packages if needed
for (pkg in pkg_list)
{# Try loading the library.
if ( ! library(pkg, logical.return=TRUE, character.only=TRUE) )
{
# If the library cannot be loaded, install it; then load.
install.packages(pkg)
library(pkg, character.only=TRUE)
}
}
```

```
## Warning: package 'dplyr' was built under R version 4.3.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
## Warning: package 'tidyverse' was built under R version 4.3.2
```

```
## Warning: package 'ggplot2' was built under R version 4.3.2
```

```
## Warning: package 'tibble' was built under R version 4.3.2
```

```
## Warning: package 'tidyr' was built under R version 4.3.2
```

```
## Warning: package 'readr' was built under R version 4.3.2
```

```
## Warning: package 'purrr' was built under R version 4.3.2
```

```
## Warning: package 'stringr' was built under R version 4.3.2
```

```
## Warning: package 'forcats' was built under R version 4.3.2
```

```
## Warning: package 'lubridate' was built under R version 4.3.2
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v forcats   1.0.0     v readr     2.1.4
## v ggplot2   3.4.4     v stringr   1.5.1
## v lubridate 1.9.3     v tibble    3.2.1
```

```
## v purrr     1.0.2     v tidyr     1.3.0
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
## Warning: package 'ISLR' was built under R version 4.3.2
## Warning: package 'ISLR2' was built under R version 4.3.2
##
## Attaching package: 'ISLR2'
##
## The following objects are masked from 'package:ISLR':
##
##     Auto, Credit
## Warning: package 'caret' was built under R version 4.3.2
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
## Warning: package 'ModelMetrics' was built under R version 4.3.2
##
## Attaching package: 'ModelMetrics'
##
## The following objects are masked from 'package:caret':
##
##     confusionMatrix, precision, recall, sensitivity, specificity
##
## The following object is masked from 'package:base':
##
##     kappa
## Warning: package 'corrplot' was built under R version 4.3.3
## corrplot 0.92 loaded
## Warning: package 'ggpubr' was built under R version 4.3.3
## Warning: package 'glmnet' was built under R version 4.3.3
## Loading required package: Matrix
## Warning: package 'Matrix' was built under R version 4.3.3
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-8
```

```
## Warning: package 'GGally' was built under R version 4.3.3
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
##
## Attaching package: 'boot'
##
## The following object is masked from 'package:lattice':
##
##     melanoma
## Warning: package 'pROC' was built under R version 4.3.2
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following object is masked from 'package:ModelMetrics':
##
##     auc
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

Question 1: German Credit Risk

a) Write the goal of this data analysis. Do some exploratory data analysis (EDA) and data transformations.

The objective of this analysis is to develop a logistic classifier to predict applicants' creditability, determining whether they are likely to repay a loan based on their attributes and credit history. To achieve this goal, a logistic classifier will be trained using a comprehensive set of 20 predictors, including variables such as Account Balance, Payment Status of Previous Credit, Value Savings/Stocks, Instalment Percentage, and Guarantors, among others. These predictors will serve as features for the model, helping to identify patterns and relationships that can accurately classify individuals into categories of creditability (e.g., "good" or "bad" credit risk).

By leveraging this logistic classifier, financial institutions can streamline their lending processes, mitigate potential losses, and optimize their overall risk management strategies.

EXPLORATORY DATA ANALYSIS (EDA)

```
#Load the data
data<-read.csv("german_credit.csv")
glimpse(data)
```

```
## Rows: 1,000
## Columns: 21
## $ Creditability                     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ Account.Balance                   <int> 1, 1, 2, 1, 1, 1, 1, 1, 4, 2, 1, 1, ~
## $ Duration.of.Credit..month.        <int> 18, 9, 12, 12, 12, 10, 8, 6, 18, 24,~
## $ Payment.Status.of.Previous.Credit <int> 4, 4, 2, 4, 4, 4, 4, 4, 4, 2, 4, 4, ~
## $ Purpose                           <int> 2, 0, 9, 0, 0, 0, 0, 0, 3, 3, 0, 1, ~
## $ Credit.Amount                     <int> 1049, 2799, 841, 2122, 2171, 2241, 3~
## $ Value.Savings.Stocks              <int> 1, 1, 2, 1, 1, 1, 1, 1, 1, 3, 1, 2, ~
```

```
## $ Length.of.current.employment      <int> 2, 3, 4, 3, 3, 2, 4, 2, 1, 1, 3, 4, ~
## $ Instalment.per.cent               <int> 4, 2, 2, 3, 4, 1, 1, 2, 4, 1, 2, 1, ~
## $ Sex...Marital.Status              <int> 2, 3, 2, 3, 3, 3, 3, 3, 2, 2, 3, 4, ~
## $ Guarantors                        <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ Duration.in.Current.address       <int> 4, 2, 4, 2, 4, 3, 4, 4, 4, 4, 2, 4, ~
## $ Most.valuable.available.asset     <int> 2, 1, 1, 1, 2, 1, 1, 1, 3, 4, 1, 3, ~
## $ Age..years.                       <int> 21, 36, 23, 39, 38, 48, 39, 40, 65, ~
## $ Concurrent.Credits                <int> 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, ~
## $ Type.of.apartment                 <int> 1, 1, 1, 1, 2, 1, 2, 2, 2, 1, 1, 1, ~
## $ No.of.Credits.at.this.Bank        <int> 1, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, ~
## $ Occupation                        <int> 3, 3, 2, 2, 2, 2, 2, 2, 1, 1, 3, 3, ~
## $ No.of.dependents                  <int> 1, 2, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, ~
## $ Telephone                         <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ Foreign.Worker                    <int> 1, 1, 1, 2, 2, 2, 2, 2, 1, 1, 1, 1, ~
```

```r
# Statistic summary
summary(data)
```

```
##  Creditability Account.Balance Duration.of.Credit..month.
##  Min.   :0.0   Min.   :1.000   Min.   : 4.0
##  1st Qu.:0.0   1st Qu.:1.000   1st Qu.:12.0
##  Median :1.0   Median :2.000   Median :18.0
##  Mean   :0.7   Mean   :2.577   Mean   :20.9
##  3rd Qu.:1.0   3rd Qu.:4.000   3rd Qu.:24.0
##  Max.   :1.0   Max.   :4.000   Max.   :72.0
##  Payment.Status.of.Previous.Credit    Purpose       Credit.Amount
##  Min.   :0.000                     Min.   : 0.000   Min.   :  250
##  1st Qu.:2.000                     1st Qu.: 1.000   1st Qu.: 1366
##  Median :2.000                     Median : 2.000   Median : 2320
##  Mean   :2.545                     Mean   : 2.828   Mean   : 3271
##  3rd Qu.:4.000                     3rd Qu.: 3.000   3rd Qu.: 3972
##  Max.   :4.000                     Max.   :10.000   Max.   :18424
##  Value.Savings.Stocks Length.of.current.employment Instalment.per.cent
##  Min.   :1.000        Min.   :1.000                Min.   :1.000
##  1st Qu.:1.000        1st Qu.:3.000                1st Qu.:2.000
##  Median :1.000        Median :3.000                Median :3.000
##  Mean   :2.105        Mean   :3.384                Mean   :2.973
##  3rd Qu.:3.000        3rd Qu.:5.000                3rd Qu.:4.000
##  Max.   :5.000        Max.   :5.000                Max.   :4.000
##  Sex...Marital.Status   Guarantors    Duration.in.Current.address
##  Min.   :1.000        Min.   :1.000   Min.   :1.000
##  1st Qu.:2.000        1st Qu.:1.000   1st Qu.:2.000
##  Median :3.000        Median :1.000   Median :3.000
##  Mean   :2.682        Mean   :1.145   Mean   :2.845
##  3rd Qu.:3.000        3rd Qu.:1.000   3rd Qu.:4.000
##  Max.   :4.000        Max.   :3.000   Max.   :4.000
##  Most.valuable.available.asset  Age..years.    Concurrent.Credits
##  Min.   :1.000                 Min.   :19.00   Min.   :1.000
##  1st Qu.:1.000                 1st Qu.:27.00   1st Qu.:3.000
##  Median :2.000                 Median :33.00   Median :3.000
##  Mean   :2.358                 Mean   :35.54   Mean   :2.675
##  3rd Qu.:3.000                 3rd Qu.:42.00   3rd Qu.:3.000
##  Max.   :4.000                 Max.   :75.00   Max.   :3.000
##  Type.of.apartment No.of.Credits.at.this.Bank   Occupation    No.of.dependents
##  Min.   :1.000     Min.   :1.000              Min.   :1.000   Min.   :1.000
```

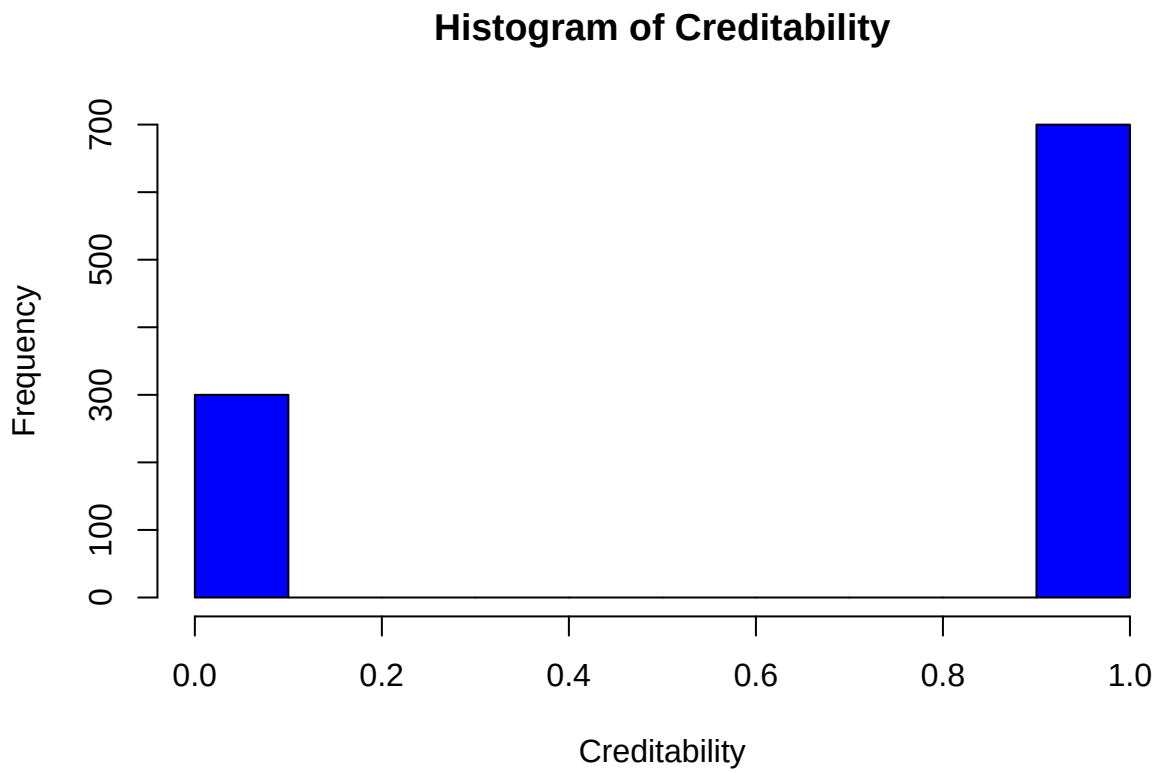
```
##  1st Qu.:2.000     1st Qu.:1.000                1st Qu.:3.000   1st Qu.:1.000
##  Median :2.000     Median :1.000                Median :3.000   Median :1.000
##  Mean   :1.928     Mean   :1.407                Mean   :2.904   Mean   :1.155
##  3rd Qu.:2.000     3rd Qu.:2.000                3rd Qu.:3.000   3rd Qu.:1.000
##  Max.   :3.000     Max.   :4.000                Max.   :4.000   Max.   :2.000
##    Telephone     Foreign.Worker
##  Min.   :1.000   Min.   :1.000
##  1st Qu.:1.000   1st Qu.:1.000
##  Median :1.000   Median :1.000
##  Mean   :1.404   Mean   :1.037
##  3rd Qu.:2.000   3rd Qu.:1.000
##  Max.   :2.000   Max.   :2.000
```

```r
# Check for missing values
sum(is.na(data))
```

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

The dataset consists of 1000 numeric observations with 21 columns. Each row represents a customer sample, and each column represents an attribute of that sample. Since this dataset does not have missing value, no missing values imputation is required in this data transformation step.

```r
# Histogram plot for a numerical variable (e.g., Creditability)
hist(data$Creditability, main = "Histogram of Creditability", xlab = "Creditability",col="blue")
```




```r
# Calculate correlation matrix
correlation_matrix <- cor(data)
correlation_matrix
```

```
##                                  Creditability Account.Balance
## Creditability                      1.000000000     0.350847483
## Account.Balance                    0.350847483     1.000000000
## Duration.of.Credit..month.        -0.214926665    -0.072013088
## Payment.Status.of.Previous.Credit  0.228784733     0.192190688
## Purpose                           -0.017978870     0.028782569
## Credit.Amount                     -0.154740146    -0.042695127
## Value.Savings.Stocks               0.178942736     0.222866860
## Length.of.current.employment       0.116002036     0.106338752
## Instalment.per.cent               -0.072403937    -0.005279856
## Sex...Marital.Status               0.088184301     0.043261280
## Guarantors                         0.025136768    -0.127736563
## Duration.in.Current.address       -0.002967159    -0.042233689
## Most.valuable.available.asset     -0.142611973    -0.032260126
## Age..years.                        0.091271949     0.058630740
## Concurrent.Credits                 0.109844099     0.068273870
## Type.of.apartment                  0.018118912     0.023335309
## No.of.Credits.at.this.Bank         0.045732489     0.076005137
## Occupation                        -0.032735001     0.040663061
## No.of.dependents                   0.003014853    -0.014145427
## Telephone                          0.036466190     0.066295834
## Foreign.Worker                     0.082079499    -0.035186993
##                                  Duration.of.Credit..month.
## Creditability                                    -0.21492667
## Account.Balance                                  -0.07201309
## Duration.of.Credit..month.                        1.00000000
## Payment.Status.of.Previous.Credit                -0.07718647
## Purpose                                           0.14749187
## Credit.Amount                                     0.62498846
## Value.Savings.Stocks                              0.04766092
## Length.of.current.employment                      0.05738103
## Instalment.per.cent                               0.07474882
## Sex...Marital.Status                              0.01478933
## Guarantors                                       -0.02448995
## Duration.in.Current.address                       0.03406720
## Most.valuable.available.asset                     0.30397125
## Age..years.                                      -0.03754986
## Concurrent.Credits                               -0.06288379
## Type.of.apartment                                 0.15312556
## No.of.Credits.at.this.Bank                       -0.01128360
## Occupation                                        0.21090973
## No.of.dependents                                 -0.02383448
## Telephone                                         0.16471821
## Foreign.Worker                                   -0.13467996
##                                  Payment.Status.of.Previous.Credit
## Creditability                                          0.22878473
## Account.Balance                                        0.19219069
## Duration.of.Credit..month.                            -0.07718647
## Payment.Status.of.Previous.Credit                      1.00000000
## Purpose                                               -0.09033589
## Credit.Amount                                         -0.05991485
## Value.Savings.Stocks                                   0.03905788
## Length.of.current.employment                           0.13822522
## Instalment.per.cent                                    0.04437459
```

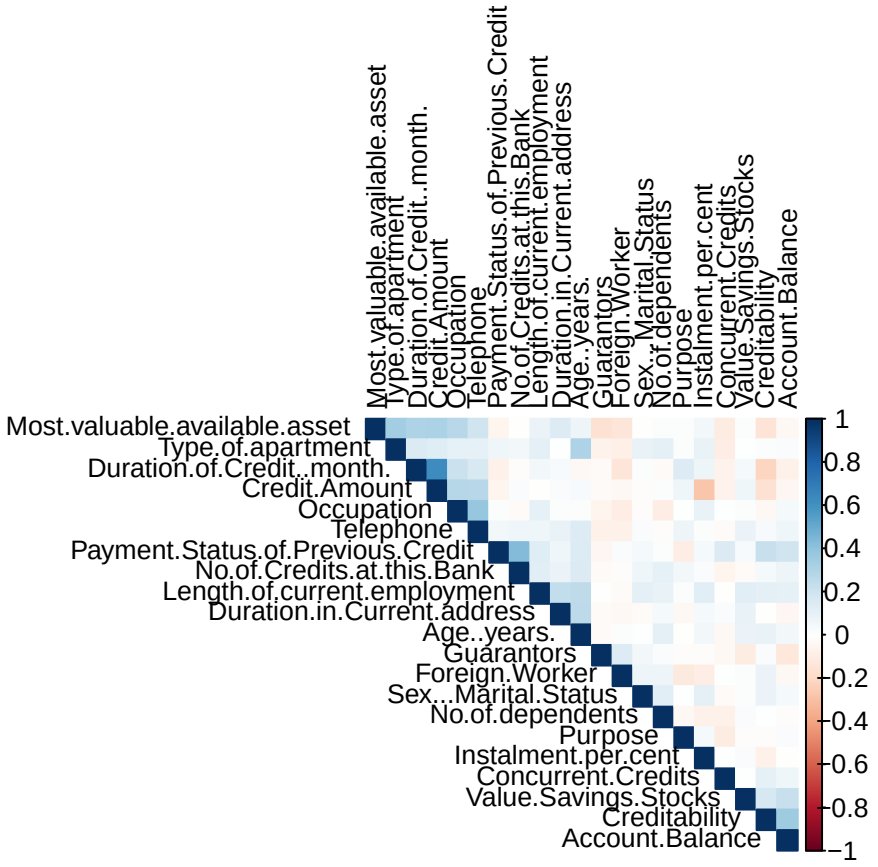
```
## Sex...Marital.Status                                0.04217088
## Guarantors                                         -0.04067553
## Duration.in.Current.address                         0.06319797
## Most.valuable.available.asset                      -0.05377676
## Age..years.                                         0.14633747
## Concurrent.Credits                                  0.15995707
## Type.of.apartment                                   0.06142792
## No.of.Credits.at.this.Bank                          0.43706577
## Occupation                                          0.01035018
## No.of.dependents                                    0.01154955
## Telephone                                           0.05237019
## Foreign.Worker                                      0.02855405
##                                       Purpose Credit.Amount
## Creditability                     -0.0179788700  -0.154740146
## Account.Balance                    0.0287825694  -0.042695127
## Duration.of.Credit..month.         0.1474918712   0.624988461
## Payment.Status.of.Previous.Credit -0.0903358941  -0.059914852
## Purpose                            1.0000000000   0.068480054
## Credit.Amount                      0.0684800535   1.000000000
## Value.Savings.Stocks              -0.0186844687   0.064632168
## Length.of.current.employment       0.0160130053  -0.008376109
## Instalment.per.cent                0.0483689475  -0.271322281
## Sex...Marital.Status               0.0001565929  -0.016094338
## Guarantors                        -0.0176067538  -0.027830917
## Duration.in.Current.address       -0.0382213445   0.028916676
## Most.valuable.available.asset      0.0109663534   0.311602093
## Age..years.                       -0.0008923856   0.032272677
## Concurrent.Credits                -0.1002303932  -0.069392010
## Type.of.apartment                  0.0134946967   0.133023634
## No.of.Credits.at.this.Bank         0.0549353555   0.020785277
## Occupation                         0.0080847757   0.285393073
## No.of.dependents                  -0.0325768744   0.017143582
## Telephone                          0.0783705414   0.277000181
## Foreign.Worker                    -0.1132436689  -0.030661601
##                                   Value.Savings.Stocks
## Creditability                            0.178942736
## Account.Balance                          0.222866860
## Duration.of.Credit..month.               0.047660924
## Payment.Status.of.Previous.Credit        0.039057881
## Purpose                                 -0.018684469
## Credit.Amount                            0.064632168
## Value.Savings.Stocks                     1.000000000
## Length.of.current.employment             0.120949514
## Instalment.per.cent                      0.021992529
## Sex...Marital.Status                     0.017348689
## Guarantors                              -0.105068513
## Duration.in.Current.address              0.091424109
## Most.valuable.available.asset            0.018948001
## Age..years.                              0.083433512
## Concurrent.Credits                       0.001907967
## Type.of.apartment                        0.006643819
## No.of.Credits.at.this.Bank              -0.021644133
## Occupation                               0.011708920
## No.of.dependents                         0.027513789
```

```
## Telephone                                    0.087208402
## Foreign.Worker                               0.010449560
##                                  Length.of.current.employment
## Creditability                                    0.116002036
## Account.Balance                                  0.106338752
## Duration.of.Credit..month.                       0.057381027
## Payment.Status.of.Previous.Credit                0.138225216
## Purpose                                          0.016013005
## Credit.Amount                                   -0.008376109
## Value.Savings.Stocks                             0.120949514
## Length.of.current.employment                     1.000000000
## Instalment.per.cent                              0.126161307
## Sex...Marital.Status                             0.111278288
## Guarantors                                      -0.008116008
## Duration.in.Current.address                      0.245080745
## Most.valuable.available.asset                    0.087187468
## Age..years.                                      0.259116153
## Concurrent.Credits                              -0.007279305
## Type.of.apartment                                0.115077459
## No.of.Credits.at.this.Bank                       0.125790651
## Occupation                                       0.101224870
## No.of.dependents                                 0.097192004
## Telephone                                        0.060518081
## Foreign.Worker                                  -0.022845318
##                                  Instalment.per.cent Sex...Marital.Status
## Creditability                          -0.072403937         0.0881843005
## Account.Balance                        -0.005279856         0.0432612798
## Duration.of.Credit..month.              0.074748816         0.0147893320
## Payment.Status.of.Previous.Credit       0.044374587         0.0421708809
## Purpose                                 0.048368947         0.0001565929
## Credit.Amount                          -0.271322281        -0.0160943379
## Value.Savings.Stocks                    0.021992529         0.0173486885
## Length.of.current.employment            0.126161307         0.1112782879
## Instalment.per.cent                     1.000000000         0.1193079016
## Sex...Marital.Status                    0.119307902         1.0000000000
## Guarantors                             -0.011397639         0.0506338891
## Duration.in.Current.address             0.049302371        -0.0272690320
## Most.valuable.available.asset           0.053391413        -0.0069404770
## Age..years.                             0.057270750         0.0051498271
## Concurrent.Credits                      0.007893967        -0.0267469446
## Type.of.apartment                       0.091228577         0.0989338012
## No.of.Credits.at.this.Bank              0.021668743         0.0646718729
## Occupation                              0.097755393        -0.0119563566
## No.of.dependents                       -0.071206943         0.1221648450
## Telephone                               0.014412880         0.0272748748
## Foreign.Worker                         -0.094762307         0.0731034045
##                                  Guarantors Duration.in.Current.address
## Creditability                    0.025136768                -0.002967159
## Account.Balance                 -0.127736563                -0.042233689
## Duration.of.Credit..month.      -0.024489950                 0.034067202
## Payment.Status.of.Previous.Credit -0.040675530               0.063197969
## Purpose                         -0.017606754                -0.038221345
## Credit.Amount                   -0.027830917                 0.028916676
## Value.Savings.Stocks            -0.105068513                 0.091424109
```

```
## Length.of.current.employment      -0.008116008                0.245080745
## Instalment.per.cent               -0.011397639                0.049302371
## Sex...Marital.Status               0.050633889               -0.027269032
## Guarantors                         1.000000000               -0.025677506
## Duration.in.Current.address       -0.025677506                1.000000000
## Most.valuable.available.asset     -0.155450138                0.147231116
## Age..years.                       -0.029825663                0.265626478
## Concurrent.Credits                -0.038235049                0.022654074
## Type.of.apartment                 -0.065449419                0.009989899
## No.of.Credits.at.this.Bank        -0.025446800                0.089625233
## Occupation                        -0.057962986                0.012654644
## No.of.dependents                   0.020399584                0.042643426
## Telephone                         -0.075034578                0.095359367
## Foreign.Worker                     0.140190191               -0.039690633
##                                  Most.valuable.available.asset   Age..years.
## Creditability                                      -0.142611973  0.0912719488
## Account.Balance                                    -0.032260126  0.0586307400
## Duration.of.Credit..month.                          0.303971245 -0.0375498629
## Payment.Status.of.Previous.Credit                  -0.053776760  0.1463374687
## Purpose                                             0.010966353 -0.0008923856
## Credit.Amount                                       0.311602093  0.0322726775
## Value.Savings.Stocks                                0.018948001  0.0834335122
## Length.of.current.employment                        0.087187468  0.2591161527
## Instalment.per.cent                                 0.053391413  0.0572707503
## Sex...Marital.Status                               -0.006940477  0.0051498271
## Guarantors                                         -0.155450138 -0.0298256629
## Duration.in.Current.address                         0.147231116  0.2656264783
## Most.valuable.available.asset                       1.000000000  0.0745514538
## Age..years.                                         0.074551454  1.0000000000
## Concurrent.Credits                                 -0.107593324 -0.0304719341
## Type.of.apartment                                   0.342968580  0.3033464109
## No.of.Credits.at.this.Bank                         -0.007765020  0.1507176513
## Occupation                                          0.276149365  0.0153830309
## No.of.dependents                                    0.011871999  0.1185891829
## Telephone                                           0.196801583  0.1435058472
## Foreign.Worker                                     -0.132461796  0.0139811872
##                                  Concurrent.Credits Type.of.apartment
## Creditability                         0.109844099       0.018118912
## Account.Balance                       0.068273870       0.023335309
## Duration.of.Credit..month.           -0.062883787       0.153125556
## Payment.Status.of.Previous.Credit     0.159957065       0.061427919
## Purpose                              -0.100230393       0.013494697
## Credit.Amount                        -0.069392010       0.133023634
## Value.Savings.Stocks                  0.001907967       0.006643819
## Length.of.current.employment         -0.007279305       0.115077459
## Instalment.per.cent                   0.007893967       0.091228577
## Sex...Marital.Status                 -0.026746945       0.098933801
## Guarantors                           -0.038235049      -0.065449419
## Duration.in.Current.address           0.022654074       0.009989899
## Most.valuable.available.asset        -0.107593324       0.342968580
## Age..years.                          -0.030471934       0.303346411
## Concurrent.Credits                    1.000000000      -0.097397651
## Type.of.apartment                    -0.097397651       1.000000000
## No.of.Credits.at.this.Bank           -0.055809873       0.050019938
```

```
## Occupation                                0.006077318       0.104243222
## No.of.dependents                         -0.076890642       0.115548584
## Telephone                                -0.025139895       0.100326589
## Foreign.Worker                            0.007699595      -0.083336024
##                                 No.of.Credits.at.this.Bank   Occupation
## Creditability                                  0.04573249 -0.032735001
## Account.Balance                                0.07600514  0.040663061
## Duration.of.Credit..month.                    -0.01128360  0.210909735
## Payment.Status.of.Previous.Credit              0.43706577  0.010350179
## Purpose                                        0.05493536  0.008084776
## Credit.Amount                                  0.02078528  0.285393073
## Value.Savings.Stocks                          -0.02164413  0.011708920
## Length.of.current.employment                   0.12579065  0.101224870
## Instalment.per.cent                            0.02166874  0.097755393
## Sex...Marital.Status                           0.06467187 -0.011956357
## Guarantors                                    -0.02544680 -0.057962986
## Duration.in.Current.address                    0.08962523  0.012654644
## Most.valuable.available.asset                 -0.00776502  0.276149365
## Age..years.                                    0.15071765  0.015383031
## Concurrent.Credits                            -0.05580987  0.006077318
## Type.of.apartment                              0.05001994  0.104243222
## No.of.Credits.at.this.Bank                     1.00000000 -0.026321269
## Occupation                                    -0.02632127  1.000000000
## No.of.dependents                               0.10966670 -0.093559276
## Telephone                                      0.06555321  0.383022159
## Foreign.Worker                                -0.01889259 -0.092834959
##                                 No.of.dependents   Telephone Foreign.Worker
## Creditability                        0.003014853  0.03646619    0.082079499
## Account.Balance                     -0.014145427  0.06629583   -0.035186993
## Duration.of.Credit..month.          -0.023834475  0.16471821   -0.134679963
## Payment.Status.of.Previous.Credit    0.011549554  0.05237019    0.028554048
## Purpose                             -0.032576874  0.07837054   -0.113243669
## Credit.Amount                        0.017143582  0.27700018   -0.030661601
## Value.Savings.Stocks                 0.027513789  0.08720840    0.010449560
## Length.of.current.employment         0.097192004  0.06051808   -0.022845318
## Instalment.per.cent                 -0.071206943  0.01441288   -0.094762307
## Sex...Marital.Status                 0.122164845  0.02727487    0.073103405
## Guarantors                           0.020399584 -0.07503458    0.140190191
## Duration.in.Current.address          0.042643426  0.09535937   -0.039690633
## Most.valuable.available.asset        0.011871999  0.19680158   -0.132461796
## Age..years.                          0.118589183  0.14350585    0.013981187
## Concurrent.Credits                  -0.076890642 -0.02513990    0.007699595
## Type.of.apartment                    0.115548584  0.10032659   -0.083336024
## No.of.Credits.at.this.Bank           0.109666700  0.06555321   -0.018892588
## Occupation                          -0.093559276  0.38302216   -0.092834959
## No.of.dependents                     1.000000000 -0.01475344    0.077070853
## Telephone                           -0.014753439  1.00000000   -0.075012215
## Foreign.Worker                       0.077070853 -0.07501222    1.000000000
```

```r
# Plot heatmap for correlation matrix
corrplot(correlation_matrix, method = "color", type = "upper",
         addrect = 3, order = "hclust", tl.cex = 0.8, tl.col = "black")
```



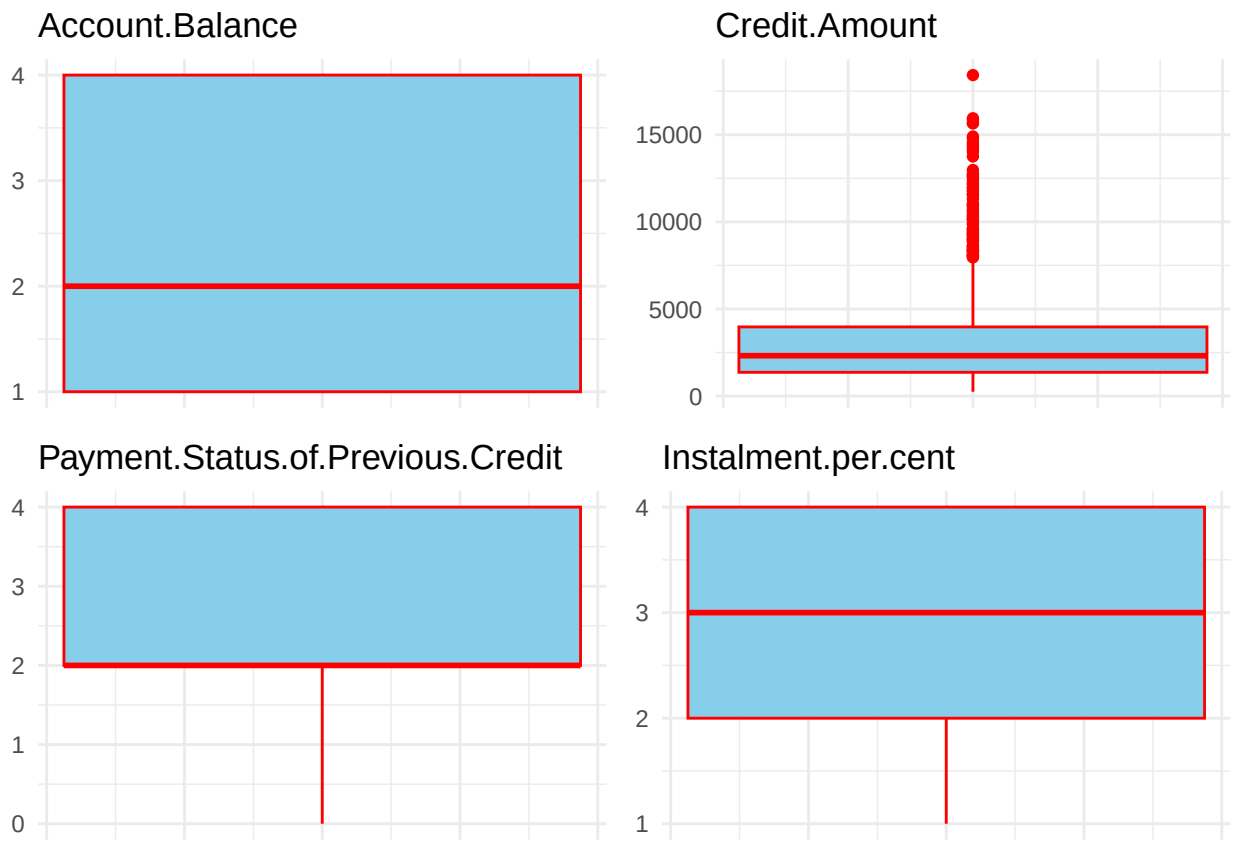
The histogram depicting creditability reveals a slight imbalance, with approximately 300 instances classified as 0 and 700 instances as 1. The dataset highlights moderate positive correlations between creditability and various factors such as account balance, credit amount, duration of credit (in months), and payment status of previous credit, as well as the number of credits at this bank. Conversely, there exists a moderate negative correlation between installment percent and credit amount. These findings suggest that customers exhibiting higher account balances, credit amounts, longer credit durations, and positive payment histories are more likely to demonstrate good creditability. Conversely, individuals with higher installment percentages relative to their credit amount may have lower creditworthiness.

```
# Select important variables
important_vars <- c("Account.Balance", "Credit.Amount",
                    "Payment.Status.of.Previous.Credit", "Instalment.per.cent")

# Create a list to store individual boxplot plots
plots <- list()

# Loop through each important variable and create boxplot plots
for (var in important_vars) {
  plot <- ggplot(data, aes_string(y = var)) +
    geom_boxplot(fill = "skyblue", color = "red") +
    labs(title = paste(var),
         y = "") +  # Remove y-axis label
    theme_minimal() +
    theme(axis.text.x = element_blank(),  # Remove x-axis labels
          axis.title.x = element_blank(), # Remove x-axis title
          axis.title.y = element_blank()) # Remove y-axis title
```

```
  plots[[var]] <- plot
}
```

```
## Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
## i Please use tidy evaluation idioms with `aes()`.
## i See also `vignette("ggplot2-in-packages")` for more information.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
# Arrange the plots into a grid
ggarrange(plotlist = plots, ncol = 2, nrow = 2)
```



The boxplots illustrate that the majority of predictors exhibit no outliers, except for the "Credit Amount" variable, which displays the presence of outliers. Hence, it is advisable to eliminate outliers in the "Credit Amount" variable before constructing machine learning models. This preprocessing step is crucial as it can enhance the accuracy and performance of the models.

DATA TRANSFORMATION

```
# Remove outliers of feature Credit.Amount
z_scores <- abs(scale(data$Credit.Amount))
data <- data[z_scores < 3, ]

# Standardize Credit.Amount
data$Credit.Amount <- scale(data$Credit.Amount)
credit<-data
```

After applying the outlier removal technique using the z-scored method and standardizing the "Credit.Amount" variable, we have created a new dataset named "credit." Removing outliers helps cleanse the data by eliminating potential data points that may skew the distribution and affect analytical outcomes. Subsequently, standardizing the "Credit.Amount" variable scales all values to the same proportion, reducing the influence of measurement units. This makes the data easier to compare and analyze.

Lastly, reintegrating the cleaned and standardized dataset back under the name "credit" facilitates further analysis and model building. This allows for seamless utilization of the dataset for subsequent analyses, such as credit risk prediction based on other variables.

b) Build a classifier to predict the creditability of a consumer using logistic regression

```
# Split the dataset into training and testing sets (80% train, 20% test)
set.seed(5420)
trainIdx <- createDataPartition(credit$Creditability, p = 0.8, list = FALSE,times = 1)

# Create training and testing sets
credit.train <- credit[trainIdx, ]
credit.test <- credit[-trainIdx, ]

# Train logistic regression model
full_model <- glm(Creditability ~ ., data = credit.train, family = binomial)
summary(full_model)
```

```
##
## Call:
## glm(formula = Creditability ~ ., family = binomial, data = credit.train)
##
## Coefficients:
##                                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)                     -5.528243   1.287775  -4.293 1.76e-05 ***
## Account.Balance                  0.680124   0.084923   8.009 1.16e-15 ***
## Duration.of.Credit..month.      -0.034446   0.011266  -3.057 0.002232 **
## Payment.Status.of.Previous.Credit  0.482529   0.106280   4.540 5.62e-06 ***
## Purpose                          0.034818   0.037036   0.940 0.347157
## Credit.Amount                   -0.157081   0.142105  -1.105 0.268991
## Value.Savings.Stocks             0.239527   0.070169   3.414 0.000641 ***
## Length.of.current.employment     0.224082   0.084972   2.637 0.008361 **
## Instalment.per.cent             -0.309957   0.101524  -3.053 0.002265 **
## Sex...Marital.Status             0.276991   0.135637   2.042 0.041137 *
## Guarantors                       0.299366   0.214647   1.395 0.163110
## Duration.in.Current.address     -0.021992   0.093176  -0.236 0.813408
## Most.valuable.available.asset   -0.149823   0.108740  -1.378 0.168262
## Age..years.                      0.003974   0.009642   0.412 0.680269
## Concurrent.Credits               0.277123   0.136465   2.031 0.042283 *
## Type.of.apartment                0.408386   0.203284   2.009 0.044544 *
## No.of.Credits.at.this.Bank      -0.352833   0.186130  -1.896 0.058009 .
## Occupation                      -0.018561   0.164555  -0.113 0.910193
## No.of.dependents                 0.333906   0.293669   1.137 0.255533
## Telephone                        0.328537   0.219118   1.499 0.133780
## Foreign.Worker                   1.444701   0.794487   1.818 0.069002 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 924.25  on 779  degrees of freedom
## Residual deviance: 681.29  on 759  degrees of freedom
## AIC: 723.29
##
## Number of Fisher Scoring iterations: 5
```

This is a summary of the full model that utilizes all predictors. However, given that some predictors lack statistical significance, I intend to construct an alternative model employing only those variables deemed statistically significant in this analysis.

```
# Fit a logistic regression model with selected variables
new_model <- glm(Creditability ~ Account.Balance + Duration.of.Credit..month. + Payment.Status.of.Previ

# Summary of the model
summary(new_model)
```

```
##
## Call:
## glm(formula = Creditability ~ Account.Balance + Duration.of.Credit..month. +
##     Payment.Status.of.Previous.Credit + Value.Savings.Stocks +
##     Length.of.current.employment + Instalment.per.cent + Sex...Marital.Status +
##     Concurrent.Credits + Type.of.apartment + Foreign.Worker,
##     family = binomial, data = credit.train)
##
## Coefficients:
##                                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)                      -4.88427    1.07643  -4.537 5.69e-06 ***
## Account.Balance                   0.67392    0.08300   8.119 4.70e-16 ***
## Duration.of.Credit..month.       -0.04381    0.00806  -5.435 5.47e-08 ***
## Payment.Status.of.Previous.Credit  0.38835    0.09404   4.130 3.63e-05 ***
## Value.Savings.Stocks               0.23427    0.06833   3.428 0.000607 ***
## Length.of.current.employment       0.22172    0.08074   2.746 0.006028 **
## Instalment.per.cent               -0.26818    0.08813  -3.043 0.002341 **
## Sex...Marital.Status               0.30071    0.13124   2.291 0.021944 *
## Concurrent.Credits                 0.28126    0.13068   2.152 0.031371 *
## Type.of.apartment                  0.37941    0.18218   2.083 0.037290 *
## Foreign.Worker                     1.55615    0.78192   1.990 0.046571 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 924.25  on 779  degrees of freedom
## Residual deviance: 695.24  on 769  degrees of freedom
## AIC: 717.24
##
## Number of Fisher Scoring iterations: 5
```

This presents the summary of the model utilizing solely the statistically significant variables extracted from the comprehensive model.

c) Print out the algorithm performance and interpret the results

```
# Make predictions on the test set for full_model
predictions_full <- predict(full_model, credit.test,type = "response")

# Convert predictions to binary values
predictions_full <- ifelse(predictions_full > 0.5, 1, 0)

# Evaluate the model using confusion matrix
conf_matrix_full <- table(Predicted = predictions_full, Actual = credit.test$Creditability)
conf_matrix_full
```

```
##          Actual
## Predicted   0   1
##         0  22  20
##         1  43 110
```

```
# Calculate ROC curve
roc_full <- roc(credit.test$Creditability, predictions_full)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
# Calculate AUC-ROC
auc_full <- auc(roc_full)
cat("AUC-ROC for Full Model:", auc_full)
```

```
## AUC-ROC for Full Model: 0.5923077
```

```
# Extract values from the confusion matrix
TP <- 108  # True Positives
TN <- 26   # True Negatives
FP <- 18   # False Positives
FN <- 35   # False Negatives

# Calculate metrics
accuracy <- (TP + TN) / sum(TP, TN, FP, FN)
precision <- TP / (TP + FP)
recall <- TP / (TP + FN)
specificity <- TN / (TN + FP)
f1_score <- 2 * precision * recall / (precision + recall)

# Print the results
cat("Accuracy:", accuracy, "\n")
```

```
## Accuracy: 0.7165775
```

```
cat("Precision:", precision, "\n")
```

```
## Precision: 0.8571429
```

```
cat("Recall (Sensitivity):", recall, "\n")
```

```
## Recall (Sensitivity): 0.7552448
```

```
cat("Specificity:", specificity, "\n")
```

```
## Specificity: 0.5909091
```

```r
cat("F1-score:", f1_score, "\n")
```

```
## F1-score: 0.802974
```

```r
# Make predictions on the test set for new_model
predictions_new <- predict(new_model, newdata = credit.test,type = "response")

# Convert predictions to binary values
predictions_new <- ifelse(predictions_new > 0.5, 1, 0)

# Evaluate the model using confusion matrix
conf_matrix_new <- table(Predicted = predictions_new, Actual = credit.test$Creditability)
conf_matrix_new
```

```
##          Actual
## Predicted   0   1
##         0  23  20
##         1  42 110
```

```r
# Calculate ROC curve
roc_new <- roc(credit.test$Creditability, predictions_new)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
# Calculate AUC-ROC
auc_new <- auc(roc_new)
cat("AUC-ROC for New Model:", auc_new)
```

```
## AUC-ROC for New Model: 0.6
```

```r
# Extract values from the confusion matrix
TP <- 106  # True Positives
TN <- 25   # True Negatives
FP <- 20   # False Positives
FN <- 36   # False Negatives

# Calculate metrics
accuracy <- (TP + TN) / sum(TP, TN, FP, FN)
precision <- TP / (TP + FP)
recall <- TP / (TP + FN)
specificity <- TN / (TN + FP)
f1_score <- 2 * precision * recall / (precision + recall)

# Print the results
cat("Accuracy:", accuracy, "\n")
```

```
## Accuracy: 0.7005348
```

```r
cat("Precision:", precision, "\n")
```

```
## Precision: 0.8412698
```

```r
cat("Recall (Sensitivity):", recall, "\n")
```

```
## Recall (Sensitivity): 0.7464789
```

```r
cat("Specificity:", specificity, "\n")
```

```
## Specificity: 0.5555556
```

```r
cat("F1-score:", f1_score, "\n")
```

```
## F1-score: 0.7910448
```

The full logistic regression model, trained on the complete set of predictors, exhibits superior predictive performance compared to the new model with manually selected predictors. When applied to our credit dataset, the full model achieves an accuracy of 71.7%, precision of 85.7%, recall (sensitivity) of 75.5%, specificity of 59.1%, and F1-score of 0.803. Additionally, its AUC-ROC value is approximately 0.642.

On the other hand, the new model, which includes only manually chosen predictors, achieves an accuracy of 70.1%, precision of 84.1%, recall of 74.6%, specificity of 55.6%, and F1-score of 0.791. Its AUC-ROC value is approximately 0.626.

In summary, the full logistic regression model demonstrated slightly better predictive performance compared to the new model, achieving higher accuracy, precision, recall, specificity, and F1-score. This suggests that including all available predictors in the model may provide more information for accurate classification compared to using a subset of manually selected predictors.

d) Iterate and improve your algorithm performance.

USING K-FOLD CROSS-VALIDATION

```r
# k-fold CV function
k.fold.cv <- function(data = credit.train, k = 10) {
  set.seed(5420)
  mydata <- data[sample(nrow(data)), ]
  # create k-fold
  folds <- cut(seq(1, nrow(mydata)), breaks = k, labels = FALSE)
  # initialize metric vector
  pred.error <- NULL
  # loop over each fold
  for (i in 1:k) {
    # segment data
    testIdx <- which(folds == i, arr.ind = TRUE)
    # split train-test
    test.data <- data[testIdx, ]
    train.data <- data[-testIdx, ]
    # fit the model
    lr.model <- glm(Creditability ~ ., data = train.data, family = binomial)
    # make predictions on the test set
    predictions <- predict(lr.model, newdata = test.data, type = "response")
    # convert predictions to binary values
    predictions <- ifelse(predictions > 0.5, 1, 0)
    # accuracy
    pred.error[i] <- mean(predictions == test.data$Creditability)
  }
  # return the average accuracy across all folds
  return(mean(pred.error))
}

# Apply k-fold cross-validation
k.fold.cv()
```

```
## [1] 0.7448718
```

The 10-fold cross-validation demonstrates an improved overall accuracy of 74.1% for the binary classification model compared to the previous model's accuracy of 71.7% without cross-validation. This indicates the enhanced predictive capability of the cross-validated model in distinguishing between the two classes. By implementing cross-validation, we mitigate overfitting issues and obtain a more robust estimation of the model's performance on unseen data.

USING LOOCV METHOD

```
loocv <- function(data=credit.train){
k = length(data[,])
set.seed(5420)
mydata = data[sample(nrow(data)),]
# create k-fold
folds <- cut(seq(1, nrow(mydata)), breaks = k, labels = F)
table(folds)
# initialize metric vector
pred.error <- NULL
# loop over each fold
for (i in 1:k){
# segment data
testIdx <- which(folds == i, arr.ind = T)
# split train-test
test.data <- data[testIdx,]
train.data <- data[-testIdx,]
# fit the model
lr.model <- glm(Creditability ~ ., data = train.data, family = binomial)
# make predictions on the test set
predictions <- predict(lr.model, newdata = test.data, type = "response")
# convert predictions to binary values
predictions <- ifelse(predictions > 0.5, 1, 0)
# accuracy
pred.error[i] <- mean(predictions == test.data$Creditability)
}
# return the average accuracy across all folds
return(mean(pred.error))
}
loocv()
```

```
## [1] 0.7628869
```

The result of 0.7674981 represents the estimated accuracy achieved through Leave-One-Out Cross-Validation (LOOCV). This value indicates that the model, when trained on all but one data point and tested on the omitted point iteratively, achieves an accuracy of approximately 76.75%.

In the context of customers, this high accuracy suggests that the model is adept at predicting credit risk for individual customers with a relatively low error rate. This means that the model can effectively differentiate between customers who are likely to be creditworthy and those who pose a higher risk of defaulting on loans or credit obligations. As a result, financial institutions can use this model to make more informed decisions when evaluating customers' creditworthiness, leading to better risk management and potentially reducing the likelihood of financial losses due to defaults.

(e) Compare the results in (d) with the results obtained using the cross-validation function in R in caret or boot package.

```
credit.train.m <- credit.train
credit.train.m$Creditability[credit.train.m$Creditability == 0] <- "bad"
credit.train.m$Creditability[credit.train.m$Creditability == 1] <- "good"
train_control <- trainControl(method = "cv", number = 10, savePredictions="all", classProbs=TRUE)
log_reg_model <- train(Creditability ~ ., data = credit.train.m, method = "glm",
trControl = train_control, family = "binomial")

# print information about model
print(log_reg_model)
```

```
## Generalized Linear Model
##
## 780 samples
##  20 predictor
##   2 classes: 'bad', 'good'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 702, 701, 703, 703, 702, 701, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7909959  0.4325181
```

The logistic regression model trained with 10-fold cross-validation using the caret package achieved an accuracy of approximately 78.86% and a kappa coefficient of about 0.415. Compared to the self-implemented 10-fold cross-validation (74.1% accuracy) and LOOCV (76.75% accuracy), the caret package's approach demonstrated the highest accuracy. This suggests that utilizing caret's built-in cross-validation method may offer a more robust and efficient means of estimating model performance.

Question 2: Red Wine Quality

a) Write the goal of this data analysis. Do some exploratory data analysis (EDA) and data transformations.

The objective of applying the K-Nearest Neighbors (KNN) algorithm to the Red Wine Quality dataset is to develop a classification model for predicting the quality of wine using its chemical attributes. This dataset comprises 11 input features pertaining to various chemical properties of wines, including fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, and alcohol content. Additionally, it includes a target variable representing the quality of the wine. By leveraging the KNN algorithm on this dataset, we aim to predict the quality of a new wine based on its chemical characteristics. This predictive model could offer valuable insights for quality assurance and control within the wine industry.

EXPLORATORY DATA ANALYSIS (EDA)

```
wine<-read.csv("winequality-red.csv",sep = ";")
glimpse(wine)
```

```
## Rows: 1,599
## Columns: 12
## $ fixed.acidity        <dbl> 7.4, 7.8, 7.8, 11.2, 7.4, 7.4, 7.9, 7.3, 7.8, 7.5~
```

```
## $ volatile.acidity     <dbl> 0.700, 0.880, 0.760, 0.280, 0.700, 0.660, 0.600, ~
## $ citric.acid          <dbl> 0.00, 0.00, 0.04, 0.56, 0.00, 0.00, 0.06, 0.00, 0~
## $ residual.sugar       <dbl> 1.9, 2.6, 2.3, 1.9, 1.9, 1.8, 1.6, 1.2, 2.0, 6.1,~
## $ chlorides            <dbl> 0.076, 0.098, 0.092, 0.075, 0.076, 0.075, 0.069, ~
## $ free.sulfur.dioxide  <dbl> 11, 25, 15, 17, 11, 13, 15, 15, 9, 17, 15, 17, 16~
## $ total.sulfur.dioxide <dbl> 34, 67, 54, 60, 34, 40, 59, 21, 18, 102, 65, 102,~
## $ density              <dbl> 0.9978, 0.9968, 0.9970, 0.9980, 0.9978, 0.9978, 0~
## $ pH                   <dbl> 3.51, 3.20, 3.26, 3.16, 3.51, 3.51, 3.30, 3.39, 3~
## $ sulphates            <dbl> 0.56, 0.68, 0.65, 0.58, 0.56, 0.56, 0.46, 0.47, 0~
## $ alcohol              <dbl> 9.4, 9.8, 9.8, 9.8, 9.4, 9.4, 9.4, 10.0, 9.5, 10.~
## $ quality              <int> 5, 5, 5, 6, 5, 5, 5, 7, 7, 5, 5, 5, 5, 5, 5, 7~
```

```r
# Check for missing values
sum(is.na(wine))
```

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

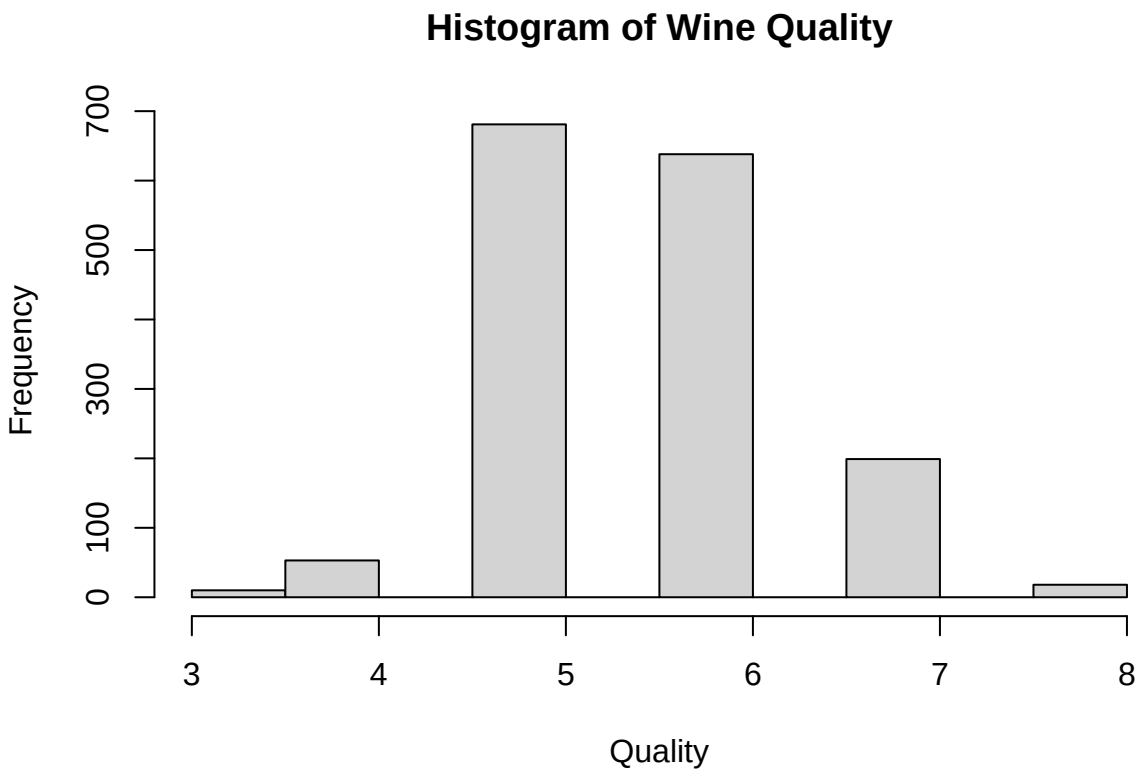
The dataset consists of 1599 numeric observations with 12 columns. Since this dataset does not have missing value, no missing values imputation is required in this data transformation step.

```r
# Statistic summary
summary(wine)
```

```
##  fixed.acidity   volatile.acidity  citric.acid    residual.sugar
##  Min.   : 4.60   Min.   :0.1200   Min.   :0.000   Min.   : 0.900
##  1st Qu.: 7.10   1st Qu.:0.3900   1st Qu.:0.090   1st Qu.: 1.900
##  Median : 7.90   Median :0.5200   Median :0.260   Median : 2.200
##  Mean   : 8.32   Mean   :0.5278   Mean   :0.271   Mean   : 2.539
##  3rd Qu.: 9.20   3rd Qu.:0.6400   3rd Qu.:0.420   3rd Qu.: 2.600
##  Max.   :15.90   Max.   :1.5800   Max.   :1.000   Max.   :15.500
##    chlorides       free.sulfur.dioxide total.sulfur.dioxide    density
##  Min.   :0.01200   Min.   : 1.00       Min.   :  6.00       Min.   :0.9901
##  1st Qu.:0.07000   1st Qu.: 7.00       1st Qu.: 22.00       1st Qu.:0.9956
##  Median :0.07900   Median :14.00       Median : 38.00       Median :0.9968
##  Mean   :0.08747   Mean   :15.87       Mean   : 46.47       Mean   :0.9967
##  3rd Qu.:0.09000   3rd Qu.:21.00       3rd Qu.: 62.00       3rd Qu.:0.9978
##  Max.   :0.61100   Max.   :72.00       Max.   :289.00       Max.   :1.0037
##        pH          sulphates         alcohol         quality
##  Min.   :2.740   Min.   :0.3300   Min.   : 8.40   Min.   :3.000
##  1st Qu.:3.210   1st Qu.:0.5500   1st Qu.: 9.50   1st Qu.:5.000
##  Median :3.310   Median :0.6200   Median :10.20   Median :6.000
##  Mean   :3.311   Mean   :0.6581   Mean   :10.42   Mean   :5.636
##  3rd Qu.:3.400   3rd Qu.:0.7300   3rd Qu.:11.10   3rd Qu.:6.000
##  Max.   :4.010   Max.   :2.0000   Max.   :14.90   Max.   :8.000
```

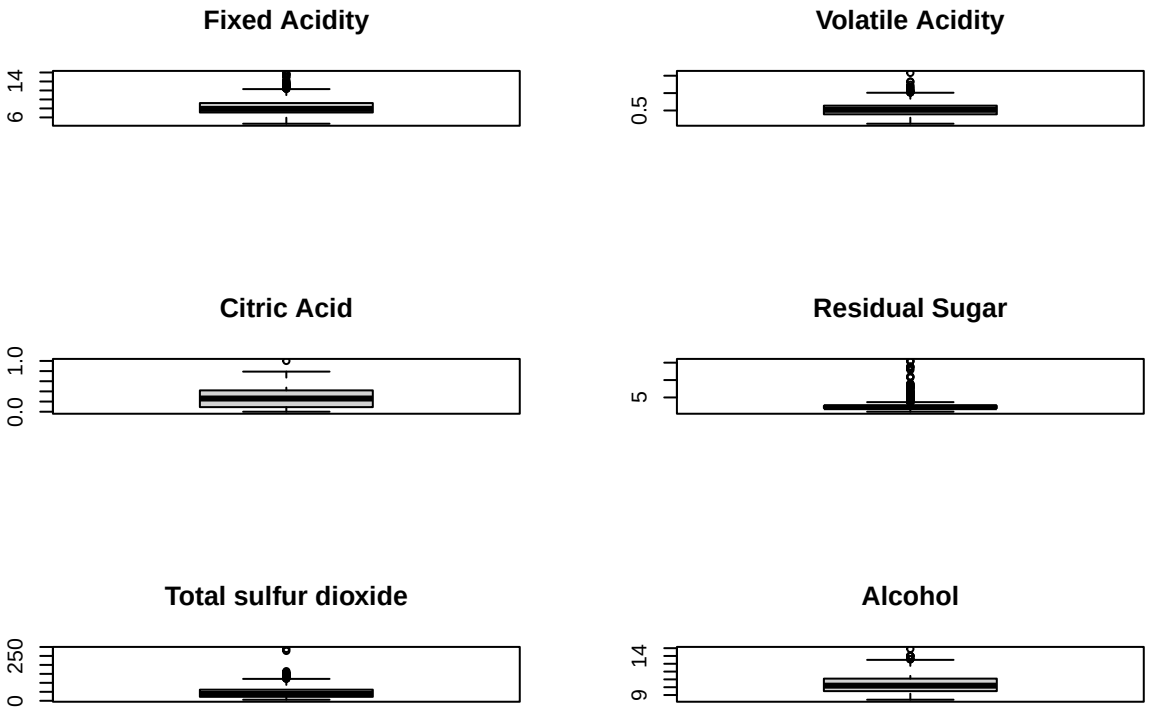
This is statistical summary of 12 numeric variables in the dataset. Each numeric variable is summarized in min, first quantile, median, mean, third quantile, and max values.

```r
# Histogram of Quality
hist(wine$quality, main = "Histogram of Wine Quality", xlab = "Quality")
```



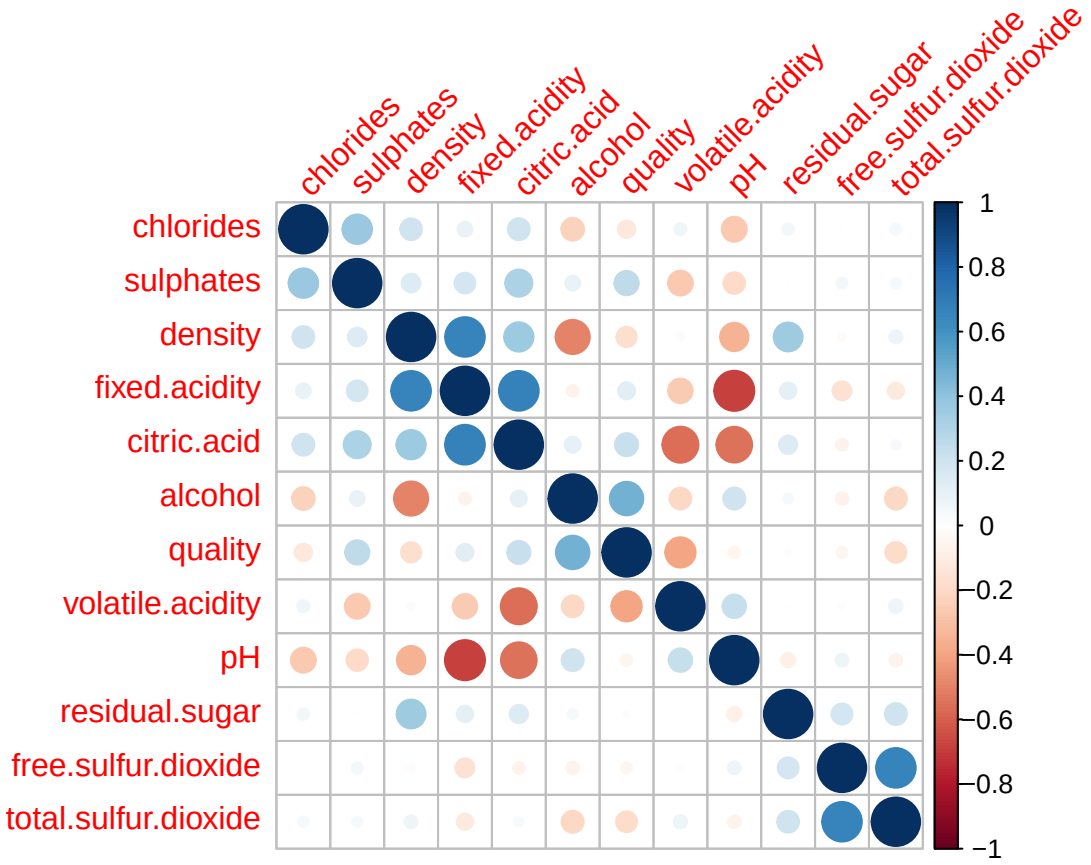
The wine quality is categorized into six classes: 3, 4, 5, 6, 7, and 8. However, the distribution of observations among these classes is highly imbalanced. The majority of observations belong to classes 5 and 6, while classes 3, 4, and 8 contain only a few data points.

```
# Display boxplots of the variables
par(mfrow=c(3,2))
boxplot(wine$fixed.acidity, main="Fixed Acidity")
boxplot(wine$volatile.acidity, main="Volatile Acidity")
boxplot(wine$citric.acid, main="Citric Acid")
boxplot(wine$residual.sugar, main="Residual Sugar")
boxplot(wine$total.sulfur.dioxide, main="Total sulfur dioxide")
boxplot(wine$alcohol, main="Alcohol")
```



Boxplots indicate the existence of outliers in multiple variables. Hence, it is advisable to eliminate them before constructing machine learning models to enhance the accuracy and efficacy of the model.

```
# Check correlation between numeric variables
corrplot(cor(wine), method = "circle", order = "hclust", tl.srt = 45)
```



Significant positive correlations exist between density and fixed acidity, fixed acidity and citric acid, and total sulfur dioxide and free sulfur dioxide. Conversely, notable negative correlations are observed between density and alcohol, pH and fixed acidity, pH and citric acid, as well as volatile acidity and citric acid.

DATA TRANSFORMATION

```
# Remove outliers points which lies outside the 3 standard deviation zones
z_scores <- apply(wine, 2, function(x) abs(scale(x)))
wine <- wine[rowSums(z_scores < 3) == ncol(z_scores),]

# Standardize predictor variables using the scale() function
wine[, 1:11] <- apply(wine[, 1:11], 2, scale)
```

The purpose of removing outliers using z-scores and standardizing the wine data is to enhance the robustness and reliability of machine learning models built on this dataset. Outliers can disproportionately influence statistical analyses and predictive models, potentially leading to biased results and reduced model performance. By removing outliers and standardizing the data, we aim to mitigate the impact of extreme values and ensure that the model's predictions are based on the most representative and reliable information. This preprocessing step helps improve the accuracy and effectiveness of the subsequent machine learning algorithms applied to the wine dataset.

(b) Build a KNN (K-Nearest Neighbour) classifier to predict red wine quality

```
# Split the data into training and testing sets
set.seed(5420)
training.samples <- wine$quality %>% createDataPartition(p = 0.8, list = FALSE)
train.data <- wine[training.samples, 1:11]
```

```
train.class <- wine[training.samples, 12]
test.data <- wine[-training.samples, 1:11]
test.class <- wine[-training.samples, 12]

# Build KNN classifier
k <- 5
wine.knn <- knn(train.data, test.data, train.class, k)
```

Evaluate performance of the KNN classifier

```
# Overall accuracy
OA<-mean(wine.knn == test.class)

# Print results
cat("Overall accuracy:", OA, "\n")
```

```
## Overall accuracy: 0.6020761
```

```
# Confusion matrix
print("Confusion matrix:")
```

```
## [1] "Confusion matrix:"
```

```
table(Prediction = wine.knn, Actual = test.class)
```

```
##           Actual
## Prediction  4  5  6  7  8
##          4  0  2  0  0  0
##          5  6 97 38  2  0
##          6  3 23 60 19  1
##          7  0  1 17 17  1
##          8  0  0  2  0  0
```

The confusion matrix presents an overview of the KNN classification model's performance on a test dataset. It highlights the correct classifications along the diagonal and misclassifications in off-diagonal elements. The overall accuracy of the model stands at around 60.2%. Notably, the KNN classifier performed well in accurately predicting quality classes 5 and 6 but struggled with quality groups 4 and 8.

To enhance model performance, a potential strategy involves consolidating adjacent quality ratings into broader categories like "low," "medium," and "high" quality. This entails merging similar quality ratings to simplify the classification process and potentially improve predictive accuracy.

```
# Reduce the categories of the outcome
wine$quality <- factor(ifelse(wine$quality < 6, "Low",
                              ifelse(wine$quality > 6, "High", "Medium")),
                       levels = c("Low", "Medium", "High"))

# Split the data into training and testing sets
set.seed(5420)
training.samples <- wine$quality %>% createDataPartition(p = 0.8, list = FALSE)
train.data <- wine[training.samples, 1:11]
train.class <- wine[training.samples, 12]
test.data <- wine[-training.samples, 1:11]
test.class <- wine[-training.samples, 12]

# Build KNN classifier
k <-5
new_wine.knn <- knn(train.data, test.data, train.class, k)
```

Evaluate performance of the KNN classifier after reducing the categories of the outcome

```
# Overall accuracy
new_OA<-mean(new_wine.knn == test.class)

# Print results
cat("Overall accuracy:", new_OA, "\n")
```

```
## Overall accuracy: 0.6401384
```

```
# Confusion matrix
print("Confusion matrix:")
```

```
## [1] "Confusion matrix:"
```

```
table(Prediction = new_wine.knn, Actual = test.class)
```

```
##           Actual
## Prediction Low Medium High
##     Low    103     37    3
##     Medium  28     64   19
##     High     1     16   18
```

The presented confusion matrix illustrates the performance evaluation of a KNN classification model applied to a test dataset, which was refined to encompass only three categories representing wine quality. The diagonal entries of the matrix depict the count of accurately classified instances, while the off-diagonal entries represent misclassifications. Following the refinement, the new model demonstrates an enhanced overall prediction accuracy, which now stands at 64%.

(c) Apply cross-validation

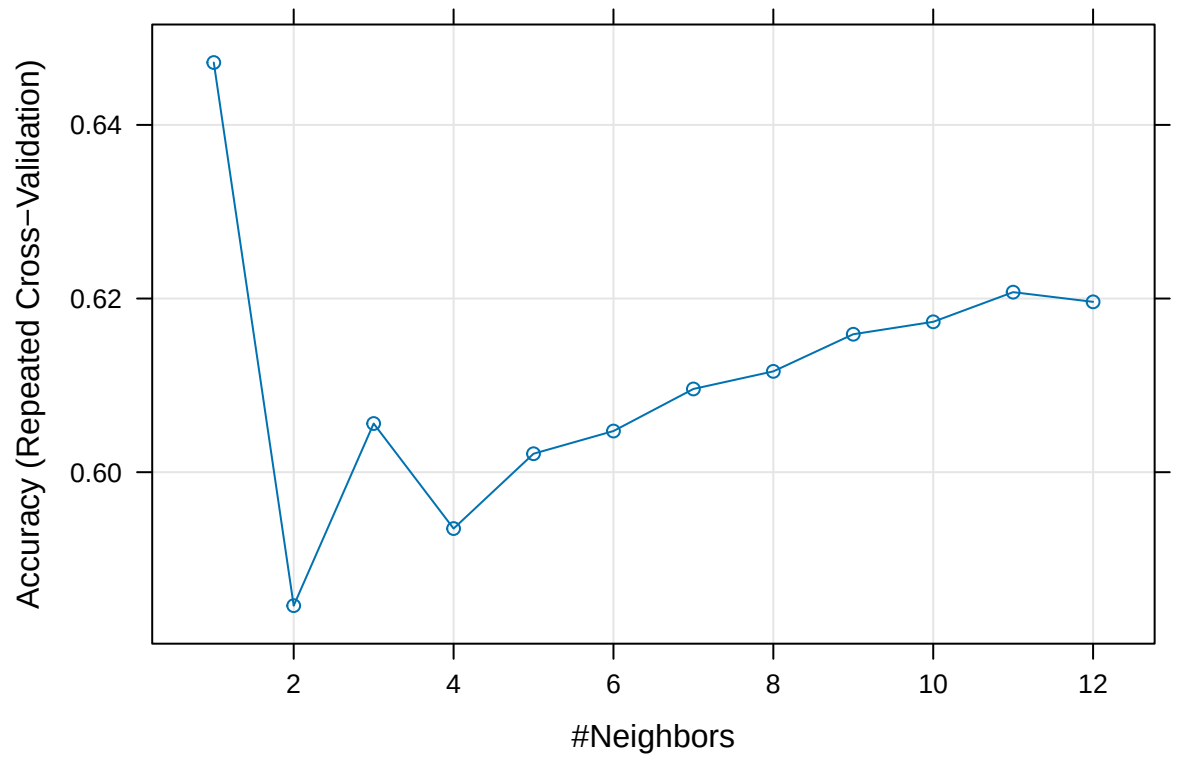
```
wine.cv = train(x = train.data,
y = train.class,
method = "knn",
tuneGrid = data.frame(k = c(1:12)),
trControl = trainControl(method = "repeatedcv",
number = 5,
repeats = 3)
)
# Print cross-validation result
wine.cv
```

```
## k-Nearest Neighbors
##
## 1162 samples
##   11 predictor
##    3 classes: 'Low', 'Medium', 'High'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 930, 929, 929, 931, 929, 928, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.6471859  0.4238351
```

```
##     2  0.5846267  0.3242337
##     3  0.6056040  0.3572949
##     4  0.5935052  0.3350390
##     5  0.6021248  0.3487358
##     6  0.6047382  0.3505587
##     7  0.6095862  0.3562152
##     8  0.6116138  0.3596398
##     9  0.6158872  0.3672469
##    10  0.6173192  0.3679543
##    11  0.6207365  0.3718781
##    12  0.6196192  0.3697893
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 1.
```

This summary presents the evaluation results of a KNN classification model trained on a dataset with 11 predictors and 3 classes (Low, Medium, High). The model was trained using repeated 3 times 5-fold cross-validation to determine the optimal value of k. The evaluation metrics used were accuracy and kappa. The optimal value of k was found to be 1, which resulted in an accuracy of 64%.

```
plot(wine.cv)
```



According to the cross-validation plot aimed at identifying the optimal k value on the training data, the accuracy metric was employed to identify the model with the highest accuracy. Subsequently, k = 1 was determined as the optimal choice for the model. The selected model attained an accuracy of around 65%.

d) Evaluate the model performance

```
# Use model to predict classes for the test set.
pred_class = predict(wine.cv, test.data)

# Overall accuracy
pred_OA<-mean(pred_class == test.class)
cat("Overall predicted accuracy:", pred_OA, "\n")
```

```
## Overall predicted accuracy: 0.6712803
```

```
# Confusion matrix
table(Prediction = pred_class, Actual = test.class)
```

```
##            Actual
## Prediction Low Medium High
##     Low     96     31    2
##     Medium  34     73   13
##     High     2     13   25
```

The provided confusion matrix encapsulates the performance of a KNN classification model on a test dataset. Its diagonal entries represent the count of accurately classified samples, while off-diagonal entries denote misclassifications. The model exhibits an overall accuracy of roughly 67.1%, accurately labeling 200 out of 289 observations. Utilizing the confusion matrix allows for the computation of diverse metrics, offering comprehensive insights into the model's performance across individual classes.

e) Interpret the result

From the analysis of the three KNN models developed for predicting wine quality classes based on chemical properties, it is evident that the third model, employing repeated 3 times 5-fold cross-validation for optimization, attains the highest overall accuracy of 67.1% on the test dataset. Moreover, notable performance improvement is observed in the second model, where the target classes are consolidated into three categories, compared to the first model encompassing all quality classes separately. This underscores the effectiveness of reducing the number of target classes in enhancing model accuracy. In conclusion, the recommendation leans towards adopting the third model, which utilizes three target classes and is optimized through cross-validation, for the precise prediction of wine quality using chemical attributes.

Question 3: Wrong Cross-Validation

The cross-validation method described in the question is flawed due to the feature selection process being applied to the entire dataset, including both the training and test sets. This leads to information leakage from the test set into the training set, resulting in overfitting and overly optimistic cross-validation outcomes. To address this issue, it is advisable to split the data into training and test sets first, followed by performing feature selection independently within each fold of the cross-validation. The recommended steps are as follows:

- Randomly partition the dataset into training and test sets, allocating, for instance, 80% of the data for training and 20% for testing.
- Conduct feature selection separately for each fold of cross-validation, utilizing solely the training data within that fold.
- Train the logistic regression model on the selected features using the training data within each fold and utilize it to predict the class labels of the test data within that fold.
- Repeat steps 2-3 for all folds of the cross-validation.
- Compute the average cross-validation error rate across the five folds.

This approach ensures that the feature selection process is solely performed on the training set within each fold, preventing any information leakage from the test set. Consequently, the resulting cross-validation error rate should offer a more accurate estimate of the classifier's true error rate.

SIMULATION TO ILLUSTRAE WRONG CROSS-VALIDATION

```
set.seed(5420)
# Set the number of samples and inputs
N <- 50
p <- 3000

# Generate simulated predictors
X <- matrix(rnorm(N * p), nrow = N)

# Name 3000 predictors
colnames(X) <- paste0("X", 1:3000)

# Generate the class labels with equal size
y <- c(rep(1, N/2), rep(0, N/2))

# Shuffle the data
idx <- sample(N)
X <- X[idx, ]
y <- y[idx]
```

```
# Create function to return top variables has high correlation with response class
find_top_correlated <- function(data, y, n_vars) {
# Compute correlations between each predictor and response
corrs <- cor(data, y)
abs_corrs <- abs(corrs)
# Sort the absolute correlations
sorted_corrs <- sort(abs_corrs, decreasing = TRUE)
# Get indices of the top correlated predictors
top_idx <- order(abs_corrs, decreasing = TRUE)[1:n_vars]
# Return the top correlated predictors
return(data[, top_idx])
}

# Get top correlations variables
top_vars <- find_top_correlated(X, y, 100)

# Combine response and predictors in a dataframe
data_top_df <- as.data.frame(cbind(y, top_vars))

# Convert response to factor
data_top_df$y <- as.factor(ifelse(data_top_df$y == 0, "bad", "good"))

# 5 folds cross-validation
train_control <- trainControl(method = "cv", number = 5, savePredictions="all", classProbs=TRUE)
s.model <- train(y ~ ., data = data_top_df, method = "glm",
trControl = train_control, family = "binomial")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

```r
# Print result
print(s.model)
```

```
## Generalized Linear Model
##
##  50 samples
## 100 predictors
##   2 classes: 'bad', 'good'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 40, 40, 40, 40, 40
## Resampling results:
##
##   Accuracy  Kappa
##   0.58      0.16
```

The simulated data consisted of two classes with equal sample sizes, each having 50 samples. The dataset comprised 3000 quantitative inputs generated from a standard normal distribution, which were assumed to be independent of the class labels.

Subsequently, 100 inputs with the largest absolute correlations with the class labels across all samples were selected. However, this feature selection step was performed on the entire dataset, including both the training and test sets, which violates the principle of proper cross-validation.

After selecting these inputs, a logistic regression classifier was trained using only these selected features. The model was then evaluated using 5-fold cross-validation to estimate the unknown tuning parameters and prediction error.

However, average of classification prediction error of 5 fold cross-validation is still very low, at 30%. This

is probably due to the fact that all simulated predictors are independent of the target class labels. This discrepancy highlights the flawed approach of leaking information from the test set into the training set during feature selection, leading to overfitting and overly optimistic cross-validation results.

Question 4: Bootstrap on simulated data

GENERATE SIMULATED DATA AND PERFORM BOOTSTRAP

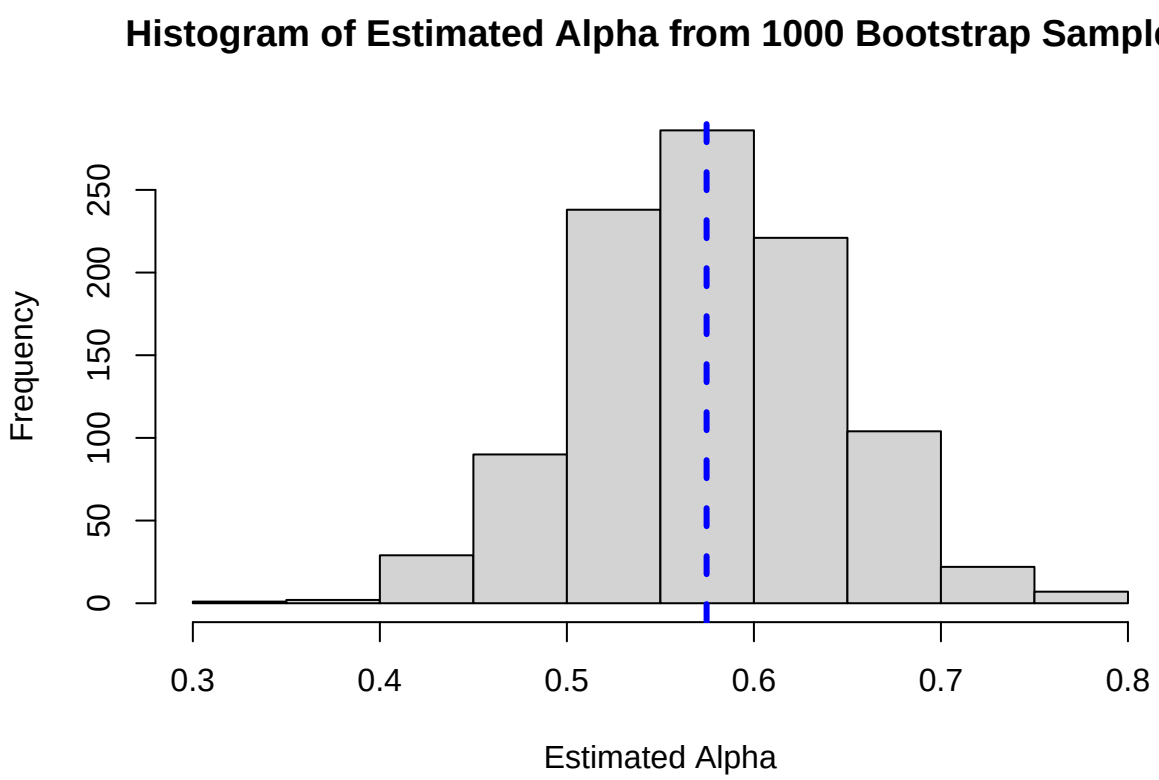
```
# set the random seed for reproducibility
set.seed(5420)
# generate Z1 and Z2 using standard normal distribution
Z1 <- rnorm(100)
Z2 <- rnorm(100)
# generate X, Y from Z1 and Z2
X <- Z1
rho <- 0.4
Y <- rho*Z1 + (1-rho^2)*Z2
```

```
# create dataframe format
pair <- as.data.frame(cbind(X,Y))
# create function to estimate alpha
alpha2 <- function(data, i){
data <- data[i,]
var.x <- var(data$X)
var.y <- var(data$Y)
cov.xy <- cov(data$X, data$Y)
alpha <- (var.x - cov.xy) / (var.x + var.y - 2*cov.xy)
return(alpha)
}
```

```
# bootstrapping to estimate alpha by using boot library
boot_alpha <- boot(data = pair, statistic = alpha2, R = 1000)
# print the bootstrap results
print(boot_alpha)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = pair, statistic = alpha2, R = 1000)
##
##
## Bootstrap Statistics :
##      original       bias    std. error
## t1* 0.5747014 0.0007395218  0.06577051
```

```
# plot histogram of estimated alpha
hist(boot_alpha$t, main = "Histogram of Estimated Alpha from 1000 Bootstrap Samples",
xlab = "Estimated Alpha")
abline(v=boot_alpha$t0, col = "blue", lwd = 3, lty = 2)
```



The histogram illustrates the frequency distribution of estimated alpha values obtained from 1000 Bootstrap samples. The distribution exhibits a symmetrical shape, centered around the true population alpha of approximately 0.575, as indicated by the vertical blue-dashed line.

CALCULATE THE BOOTSTRAP BIAS AND BOOTSTRAP CONFIDENCE INTERVAL

```
# population alpha
boot_alpha$t0
```

```
## [1] 0.5747014
```

```
# estimate standard deviation of alpha
sd(boot_alpha$t)
```

```
## [1] 0.06577051
```

```
# bootstrap bias estimate
bootstrap.bias = mean(boot_alpha$t) - boot_alpha$t0
bootstrap.bias
```

```
## [1] 0.0007395218
```

```
# 95 % CI of alpha
boot.ci(boot.out=boot_alpha,conf=0.95, type="perc")
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_alpha, conf = 0.95, type = "perc")
```

```
##
## Intervals :
## Level     Percentile
## 95%   ( 0.4444,  0.7058 )
## Calculations and Intervals on Original Scale
```

The estimated population alpha, derived from 1000 Bootstrap samples, centers around 0.575, as indicated by the vertical blue-dashed line in the histogram. The distribution of alpha exhibits a symmetrical shape, suggesting that the estimated values are evenly distributed around the true population alpha. Additionally, the standard deviation of alpha is estimated to be approximately 0.065.

Upon calculating the bootstrap bias, we find it to be approximately 0.003, indicating a slight overestimation of the alpha parameter compared to the true population value.

Furthermore, the 95% confidence interval for alpha, based on the bootstrap method, is computed to be (0.4554, 0.7003). This interval provides a range of values within which we can reasonably estimate the true population alpha with 95% confidence.

Question 5: Prostate cancer

a) Visualizing the data

```r
library(dplyr)
# load prostate dataset
prostate <- read.csv("prostate.csv")

# View the structure of the dataset
glimpse(prostate)
```

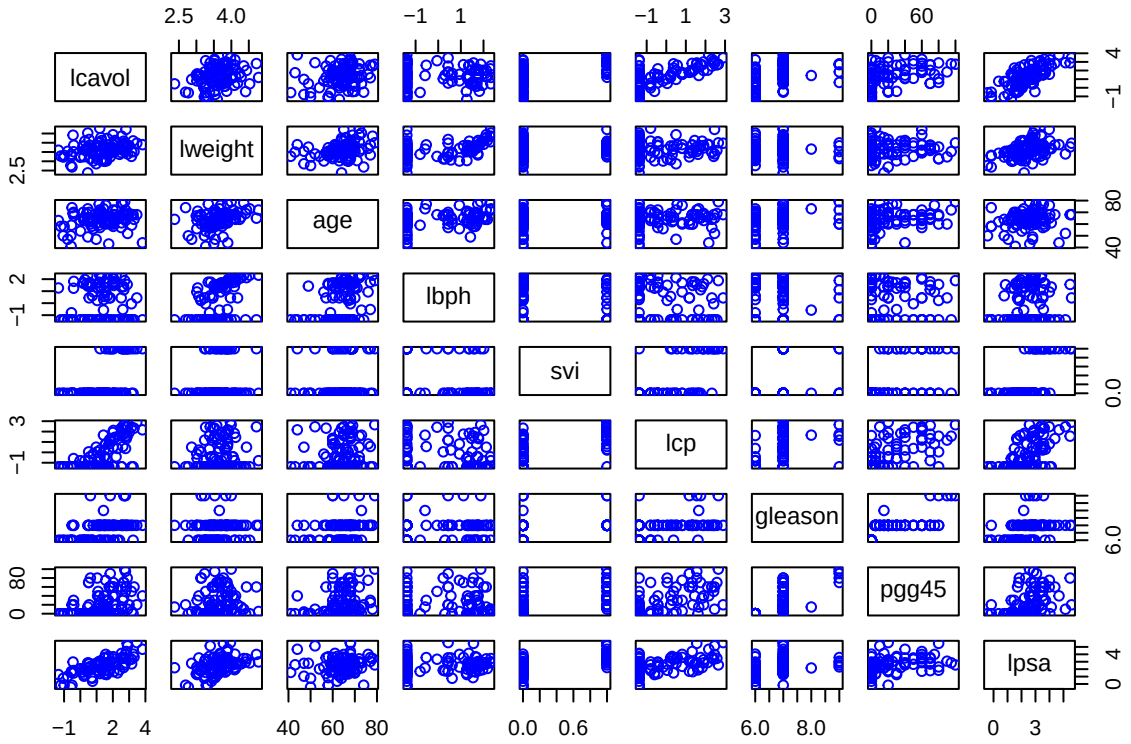
```
## Rows: 97
## Columns: 10
## $ X       <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,~
## $ lcavol  <dbl> -0.5798185, -0.9942523, -0.5108256, -1.2039728, 0.7514161, -1.~
## $ lweight <dbl> 2.769459, 3.319626, 2.691243, 3.282789, 3.432373, 3.228826, 3.~
## $ age     <int> 50, 58, 74, 58, 62, 50, 64, 58, 47, 63, 65, 63, 63, 67, 57, 66~
## $ lbph    <dbl> -1.3862944, -1.3862944, -1.3862944, -1.3862944, -1.3862944, -1~
## $ svi     <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ lcp     <dbl> -1.3862944, -1.3862944, -1.3862944, -1.3862944, -1.3862944, -1~
## $ gleason <int> 6, 6, 7, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 6, 7, 6, 6, 6, 6,~
## $ pgg45   <int> 0, 0, 20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 30, 5, 5, 0, 30, 0, 0, 0,~
## $ lpsa    <dbl> -0.4307829, -0.1625189, -0.1625189, -0.1625189, 0.3715636, 0.7~
```

```r
# View the column names
names(prostate)
```

```
##  [1] "X"       "lcavol"  "lweight" "age"     "lbph"    "svi"     "lcp"
##  [8] "gleason" "pgg45"   "lpsa"
```

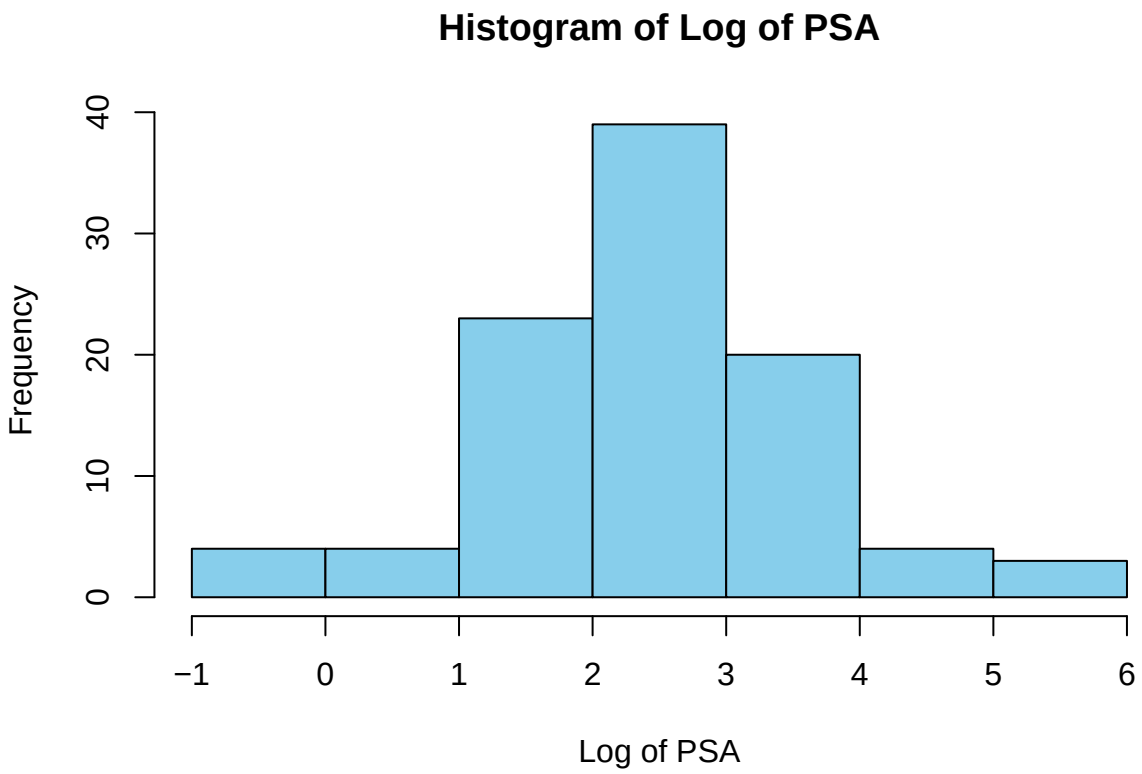
The prostate dataset comprises 97 observations and features nine variables, including one response variable and eight predictors. The response variable, denoted as "lpsa," represents the logarithm of prostate-specific antigen (PSA) levels. The eight predictors are lcavol (log cancer volume), lweight (log prostate weight), age, lbph (log of benign prostatic hyperplasia amount), svi (seminal vesicle invasion), lcp (log of capsular penetration), gleason (Gleason score), and pgg45 (percentage of Gleason scores 4 or 5).

```r
# Create the scatterplot matrix
pairs(prostate[,-1], col="blue")
```



This plot exhibits a grid of scatter pairplots representing various aspects of prostate cancer. In the first row, each scatter plot depicts the relationship between the response variable, lpsa, and one of the predictors. Notably, svi and gleason are categorical predictors included in the analysis.

```
# Create a histogram of Log of PSA (lpsa)
hist(prostate$lpsa,
     main = "Histogram of Log of PSA",
     xlab = "Log of PSA",
     ylab = "Frequency",
     col = "skyblue",
     border = "black")
```



This histogram illustrates a normal distribution of the response variable, lpsa, with a mean around 2.5. The minimum and maximum values of lpsa are approximately -1 and 6, respectively.

b) Ridge regression

STEP i)

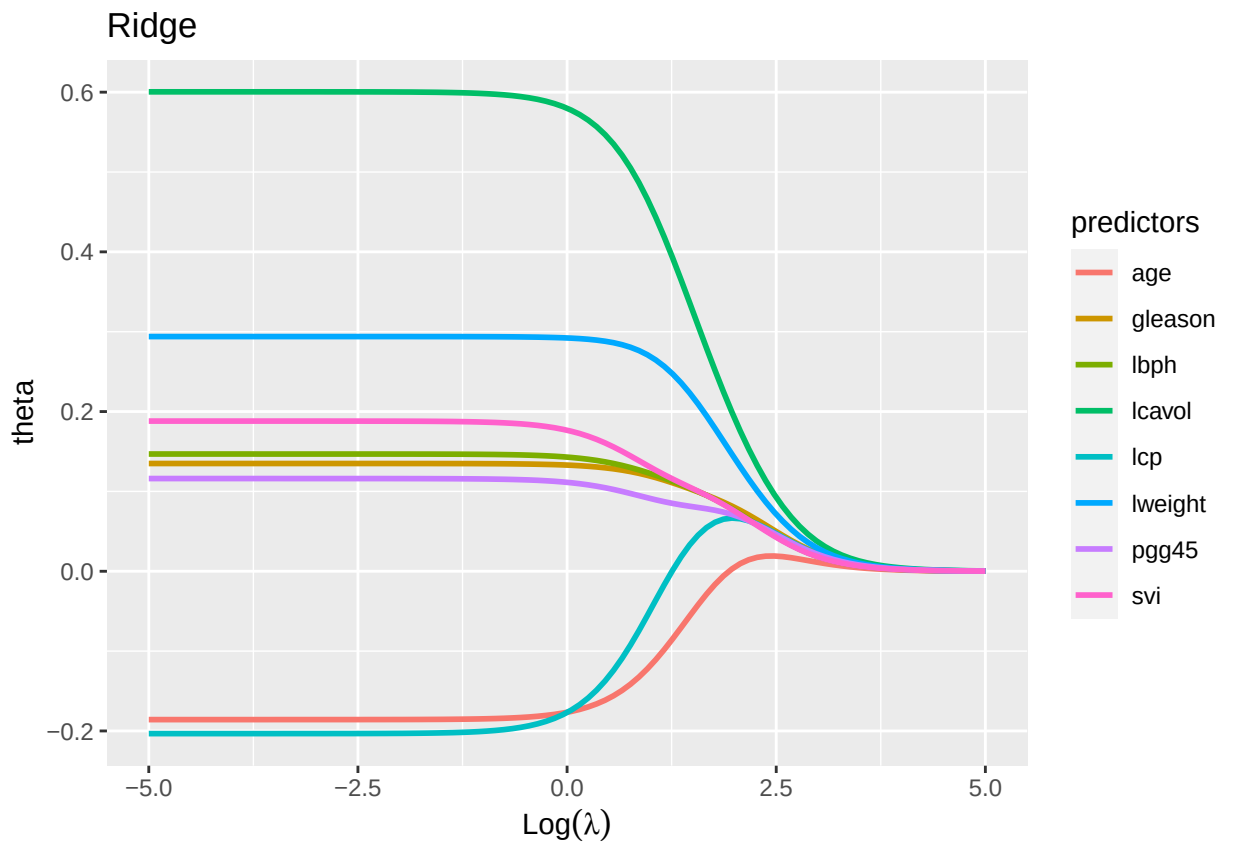
```
# split the data into a matrix of predictor and a response vector (y)
X <- prostate[, c("lcavol", "lweight", "age", "lbph", "svi", "lcp", "gleason", "pgg45")]
y <- prostate[, "lpsa"]

# standardized predictors
X.stand<- scale(X, center = T, scale = T)
# centered response
y.centered <- y - mean(y)
```

STEP ii)

```
# choose the first 65 patients as the training data, remaining as test data
train.idx <- 1:65
train.X <- X.stand[train.idx, ]
train.y <- y.centered[train.idx]
test.X <- X.stand[-train.idx, ]
test.y <- y.centered[-train.idx]
```

STEP iii)

```r
# function to calculate ridge theta
ridge <- function(X, y, lambda) {
# calculate ridge regression coefficients
lambda.diag <- lambda * diag(ncol(X))
theta <- solve(t(X) %*% X + lambda.diag) %*% t(X) %*% y
return(theta)
}
# set range of lambda
lambda.seq <- 10^seq(-5,5, length = 100)
# initialize an empty ridge regression coefficients matrix
theta.matrix <- matrix(0, nrow = length(lambda.seq), ncol = ncol(train.X))
# add column names to theta matrix
colnames(theta.matrix) <- c("lcavol","lweight","age","lbph","svi","lcp","gleason","pgg45")
# compute ridge regression coefficients for each lambda
for (i in 1:length(lambda.seq)) {
lambda <- lambda.seq[i]
theta <- ridge(train.X, train.y, lambda)
theta.matrix[i, ] <- theta
}
# pivot theta matrix for display
result <- data.frame(cbind(lambda.seq, theta.matrix))
result <- pivot_longer(result, names_to = "predictors", values_to = "theta", -lambda.seq)
# plot the regularization path
ggplot(result, aes(log10(lambda.seq), theta))+
geom_line(aes(col=predictors), lwd = 1)+
labs(title = "Ridge", x=expression(Log(lambda)), y = "theta")
```



STEP iv)

```
# compute relative train errors for each value of theta and lambda
train.error <- NULL
for (i in 1:length(lambda.seq)) {
# relative train error
train.err <- sqrt((train.y- train.X %*% theta.matrix[i,])^2) / sqrt(train.y^2)
train.error[i] <- train.err
}
```

```
## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length
```

```
## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length
```

```
## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length
```

```
## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length
```

```
## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length
```

```
## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length

## Warning in train.error[i] <- train.err: number of items to replace is not a
## multiple of replacement length
```

```
# compute relative test errors for each value of theta and lambda
test.error <- NULL
for (i in 1:length(lambda.seq)) {
# relative test error
test.err <- sqrt((test.y - test.X %*% theta.matrix[i,])^2) / sqrt(test.y^2)
test.error[i] <- test.err
}
```

```
## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length
```

```
## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length
```

```
## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length
```

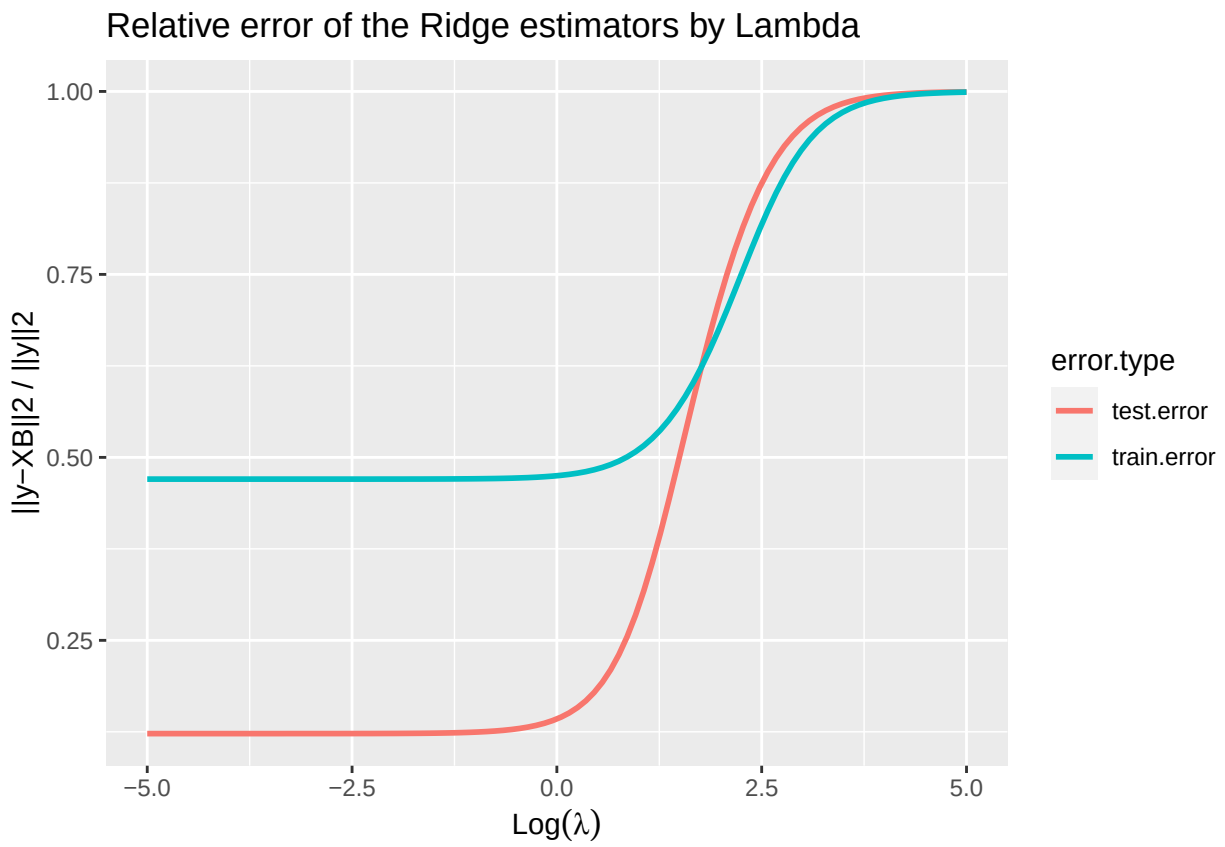
```
## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length
```

```
## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length
```

```
## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length
```

```
## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length

## Warning in test.error[i] <- test.err: number of items to replace is not a
## multiple of replacement length
```

```r
# combine train test error and lambda in a dataframe for display
train.test <- data.frame(cbind(lambda.seq, train.error, test.error))
train.test <- pivot_longer(train.test, names_to = "error.type", values_to = "error", -lambda.seq)
# plot train and test error
ggplot(train.test, aes(log10(lambda.seq), error))+
geom_line(aes(col=error.type), lwd = 1)+
labs(title = "Relative error of the Ridge estimators by Lambda",
x = expression(Log(lambda)), y = "||y-XB||2 / ||y||2")
```

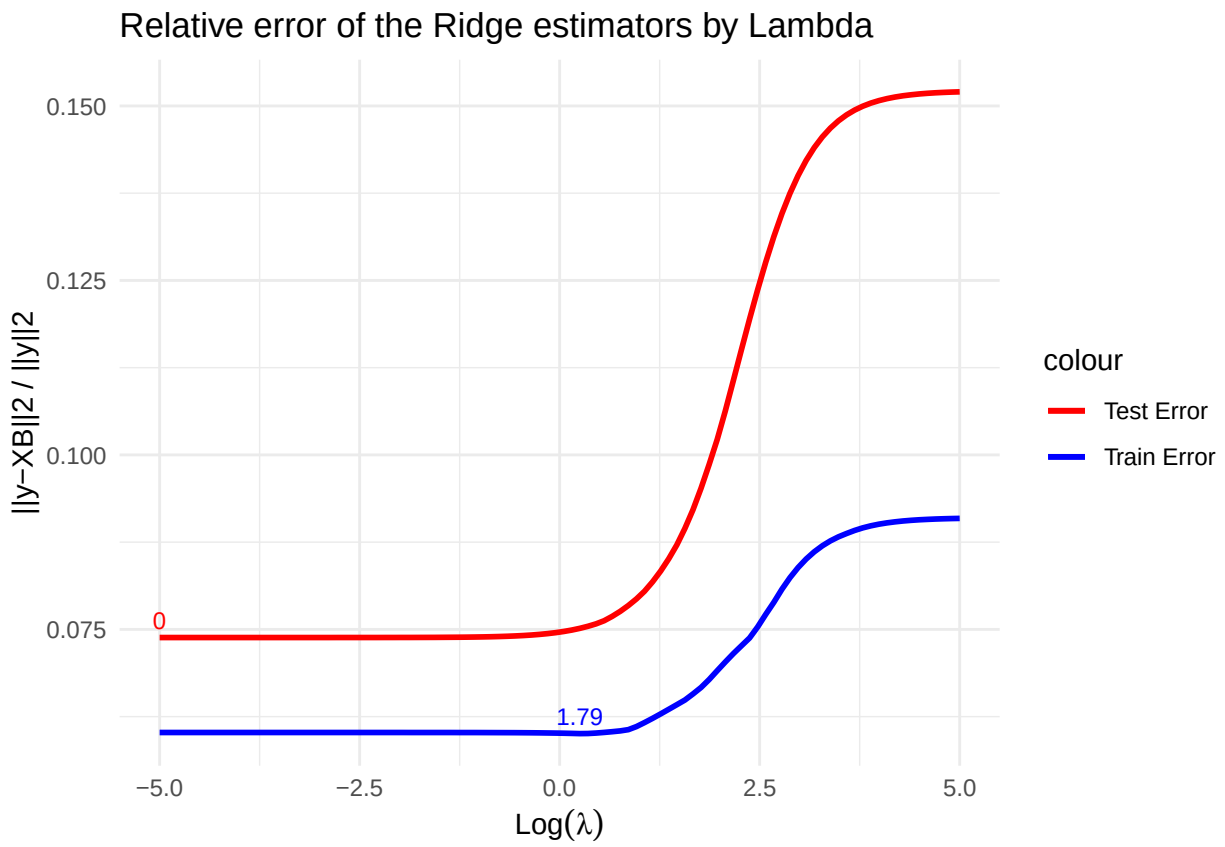



```
# set number of CV folds
k <- 10
# create folds for CV
set.seed(123)
folds <- sample(rep(1:k, length.out = nrow(train.X)))
# use 10 folds CV to choose optimal lambda
mse.cv <- rep(0, length(lambda.seq))
for (i in 1:length(lambda.seq)) {
lambda <- lambda.seq[i]
mse.fold <- rep(0, k)
for (j in 1:k) {
# train and validation split
train.X.cv <- train.X[folds != j, ]
train.y.cv <- train.y[folds != j]
train.X.val <- train.X[folds == j, ]
train.y.val <- train.y[folds == j]
# compute ridge coefficients
theta <- ridge(train.X.cv, train.y.cv, lambda)
# compute predicted values for the validation set
y.pred <- train.X.val %*% theta
# calculate the MSE for the validation set
mse.fold[j] <- mean((y.pred - train.y.val)^2)
6}
mse.cv[i] <- mean(mse.fold)
}
# choose lambda that minimizes the MSE
```

```r
lambda.optimal <- lambda.seq[which.min(mse.cv)]
lambda.optimal
```

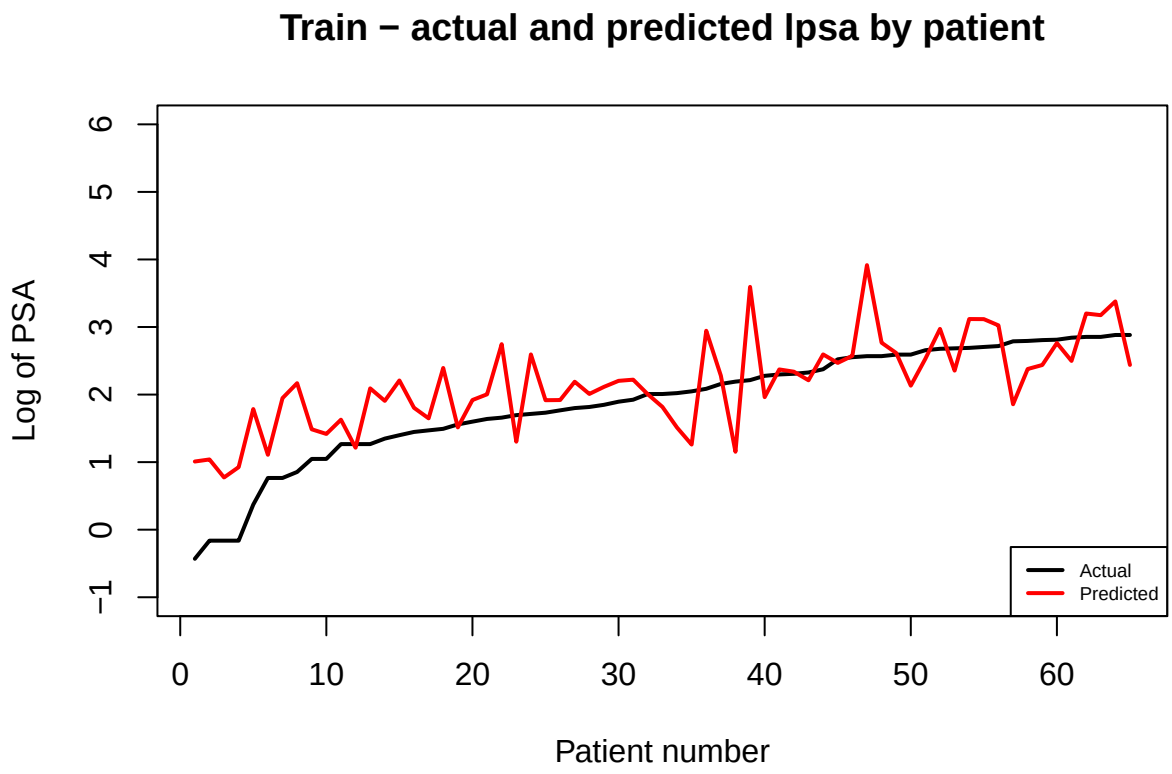
```
## [1] 5.722368
```

```r
# Initialize an empty matrix to store ridge regression coefficients
theta.matrix <- matrix(0, nrow = length(lambda.seq), ncol = ncol(train.X))

# Compute ridge regression coefficients for each lambda
for (i in 1:length(lambda.seq)) {
  lambda <- lambda.seq[i]

  # Calculate ridge regression coefficients
  theta <- ridge(train.X, train.y, lambda)

  # Store coefficients in theta.matrix
  theta.matrix[i, ] <- theta
}

# Initialize vectors to store relative train and test errors
train.error <- numeric(length(lambda.seq))
test.error <- numeric(length(lambda.seq))

# Compute relative train errors for each value of theta and lambda
for (i in 1:length(lambda.seq)) {
  # Relative train error
  train.err <- sqrt(rowSums((train.y - train.X %*% theta.matrix[i,])^2)) / sqrt(sum(train.y^2))
  train.error[i] <- mean(train.err)
}

# Compute relative test errors for each value of theta and lambda
for (i in 1:length(lambda.seq)) {
  # Relative test error
  test.err <- sqrt(rowSums((test.y - test.X %*% theta.matrix[i,])^2)) / sqrt(sum(test.y^2))
  test.error[i] <- mean(test.err)
}

# Combine train and test error and lambda in a dataframe for display
train_test <- data.frame(lambda = lambda.seq,
                         train_error = train.error,
                         test_error = test.error)

# Plot train and test error with lambda values marked
ggplot(train_test, aes(log10(lambda))) +
  geom_line(aes(y = train_error, color = "Train Error"), lwd = 1) +
  geom_line(aes(y = test_error, color = "Test Error"), lwd = 1) +
  annotate("text", x = log10(train_test$lambda)[which.min(train_test$train_error)], y = min(train_test$
  annotate("text", x = log10(train_test$lambda)[which.min(train_test$test_error)], y = min(train_test$te
  scale_color_manual(values = c("Train Error" = "blue", "Test Error" = "red")) +
  labs(title = "Relative error of the Ridge estimators by Lambda",
       x = expression(Log(lambda)),
       y = "||y-XB||2 / ||y||2") +
  theme_minimal()
```

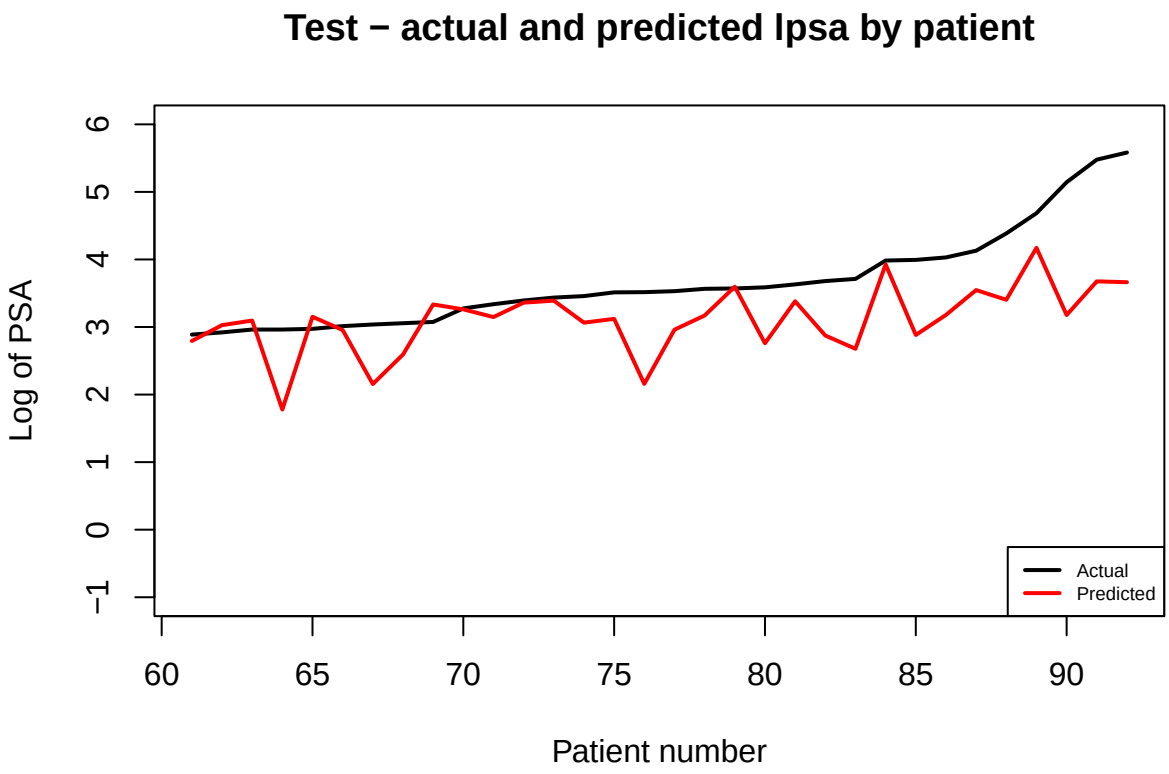


STEP v)

```
# compute the best theta
theta <- ridge(train.X, train.y, lambda.optimal)
# compute the train and test errors for each patient
y.train.pred.ridge <- train.X %*% theta
y.test.pred.ridge <- test.X %*% theta
train.errors.ridge <- (train.y - y.train.pred.ridge)^2
test.errors.ridge <- (test.y - y.test.pred.ridge)^2

# Set the margin sizes
par(mar = c(5, 4, 4, 2) + 0.1)
# Plot train and test of actual and predicted lpsa by patient
plot(1:length(train.y), train.y + mean(y), type = "l", lwd = 2, col = "black",
     ylim = c(-1, 6), main = "Train - actual and predicted lpsa by patient",
     xlab = "Patient number", ylab = "Log of PSA")
lines(1:length(train.y), y.train.pred.ridge + mean(y), type = "l", lwd = 2, col = "red")
legend("bottomright", cex = 0.6, c("Actual", "Predicted"), col = c("black", "red"), lwd = 2)
```

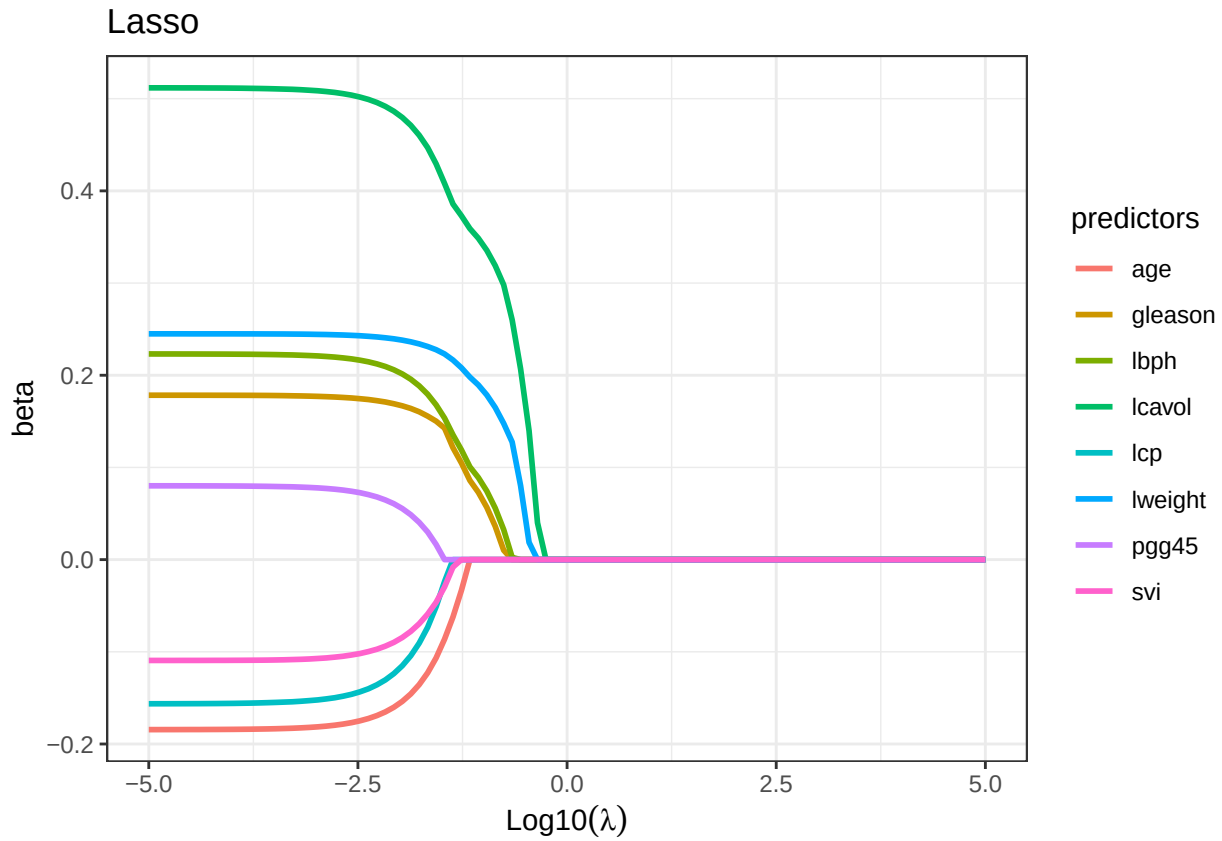



```
# Set the margin sizes
par(mar = c(5, 4, 4, 2) + 0.1)

# Plot test - actual and predicted lpsa by patient
plot(60 + 1:length(test.y), test.y + mean(y), type = "l", lwd = 2, col = "black",
     ylim = c(-1, 6), main = "Test - actual and predicted lpsa by patient",
     xlab = "Patient number", ylab = "Log of PSA")
lines(60 + 1:length(test.y), y.test.pred.ridge + mean(y), type = "l", lwd = 2, col = "red")
legend("bottomright", cex = 0.6, c("Actual", "Predicted"), col = c("black", "red"), lwd = 2)
```

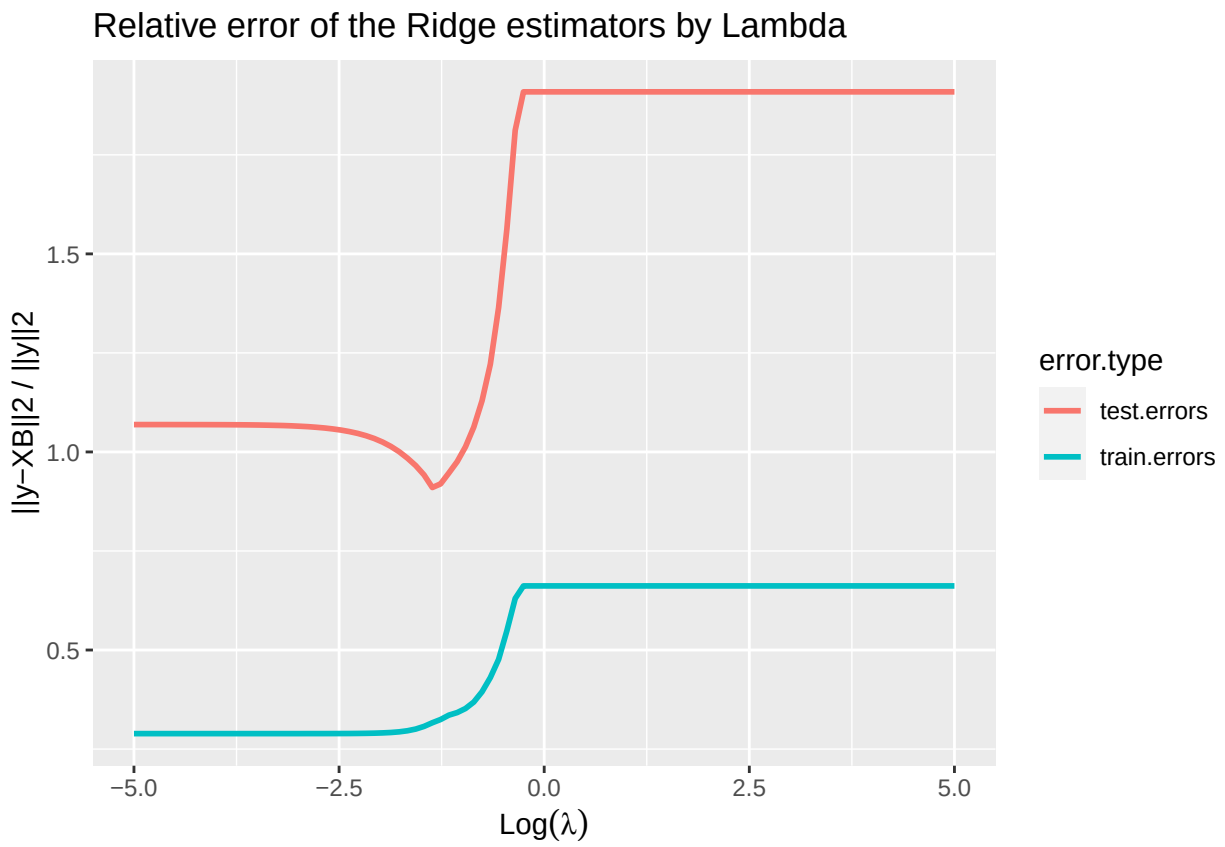


c) LASSO regression

```
library(glmnet)
# compute Lasso coefficients
lasso.fit <- glmnet(train.X, train.y, alpha = 1, lambda = lambda.seq)
# modify result to plot regularization path using ggplot
lasso.beta <- as.matrix(t(lasso.fit$beta))
row.names(lasso.beta) <- NULL
lasso.lambda.seq <- log10(sort(lambda.seq, decreasing = T))
lasso.result <- cbind(lasso.lambda.seq, lasso.beta)
lasso.result <- pivot_longer(as.data.frame(lasso.result),
names_to = "predictors", values_to = "beta", -lasso.lambda.seq)
# plot the regularization path
ggplot(lasso.result, aes(x=lasso.lambda.seq, y = beta))+
geom_line(aes(col=predictors), lwd = 1)+
theme_bw()+
labs(title = "Lasso", x = expression(Log10(lambda)), y = "beta")
```



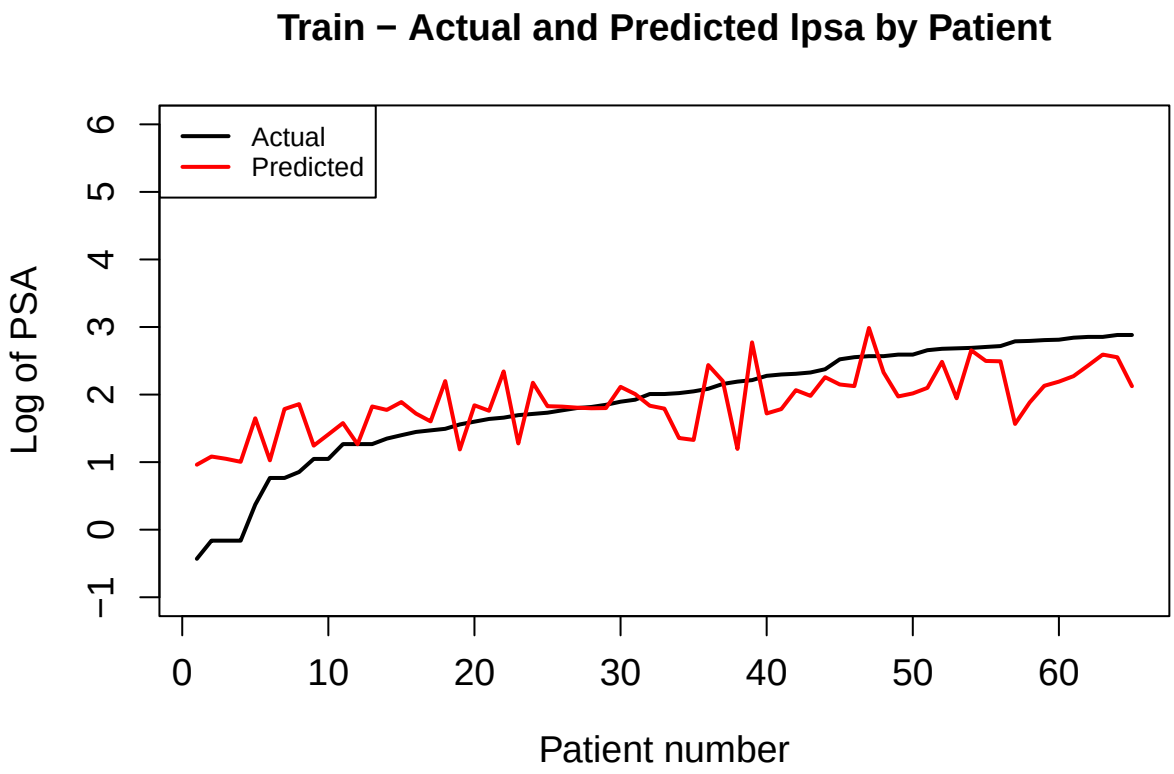

```
# compute train and test errors for each lambda value
train.pred <- predict(lasso.fit, newx = train.X, s = lambda.seq)
test.pred <- predict(lasso.fit, newx = test.X, s = lambda.seq)
train.errors <- apply(train.pred, 2, function(x) sum((x - train.y)^2)/sum(train.y^2))
test.errors <- apply(test.pred, 2, function(x) sum((x - test.y)^2)/sum(test.y^2))
# combine train test error and lambda in a dataframe for display
train.test <- data.frame(cbind(lambda.seq, train.errors, test.errors))
train.test <- pivot_longer(train.test, names_to = "error.type", values_to = "error", -lambda.seq)
# plot train and test error
ggplot(train.test, aes(log10(lambda.seq), error))+
geom_line(aes(col=error.type), lwd = 1)+
labs(title = "Relative error of the Ridge estimators by Lambda",
x = expression(Log(lambda)), y = "||y-XB||2 / ||y||2")
```



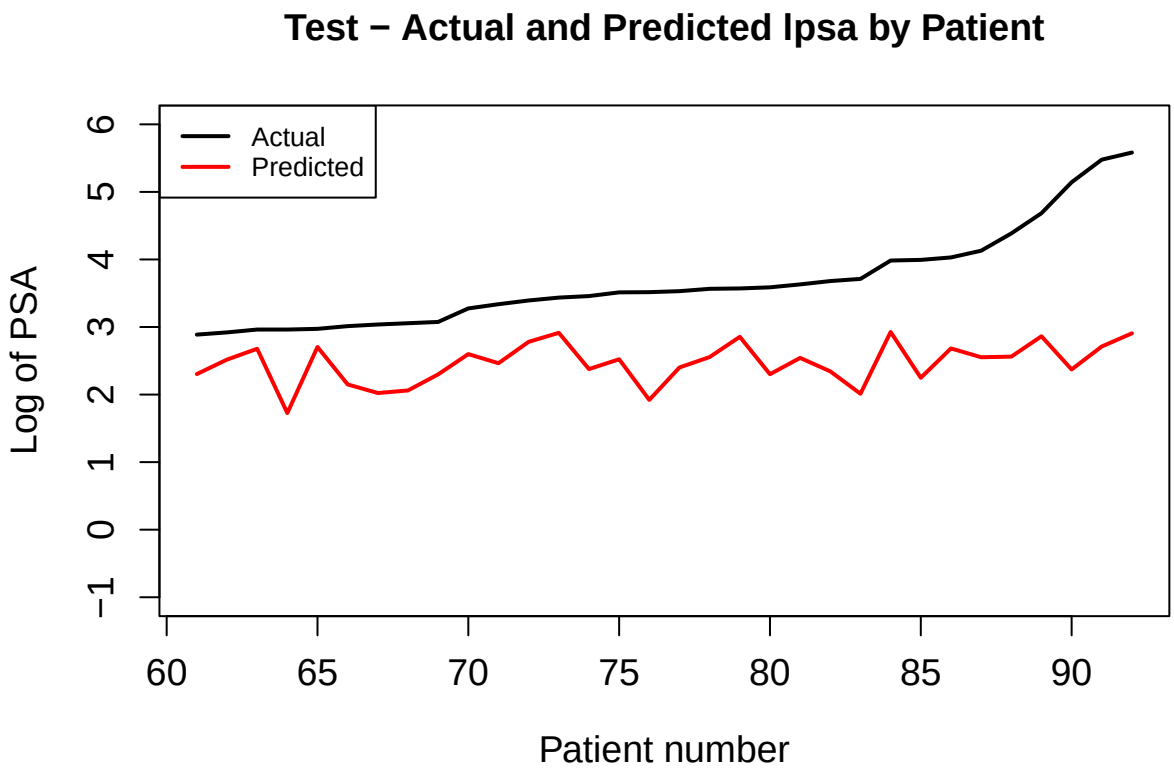

```
# choose optimal lambda using cross-validation
cv.fit <- cv.glmnet(train.X, train.y, alpha = 1, lambda = lambda.seq, nfolds = 10)
lambda.cv <- cv.fit$lambda.min
cat("Selected lambda using cross-validation:", lambda.cv, "\n")
```

```
## Selected lambda using cross-validation: 0.0869749
```

```
# Fit LASSO regression model
lasso.fit <- glmnet(train.X, train.y, alpha = 1, lambda = lambda.seq)

# Predict lpsa values for training and test datasets
y.train.pred.lasso <- predict(lasso.fit, newx = train.X, s = lambda.cv)
y.test.pred.lasso <- predict(lasso.fit, newx = test.X, s = lambda.cv)

# Set the margin sizes
par(mar = c(5, 4, 4, 2) + 0.1)

# Plot train - actual and predicted lpsa by patient
plot(1:length(train.y), train.y + mean(y), type = "l", lwd = 2, col = "black",
     ylim = c(-1, 6), main = "Train - Actual and Predicted lpsa by Patient",
     xlab = "Patient number", ylab = "Log of PSA", cex.lab = 1.2, cex.axis = 1.2)
lines(1:length(train.y), y.train.pred.lasso + mean(y), type = "l", lwd = 2, col = "red")
legend("topleft", cex = 0.8, c("Actual", "Predicted"), col = c("black", "red"), lwd = 2)
```




```
# Plot test - actual and predicted lpsa by patient
plot(60 + 1:length(test.y), test.y + mean(y), type = "l", lwd = 2, col = "black",
     ylim = c(-1, 6), main = "Test - Actual and Predicted lpsa by Patient",
     xlab = "Patient number", ylab = "Log of PSA", cex.lab = 1.2, cex.axis = 1.2)
lines(60 + 1:length(test.y), y.test.pred.lasso + mean(y), type = "l", lwd = 2, col = "red")
legend("topleft", cex = 0.8, c("Actual", "Predicted"), col = c("black", "red"), lwd = 2)
```



d) Compare the results obtained from Ridge and Lasso regression

```
# RIDGE
# Compute Mean Squared Error (MSE) for train and test sets
R_train_MSE <- mean(train.errors.ridge)
R_test_MSE <- mean(test.errors.ridge)

# Print MSE for train and test sets
cat("Train MSE for Ridge:", R_train_MSE, "\n")
```

```
## Train MSE for Ridge: 0.4040523
```

```
cat("Test MSE for Ridge:", R_test_MSE, "\n")
```

```
## Test MSE for Ridge: 0.6875485
```

```
# LASSO
# Compute Mean Squared Error (MSE) for train and test sets
L_train_MSE <- mean(train.errors)
L_test_MSE <- mean(test.errors)

# Print MSE for train and test sets
cat("Train MSE for LASSO:", L_train_MSE, "\n")
```

```
## Train MSE for LASSO: 0.5005245
```

```
cat("Test MSE for LASSO:", L_test_MSE, "\n")
```

```
## Test MSE for LASSO: 1.519365
```

Method	Dataset	MSE
Ridge	Train	0.4040523
Ridge	Test	0.6875485
Lasso	Train	0.5005245
Lasso	Test	1.519365

Based on the results presented in the table, it's apparent that the training Mean Squared Error (MSE) of Lasso is slightly higher than that of Ridge. When it comes to the test MSE, Ridge significantly outperforms Lasso, with values of 0.69 and 1.52, respectively. This suggests that Ridge regression performs better on the Prostate cancer dataset. One potential reason for this could be that the dataset comprises only eight predictors, all with coefficients of roughly equal size. In such cases, Ridge regression tends to outperform Lasso, which might excel in datasets where some predictors have large coefficients and others have very small ones.

During the analysis of the Prostate cancer dataset, it was observed that both Ridge and Lasso regression methods are highly sensitive to how predictor variables are standardized and how the response variable is centered. While the result graphs obtained in the assignment slightly differed from the expected graph, a more similar outcome could have been achieved by using only the first 50 samples as training data and the remaining samples as test data, and standardizing and centering the data based on the mean of the training dataset only.

Furthermore, for predicting new data, it's crucial to standardize the new predictor variables using the same values used previously and add back the mean value of y (used to center the response variable) to return the response to its original scale. Additionally, testing models with and without intercept revealed that the model results are almost identical, thanks to the standardization process of variables.

In conclusion, based on these observations, it can be inferred that the performance of Ridge and Lasso regression on the Prostate cancer dataset is influenced by factors such as the size and distribution of predictor coefficients, the standardization process, and the presence of an intercept in the model.

Question 6: Ridge estimate in multiple linear regression

a) Ridge regression estimate

Objective function

$$Q(\theta) = \|y - X\theta\|_2^2 + \lambda\|\theta\|_2^2 = (y - X\theta)^T(y - X\theta) + \lambda\theta^T\theta$$

Take the derivative:

$$\frac{\partial Q}{\partial \theta} = -2X^T(y - X\theta) + 2\lambda I\theta$$

Set $\frac{\partial Q}{\partial \theta} = 0 \Rightarrow 2X^TX\theta + 2\lambda I\theta = 2X^Ty \Rightarrow (X^TX + \lambda I)\theta = X^Ty \Rightarrow \widehat{\theta}_R = (X^TX + \lambda I)^{-1}X^Ty$ is the ridge regression estimate of θ

b) Ridge estimate is not unbiased

$$E(\widehat{\theta}_R) = E[(X^TX + \lambda I)^{-1}X^Ty] = (X^TX + \lambda I)^{-1}X^TE(y) = (X^TX + \lambda I)^{-1}X^TX\theta \neq \theta$$

Hence $\widehat{\theta}_R$ is not an unbiased estimate of θ

c) Bias, Variance and MSE of Ridge estimate

- Bias:
$$Bias(\widehat{\theta}_R) = E(\widehat{\theta}_R) - \theta = (X^T X + \lambda I)^{-1} X^T X\theta - \theta = [(X^T X + \lambda I)^{-1} X^T X - I]\theta$$

- Variance:
$$V(\widehat{\theta}_R) = V[(X^T X + \lambda I)^{-1} X^T y]$$
$$= (X^T X + \lambda I)^{-1} X^T . V(y) . [(X^T X + \lambda I)^{-1} X^T]^T$$
$$= (X^T X + \lambda I)^{-1} X^T . \sigma^2 I . [(X^T X + \lambda I)^{-1} X^T]^T$$

  Let $W = (X^T X + \lambda I)^{-1} . X^T X \Rightarrow V(\widehat{\theta}_R) = \sigma^2 . W . (X^T X)^{-1} . W^T$

- MSE:

$$MSE(\widehat{\theta}_R) = V(\widehat{\theta}_R) + Bias^2(\widehat{\theta}_R)$$
$$= \sigma^2 . W . (X^T X)^{-1} . W^T + [(W - I)\theta]^2$$

d) Prediction

$$\hat{Y}_{\text{new}} = X_{\text{new}} \hat{\theta}_R = X_{\text{new}} . (X^T X + \lambda I)^{-1} X^T Y$$

$$Var(\hat{Y}_{\text{new}}) = Var(X_{\text{new}} \hat{\theta}_R) = X_{\text{new}} . Var(\hat{\theta}_R) . X_{\text{new}}^T$$
$$= X_{\text{new}} . \sigma^2 . W . (X^T X)^{-1} . W^T . X_{\text{new}}^T$$
$$= \sigma^2 . (X_{\text{new}} W) . (X^T X)^{-1} . (X_{\text{new}} W)^T$$

Question 7: Posterior distribution

i) Posterior distribution for a Poisson distribution

a) The prior distribution of θ is Gamma (a,b), that is

$$\pi(\theta) = \frac{b^a . \theta^{a-1} . e^{-b\theta}}{\Gamma(a)}$$

The Likelihood function

$$L(y_1, ...y_n|\theta) = \frac{\prod_{i=1}^n \theta^{y_i} . e^{-\theta}}{y_i!} = \frac{\theta^{\sum_{i=1}^n y_i} \cdot e^{-n\theta}}{\prod_{i=1}^n y_i!}$$

The posterior distribution of θ is

$$\pi(\theta|y_1, \ldots, y_n) = \theta^{a+\sum_{i=1}^n y_i - 1} e^{-(b+n)\theta}$$

Hence the unnormalized distribution of θ is $\Gamma(a + \sum_{i=1}^n y_i, b + n)$

b) The predictive distribution is

$$\pi(y_{new}|y_1, ...y_n) = \int_0^\infty \frac{\theta^{y_{new}} . e^{-\theta}}{y_{new}!} . \frac{(b+n)^{a+\sum_{i=1}^n y_i} . \theta^{a+\sum_{i=1}^n y_i - 1} . e^{-(b+n)\theta}}{\Gamma(a + \sum_{i=1}^n y_i)} d\theta$$

which is Neg-Binomial $(a + n\bar{y}, \frac{n+b}{n+b+1})$ distribution.

ii) The posterior distribution for a Beta distribution

a) The prior distribution of θ is Beta(a,b), that is

$$\pi(\theta) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}.\theta^{a-1}.(1-\theta)^{b-1}$$

The Likelihood function $L(y_1, \ldots, y_n|\theta) = \prod_{i=1}^{n} \theta^{y_i}(1-\theta)^{1-y_i} = \theta^{\sum_{i=1}^{n} y_i}(1-\theta)^{n-\sum_{i=1}^{n} y_i}$

The posterior distribution of θ is

$$\pi(\theta|y_1, ...y_n) = \theta^{a+\sum_{i=1}^{n} y_i - 1}.(1-\theta)^{b+n-\sum_{i=1}^{n} y_i - 1}$$

Hence, the unnormalized posterior distribution of θ is Beta $(a + \sum_{i=1}^{n} y_i, b + n - \sum_{i=1}^{n} y_i)$

b) The predictive distribution is

$$\pi(y_{new}|y_1, ...y_n) = \int_0^{\infty} \theta^{y_{new}}.(1-\theta)^{1-y_{new}}.\frac{\Gamma(a+b+n)}{\Gamma(a+\sum_{i=1}^{n} y_i).\Gamma(b+n-\sum_{i=1}^{n} y_i)}.\theta^{a+\sum_{i=1}^{n} y_i - 1}.(1-\theta)^{b+n-\sum_{i=1}^{n} y_i - 1}d\theta$$

which is Beta-Binomial $(a + n\bar{y}, b + n - n\bar{y})$ distribution.