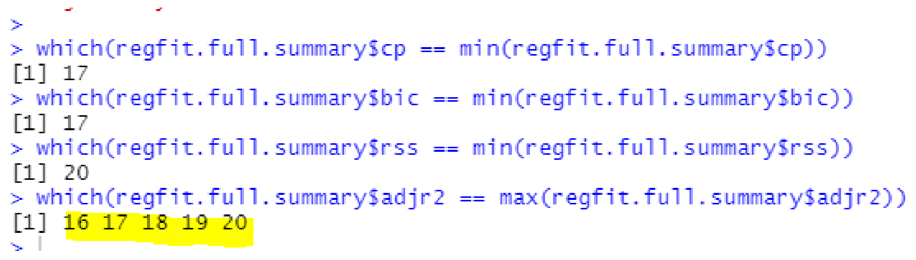
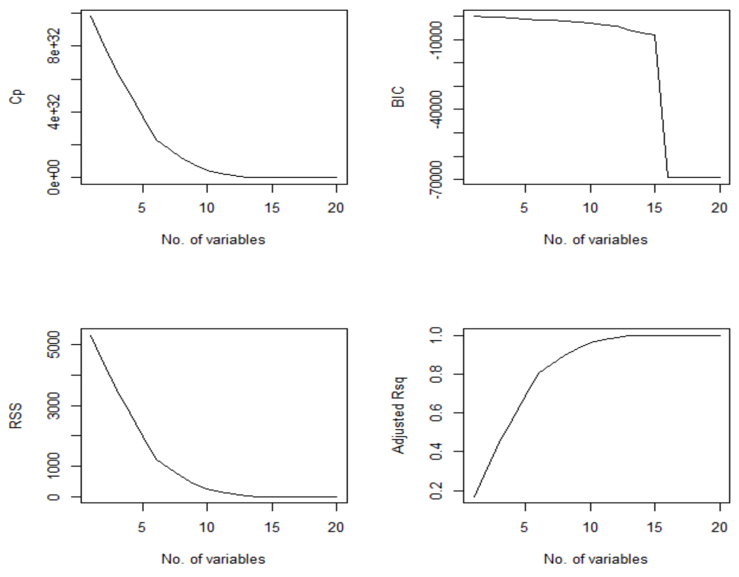
Task1 (Initial visualization after splitting the data)

a)Generated a data set with p = 20 features, n = 1,000 observations, and an associated quantitative response vector generated according to the model: Y = Xβ + ϵ.

b)Splitted the data set into a training set containing 900 observations and a test set containing 100 observations.



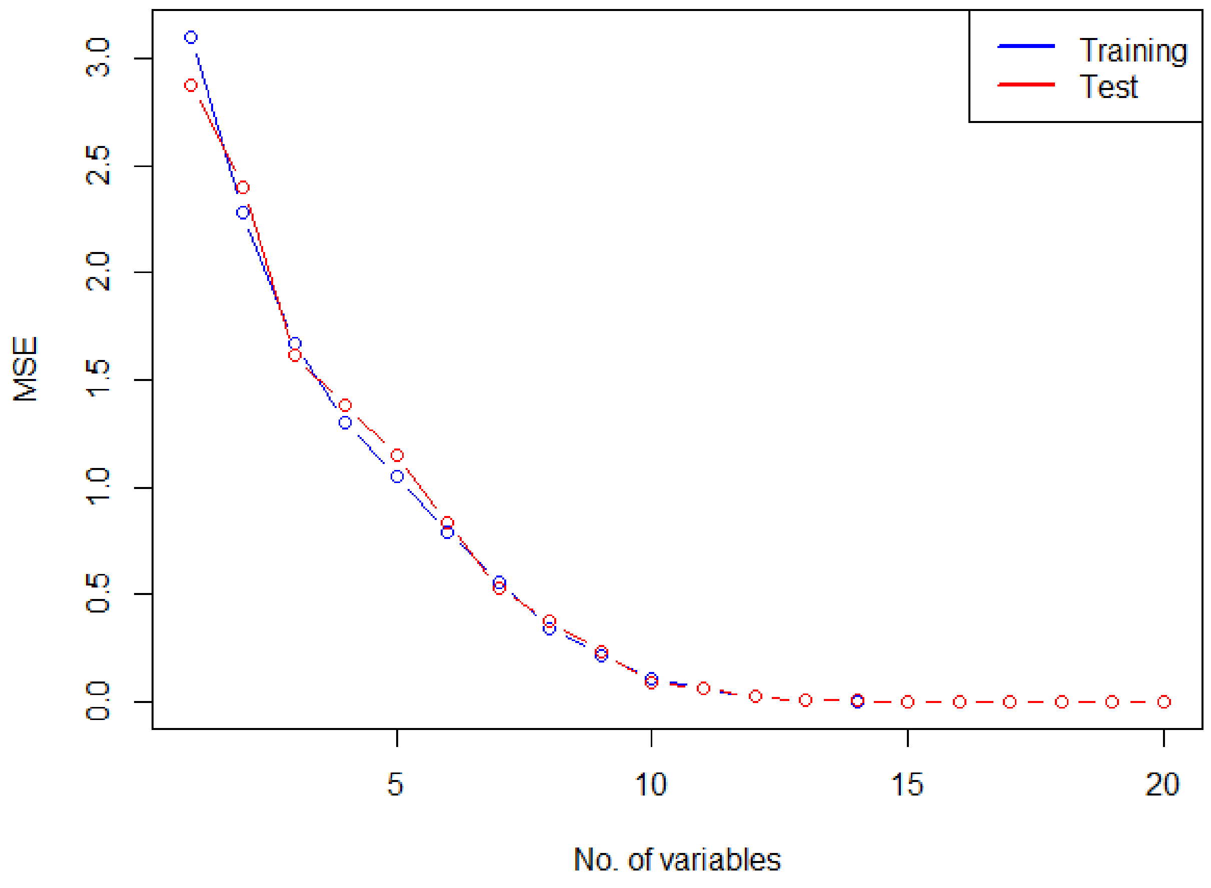
```
>
> which(regfit.full.summary$cp == min(regfit.full.summary$cp))
[1] 17
> which(regfit.full.summary$bic == min(regfit.full.summary$bic))
[1] 17
> which(regfit.full.summary$rss == min(regfit.full.summary$rss))
[1] 20
> which(regfit.full.summary$adjr2 == max(regfit.full.summary$adjr2))
[1] 16 17 18 19 20
>
```



The above plots suggest as the p variable model size increases the Cp,BIC and RSS decreases while the adjusted R2 increases which is quite obvious because as the MSE decreases shown in the below plot the adjusted R2 is supposed to increase.

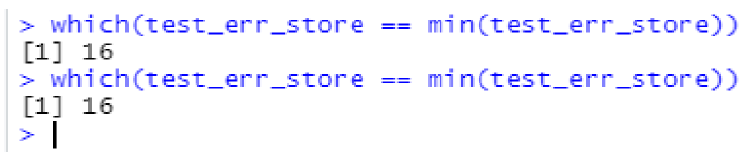
Task1 Part c) and Part d)

Mean Squared error plot for training and test data set model with respect to each of the p variable model



This Mean squared error plot of training and test error reflects that as the number of variables is increased gradually both of these errors decreases. Both the test and training error is minimum for Sixteen variable model.

Task1) Part e



```
> which(test_err_store == min(test_err_store))
[1] 16
> which(test_err_store == min(test_err_store))
[1] 16
> 
```

Thus, the minimum MSE is obtained from 16 variable model.

Some of the Beta value were set to 0 while simulating the response variable which in this case were for variables x3,x9,x13,x20

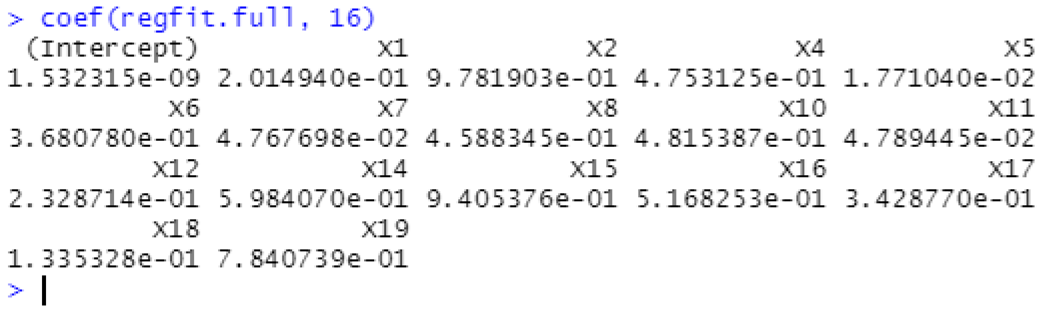
beta[c(3,9,13,20)]=0

Now since its already known that the betas of these variables are 0 then the best p variable model should not contain these variables

So, from the output of this command which(test_err_store == min(test_err_store)) is 16

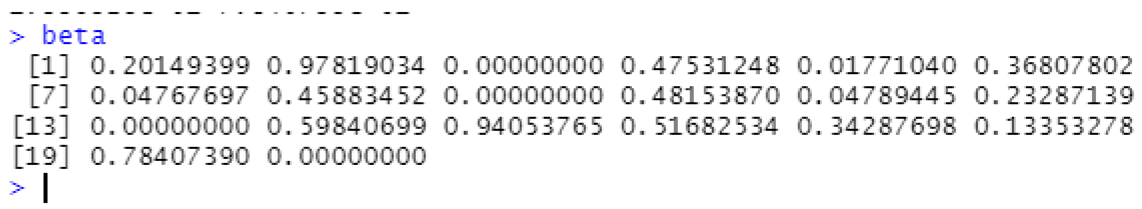
Hence these 16 variables should not contain x3,x9,x13 and x20

Coefficients of p variable model which provides minimum MSE



```
> coef(regfit.full, 16)
 (Intercept)           x1           x2           x4           x5
1.532315e-09 2.014940e-01 9.781903e-01 4.753125e-01 1.771040e-02
          x6           x7           x8          x10          x11
3.680780e-01 4.767698e-02 4.588345e-01 4.815387e-01 4.789445e-02
         x12          x14          x15          x16          x17
2.328714e-01 5.984070e-01 9.405376e-01 5.168253e-01 3.428770e-01
         x18          x19
1.335328e-01 7.840739e-01
> 
```

Coefficients of the model



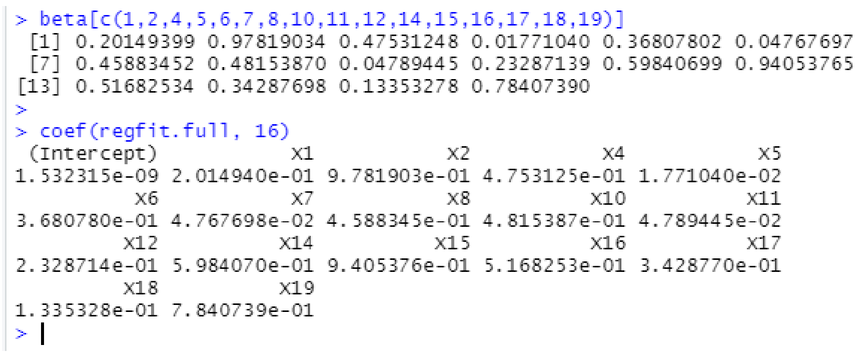
```
> beta
 [1] 0.20149399 0.97819034 0.00000000 0.47531248 0.01771040 0.36807802
 [7] 0.04767697 0.45883452 0.00000000 0.48153870 0.04789445 0.23287139
[13] 0.00000000 0.59840699 0.94053765 0.51682534 0.34287698 0.13353278
[19] 0.78407390 0.00000000
> 
```

Comparing the old and original coefficients it’s obvious that variables for which beta was set to 0 while simulating the original model are not included in the best model

Task 1 – Part (f)

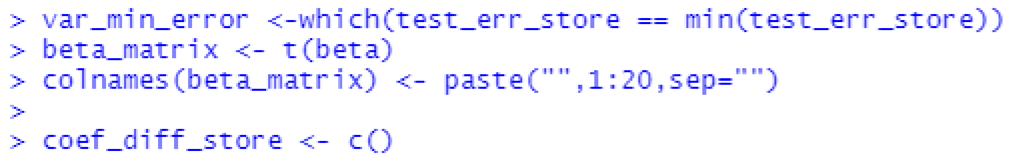
coef(regfit.full, 16)

It seems the new coefficients that is coefficients of best p variable model and the original coefficients of the simulated model are almost same

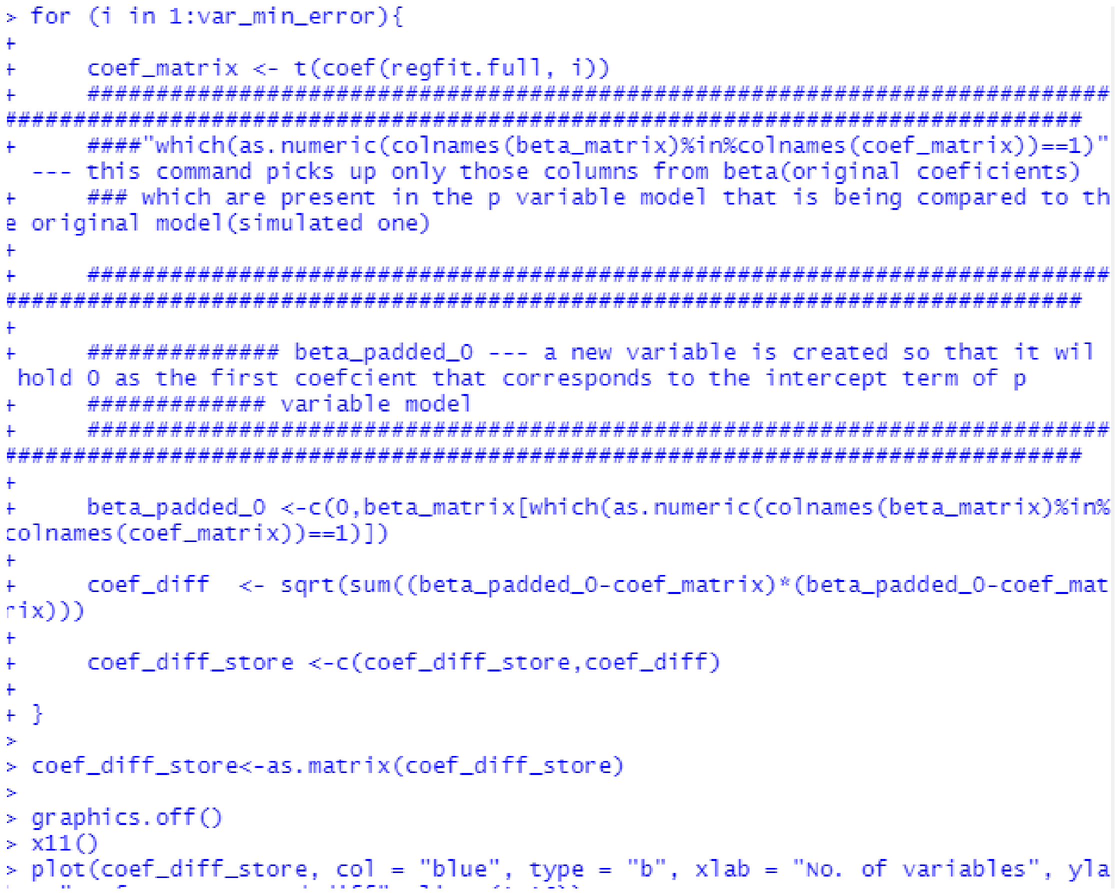


```
> beta[c(1,2,4,5,6,7,8,10,11,12,14,15,16,17,18,19)]
 [1] 0.20149399 0.97819034 0.47531248 0.01771040 0.36807802 0.04767697
 [7] 0.45883452 0.48153870 0.04789445 0.23287139 0.59840699 0.94053765
[13] 0.51682534 0.34287698 0.13353278 0.78407390
> 
> coef(regfit.full, 16)
 (Intercept)           x1           x2           x4           x5
1.532315e-09 2.014940e-01 9.781903e-01 4.753125e-01 1.771040e-02
          x6           x7           x8          x10          x11
3.680780e-01 4.767698e-02 4.588345e-01 4.815387e-01 4.789445e-02
         x12          x14          x15          x16          x17
2.328714e-01 5.984070e-01 9.405376e-01 5.168253e-01 3.428770e-01
         x18          x19
1.335328e-01 7.840739e-01
> 
```

Task1-part(g)



```
> var_min_error <-which(test_err_store == min(test_err_store))
> beta_matrix <- t(beta)
> colnames(beta_matrix) <- paste("",1:20,sep="")
>
> coef_diff_store <- c()
```

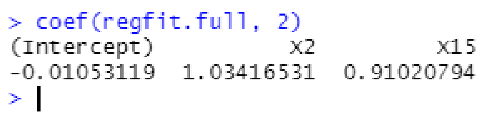


```
> for (i in 1:var_min_error){
+
+     coef_matrix <- t(coef(regfit.full, i))
+     #########################################################################
#########################################################################
+     ####"which(as.numeric(colnames(beta_matrix)%in%colnames(coef_matrix))==1)"
  --- this command picks up only those columns from beta(original coeficients)
+     ### which are present in the p variable model that is being compared to th
e original model(simulated one)
+
+     #########################################################################
#########################################################################
+
+     ############## beta_padded_0 --- a new variable is created so that it wil
 hold 0 as the first coefcient that corresponds to the intercept term of p
+     ############# variable model
+     #########################################################################
#########################################################################
+
+     beta_padded_0 <-c(0,beta_matrix[which(as.numeric(colnames(beta_matrix)%in%
colnames(coef_matrix))==1)])
+
+     coef_diff  <- sqrt(sum((beta_padded_0-coef_matrix)*(beta_padded_0-coef_mat
rix)))
+
+     coef_diff_store <-c(coef_diff_store,coef_diff)
+
+ }
>
> coef_diff_store<-as.matrix(coef_diff_store)
>
> graphics.off()
> x11()
> plot(coef_diff_store, col = "blue", type = "b", xlab = "No. of variables", yla
```

Explanation of the code which is calculating the square root of difference between the coefficients of any p variable model and the coefficients in the original simulated data set.

Here var_min_error contains the no. of variables that is coming out of best p variable model which is providing minimum MSE and in this case it is 16.

The for loop will iterate from 1 variable model till best p variable model which gives the minimum MSE and calculated the square root of sum of difference between the coefficients of each of the p variable models and the corresponding coefficients present in the original simulated model so for example for 2 variable model in the below screenshot for loop is going to extract only X2 and X3 coefficients corresponding to only X2 and X3 variables from beta matrix that contains the coefficients of all the variables in my simulated data set.



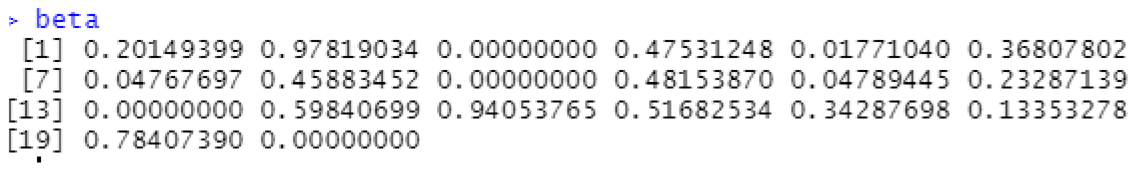
```
> coef(regfit.full, 2)
(Intercept)          X2         X15 
-0.01053119  1.03416531  0.91020794 
> 
```

Going step wise would make understanding of the code better, lets assume we have a 2 variable model ----→

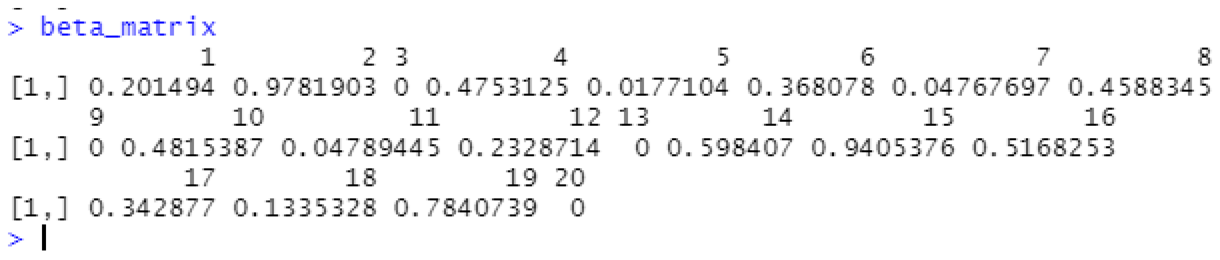
1)As mentioned above var_min_error=16

2)beta_matrix variable is created which is basically 1 d matrix that contains all the coefficients of my original simulated dataset.

beta_matrix <- t(beta)



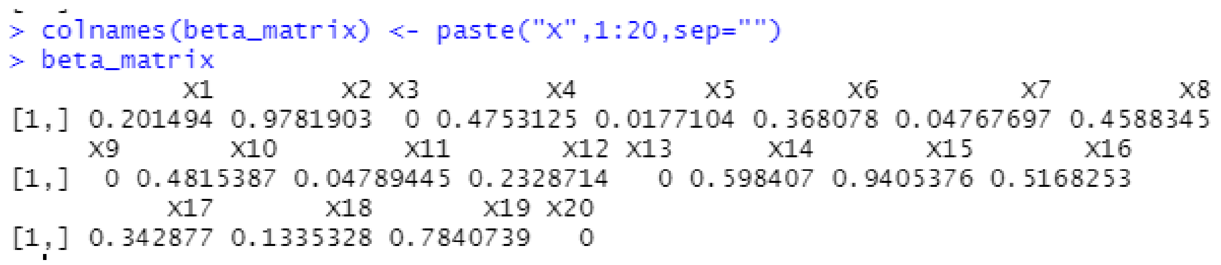
```
> beta
 [1] 0.20149399 0.97819034 0.00000000 0.47531248 0.01771040 0.36807802
 [7] 0.04767697 0.45883452 0.00000000 0.48153870 0.04789445 0.23287139
[13] 0.00000000 0.59840699 0.94053765 0.51682534 0.34287698 0.13353278
[19] 0.78407390 0.00000000
```



```
> beta_matrix
            1         2 3         4         5        6          7         8
[1,] 0.201494 0.9781903 0 0.4753125 0.0177104 0.368078 0.04767697 0.4588345
     9        10         11        12 13       14        15        16
[1,] 0 0.4815387 0.04789445 0.2328714  0 0.598407 0.9405376 0.5168253
           17        18        19 20
[1,] 0.342877 0.1335328 0.7840739  0
> 
```

4) colnames(beta_matrix) <- paste("X",1:20,sep="")

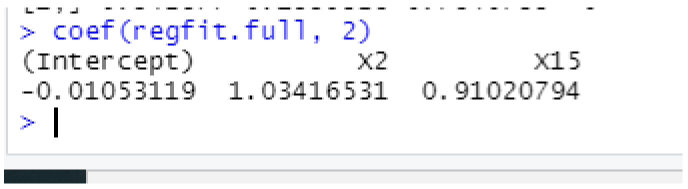
The above command changes the coeffcients names to exactly the same column names that are present in my original data set.



```
> colnames(beta_matrix) <- paste("X",1:20,sep="")
> beta_matrix
           X1        X2 X3        X4        X5       X6         X7        X8
[1,] 0.201494 0.9781903  0 0.4753125 0.0177104 0.368078 0.04767697 0.4588345
     X9       X10        X11       X12 X13      X14       X15       X16
[1,]  0 0.4815387 0.04789445 0.2328714   0 0.598407 0.9405376 0.5168253
          X17       X18       X19 X20
[1,] 0.342877 0.1335328 0.7840739   0
```

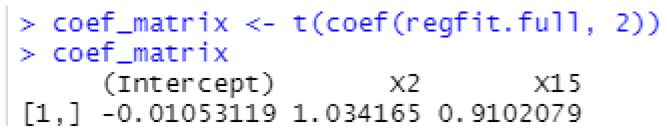
5)Inside the for loop for 2 variable model the small i value would be equal to 2 that is i = 2 in other words for the second iteration of my for loop

The value of   coef(regfit.full, i) would be –

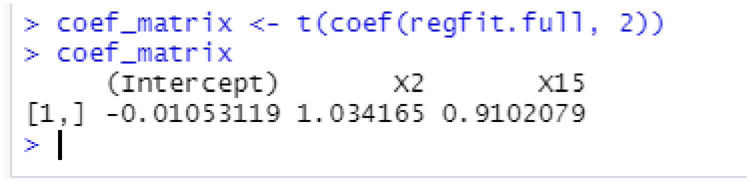


```
> coef(regfit.full, 2)
(Intercept)          X2         X15
-0.01053119  1.03416531  0.91020794
> 
```

Hence the value of coef_matrix <- t(coef(regfit.full, i)) would be –



```
> coef_matrix <- t(coef(regfit.full, 2))
> coef_matrix
     (Intercept)       X2       X15
[1,] -0.01053119 1.034165 0.9102079
```

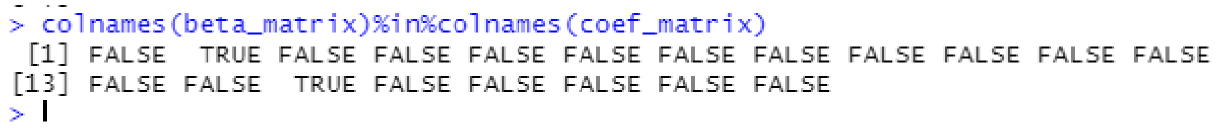


```
> coef_matrix <- t(coef(regfit.full, 2))
> coef_matrix
     (Intercept)       X2       X15
[1,] -0.01053119 1.034165 0.9102079
> 
```

4) beta_padded_0 <-
c(0,beta_matrix[which(as.numeric(colnames(beta_matrix)%in%colnames(coef_matrix))==1)])

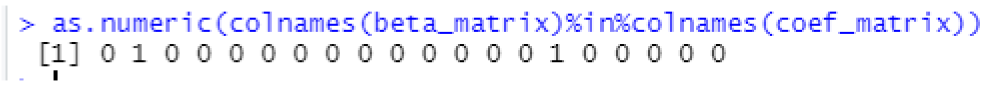
Lets understand what exactly this variable contains so lets break it down.

- So keeping in mind that we are just looking into a 2 variable model now as mentioned earlier , the output of this command colnames(beta_matrix)%in%colnames(coef_matrix) extracts the colnames from original data set that matches with those of the p variable model.



```
> colnames(beta_matrix)%in%colnames(coef_matrix)
 [1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[13] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
> 
```

Hence False are those variable names which are not present in my p variable model. I will try to assign a numeric value inside of logical True or False by the below command

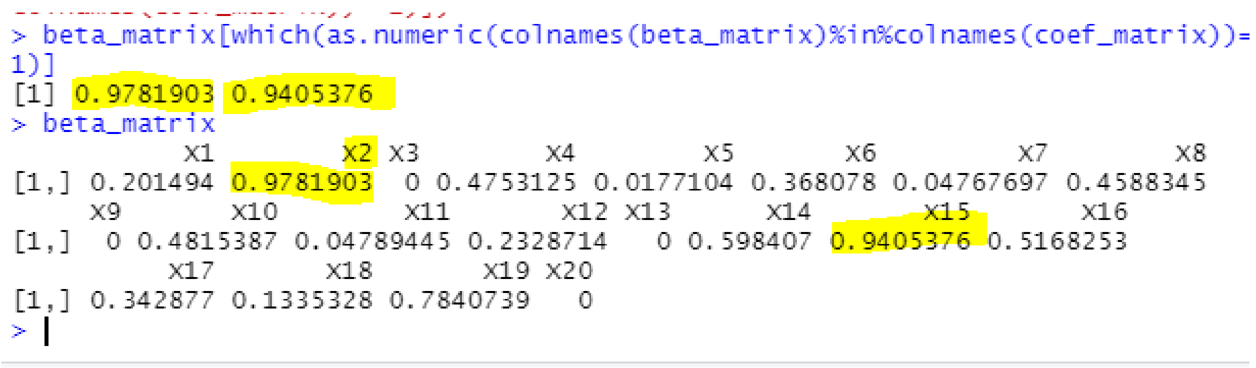


```
> as.numeric(colnames(beta_matrix)%in%colnames(coef_matrix))
 [1] 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
```

as.numeric(colnames(beta_matrix)%in%colnames(coef_matrix)

Comparing these two screenshots logical False in first screenshot corresponds to 0 in second screenshot and 1 and logical True corresponds to 1

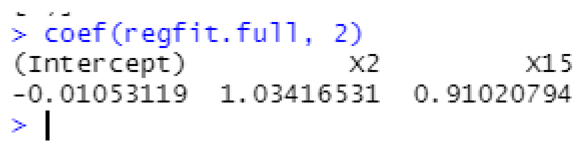
I want to pick up only the True from first screenshot which is equivalent to 1 in the second screenshot. The below command fulfills the purpose
beta_matrix[which(as.numeric(colnames(beta_matrix)%in%colnames(coef_matrix))==1)]



```
> beta_matrix[which(as.numeric(colnames(beta_matrix)%in%colnames(coef_matrix))=
1)]
[1] 0.9781903 0.9405376
> beta_matrix
            x1        x2 x3        x4        x5       x6         x7        x8
[1,] 0.201494 0.9781903  0 0.4753125 0.0177104 0.368078 0.04767697 0.4588345
      x9       x10        x11       x12 x13      x14       x15       x16
[1,]  0 0.4815387 0.04789445 0.2328714   0 0.598407 0.9405376 0.5168253
          x17       x18       x19 x20
[1,] 0.342877 0.1335328 0.7840739   0
> |
```

Here if we compare the output of command beta_matrix[which(as.numeric(colnames(beta_matrix)%in%colnames(coef_matrix))==1)] with the original beta_matrix, we see that the coefficients correspond to the variables correspond to X2 and X15.

Now checking if my two variable model also contains the same coefficients

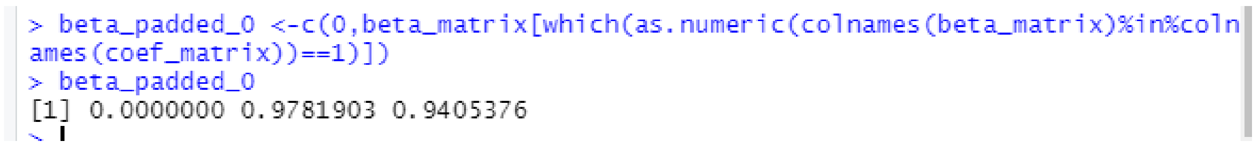


```
> coef(regfit.full, 2)
(Intercept)          x2         x15
-0.01053119  1.03416531  0.91020794
> |
```

So the for loop has extracted the coefficients all the variables from the original dataset that matches with those in the p variable model.

Now the original data set does not contain any intercept term so here 0 is padded to the original dataset after extracting the matching column names to ease my further calculation

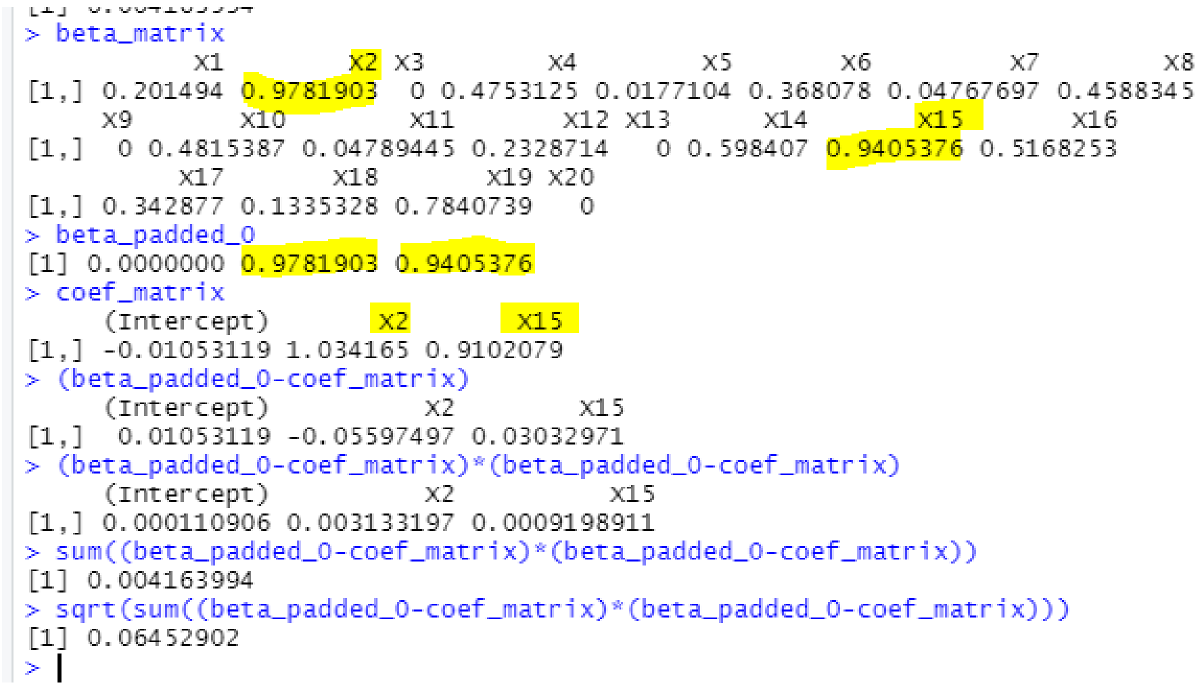
c(0,beta_matrix[which(as.numeric(colnames(beta_matrix)%in%colnames(coef_matrix))==1)])



```
> beta_padded_0 <-c(0,beta_matrix[which(as.numeric(colnames(beta_matrix)%in%coln
ames(coef_matrix))==1)])
> beta_padded_0
[1] 0.0000000 0.9781903 0.9405376
```

5) coef_diff <- sqrt(sum((beta_padded_0-coef_matrix)*(beta_padded_0-coef_matrix)))

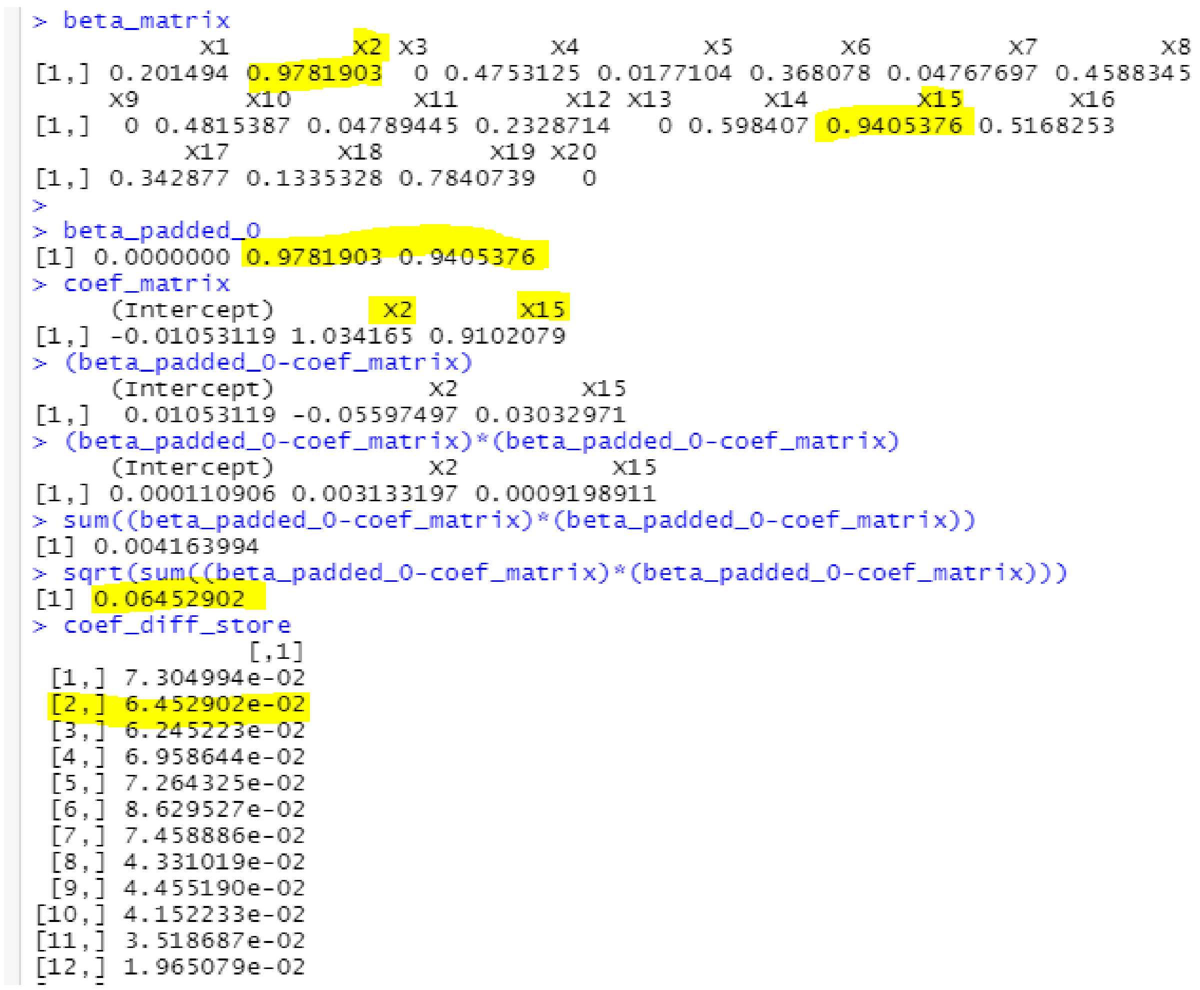
Let’s understand this command step by step. If we look at the question no 1 part g) compare the same with this command.



```
> beta_matrix
           x1        x2 x3        x4        x5       x6         x7        x8
[1,] 0.201494 0.9781903  0 0.4753125 0.0177104 0.368078 0.04767697 0.4588345
     x9       x10        x11       x12 x13      x14       x15       x16
[1,]  0 0.4815387 0.04789445 0.2328714   0 0.598407 0.9405376 0.5168253
          x17       x18       x19 x20
[1,] 0.342877 0.1335328 0.7840739   0
> beta_padded_0
[1] 0.0000000 0.9781903 0.9405376
> coef_matrix
     (Intercept)       x2       x15
[1,] -0.01053119 1.034165 0.9102079
> (beta_padded_0-coef_matrix)
     (Intercept)          x2        x15
[1,]  0.01053119 -0.05597497 0.03032971
> (beta_padded_0-coef_matrix)*(beta_padded_0-coef_matrix)
     (Intercept)          x2          x15
[1,] 0.000110906 0.003133197 0.0009198911
> sum((beta_padded_0-coef_matrix)*(beta_padded_0-coef_matrix))
[1] 0.004163994
> sqrt(sum((beta_padded_0-coef_matrix)*(beta_padded_0-coef_matrix)))
[1] 0.06452902
> 
```

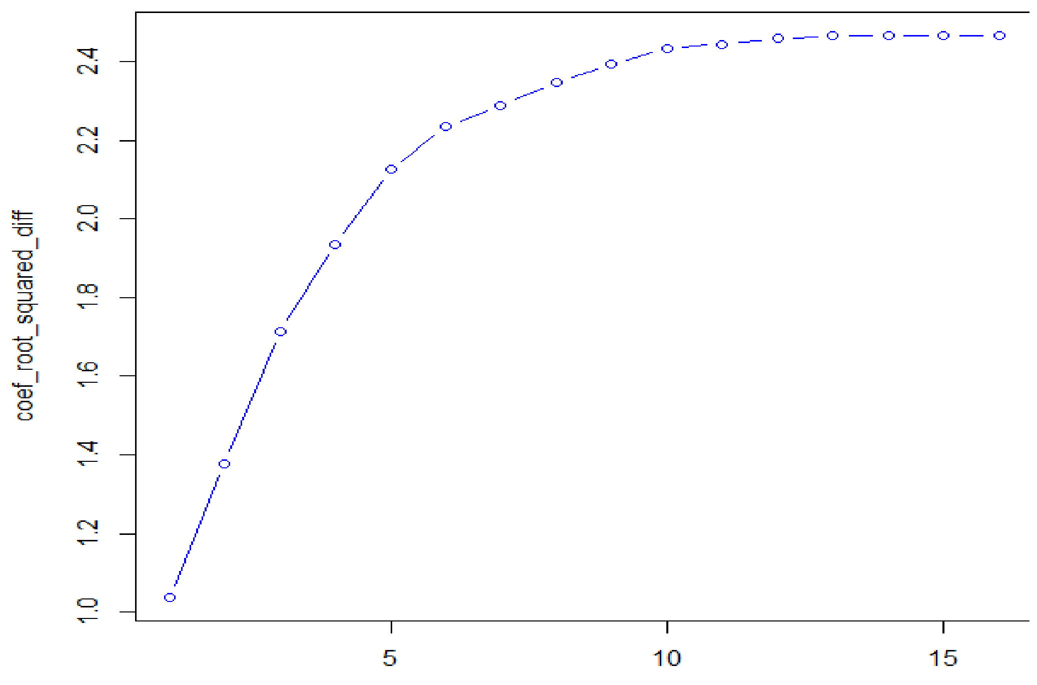
6) So till now its understood how the for loop is going the calculate the square root of sum of differences of coefficients of a p variable model and the corresponding coefficients in the original dataset.

7)The same steps will be repeated for each of the p variable model and the result for each of the p variable model would be stored in in a matrix coef_diff_store.Lets understand once again step by step for a two variable model.



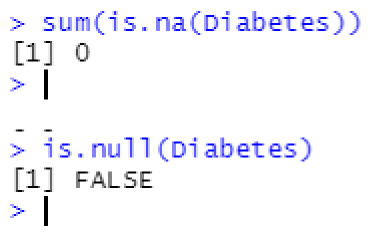
```
> beta_matrix
           x1        x2 x3        x4        x5       x6         x7        x8
[1,] 0.201494 0.9781903  0 0.4753125 0.0177104 0.368078 0.04767697 0.4588345
     x9       x10        x11       x12 x13      x14       x15       x16
[1,]  0 0.4815387 0.04789445 0.2328714   0 0.598407 0.9405376 0.5168253
          x17       x18       x19 x20
[1,] 0.342877 0.1335328 0.7840739   0
>
> beta_padded_0
[1] 0.0000000 0.9781903 0.9405376
> coef_matrix
     (Intercept)       x2       x15
[1,] -0.01053119 1.034165 0.9102079
> (beta_padded_0-coef_matrix)
     (Intercept)          x2        x15
[1,]  0.01053119 -0.05597497 0.03032971
> (beta_padded_0-coef_matrix)*(beta_padded_0-coef_matrix)
     (Intercept)          x2          x15
[1,] 0.000110906 0.003133197 0.0009198911
> sum((beta_padded_0-coef_matrix)*(beta_padded_0-coef_matrix))
[1] 0.004163994
> sqrt(sum((beta_padded_0-coef_matrix)*(beta_padded_0-coef_matrix)))
[1] 0.06452902
> coef_diff_store
              [,1]
 [1,] 7.304994e-02
 [2,] 6.452902e-02
 [3,] 6.245223e-02
 [4,] 6.958644e-02
 [5,] 7.264325e-02
 [6,] 8.629527e-02
 [7,] 7.458886e-02
 [8,] 4.331019e-02
 [9,] 4.455190e-02
[10,] 4.152233e-02
[11,] 3.518687e-02
[12,] 1.965079e-02
```

It seems with the increase in the no of p variables the root squared sum difference between the p variable models coefficients and the original Beta value increases as shown in the below graph



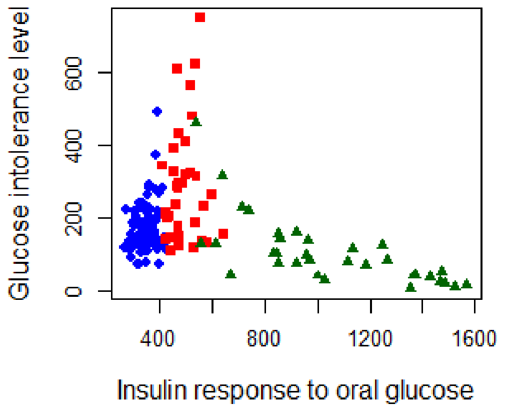
Task 3 --- Diabetes

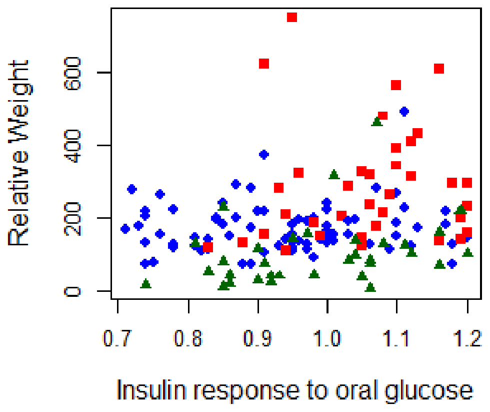
Part A)Initial checks for missing and null values

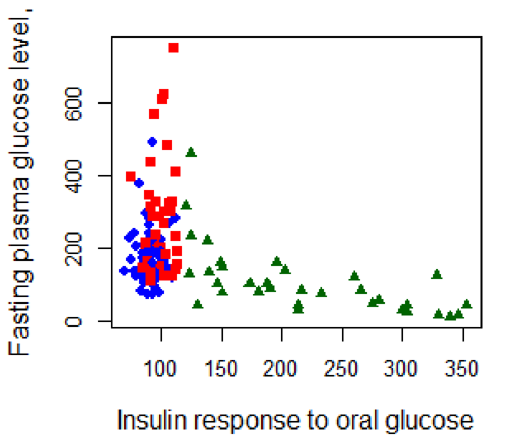


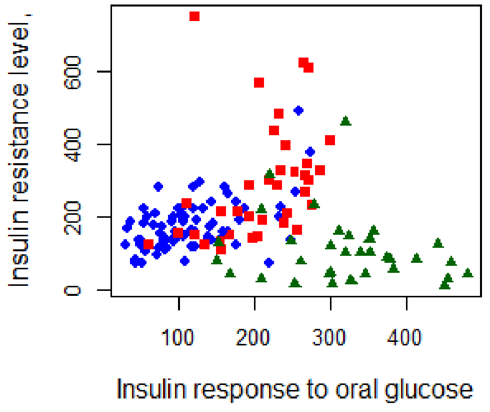
```
> sum(is.na(Diabetes))
[1] 0
> 

> is.null(Diabetes)
[1] FALSE
> 
```

Intest ~ All variables

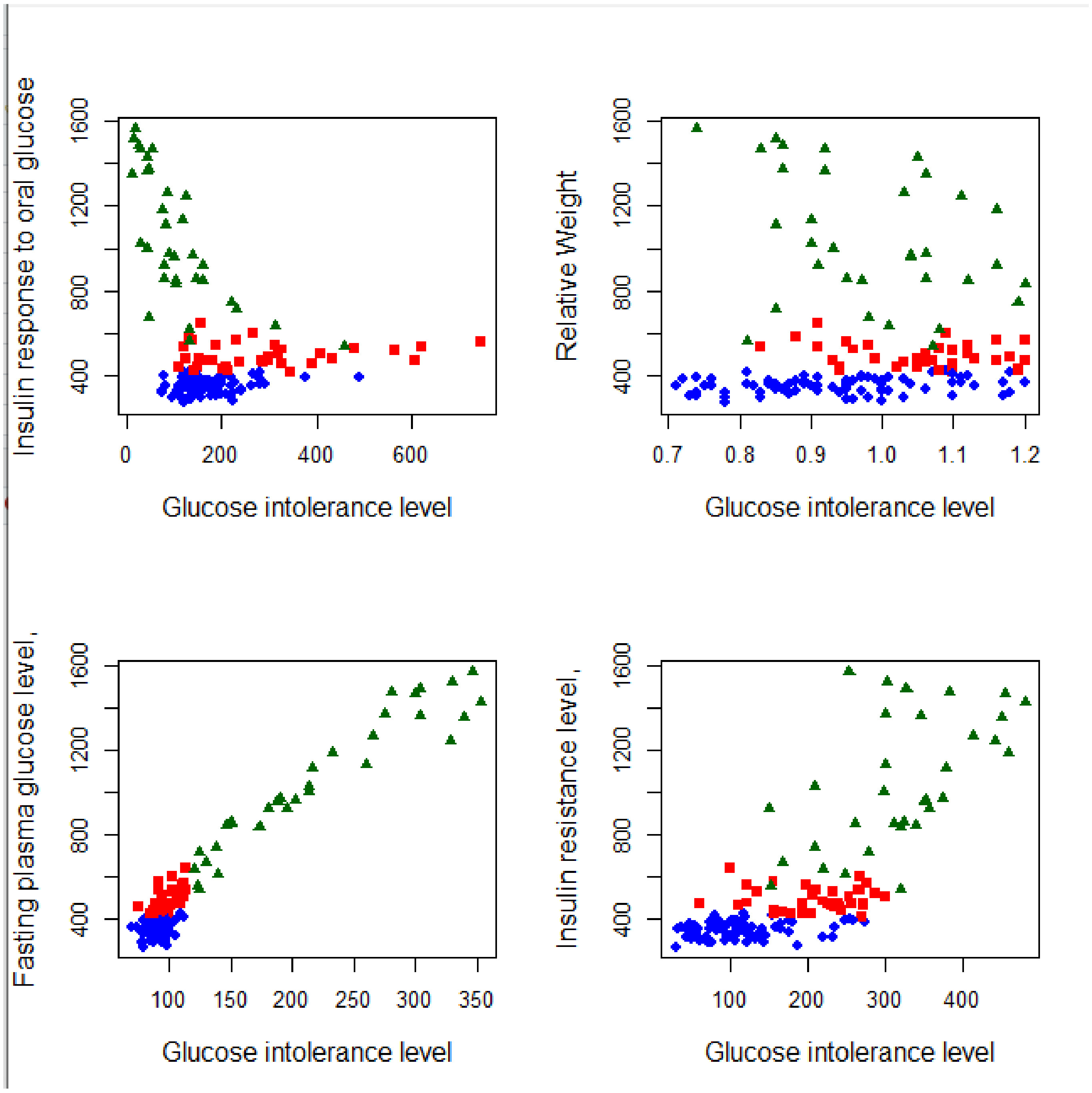




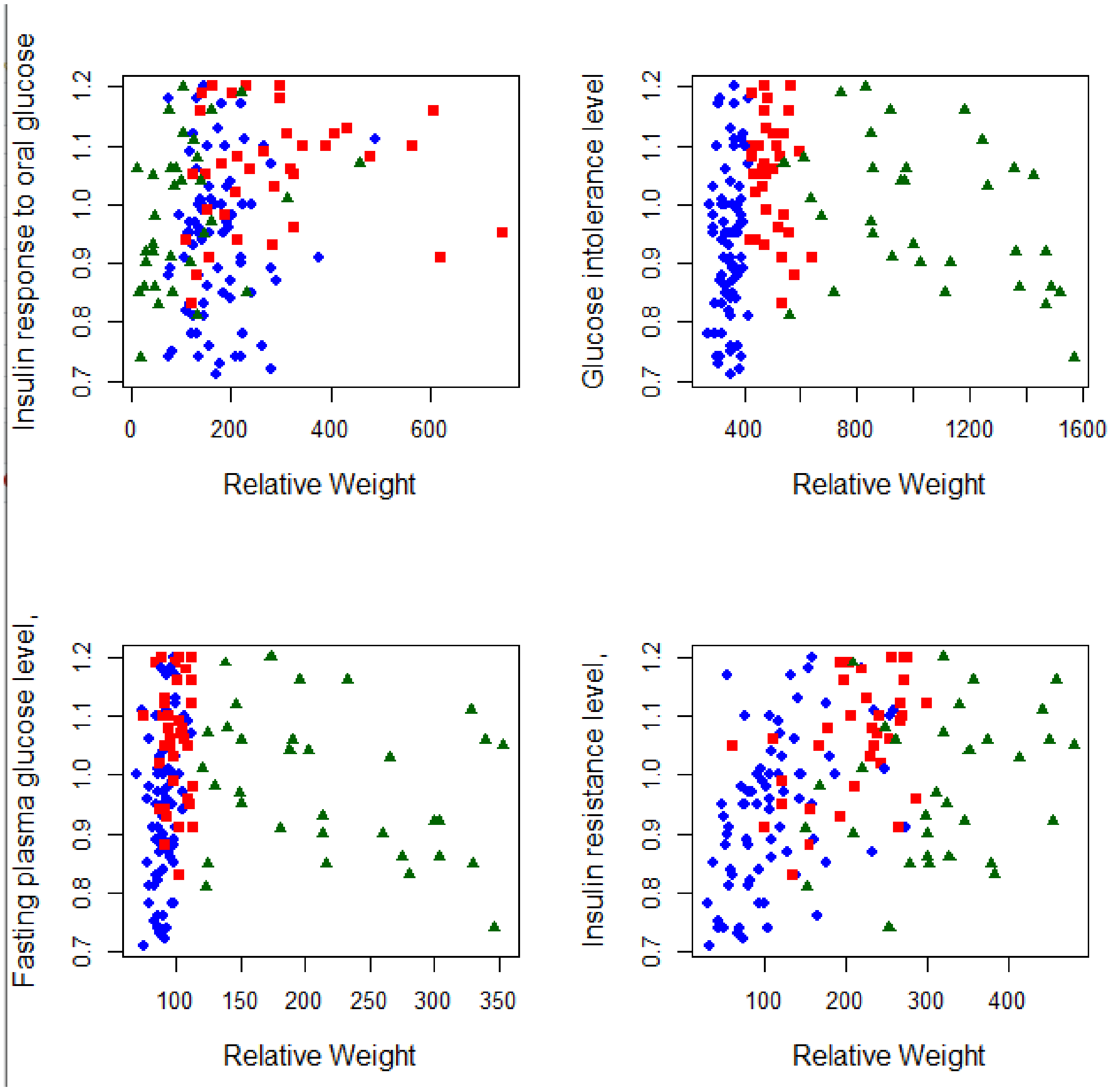




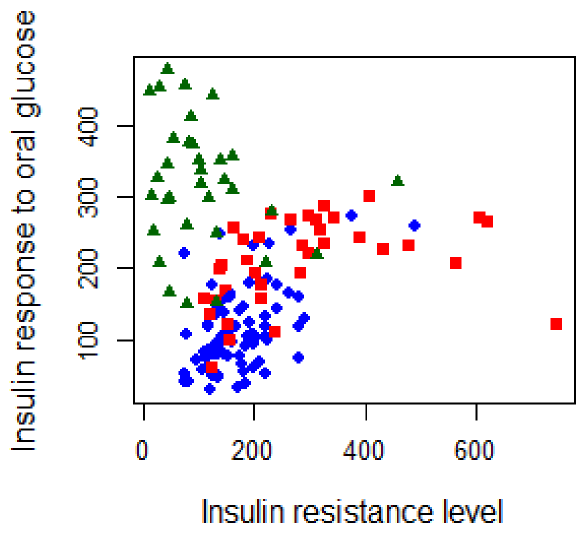
Glutest ~ All variables

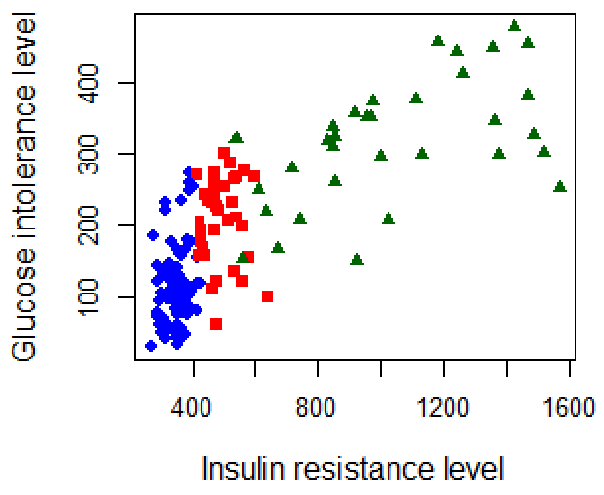


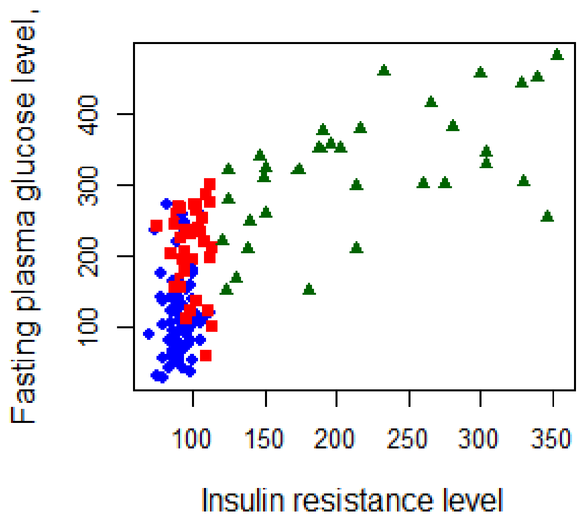
Relative weight versus all variables

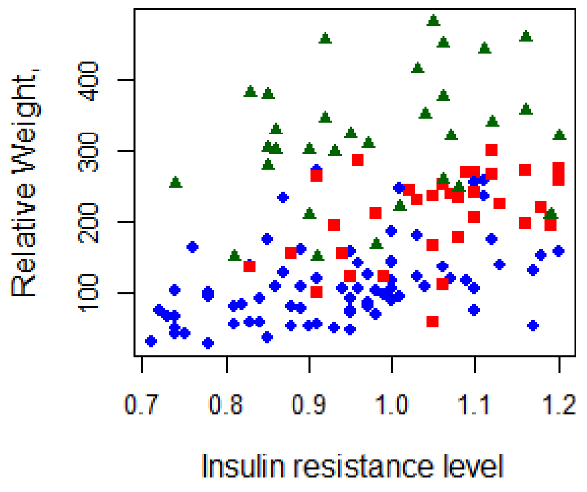


Insulin Resistance level(SSPG) vs all variables



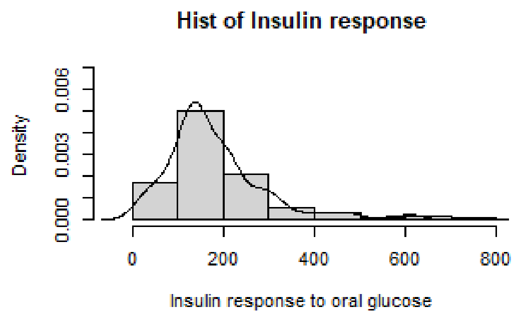


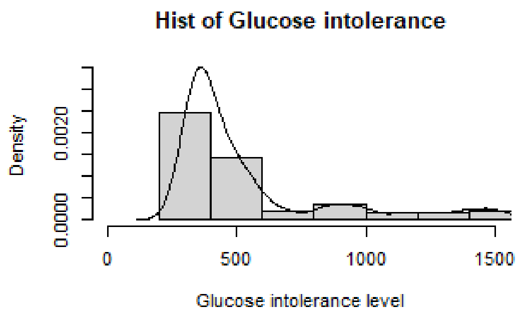


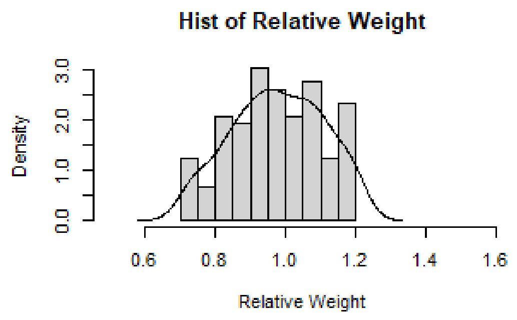


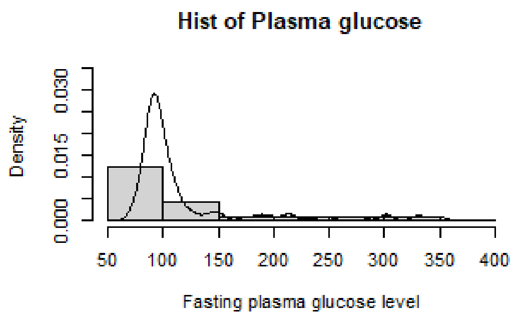
A fundamental property of multivariate normal distribution states that if the data has multivariate normal distribution, then each of its variables has a univariate normal distribution however the vice versa might not be true

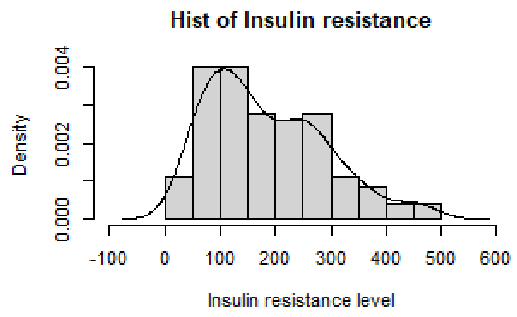
From the density distribution of each of the variables given below it seems that mostly the variables are normally distributed hence overall distribution of data can be multivariate normal



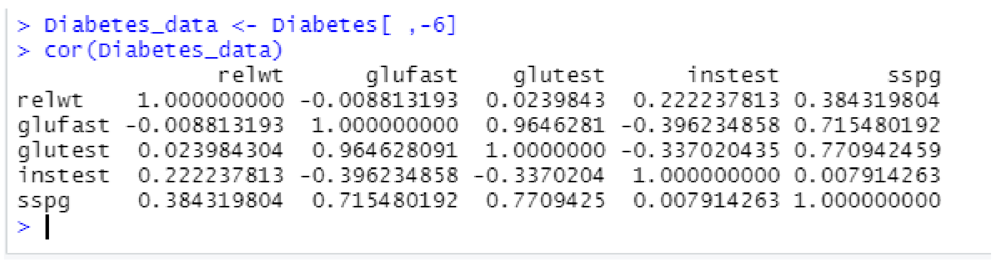








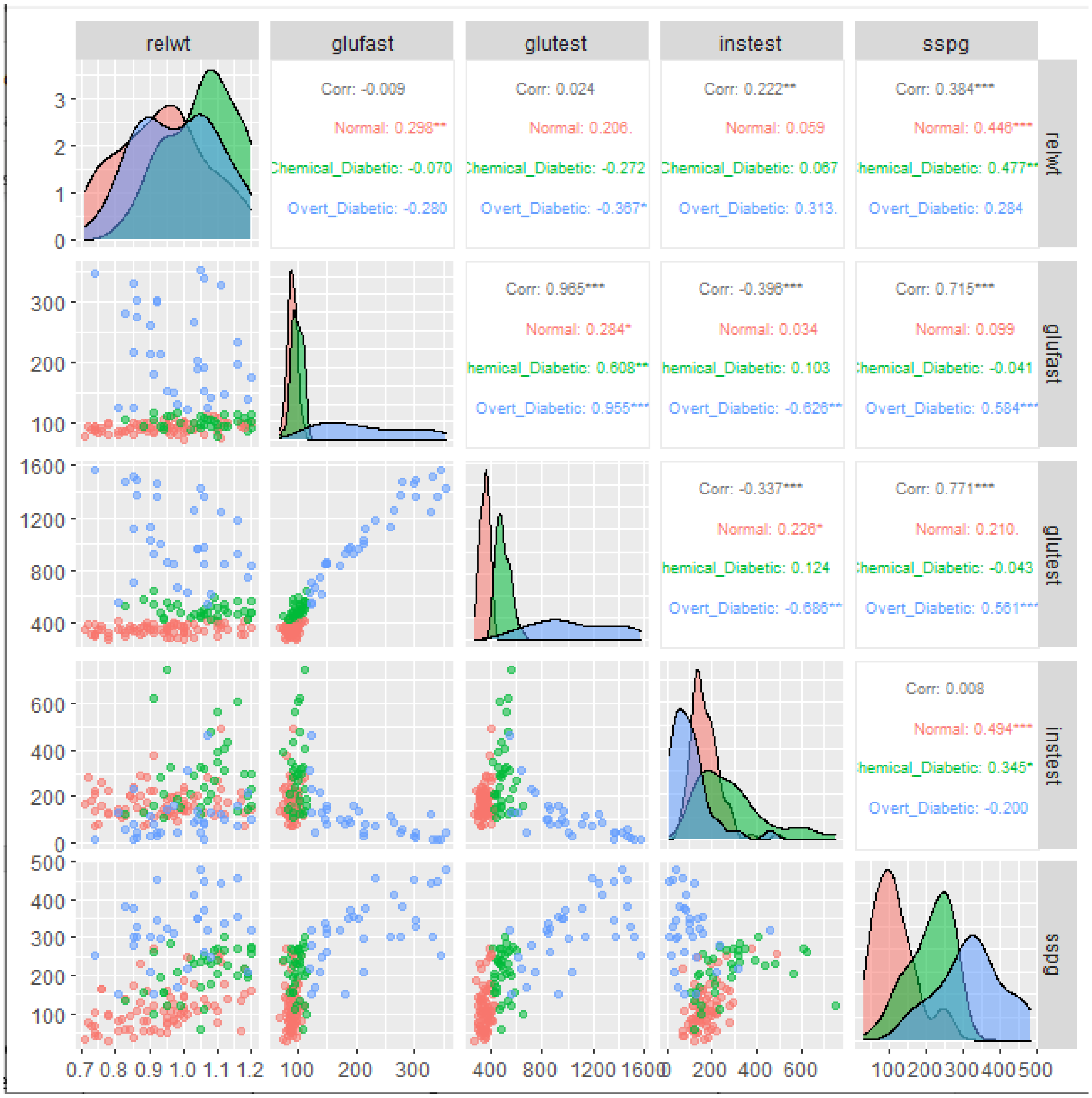
Plotting the correlation of data:



```
> Diabetes_data <- Diabetes[ ,-6]
> cor(Diabetes_data)
              relwt      glufast    glutest      instest        sspg
relwt    1.000000000 -0.008813193  0.0239843  0.222237813 0.384319804
glufast -0.008813193  1.000000000  0.9646281 -0.396234858 0.715480192
glutest  0.023984304  0.964628091  1.0000000 -0.337020435 0.770942459
instest  0.222237813 -0.396234858 -0.3370204  1.000000000 0.007914263
sspg     0.384319804  0.715480192  0.7709425  0.007914263 1.000000000
> 
```

From the correlation data it seem that sspg - insulin resistance is highly correlated to glufast - plasma glucose level and glutest - glucose intolerance which is obvious because if the insulin secretion is low then naturally the glucose level inside body would be high. Similarly, glucose intolerance might vary person to person, however if the patient on the top of that is having low insulin secretion, then quite naturally his/her glucose intolerance level would be reflected high which might not have been the case provided his/her insulin secretion is normal.

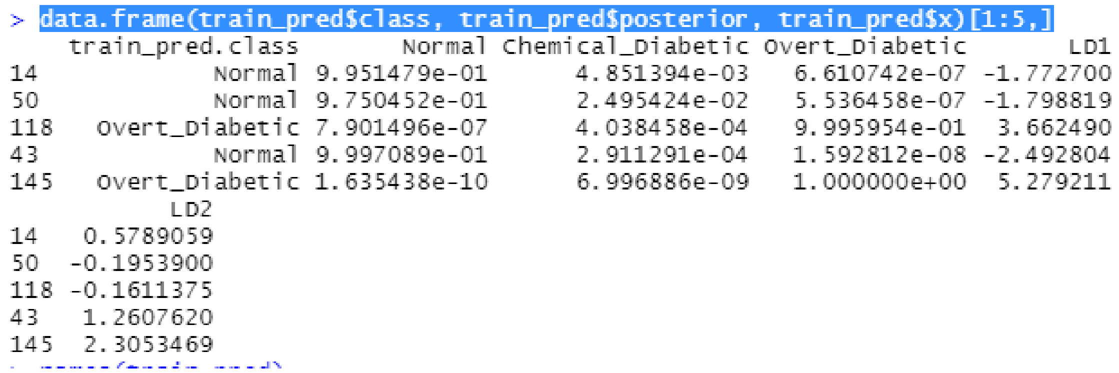
GGpairs plot all the feature variables of Diabetes



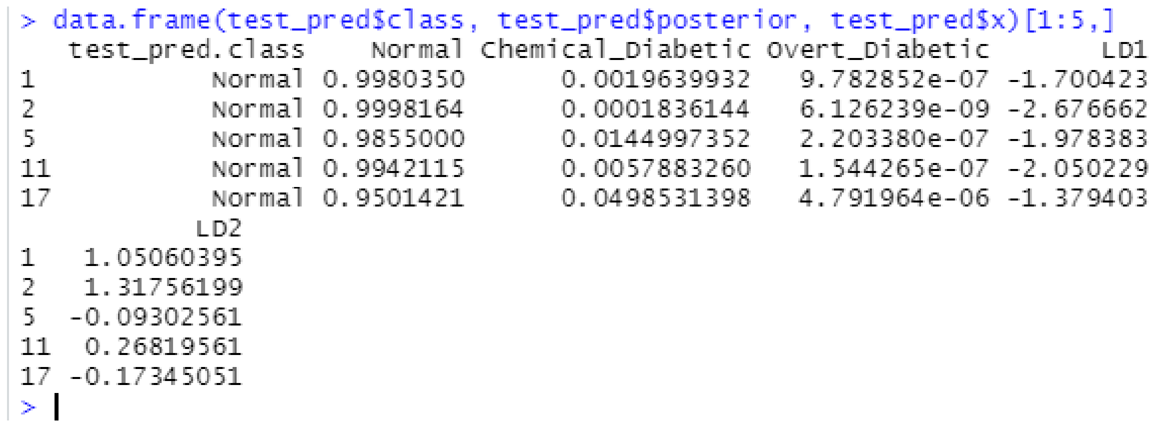
If we take any of the above ggpairs plot and compare all the variables with respect to a particular diabetic group, it seems that for certain groups say Normal and Chemical Diabetic these variables are not varying much compared to what we see for Over_Diabetic. Maximum variance between the variables is observed for Diabetic group Over_Diabetic. Though there is variation amongst the feature variables of the dataset for other diabetic groups, but it doesn't seem to be as much as reflected from the plot above as for diabetic group Over_Diabetic. So, from an overall analysis of above plot I come to form the hypotheses that the covariances are different for each of this diabetic group.

Task 3 Partb)

Fitting and predicting test and training data with LDA.

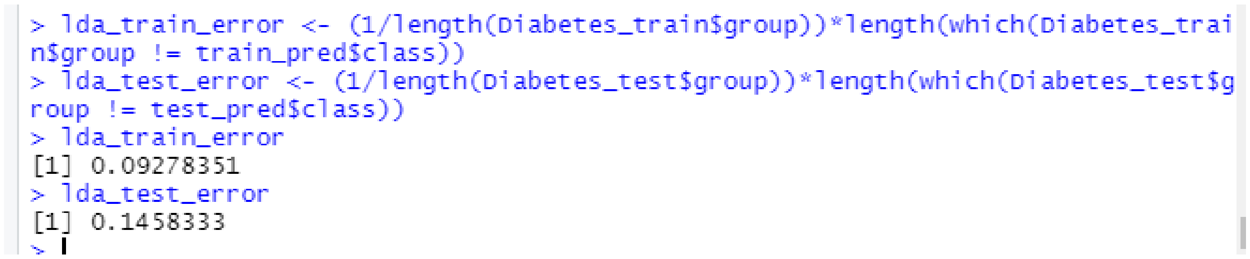


```
> data.frame(train_pred$class, train_pred$posterior, train_pred$x)[1:5,]
    train_pred.class       Normal Chemical_Diabetic Overt_Diabetic       LD1
14            Normal 9.951479e-01      4.851394e-03   6.610742e-07 -1.772700
50            Normal 9.750452e-01      2.495424e-02   5.536458e-07 -1.798819
118   Overt_Diabetic 7.901496e-07      4.038458e-04   9.995954e-01  3.662490
43            Normal 9.997089e-01      2.911291e-04   1.592812e-08 -2.492804
145   Overt_Diabetic 1.635438e-10      6.996886e-09   1.000000e+00  5.279211
           LD2
14   0.5789059
50  -0.1953900
118 -0.1611375
43   1.2607620
145  2.3053469
```



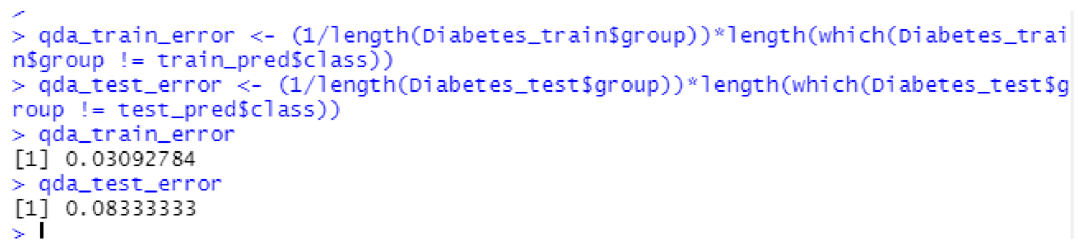
```
> data.frame(test_pred$class, test_pred$posterior, test_pred$x)[1:5,]
   test_pred.class    Normal Chemical_Diabetic Overt_Diabetic       LD1
1           Normal 0.9980350      0.0019639932   9.782852e-07 -1.700423
2           Normal 0.9998164      0.0001836144   6.126239e-09 -2.676662
5           Normal 0.9855000      0.0144997352   2.203380e-07 -1.978383
11          Normal 0.9942115      0.0057883260   1.544265e-07 -2.050229
17          Normal 0.9501421      0.0498531398   4.791964e-06 -1.379403
           LD2
1   1.05060395
2   1.31756199
5  -0.09302561
11  0.26819561
17 -0.17345051
> 
```

Test and Train Misclassification error with LDA.



```
> lda_train_error <- (1/length(Diabetes_train$group))*length(which(Diabetes_trai
n$group != train_pred$class))
> lda_test_error <- (1/length(Diabetes_test$group))*length(which(Diabetes_test$g
roup != test_pred$class))
> lda_train_error
[1] 0.09278351
> lda_test_error
[1] 0.1458333
> 
```

Test and Train Misclassification error with QDA.



```
> qda_train_error <- (1/length(Diabetes_train$group))*length(which(Diabetes_trai
n$group != train_pred$class))
> qda_test_error <- (1/length(Diabetes_test$group))*length(which(Diabetes_test$g
roup != test_pred$class))
> qda_train_error
[1] 0.03092784
> qda_test_error
[1] 0.08333333
> 
```

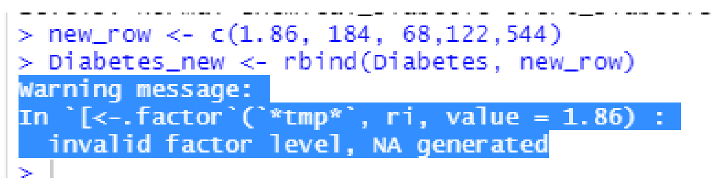
From the above misclassification error values it seems the test and training error is low for QDA compared to LDA with Diabetes data hence QDA is performing better than LDA for this dataset.

Task3 Part C)

Added a new row as per the question a new row was added to the Diabetes data with the values mentioned in part c of question 3

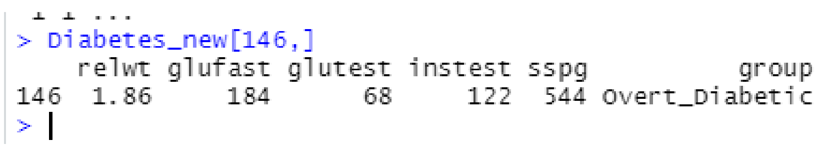
Though group was the response variable to be predicted however it was to avoid the warning message as mentioned below

Warning message:

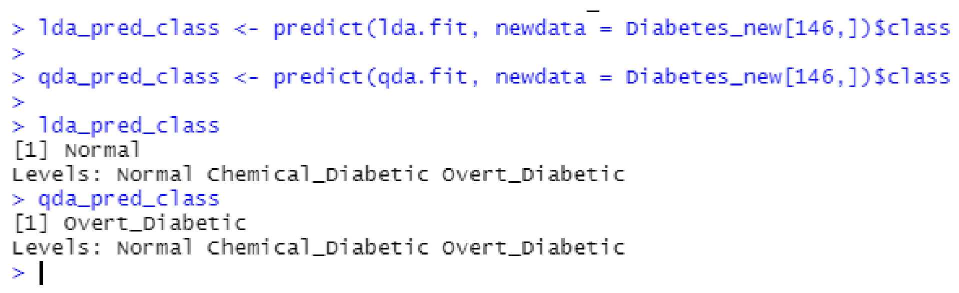


```
> new_row <- c(1.86, 184, 68,122,544)
> Diabetes_new <- rbind(Diabetes, new_row)
Warning message:
In `[<-.factor`(`*tmp*`, ri, value = 1.86) :
  invalid factor level, NA generated
> 
```

New data:



```
> Diabetes_new[146,]
    relwt glufast glutest instest sspg          group
146  1.86     184      68     122  544 Overt_Diabetic
> 
```

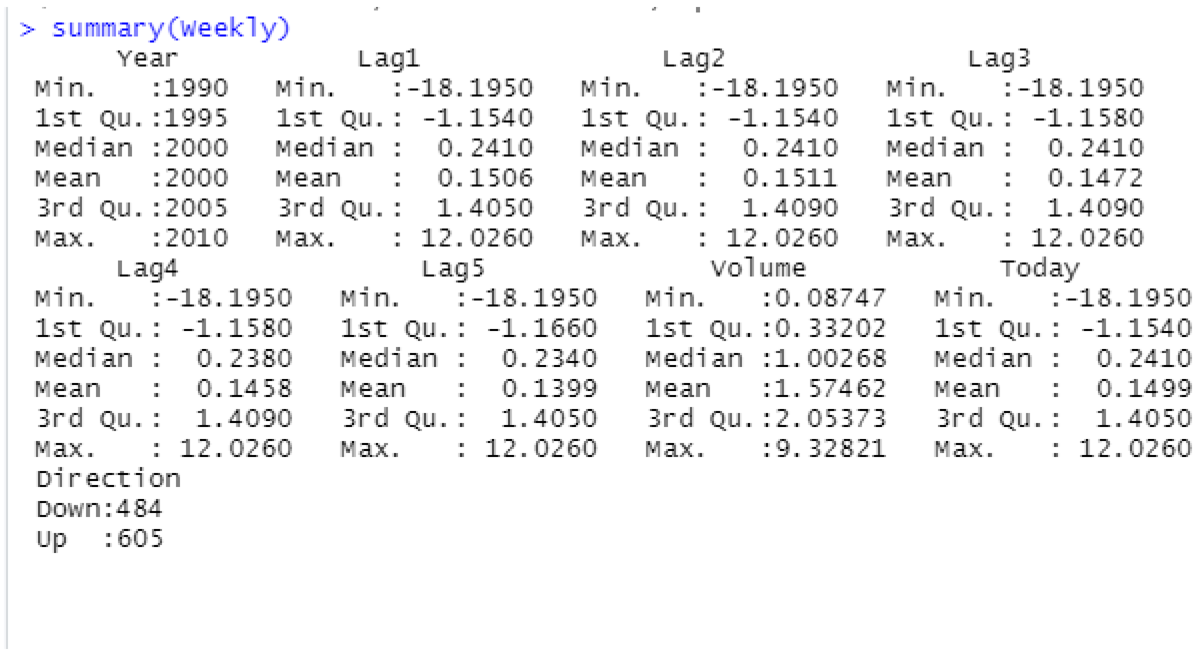


```
> lda_pred_class <- predict(lda.fit, newdata = Diabetes_new[146,])$class
>
> qda_pred_class <- predict(qda.fit, newdata = Diabetes_new[146,])$class
>
> lda_pred_class
[1] Normal
Levels: Normal Chemical_Diabetic Overt_Diabetic
> qda_pred_class
[1] Overt_Diabetic
Levels: Normal Chemical_Diabetic Overt_Diabetic
> 
```

From the above screenshot it seems that LDA predicted “Normal” as the group for the new data and QDA predicted Overt Diabetic for the new data.

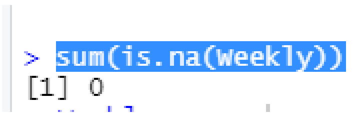
Task 2---------

Part a) Summary of the Weekly data



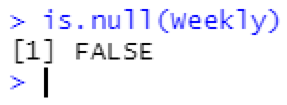
```
> summary(Weekly)
      Year           Lag1               Lag2               Lag3
 Min.   :1990   Min.   :-18.1950   Min.   :-18.1950   Min.   :-18.1950
 1st Qu.:1995   1st Qu.: -1.1540   1st Qu.: -1.1540   1st Qu.: -1.1580
 Median :2000   Median :  0.2410   Median :  0.2410   Median :  0.2410
 Mean   :2000   Mean   :  0.1506   Mean   :  0.1511   Mean   :  0.1472
 3rd Qu.:2005   3rd Qu.:  1.4050   3rd Qu.:  1.4090   3rd Qu.:  1.4090
 Max.   :2010   Max.   : 12.0260   Max.   : 12.0260   Max.   : 12.0260
      Lag4               Lag5              Volume            Today
 Min.   :-18.1950   Min.   :-18.1950   Min.   :0.08747   Min.   :-18.1950
 1st Qu.: -1.1580   1st Qu.: -1.1660   1st Qu.:0.33202   1st Qu.: -1.1540
 Median :  0.2380   Median :  0.2340   Median :1.00268   Median :  0.2410
 Mean   :  0.1458   Mean   :  0.1399   Mean   :1.57462   Mean   :  0.1499
 3rd Qu.:  1.4090   3rd Qu.:  1.4050   3rd Qu.:2.05373   3rd Qu.:  1.4050
 Max.   : 12.0260   Max.   : 12.0260   Max.   :9.32821   Max.   : 12.0260
 Direction
 Down:484
 Up  :605
```

No NA values



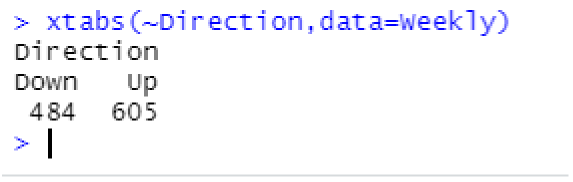
```
> sum(is.na(Weekly))
[1] 0
```

No Null values



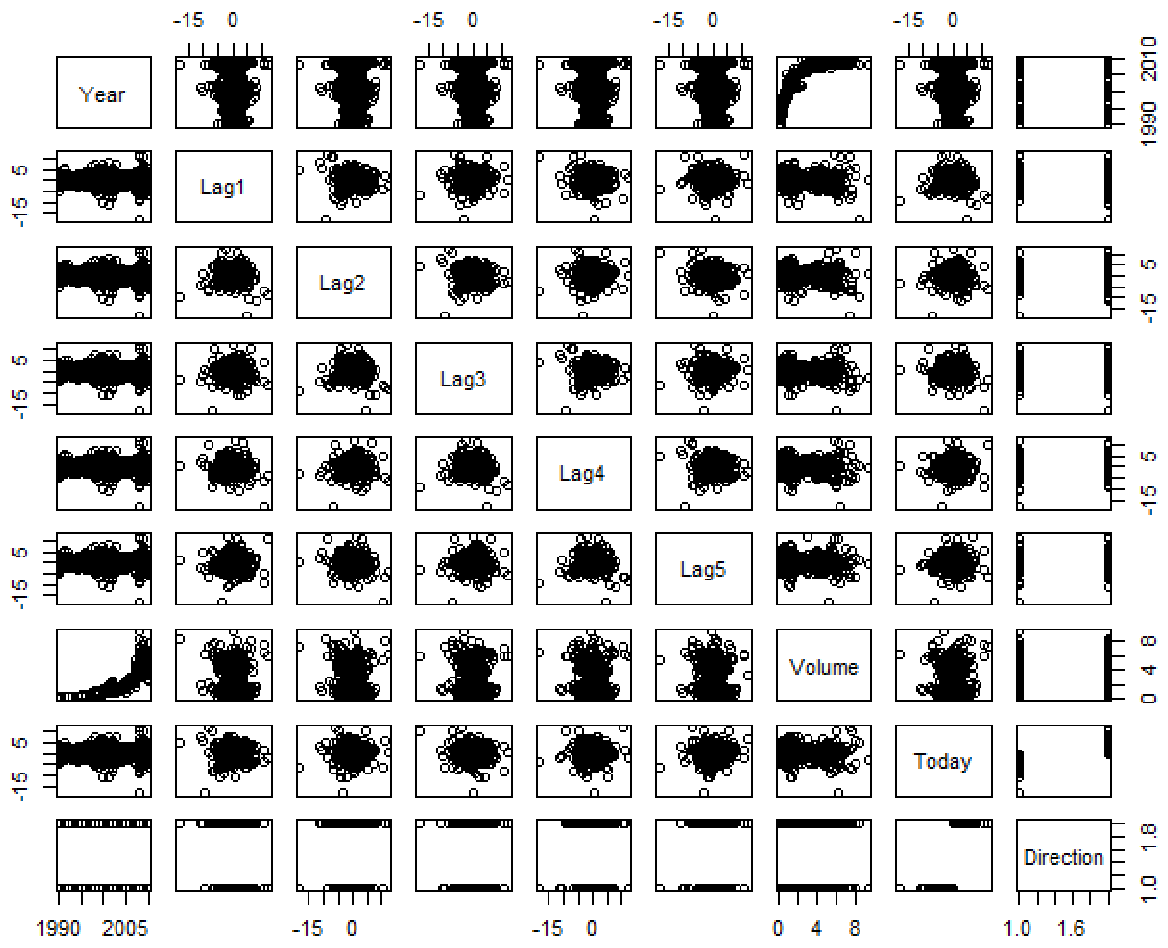
```
> is.null(Weekly)
[1] FALSE
> 
```

A tally of Down and Up from Direction Categorical variable



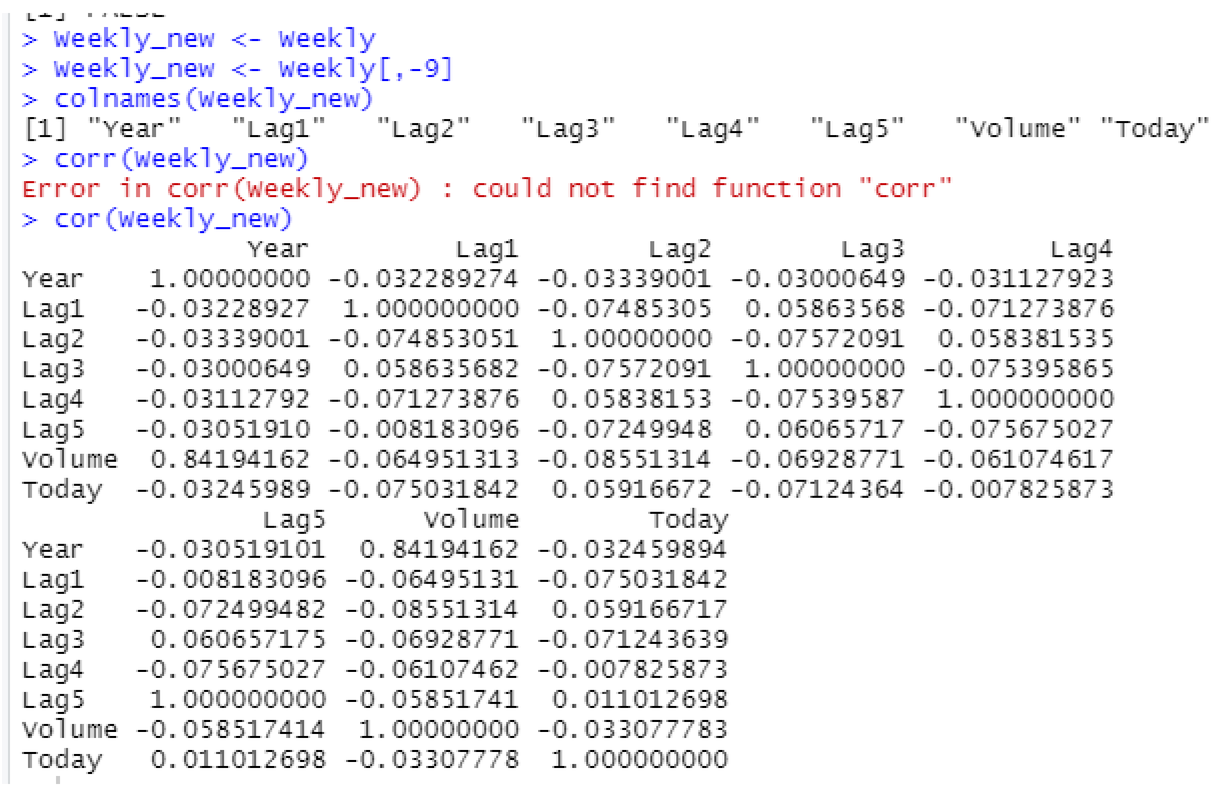
```
> xtabs(~Direction,data=Weekly)
Direction
Down   Up
 484  605
> 
```

Pairs Plot:



From the initial summary statistics and scatterplot matrix, there don't appear to be any obvious patterns aside from the fact the volume of shares traded each week has grown quite a lot from 1990 to 2010. Looking more closely at the scatterplot of just volume over time, we can see that the number of shares traded each week has grown exponentially over the 21 years covered by the data.

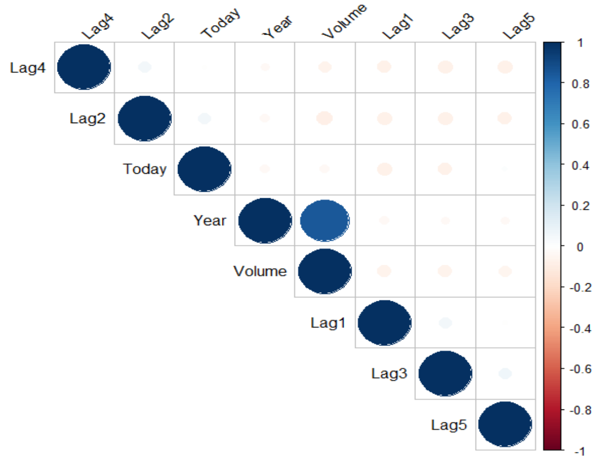
Correlation summary:



```
> weekly_new <- weekly
> weekly_new <- weekly[,-9]
> colnames(weekly_new)
[1] "Year"    "Lag1"    "Lag2"    "Lag3"    "Lag4"    "Lag5"    "Volume" "Today"
> corr(weekly_new)
Error in corr(weekly_new) : could not find function "corr"
> cor(weekly_new)
              Year         Lag1        Lag2        Lag3         Lag4
Year    1.00000000 -0.032289274 -0.03339001 -0.03000649 -0.031127923
Lag1   -0.03228927  1.000000000 -0.07485305  0.05863568 -0.071273876
Lag2   -0.03339001 -0.074853051  1.00000000 -0.07572091  0.058381535
Lag3   -0.03000649  0.058635682 -0.07572091  1.00000000 -0.075395865
Lag4   -0.03112792 -0.071273876  0.05838153 -0.07539587  1.000000000
Lag5   -0.03051910 -0.008183096 -0.07249948  0.06065717 -0.075675027
Volume  0.84194162 -0.064951313 -0.08551314 -0.06928771 -0.061074617
Today  -0.03245989 -0.075031842  0.05916672 -0.07124364 -0.007825873
               Lag5      Volume        Today
Year   -0.030519101  0.84194162 -0.032459894
Lag1   -0.008183096 -0.06495131 -0.075031842
Lag2   -0.072499482 -0.08551314  0.059166717
Lag3    0.060657175 -0.06928771 -0.071243639
Lag4   -0.075675027 -0.06107462 -0.007825873
Lag5    1.000000000 -0.05851741  0.011012698
Volume -0.058517414  1.00000000 -0.033077783
Today   0.011012698 -0.03307778  1.000000000
```

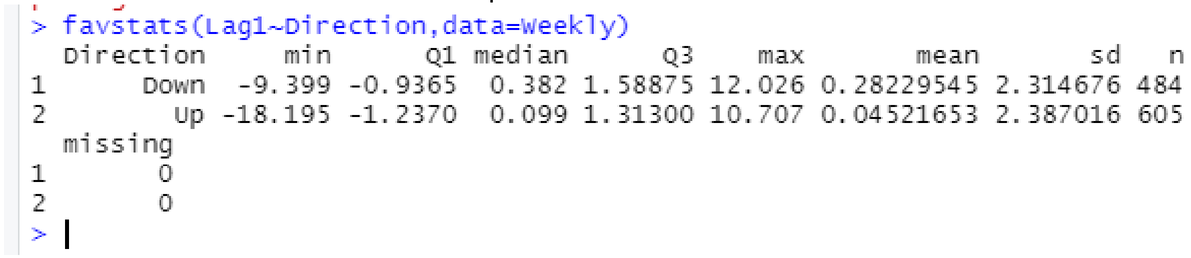
Looking at the last row, we can see that each of the lag variables is only correlated very weakly with today's returns. The sole substantial value of 0.842, between `Volume` and `Year`, aligns with the strong correlation we saw in the above scatterplot.

Graphical Visualization(Correlation Plot)



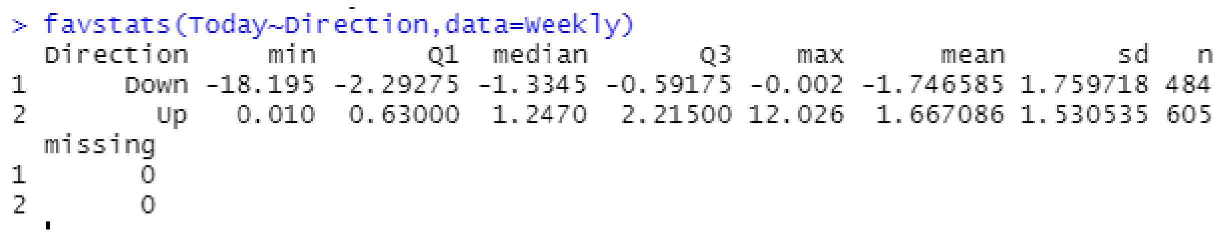
Again, the sole substantial value of 0.842, between `Volume` and `Year`, aligns with the strong correlation we saw in the above scatterplot.

From the correlation matrix above and plot of the same correlations between the variables. correlation we saw in the above scatterplot.



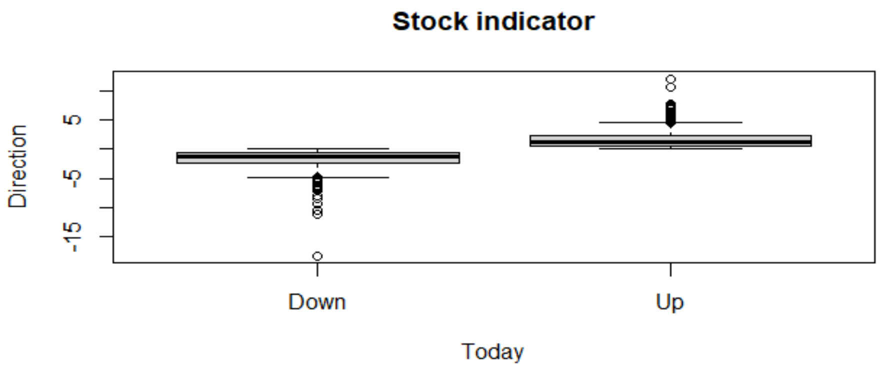
```
> favstats(Lag1~Direction,data=Weekly)
  Direction     min      Q1 median      Q3    max       mean       sd   n
1      Down  -9.399 -0.9365  0.382 1.58875 12.026 0.28229545 2.314676 484
2        Up -18.195 -1.2370  0.099 1.31300 10.707 0.04521653 2.387016 605
  missing
1       0
2       0
> 
```

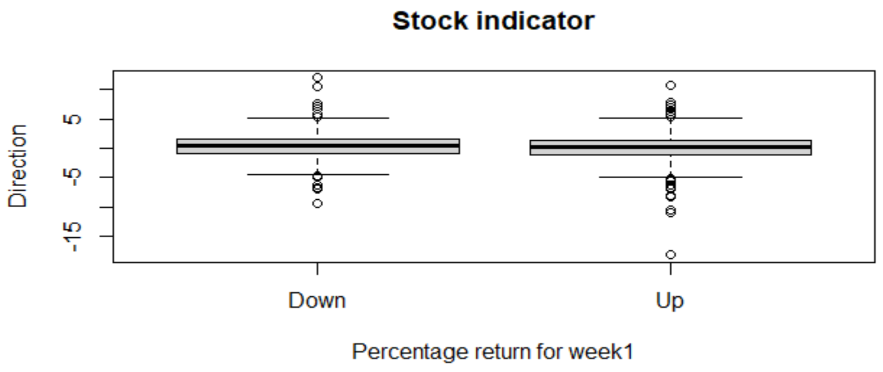
From the below screen shot again if we look at the mean it seems the market has mostly a positive return for todays data



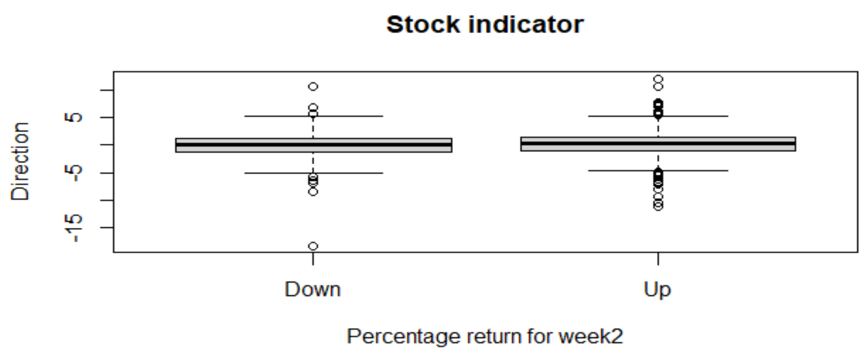
```
> favstats(Today~Direction,data=Weekly)
  Direction     min       Q1  median       Q3    max      mean       sd   n
1      Down -18.195 -2.29275 -1.3345 -0.59175 -0.002 -1.746585 1.759718 484
2        Up   0.010  0.63000  1.2470  2.21500 12.026  1.667086 1.530535 605
  missing
1       0
2       0
```

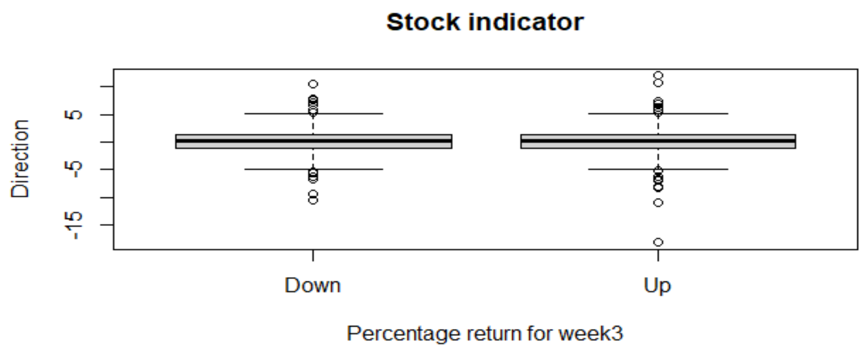
The below two box plots contains information for stock return for today and week1 respectively.



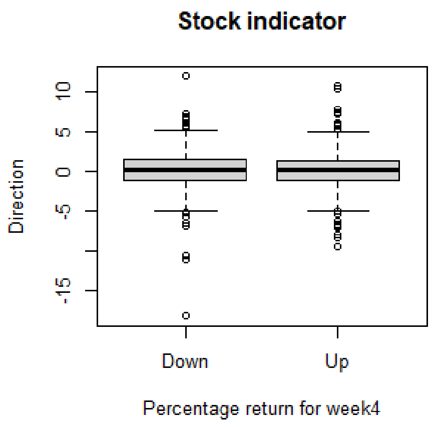


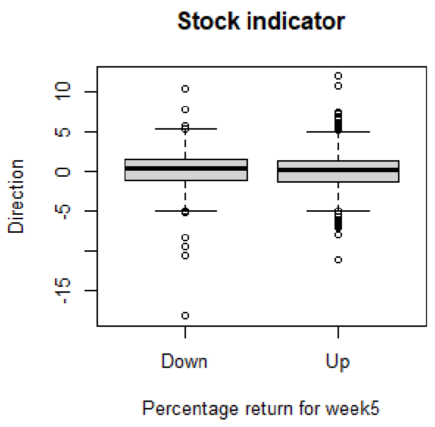
The below two box plots contains information for stock return for week2 and week3 respectively

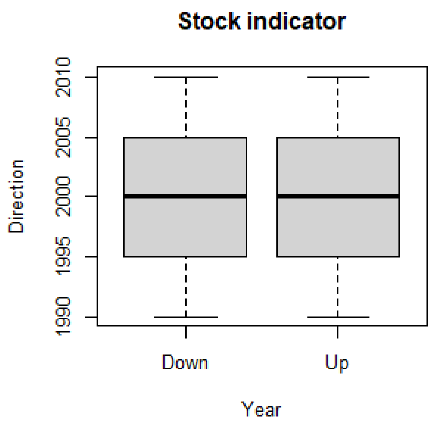




The below two box plots contains information for stock return for week4 and week5 and yearly data respectively.

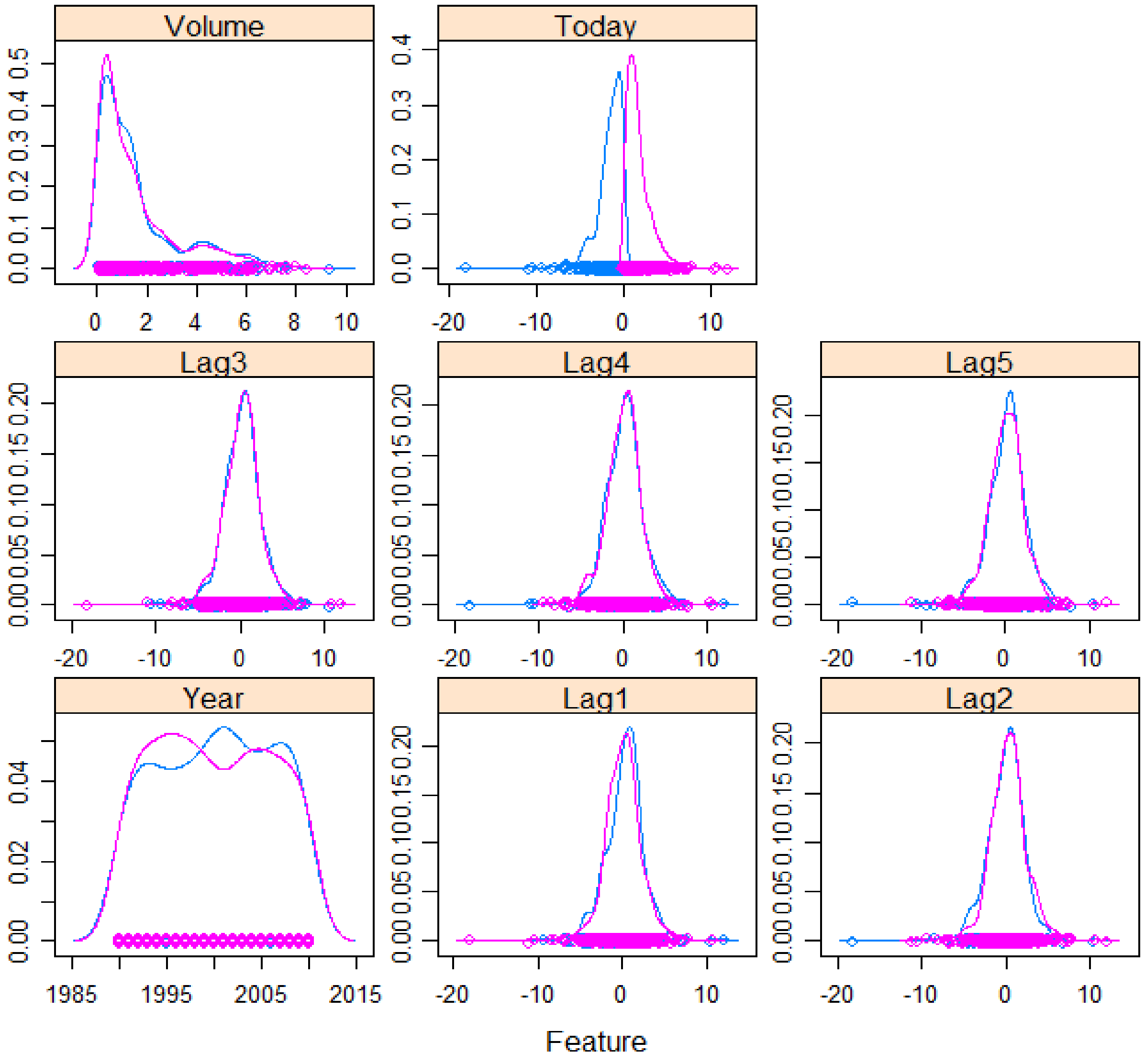




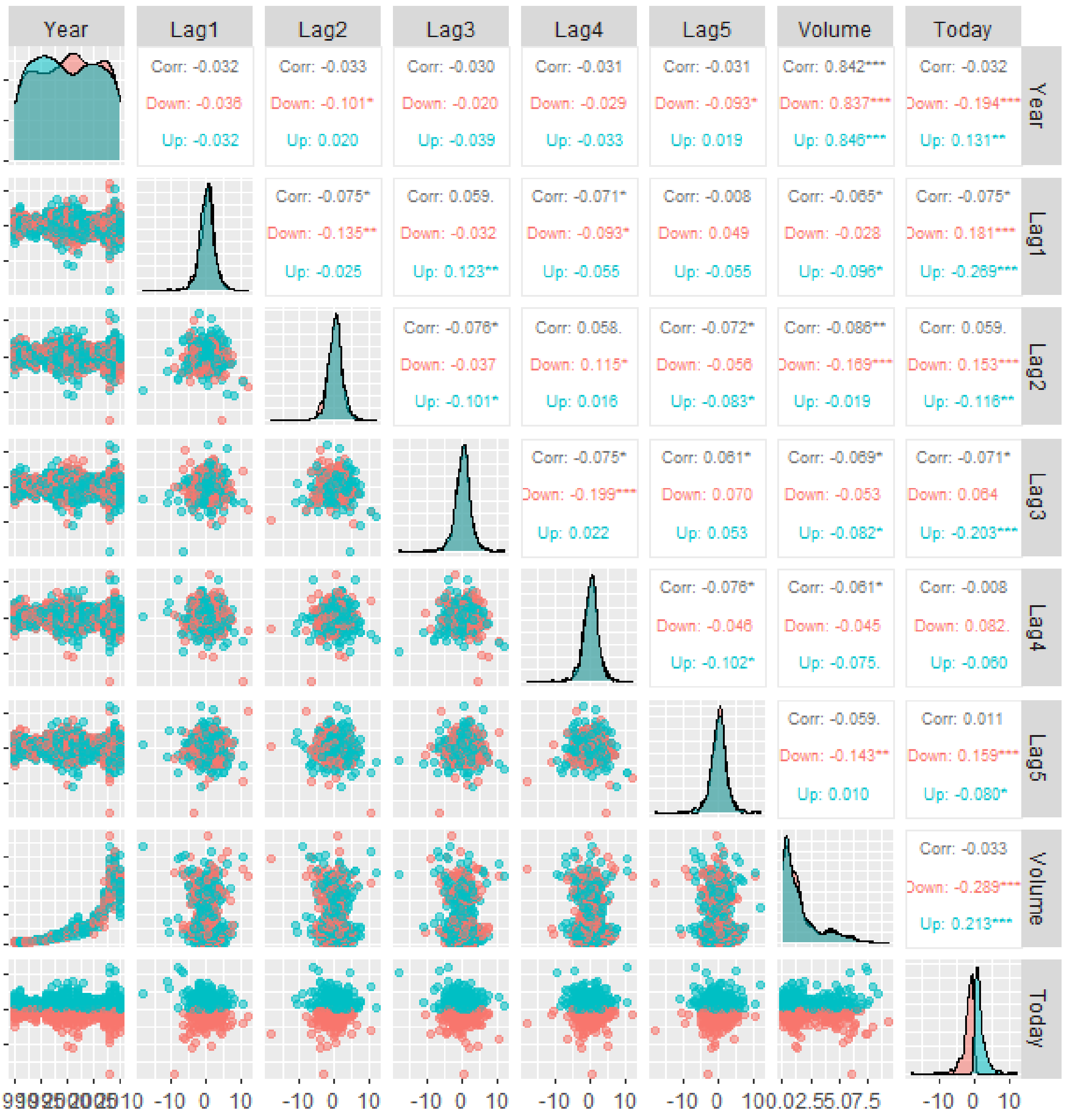


From the box plots of all the variables with respect to Direction it seems that there is almost no difference between positive and negative returns except today's data. The positive and negative returns are mostly cancelling each other except for today's data. For Today's data it seems mostly the returns were positive. For today's data the median of the positive returns lies above the median of negative returns.

Density Distribution of Each variable broken down by Direction Response Variable



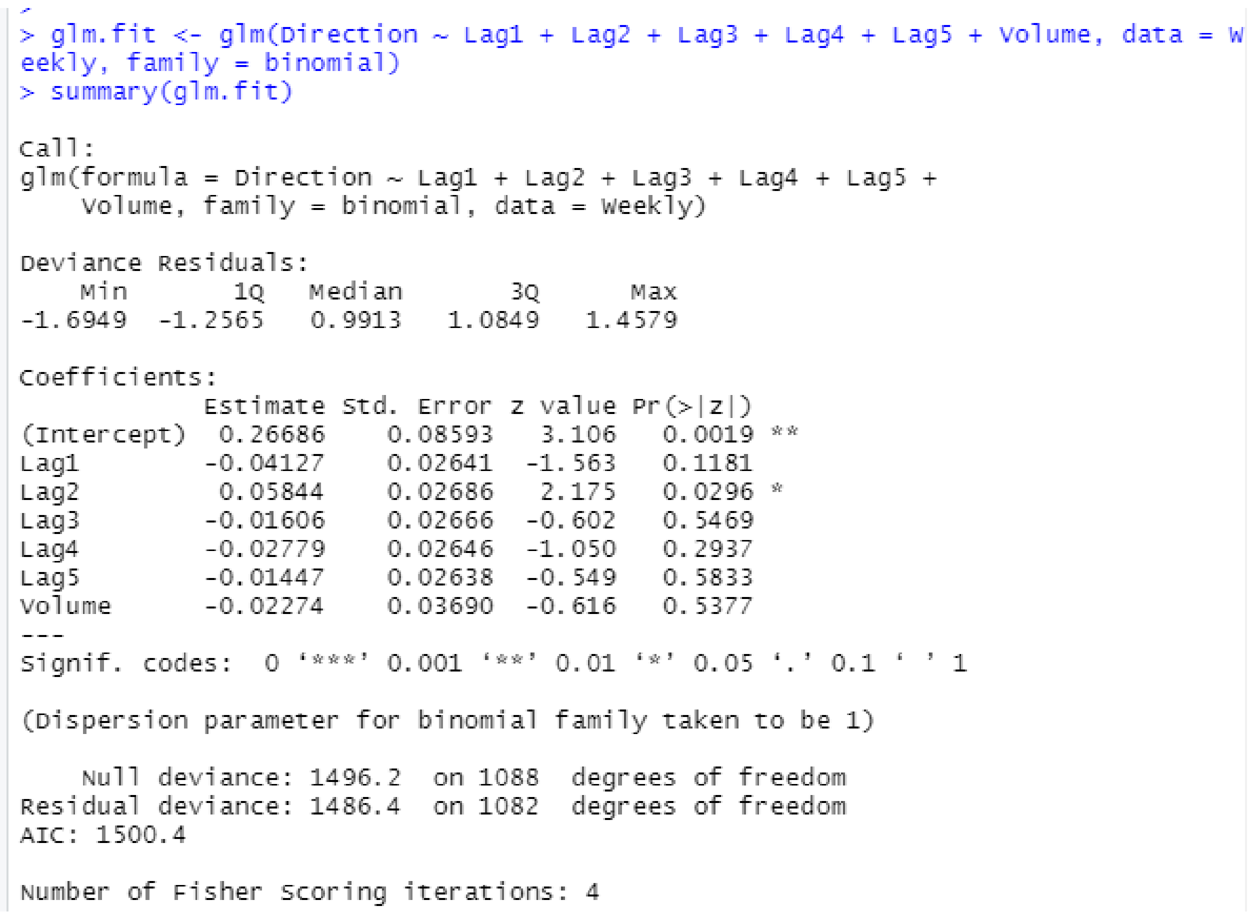
The density plot by Direction can help see the separation of Up and Down. It can also help to understand the overlap in Direction values for a variable. It seems that the Direction values overlap for all of these variables, meaning that it's hard to predict Up or Down based on just one or two variables.



We can observe that the variables Year and Volume are highly correlated, and it might look like the Volume increases quadratically with increasing Year. No other clear patterns is observed.

Task 2 b)

Printing the summary of Logistic regression fit model



```
> glm.fit <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume, data = W
eekly, family = binomial)
> summary(glm.fit)

Call:
glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + 
    Volume, family = binomial, data = Weekly)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-1.6949  -1.2565   0.9913   1.0849   1.4579  

Coefficients:
            Estimate Std. Error z value Pr(>|z|)   
(Intercept)  0.26686    0.08593   3.106   0.0019 **
Lag1        -0.04127    0.02641  -1.563   0.1181   
Lag2         0.05844    0.02686   2.175   0.0296 * 
Lag3        -0.01606    0.02666  -0.602   0.5469   
Lag4        -0.02779    0.02646  -1.050   0.2937   
Lag5        -0.01447    0.02638  -0.549   0.5833   
Volume      -0.02274    0.03690  -0.616   0.5377   
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1496.2  on 1088  degrees of freedom
Residual deviance: 1486.4  on 1082  degrees of freedom
AIC: 1500.4

Number of Fisher Scoring iterations: 4
```

summary() returns the estimate, standard errors, z-score, and p-values on each of the coefficients. It seems like none of the coefficients are significant here. It also gives the null deviance (the deviance just for the mean) and the residual deviance (the deviance for the model with all the predictors). There's a very small difference between the 2, along with 6 degrees of freedom.

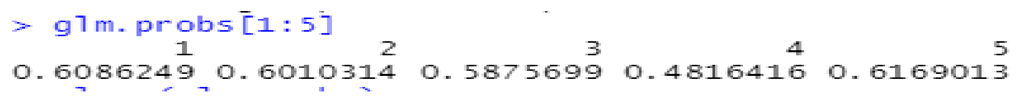
The only statistically significant predictor is Lag2, with a p-value of 0.0296 providing evidence at the 5% significance level to reject the null hypothesis that it is not related to the response Direction. None of the other predictors are statistically significant, though Lag1 is somewhat near the border of being significant at the 10% level, with a p-value of 0.1181.

Part C)

In order to calculate the confusion matrix we perform the below operations

After fitting the model, I am using this command glm.probs <- predict(glm.fit,type = "response").

This will make predictions on the training data that we used to fit the model and give me a vector of fitted probabilities.



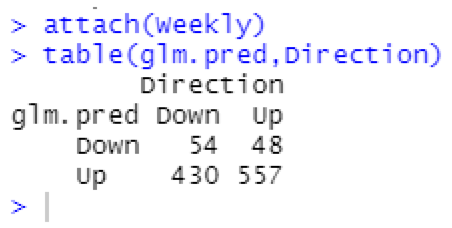
```
> glm.probs[1:5]
        1         2         3         4         5 
0.6086249 0.6010314 0.5875699 0.4816416 0.6169013 
```

Now we would be converting these probabilities into classifications by thresholding at 0.5 by using the below command.

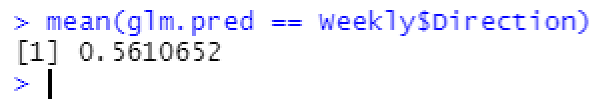
glm.pred <- ifelse(glm.probs > 0.5, "Up", "Down")

Here glm.pred is a vector of trues and false. If glm.probs is bigger than 0.5, glm.pred calls "Up"; otherwise, it calls "Down".

Now the below command is used to show how many predictions were correct and how many were false. It's a confusion matrix of correctly and incorrectly classified data



```
> attach(Weekly)
> table(glm.pred,Direction)
        Direction
glm.pred Down  Up
    Down   54  48
    Up    430 557
> 
```



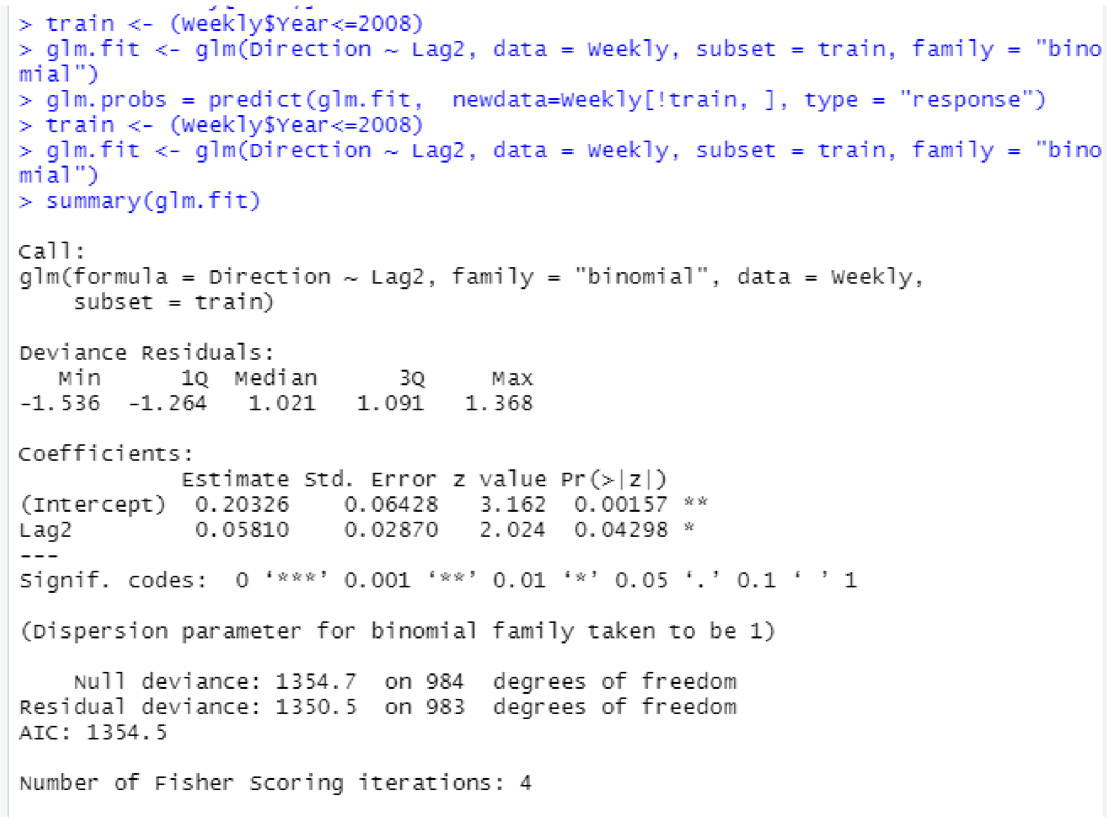
```
> mean(glm.pred == Weekly$Direction)
[1] 0.5610652
> 
```

As we can see in the confusion matrix, the logistic regression model using the five lag variables along with Volume as the predictors, and a prediction threshold of 0.5, correctly predicted 54 down weeks out of a total of 484 actual down weeks and 557 up days out of a total of 605 actual up weeks. This means that the model correctly predicted the direction for 611 weeks out of the 1089 for an accuracy of 0.5612. While this seems to be better than random chance, it is important to note that the model was trained on the entire data set, so 0.5612 is the training accuracy. Moreover, a naive strategy of simply saying that every week will be an up week would have resulted in 605 correctly predicted weeks out of 1089, which is a very similar level of overall accuracy.

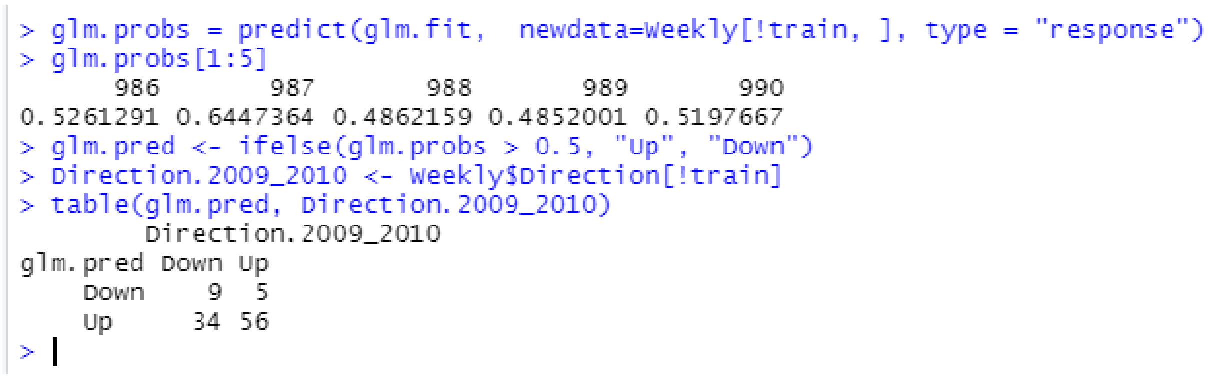
To look a little closer at the confusion matrix, let's assume that our goal is to correctly predict when the market will go up. In this case, up weeks will be considered as positive (++) and down weeks as negative (−−). Having set this convention, we can now consider four important quantities associated with the confusion matrix: true positive rate (i.e. sensitivity or recall), false positive rate, positive predictive value (i.e. precision), and negative predictive value. The true positive rate is the number of correctly predicted positives divided by the overall number of positives -- the number of correctly predicted up weeks (557) over the total number of up weeks (605) for a value of 557/605≈0.92557/605≈0.92 for this model. While this is a pretty high value, which is good, the false positive rate -- the number of incorrectly predicted positives (weeks incorrectly predicted to be up weeks = 430 weeks) divided by the overall number of negatives (the total number of down weeks = 484 weeks) -- is comparably high at 430/484≈0.888430/484≈0.888, which might be quite bad depending on our sensitivity to losing money on an incorrectly predicted down week. Next is the positive predictive value, which is the number of true positives divided by the total number of predicted positives; in our case this is 557/987≈0.564557/987≈0.564. This is better than chance, but as already noted we would have a comparable positive predictive value if we just predicted that every week would be an up week. Lastly is the negative predictive value, which is the number of true negatives divided by the total number of predicted negatives; in our case this is 54/102≈0.52954/102≈0.529.

From the table, instances on the elements gives us get the correct classification, and off the diagonals gives the mistakes that the model makes. It seems that are around 430 values which were actually negative stock returns however predicted as positive which is quite a big number.

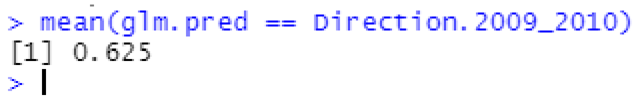
Part D) So as per the question the Weekly data which comes within the period 1990 to 2008 has been used to fit the logistic regression model.



```
> train <- (Weekly$Year<=2008)
> glm.fit <- glm(Direction ~ Lag2, data = Weekly, subset = train, family = "bino
mial")
> glm.probs = predict(glm.fit,  newdata=Weekly[!train, ], type = "response")
> train <- (Weekly$Year<=2008)
> glm.fit <- glm(Direction ~ Lag2, data = Weekly, subset = train, family = "bino
mial")
> summary(glm.fit)

Call:
glm(formula = Direction ~ Lag2, family = "binomial", data = Weekly,
    subset = train)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-1.536  -1.264   1.021   1.091   1.368

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.20326    0.06428   3.162  0.00157 **
Lag2         0.05810    0.02870   2.024  0.04298 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1354.7  on 984  degrees of freedom
Residual deviance: 1350.5  on 983  degrees of freedom
AIC: 1354.5

Number of Fisher Scoring iterations: 4
```



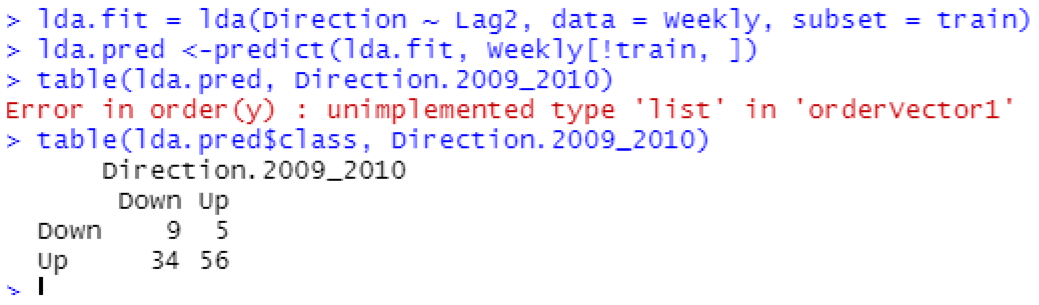
```
> glm.probs = predict(glm.fit,  newdata=Weekly[!train, ], type = "response")
> glm.probs[1:5]
      986       987       988       989       990 
0.5261291 0.6447364 0.4862159 0.4852001 0.5197667 
> glm.pred <- ifelse(glm.probs > 0.5, "Up", "Down")
> Direction.2009_2010 <- Weekly$Direction[!train]
> table(glm.pred, Direction.2009_2010)
        Direction.2009_2010
glm.pred Down Up
    Down    9  5
    Up     34 56
> 
```



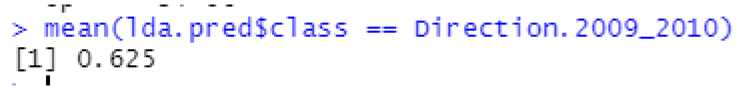
```
> mean(glm.pred == Direction.2009_2010)
[1] 0.625
> 
```

After fitting a logistic regression model on the data from 1990 through 2008 using only Lag2 as the predictor, the model correctly predicted the market direction for 62.5% of the weeks in the held-out data (the data from 2009 and 2010). While this is better than chance, it still is less than a 10% improvement over naively predicting that every week will be an up week. Continuing with the convention that an up week is a positive result, the true positive rate is 56/61≈0.91856/61≈0.918, while the false positive rate is 34/43≈0.79134/43≈0.791. In addition, the positive predictive value is 56/90≈0.62256/90≈0.622 and the negative predictive value is 9/14≈0.6439/14≈0.643.

Part e) So as per the question the Weekly data which comes within the period 1990 to 2008 has been used to fit the linear discriminant model.



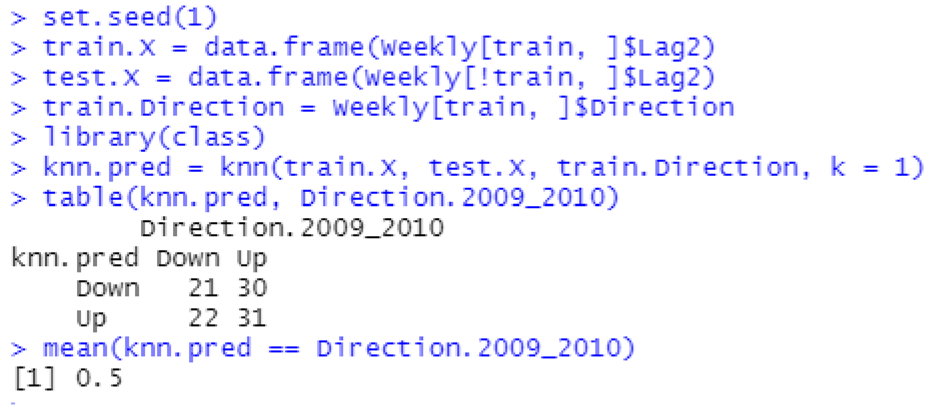
```
> lda.fit = lda(Direction ~ Lag2, data = Weekly, subset = train)
> lda.pred <-predict(lda.fit, Weekly[!train, ])
> table(lda.pred, Direction.2009_2010)
Error in order(y) : unimplemented type 'list' in 'orderVector1'
> table(lda.pred$class, Direction.2009_2010)
      Direction.2009_2010
       Down Up
  Down    9  5
  Up     34 56
> 
```



```
> mean(lda.pred$class == Direction.2009_2010)
[1] 0.625
```

After performing linear discriminant analysis on the data from 1990 through 2008 using only Lag2 as the predictor, we ended up with an identical confusion matrix to the one from Part d with the logistic regression model. As we saw in Part d, the model correctly predicted the market direction for 62.5% of the weeks in the held-out data (the data from 2009 and 2010). While this is better than chance, it still is less than a 10% improvement over naively predicting that every week will be an up week. Continuing with the convention from Part b that an up week is a positive result, the true positive rate is 56/61≈0.91856/61≈0.918, while the false positive rate is 34/43≈0.79134/43≈0.791. In addition, the positive predictive value is 56/90≈0.62256/90≈0.622 and the negative predictive value is 9/14≈0.6439/14≈0.643.

Part f) So as per the question the Weekly data which comes within the period 1990 to 2008 has been used to fit the KNN model with k=1



```
> set.seed(1)
> train.X = data.frame(Weekly[train, ]$Lag2)
> test.X = data.frame(Weekly[!train, ]$Lag2)
> train.Direction = Weekly[train, ]$Direction
> library(class)
> knn.pred = knn(train.X, test.X, train.Direction, k = 1)
> table(knn.pred, Direction.2009_2010)
        Direction.2009_2010
knn.pred Down Up
    Down   21 30
    Up     22 31
> mean(knn.pred == Direction.2009_2010)
[1] 0.5
```

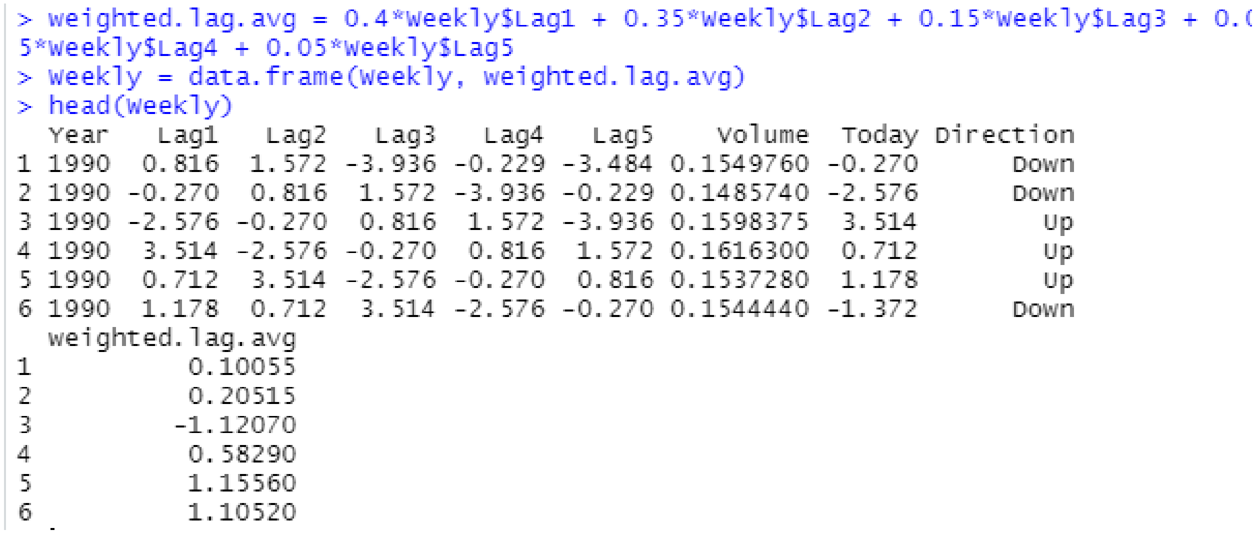
After performing kk-nearest neighbors' classification with k=1k=1 on the data from 1990 through 2008 using only Lag2 as the predictor, the model correctly predicted the market direction for 50% of the weeks in the held-out data (the data from 2009 and 2010). While this only as good as picking the direction randomly, it had worse performance than naively predicting that every week will be an up week. Continuing with the convention from Part 3 that an up week is a positive result, the true positive rate is 31/61≈0.50831/61≈0.508, while the false positive rate is 22/43≈0.51222/43≈0.512. In addition, the positive predictive value is 31/53≈0.58531/53≈0.585 and the negative predictive value is 21/51≈0.41221/51≈0.412.

Part g)

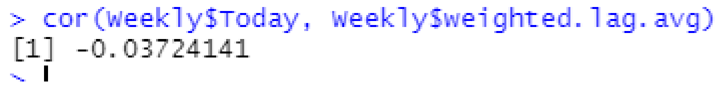
If we are only considering overall prediction accuracy, it appears that logistic regression and linear discriminant analysis were equally good as the models that performed the best on this data. k-nearest neighbors didn't perform any better than randomly guessing, and in fact performed worse than naively predicting every week would be an up week.

Part h)

The combination of predictors I will try out is a weighted average of the lag variables where the recent lag values are weighted more heavily than the ones further in the past. More specifically, I will try out giving Lag1 a weight of 40%, Lag2 a weight of 35%, Lag3 a weight of 15%, and Lag4 and Lag5 each weights of 5%.



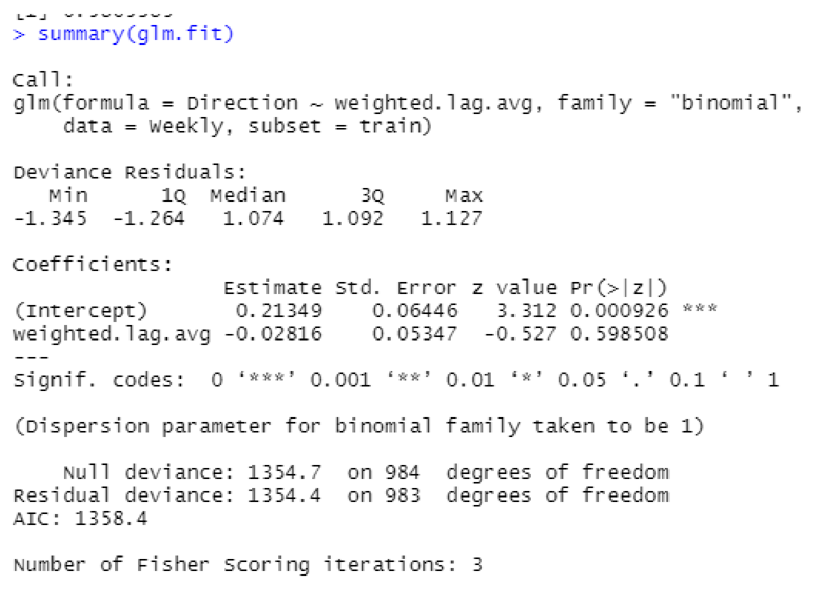
```
> weighted.lag.avg = 0.4*Weekly$Lag1 + 0.35*Weekly$Lag2 + 0.15*Weekly$Lag3 + 0.0
5*Weekly$Lag4 + 0.05*Weekly$Lag5
> Weekly = data.frame(Weekly, weighted.lag.avg)
> head(Weekly)
  Year   Lag1   Lag2   Lag3   Lag4   Lag5    Volume  Today Direction
1 1990  0.816  1.572 -3.936 -0.229 -3.484 0.1549760 -0.270      Down
2 1990 -0.270  0.816  1.572 -3.936 -0.229 0.1485740 -2.576      Down
3 1990 -2.576 -0.270  0.816  1.572 -3.936 0.1598375  3.514        Up
4 1990  3.514 -2.576 -0.270  0.816  1.572 0.1616300  0.712        Up
5 1990  0.712  3.514 -2.576 -0.270  0.816 0.1537280  1.178        Up
6 1990  1.178  0.712  3.514 -2.576 -0.270 0.1544440 -1.372      Down
  weighted.lag.avg
1          0.10055
2          0.20515
3         -1.12070
4          0.58290
5          1.15560
6          1.10520
```



```
> cor(Weekly$Today, Weekly$weighted.lag.avg)
[1] -0.03724141
> 
```

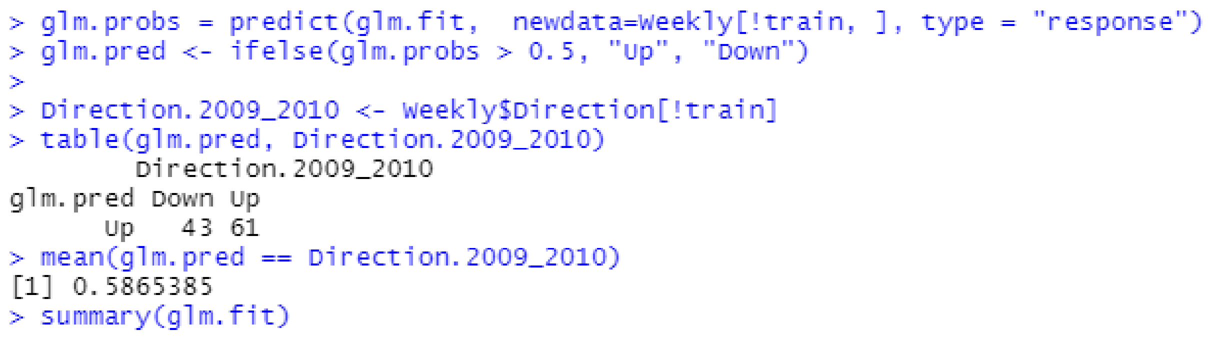
Computing the correlation between this weighted average and the value of the current week's return, we see that there only a very weak correlation between the two quantities. It is smaller in magnitude than the correlations between Today and the first the lag variables individually. That seems to suggest that this weighting might not be too useful, but I will still try out each of the classification methods using this transformation of the predictors. I will start out with logistic regression.

Fitting with logistic regression model where predictor variable is this new variable Weekly$weighted.lag.avg and printing the summary:



```
> summary(glm.fit)

Call:
glm(formula = Direction ~ weighted.lag.avg, family = "binomial",
    data = Weekly, subset = train)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-1.345  -1.264   1.074   1.092   1.127

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)       0.21349    0.06446   3.312 0.000926 ***
weighted.lag.avg -0.02816    0.05347  -0.527 0.598508
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1354.7  on 984  degrees of freedom
Residual deviance: 1354.4  on 983  degrees of freedom
AIC: 1358.4

Number of Fisher Scoring iterations: 3
```

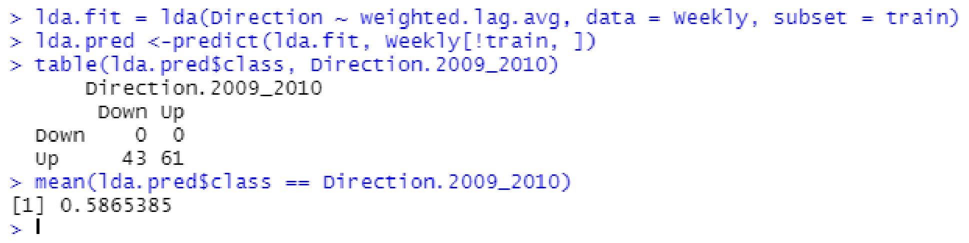
Predicting the model using train data



```
> glm.probs = predict(glm.fit,  newdata=Weekly[!train, ], type = "response")
> glm.pred <- ifelse(glm.probs > 0.5, "Up", "Down")
>
> Direction.2009_2010 <- Weekly$Direction[!train]
> table(glm.pred, Direction.2009_2010)
        Direction.2009_2010
glm.pred Down Up
      Up   43 61
> mean(glm.pred == Direction.2009_2010)
[1] 0.5865385
> summary(glm.fit)
```

As we can see, the results with logistic regression aren't particularly encouraging, as with a prediction threshold of 50% this method was equivalent to always predicting that the market would go up when evaluated on the test set. In addition, the p-value for the coefficient of `weighted.lag.avg` is 0.598, which means that there isn't evidence to say that it is statistically significant. Thinking a little more about the idea of using a weighted average, it makes sense that the performance with logistic regression won't provide an improvement over what we did in Part 4, since the weighted average is still a linear combination of the variables. We already saw that in a logistic regression model, `Lag2` was the only statistically significant coefficient, and even then it is borderline at the 5% significance level, so the weighted average includes variables which we already had reason to believe weren't particularly helpful in making a strong model. I'll still try out the remaining methods with `weighted.lag.avg` before trying out one other combination of the predictors. Next up is linear discriminant analysis.

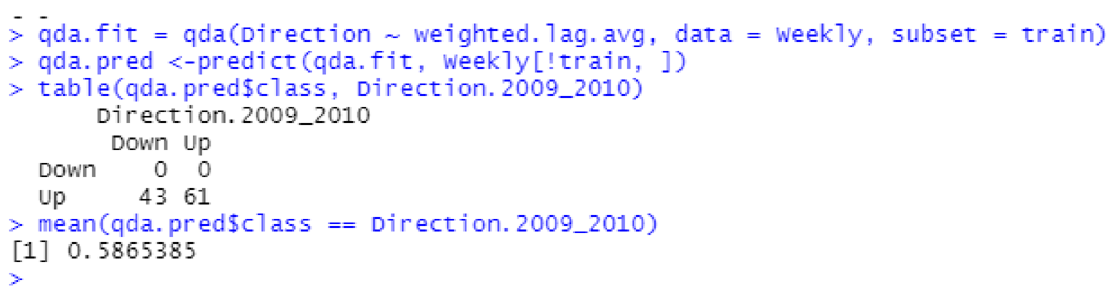
Fitting the model with Linear Discriminant Analysis where the predictor is Weekly$weighted.lag.avg and predicting on train data:



```
> lda.fit = lda(Direction ~ weighted.lag.avg, data = Weekly, subset = train)
> lda.pred <-predict(lda.fit, Weekly[!train, ])
> table(lda.pred$class, Direction.2009_2010)
      Direction.2009_2010
       Down Up
  Down    0  0
  Up     43 61
> mean(lda.pred$class == Direction.2009_2010)
[1] 0.5865385
> 
```

Linear discriminant analysis has the same performance as logistic regression. This is reasonable since the two methods often perform similarly. Now we'll consider quadratic discriminant analysis.

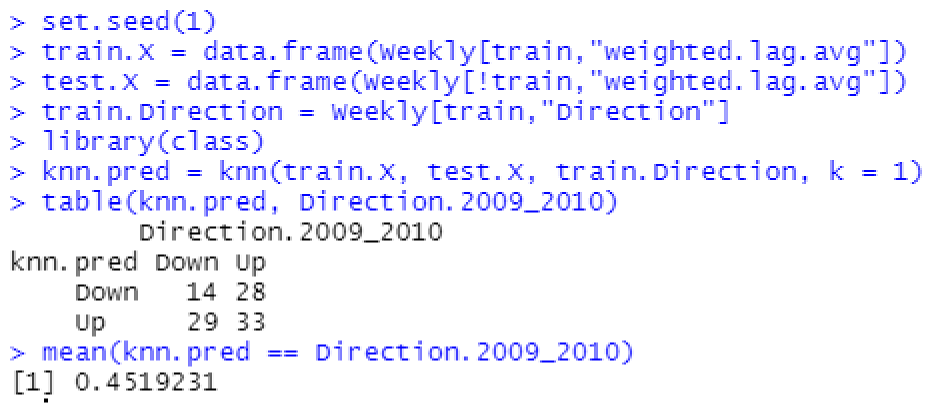
Fitting the model with quadratic discriminant analysis where the predictor is Weekly$weighted.lag.avg and predicting on train data:



```
> qda.fit = qda(Direction ~ weighted.lag.avg, data = Weekly, subset = train)
> qda.pred <-predict(qda.fit, Weekly[!train, ])
> table(qda.pred$class, Direction.2009_2010)
      Direction.2009_2010
       Down Up
  Down    0  0
  Up     43 61
> mean(qda.pred$class == Direction.2009_2010)
[1] 0.5865385
> 
```

Quadratic discriminant analysis has the same performance as logistic regression and Linear discriminant analysis. Now we'll consider k nearest neighbor with k= 1 ,k=3 and k =5

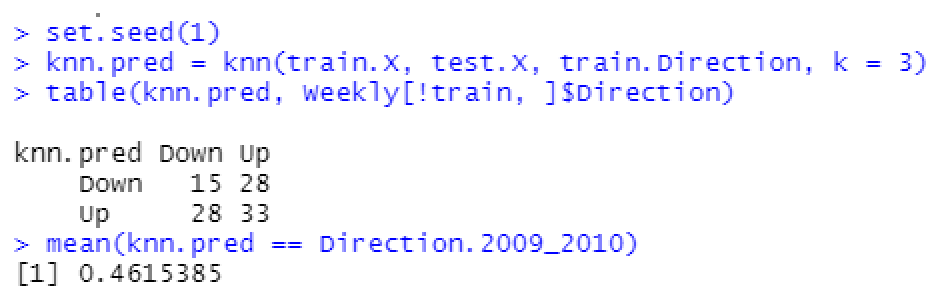
Fitting the model with Knn where k=1 and predictor is Weekly$weighted.lag.avg and predicting on training data



```
> set.seed(1)
> train.X = data.frame(Weekly[train,"weighted.lag.avg"])
> test.X = data.frame(Weekly[!train,"weighted.lag.avg"])
> train.Direction = Weekly[train,"Direction"]
> library(class)
> knn.pred = knn(train.X, test.X, train.Direction, k = 1)
> table(knn.pred, Direction.2009_2010)
        Direction.2009_2010
knn.pred Down Up
    Down   14 28
    Up     29 33
> mean(knn.pred == Direction.2009_2010)
[1] 0.4519231
```

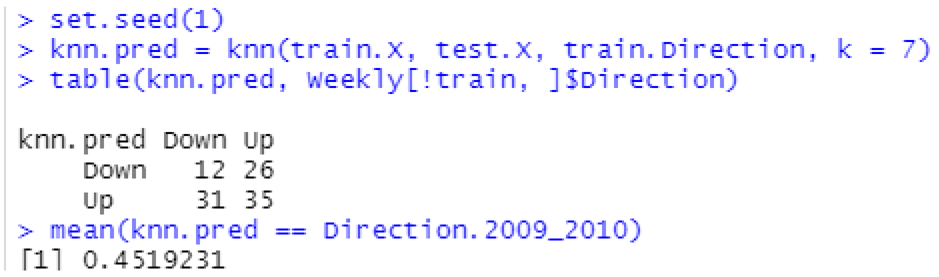
With k=1k=1, KNN performs even worse than random guessing when it comes to overall prediction accuracy, and the true positive rate (0.541), false positive rate (0.674), and positive predictive rate (0.532) aren't encouraging when compared to the result from Part f. Before moving on, let's try out two more values for kk: k=3k=3 and k=7.

Fitting the model with Knn where k=3 and predictor is Weekly$weighted.lag.avg and predicting on training data



```
> set.seed(1)
> knn.pred = knn(train.X, test.X, train.Direction, k = 3)
> table(knn.pred, Weekly[!train, ]$Direction)

knn.pred Down Up
    Down   15 28
    Up     28 33
> mean(knn.pred == Direction.2009_2010)
[1] 0.4615385
```

Fitting the model with Knn where k=7and predictor is Weekly$weighted.lag.avg and predicting on training data



```
> set.seed(1)
> knn.pred = knn(train.X, test.X, train.Direction, k = 7)
> table(knn.pred, Weekly[!train, ]$Direction)

knn.pred Down Up
    Down   12 26
    Up     31 35
> mean(knn.pred == Direction.2009_2010)
[1] 0.4519231
```

Even when we increase the value of kk, the results are largely the same.