ΣΧΟΛΗ ΧΡΗΜΑΤΟΟΙΚΟΝΟΜΙΚΗΣ & ΣΤΑΤΙΣΤΙΚΗΣ

ΤΜΗΜΑ ΣΤΑΤΙΣΤΙΚΗΣ & ΑΣΦΑΛΙΣΤΙΚΗΣ ΕΠΙΣΤΗΜΗΣ

Π.Μ.Σ. ΕΦΑΡΜΟΣΜΕΝΗΣ ΣΤΑΤΙΣΤΙΚΗΣ

ΣΤΑΤΙΣΤΙΚΗ ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ

ΕΡΓΑΣΙΑ 1η

ΜΟΝΤΕΛΑ ΠΑΛΙΝΔΡΟΜΗΣΗΣ ΚΑΙ ΤΑΞΙΝΟΜΙΣΗΣ ΣΕ R & PYTHON

Εμμανουήλ Φλωράκης florakis.emm@gmail.com ΜΕΣ22024

Αθήνα, 2023

Περιεχόμενα

ΑΣΚΗΣΗ 1η

Βασισμένοι σε ένα δοθέν σύνολο δεδομένων όπου περιέχονται πληροφορίες για τις τιμές και τα χαρακτηριστικά μεταχειρισμένων αυτοκινήτων προς πώληση, θα προχωρήσουμε σε κατασκευή και αξιολόγηση μοντέλων παλινδρόμησης για την πρόβλεψη της τιμής ενός μεταχειρισμένου αυτοκινήτου με βάση τα διαθέσιμα χαρακτηριστικά του. Οι μέθοδοι θα γίνουν επιμέρους μέσω του στατιστικού πακέτου R και της γλώσσας προγραμματισμού Python.

- Γραμμική Παλινδρόμηση

Ξεκινώντας θα παρατηρήσουμε την βασική καταχώρηση των δεδομένων στην εκάστοτε πλατφόρμα και κατόπιν της επεξεργασίας τους θα περάσουμε στην κατασκευή και αξιολόγηση του μοντέλου:

Στατιστικό πακέτο R

```
> ###Statistical Machine Learning
> ##Assingment 1
>
> #Neccesary Libraries
> library(data.table)
> library(caret)
> library(leaps)
> library(glmnet)
> library(plotly)
> library(e1071)
> library(pROC)
>
> #Exercise 1
>
> #Load Data
> data <- fread(file = "carsales.csv")
> #First Look
> str(data)
Classes 'data.table' and 'data.frame':  301 obs. of  9 variables:
 $ Car_Name     : chr  "ritz" "sx4" "ciaz" "wagon r" ...
 $ Year         : int  2014 2013 2017 2011 2014 2018 2015 2015 2016 2015 ...
 $ Selling_Price: num  3.35 4.75 7.25 2.85 4.6 9.25 6.75 6.5 8.75 7.45 ...
 $ Present_Price: num  5.59 9.54 9.85 4.15 6.87 9.83 8.12 8.61 8.89 8.92 ...
 $ Kms_Driven   : int  27000 43000 6900 5200 42450 2071 18796 33429 20273 42367 ...
 $ Fuel_Type    : chr  "Petrol" "Diesel" "Petrol" "Petrol" ...
 $ Seller_Type  : chr  "Dealer" "Dealer" "Dealer" "Dealer" ...
 $ Transmission : chr  "Manual" "Manual" "Manual" "Manual" ...
 $ Owner        : int  0 0 0 0 0 0 0 0 0 0 ...
 - attr(*, ".internal.selfref")=<externalptr>
> psych::describe(data)
             vars   n     mean       sd  median  trimmed      mad     min      max     range  skew kurtosis      se
Car_Name*       1 301    47.54    27.07    39.0    46.96    31.13    1.00     98.0     97.00  0.25    -1.21    1.56
Year            2 301  2013.63     2.89  2014.0  2014.00     2.97 2003.00   2018.0     15.00 -1.23     1.46    0.17
Selling_Price   3 301     4.66     5.08     3.6     3.74     3.85    0.10     35.0     34.90  2.47     8.66    0.29
Present_Price   4 301     7.63     8.64     6.4     6.18     6.89    0.32     92.6     92.28  4.04    30.93    0.50
Kms_Driven      5 301 36947.21 38886.88 32000.0 32133.00 25204.20  500.00 500000.0 499500.00  6.37    66.94 2241.40
Fuel_Type*      6 301     2.79     0.43     3.0     2.87     0.00    1.00      3.0      2.00 -1.65     1.44    0.02
Seller_Type*    7 301     1.35     0.48     1.0     1.32     0.00    1.00      2.0      1.00  0.62    -1.63    0.03
Transmission*   8 301     1.87     0.34     2.0     1.96     0.00    1.00      2.0      1.00 -2.15     2.64    0.02
Owner           9 301     0.04     0.25     0.0     0.00     0.00    0.00      3.0      3.00  7.54    71.59    0.01
> names(data)
[1] "Car_Name"      "Year"          "Selling_Price" "Present_Price" "Kms_Driven"    "Fuel_Type"     "Seller_Type"
[8] "Transmission"  "Owner"
```

```
> #Removing the names columns
> data1 <- subset(data,select=-c(Car_Name))
>
> #set all columns as numeric and factor
> data1$Year <- as.numeric(data1$Year)
> data1$Kms_Driven <- as.numeric(data1$Kms_Driven)
> data1$Fuel_Type <- as.factor(data1$Fuel_Type)
> data1$Seller_Type <- as.factor(data1$Seller_Type)
> data1$Transmission <- as.factor(data1$Transmission)
> data1$Owner <- as.factor(data1$Owner)
> str(data1)
Classes 'data.table' and 'data.frame':  301 obs. of  8 variables:
 $ Year         : num  2014 2013 2017 2011 2014 ...
 $ Selling_Price: num  3.35 4.75 7.25 2.85 4.6 9.25 6.75 6.5 8.75 7.45 ...
 $ Present_Price: num  5.59 9.54 9.85 4.15 6.87 9.83 8.12 8.61 8.89 8.92 ...
 $ Kms_Driven   : num  27000 43000 6900 5200 42450 ...
 $ Fuel_Type    : Factor w/ 3 levels "CNG","Diesel",..: 3 2 3 3 2 2 3 2 2 2 ...
 $ Seller_Type  : Factor w/ 2 levels "Dealer","Individual": 1 1 1 1 1 1 1 1 1 1 ...
 $ Transmission : Factor w/ 2 levels "Automatic","Manual": 2 2 2 2 2 2 2 2 2 2 ...
 $ Owner        : Factor w/ 3 levels "0","1","3": 1 1 1 1 1 1 1 1 1 1 ...
 - attr(*, ".internal.selfref")=<externalptr>
```

Διαμορφώνοντας ως άνω το dataset προχωρούμε στην κατασκευή του μοντέλου

```
> ##Linear Regression
> # Reproducible random sampling
> set.seed(123)
>
> # Creating training data as 75% of the dataset
> random_sample <- createDataPartition(data1$Selling_Price, p = 0.75, list = FALSE)
>
> # Generating training dataset from the random_sample
> training_dataset  <- data1[random_sample, ]
>
> # Generating testing dataset from rows which are not included in random_sample
> testing_dataset <- data1[-random_sample, ]
>
> # Defining training control as cross-validation and value of K equal to 10
> train_control <- trainControl(method = "cv",number = 10)
>
>
> #Training the model by assigning Selling_Price column as target variable and rest other column
> # as independent variable
> model <- train(Selling_Price ~., data = training_dataset, method = "lm", trControl = train_control)
```

Διαχωρίζουμε τυχαία το σύνολο δεδομένων σε 75% 25% σύνολα εκπαίδευσης και ελέγχου και θέτουμε 10-Cross-Validation

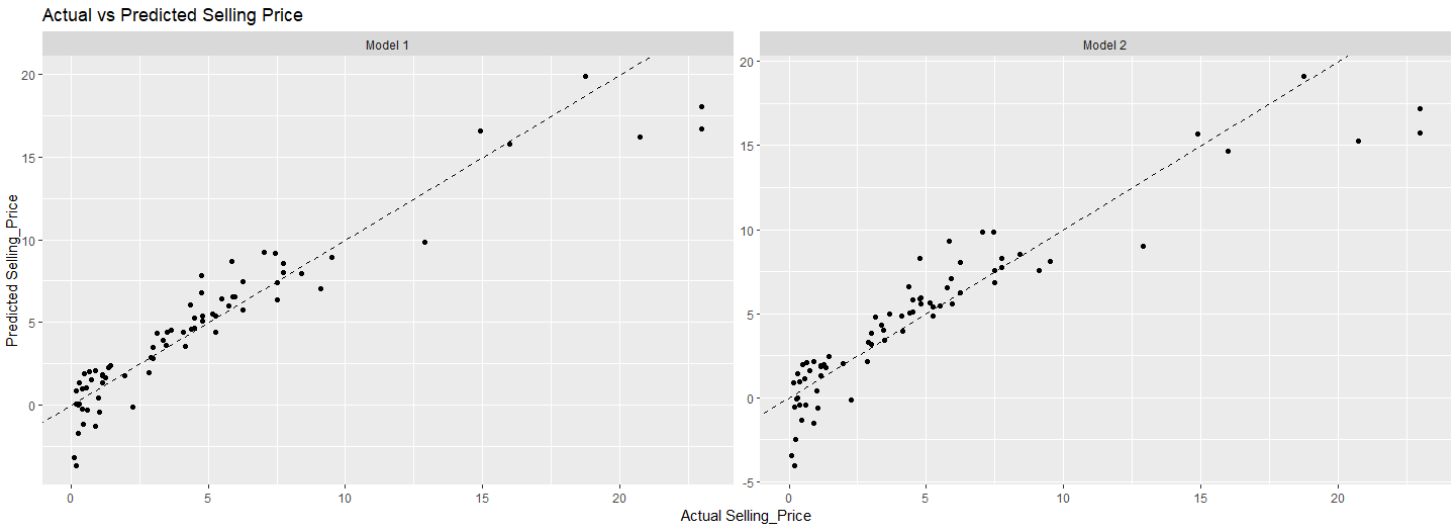
```
> summary(model)

Call:
lm(formula = .outcome ~ ., data = dat)

Residuals:
    Min      1Q  Median      3Q     Max
-7.2515 -0.8628 -0.1822  0.7172 11.4505

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)          -9.230e+02  1.024e+02  -9.014  < 2e-16 ***
Year                  4.592e-01  5.081e-02   9.038  < 2e-16 ***
Present_Price         4.462e-01  1.755e-02  25.422  < 2e-16 ***
Kms_Driven           -1.795e-06  3.512e-06  -0.511 0.609702
Fuel_TypeDiesel       2.141e+00  1.298e+00   1.650 0.100444
Fuel_TypePetrol       6.872e-01  1.276e+00   0.538 0.590797
Seller_TypeIndividual -1.172e+00  3.006e-01  -3.899 0.000129 ***
TransmissionManual    -1.104e+00  3.694e-01  -2.988 0.003131 **
Owner1                4.475e-01  7.480e-01   0.598 0.550272
Owner3               -5.552e+00  1.876e+00  -2.959 0.003424 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.784 on 217 degrees of freedom
Multiple R-squared:  0.8795,    Adjusted R-squared:  0.8745
F-statistic:   176 on 9 and 217 DF,  p-value: < 2.2e-16

>
> # Predicting the target variable
> predictions <- predict(model, testing_dataset)
>
> # Computing model performance metrics
> data.frame( R2 = R2(predictions, testing_dataset$Selling_Price),
+             RMSE = RMSE(predictions, testing_dataset$Selling_Price),
+             MAE = MAE(predictions, testing_dataset$Selling_Price))
        R2     RMSE      MAE
1 0.8974604 1.676636 1.178818
>
> #Variables Fuel_Type, Kms_Driven and Owners aren't in total significant so we will test the above again
```

Παρατηρούμε τα coefficients και τους σχετικούς ελέγχους Wald όπου δεν είναι όλες οι μεταβλητές σημαντικές (θα δοκιμάσουμε μια προσπάθεια βελτίωσης)

Παρατηρούμε πολύ καλά αποτελέσματα με υψηλό $R^2 = 0.897$ και χαμηλά:
RMSE = 1.677 και
MAE = 1.179

```
> data2 <- subset(data1,select=c(Year,Selling_Price,Present_Price,Seller_Type,Transmission))
> # Reproducible random sampling
> set.seed(123)
> random_sample1 <- createDataPartition(data2$Selling_Price, p = 0.75, list = FALSE)
> training_dataset2  <- data2[random_sample1, ]
> testing_dataset2 <- data2[-random_sample1, ]
> #train_control remains the same
> model2 <- train(Selling_Price ~., data = training_dataset2, method = "lm", trControl = train_control)
> summary(model2)

Call:
lm(formula = .outcome ~ ., data = dat)

Residuals:
    Min      1Q  Median      3Q     Max
-7.6127 -0.8990 -0.1699  0.6748 12.2148

Coefficients:
                       Estimate Std. Error t value Pr(>|t|)
(Intercept)          -1.012e+03  8.992e+01 -11.256  < 2e-16 ***
Year                  5.038e-01  4.465e-02  11.284  < 2e-16 ***
Present_Price         4.633e-01  1.728e-02  26.804  < 2e-16 ***
Seller_TypeIndividual -1.512e+00  3.067e-01  -4.929 1.62e-06 ***
TransmissionManual    -7.605e-01  3.800e-01  -2.001   0.0466 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.881 on 222 degrees of freedom
Multiple R-squared:  0.8629,    Adjusted R-squared:  0.8605
F-statistic: 349.4 on 4 and 222 DF,  p-value: < 2.2e-16

> predictions2 <- predict(model2, testing_dataset2)
> data.frame( R2 = R2(predictions2, testing_dataset2$Selling_Price),
+             RMSE = RMSE(predictions2, testing_dataset2$Selling_Price),
+             MAE = MAE(predictions2, testing_dataset2$Selling_Price))
         R2    RMSE      MAE
1 0.8651933 1.92326 1.325447
>
> # Plotting
> plot_data1 <- data.frame(Actual = testing_dataset$Selling_Price, Predicted = predictions)
> plot_data2 <- data.frame(Actual = testing_dataset2$Selling_Price, Predicted = predictions2)
> # Combining data for both models
> plot_data_combined <- rbind(cbind(plot_data1, Model = "Model 1"), cbind(plot_data2, Model = "Model 2"))
>
> # Creating a combined scatterplot using facets
> ggplot(plot_data_combined, aes(x = Actual, y = Predicted)) +
+   geom_point() +
+   geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
+   labs(x = "Actual Selling_Price", y = "Predicted Selling_Price") +
+   ggtitle("Actual vs Predicted Selling Price") +
+   facet_wrap(~ Model, scales = "free")  # Faceting by model
>
> #As we can see we can gain the same amount of accuracy with lesser variables so we can conclude
> #that the optimal model will have Year,Selling_Price,Present_Price,Seller_Type and Transmission
> #as the most significant variables
```

Παραπάνω δοκιμάσαμε ενδεικτικά ένα μοντέλο με μεταβλητές τις Year, Selling_Price, Present_Price, Seller_Type and Transmission όπου μπορούμε να παρατηρήσουμε αρκετά παρεμφερή αποτελέσματα. Τελικά και σε συνέχεια των ως άνω ελέγχων όπου χαρακτηρίζουν μία πολύ καλή κατασκευή μοντέλου δημιουργήσαμε τα δύο επόμενα Scatter Plots.



Όπου πράγματι παρατηρούμε δύο αρκετά καλές εφαρμογές με ελάχιστες ενδείξεις για ακραίες τιμές κοντά στην γραμμή παλινδρόμησης.

Γλώσσα Προγραμματισμού Python

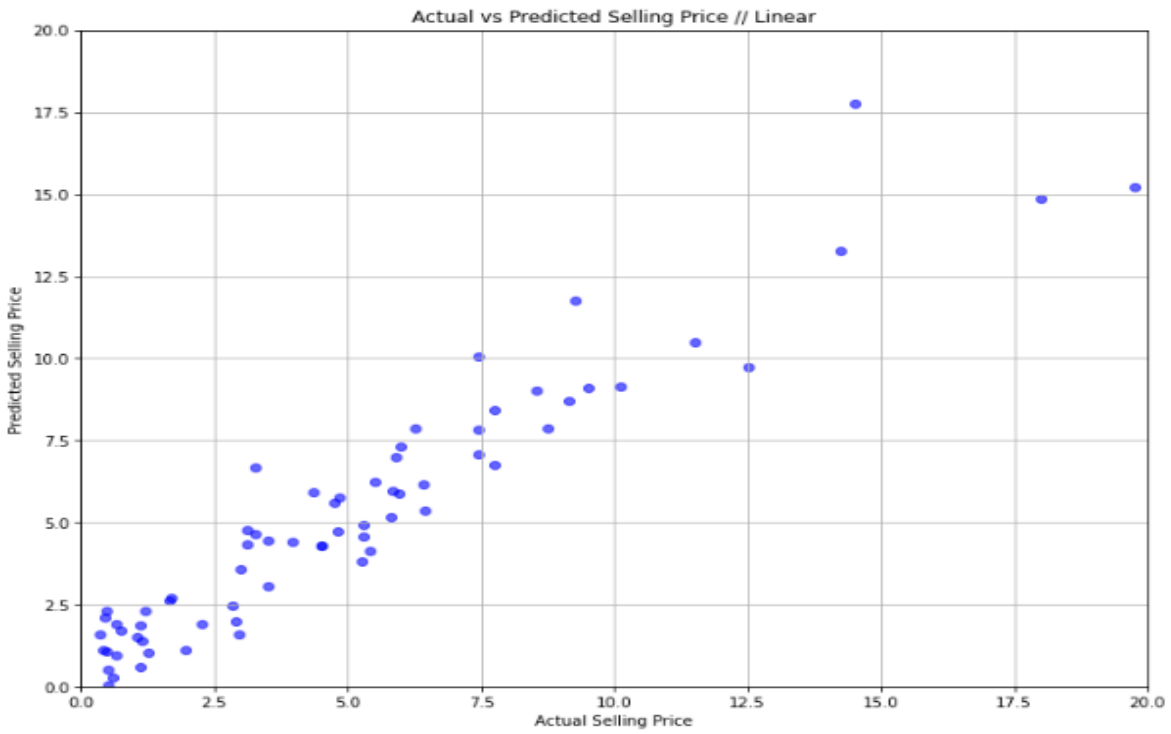
```
>>> ###Statistical Machine Learning
>>> ##Assingment 1
>>>
>>> #Neccesary Libraries
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
>>> from sklearn.linear_model import LinearRegression, RidgeCV, LassoCV, ElasticNetCV, LogisticRegression
>>> from sklearn.preprocessing import StandardScaler
>>> import warnings
>>> from numpy import arange
>>> from sklearn.model_selection import RepeatedKFold
>>> from sklearn.preprocessing import LabelEncoder
>>> import matplotlib.pyplot as plt
>>> from sklearn.svm import SVC
>>> from sklearn import metrics
>>> from sklearn.metrics import roc_curve, auc
>>> from sklearn import datasets
>>> warnings.filterwarnings('ignore')
>>>
>>> #Exercise 1
>>>
>>> #Load Data
>>> data = pd.read_csv('carsales.csv')
>>> data.head()
  Car_Name  Year  Selling_Price  ...  Seller_Type  Transmission Owner
0     ritz  2014           3.35  ...       Dealer        Manual     0
1      sx4  2013           4.75  ...       Dealer        Manual     0
2     ciaz  2017           7.25  ...       Dealer        Manual     0
3  wagon r  2011           2.85  ...       Dealer        Manual     0
4    swift  2014           4.60  ...       Dealer        Manual     0

[5 rows x 9 columns]
>>>
>>>
>>> # Create a LabelEncoder object for each column
>>> label_encoder_fuel = LabelEncoder()
>>> label_encoder_seller = LabelEncoder()
>>> label_encoder_transmission = LabelEncoder()
>>>
>>> # Apply Label Encoding to each categorical column
>>> data['Fuel_Type'] = label_encoder_fuel.fit_transform(data['Fuel_Type'])
>>> data['Seller_Type'] = label_encoder_seller.fit_transform(data['Seller_Type'])
>>> data['Transmission'] = label_encoder_transmission.fit_transform(data['Transmission'])
>>> data.head()
  Car_Name  Year  Selling_Price  ...  Seller_Type  Transmission  Owner
0     ritz  2014           3.35  ...            0             1      0
1      sx4  2013           4.75  ...            0             1      0
2     ciaz  2017           7.25  ...            0             1      0
3  wagon r  2011           2.85  ...            0             1      0
4    swift  2014           4.60  ...            0             1      0

[5 rows x 9 columns]
>>>
>>> # Define the StandardScaler
>>> scale = StandardScaler()
```

Μετά από την καταχώρηση και επεξεργασία των δεδομένων θα προχωρήσουμε σε μορφοποίηση και τυποποίηση ως κάτωθι:

```
>>> x = data.iloc[:, [1, 3, 4, 5, 6, 7, 8]]
>>> y = data['Selling_Price']
>>>
>>> # Scale the features
>>> x_scaled = scale.fit_transform(x)
>>>
>>> # Split the scaled data into training and testing sets
>>> x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, train_size=0.75)
>>>
>>> modelLinear = LinearRegression()
>>>
>>> # Perform cross-validation
>>> cv_scores = cross_val_score(modelLinear, x_train, y_train, cv=10)  # Using 10-fold cross-validation
>>>
>>> # Fit the model on the training data
>>> modelLinear.fit(x_train, y_train)
LinearRegression()
>>>
>>> # Retrieve feature names
>>> feature_names = x.columns.tolist()
>>>
>>> # Retrieve coefficients of the features
>>> coefficients = modelLinear.coef_
>>>
>>> # Compute R-squared scores
>>> R2_train_score_lr = modelLinear.score(x_train, y_train)
>>> R2_test_score_lr = modelLinear.score(x_test, y_test)
>>>
>>> # Compute mean cross-validation score
>>> mean_cv_score = np.mean(cv_scores)
>>>
>>> print("Feature Names:", feature_names)
Feature Names: ['Year', 'Present_Price', 'Kms_Driven', 'Fuel_Type', 'Seller_Type', 'Transmission', 'Owner']
>>> print("Coefficients:", coefficients)
Coefficients: [ 1.08686078  4.74813278 -0.28584462 -0.45346472 -0.23915798 -0.36977535
 -0.28889157]
>>> print("The train R-squared score for LR model is {:.4f}".format(R2_train_score_lr))
The train R-squared score for LR model is 0.8894
>>> print("The test R-squared score for LR model is {:.4f}".format(R2_test_score_lr))
The test R-squared score for LR model is 0.7738
>>> print("Mean Cross-validated R-squared score for LR model is {:.4f}".format(mean_cv_score))
Mean Cross-validated R-squared score for LR model is 0.6390
>>>
>>> # Predictions on the test set
>>> y_pred = modelLinear.predict(x_test)
>>> # Scatter plot of predicted vs actual values
>>> plt.figure(figsize=(8, 6))
<Figure size 800x600 with 0 Axes>
>>> plt.scatter(y_test, y_pred, color='blue', alpha=0.6)
<matplotlib.collections.PathCollection object at 0x0000027A9F556F10>
>>> plt.title('Actual vs Predicted Selling Price // Linear')
Text(0.5, 1.0, 'Actual vs Predicted Selling Price // Linear')
>>> plt.xlabel('Actual Selling Price')
Text(0.5, 0, 'Actual Selling Price')
>>> plt.ylabel('Predicted Selling Price')
Text(0, 0.5, 'Predicted Selling Price')
>>> plt.xlim(0,20)
(0.0, 20.0)
>>> plt.ylim(0,20)
(0.0, 20.0)
>>> plt.grid(True)
>>> plt.show()
```

Χωρίζοντας πάλι το δείγμα μας σε εκπαίδευσης και ελέγχου προχωρούμε στην γραμμική παλινδρόμηση με τα ως άνωθεν αποτελέσματα. Συγκεκριμένα έχουμε μία εξίσου καλή εφαρμογή με R^2 = 0.889 για το Train και 0.773 για το Test καθώς και 0.64 το score του CV.

Για την ορθή απεικόνιση θα χρησιμοποιήσουμε το επόμενο Scatterplot:



Όπου παρατηρείται εύκολα μία γραμμική διαγράμμιση των δεδομένων με ελάχιστες ενδείξεις για ακραίες τιμές.

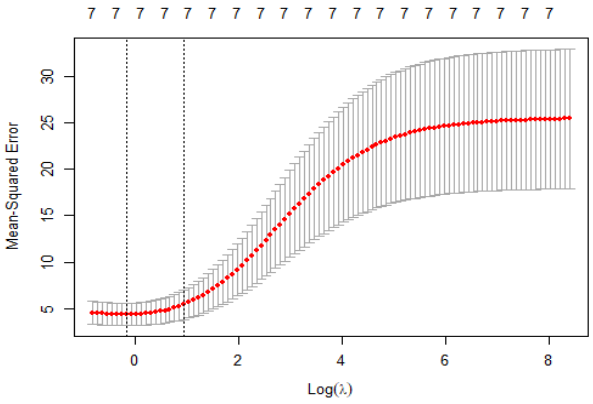
- Ridge

Προχωρούμε με την επόμενη μέθοδο ως κάτωθι:

Για το στατιστικό πακέτο R

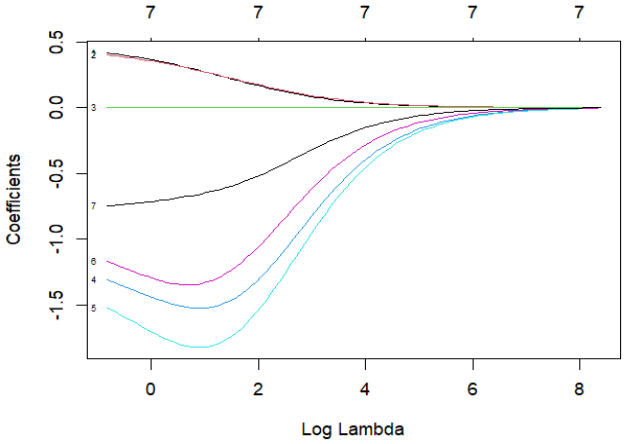
```
> ##Ridge
>
> # Reproducible random sampling
> set.seed(123)
>
> #Provide only numeric data
> data_Ridge <- as.data.frame(sapply(data1, as.numeric))
>
> # Generating training dataset from the random_sample
> training_dataset  <- data_Ridge[random_sample, ]
> train_x <- as.matrix(training_dataset[,-2])
> train_y <- training_dataset$Selling_Price
>
> # Generating testing dataset from rows which are not included in random_sample
> testing_dataset <- data_Ridge[-random_sample, ]
> test_x <- as.matrix(testing_dataset[,-2])
> test_y <- testing_dataset$Selling_Price
>
> #perform k-fold cross-validation to find optimal lambda value
> cv_model <- cv.glmnet(x = train_x, y = train_y, alpha = 0,nfolds = 10)
> cv_model

Call:  cv.glmnet(x = train_x, y = train_y, nfolds = 10, alpha = 0)

Measure: Mean-Squared Error

    Lambda Index Measure    SE Nonzero
min 0.8413    93   4.405 1.181       7
1se 2.5691    81   5.441 1.568       7
>
> #find optimal lambda value that minimizes test MSE
> best_lambda <- cv_model$lambda.min
> best_lambda
[1] 0.8412742
>
> #produce plot of test MSE by lambda value
> plot(cv_model)
> c(log(cv_model$lambda.min), log(cv_model$lambda.1se))
[1] -0.1728376  0.9435673
```

Όπου κατόπιν σχετικής διαμόρφωσης δείγματος εκπαίδευσης και ελέγχου προχωράμε σε έλεγχο αναζήτησης καλύτερης τιμής για την μεταβλητή λάμδα. Με αποτέλεσμα για το μικρότερο MSE, το λ = 0.841 , όπως στο διάγραμμα:

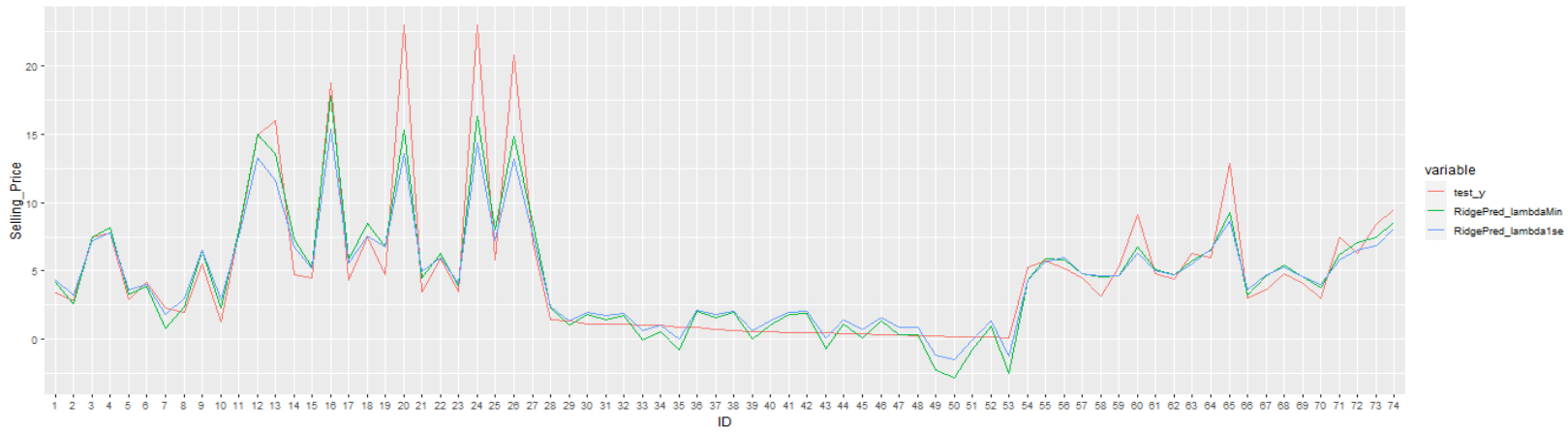


```
> #produce Ridge trace plot
> plot(cv_model$glmnet.fit,xvar = "lambda",label = TRUE)
> t(colnames(train_x))
     [,1]   [,2]           [,3]         [,4]        [,5]          [,6]           [,7]
[1,] "Year" "Present_Price" "Kms_Driven" "Fuel_Type" "Seller_Type" "Transmission" "Owner"
> #find coefficients and R2 of best model (lambda min)
> best_model <- glmnet(x = train_x, y = train_y, alpha = 0, lambda = best_lambda)
> coef(best_model)
8 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept)   -7.459698e+02
Year           3.760586e-01
Present_Price  3.650066e-01
Kms_Driven    -3.472242e-06
Fuel_Type     -1.412138e+00
Seller_Type   -1.667034e+00
Transmission  -1.270807e+00
Owner         -7.237631e-01
> y_pred <- as.numeric( predict(object = best_model,newx = test_x) )
> R2(y_pred, test_y)
[1] 0.8930482
> #find coefficients and R2 of best model (lambda 1se)
> best_model_1se <- glmnet(x = train_x, y = train_y, alpha = 0, lambda = cv_model$lambda.1se)
> coef(best_model_1se)
8 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept)   -5.458306e+02
Year           2.772955e-01
Present_Price  2.756621e-01
Kms_Driven    -3.494015e-06
Fuel_Type     -1.528669e+00
Seller_Type   -1.824673e+00
Transmission  -1.338314e+00
Owner         -6.555265e-01
> #We can see that 'Kms_Driven' have coefficient close to zero
> #suggesting lesser influence on the model.
> y_pred_1se <- as.numeric( predict(object = best_model_1se,newx = test_x) )
> R2(y_pred_1se, test_y)
[1] 0.8921119
> dataForPlot <- data.frame(test_y,RidgePred_lambdaMin = y_pred, RidgePred_lambda1se = y_pred_1se)
> dataForPlot$ID <- as.factor(1:nrow(dataForPlot))
> dataForPlotMelted <- reshape2::melt(data = dataForPlot,value.name = "Selling_Price")
Using ID as id variables
> ggplot(data = dataForPlotMelted,aes(x=ID,y=Selling_Price,group=variable,col=variable))+geom_line()
```

Στο οποίο είναι και φανερή η εξέλιξη του μέσου τετράγ. σφάλματος για τα διάφορα λάμδα. Συνεχίζοντας τους ελέγχους για την μέθοδο Ridge θα κατασκευάσουμε διαγράμματα για τα αναφορικά για τα coefficients αλλά και για εκτιμήσεις μεταξύ των δεδομένων και των λάμδα που εντοπίζουμε στον κώδικα. Επιπροσθέτως το R^2 του μοντέλου είναι αρκετά υψηλό στο 0.89 και για τους δύο ελέγχους.



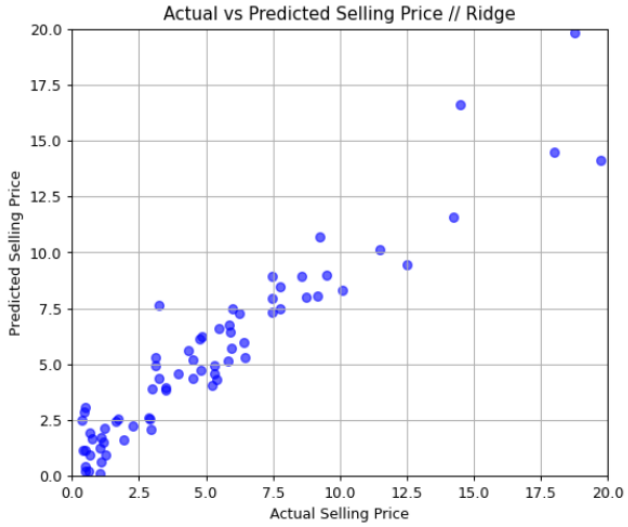
Στο αριστερό διάγραμμα μπορεί να παρατηρηθεί η διανομή των coefficients σύμφωνα με το εκάστοτε λάμδα όπου πράγματι στην καλύτερη τιμή του έχουμε την μεγαλύτερη αλληλεπίδραση μεταξύ των μεταβλητών τόσο για θετική όσο και για αρνητική επιρροή. Στο επόμενο σχήμα βλέπουμε αρκετά κοντινές τιμές μεταξύ των υπολογισθέντων τιμών και των πραγματικών. Άξιος λόγου είναι ο μηδενισμός «ασήμαντων» coefficients όπως της μεταβλητής Kms_Driven βάσει της μεθόδου.



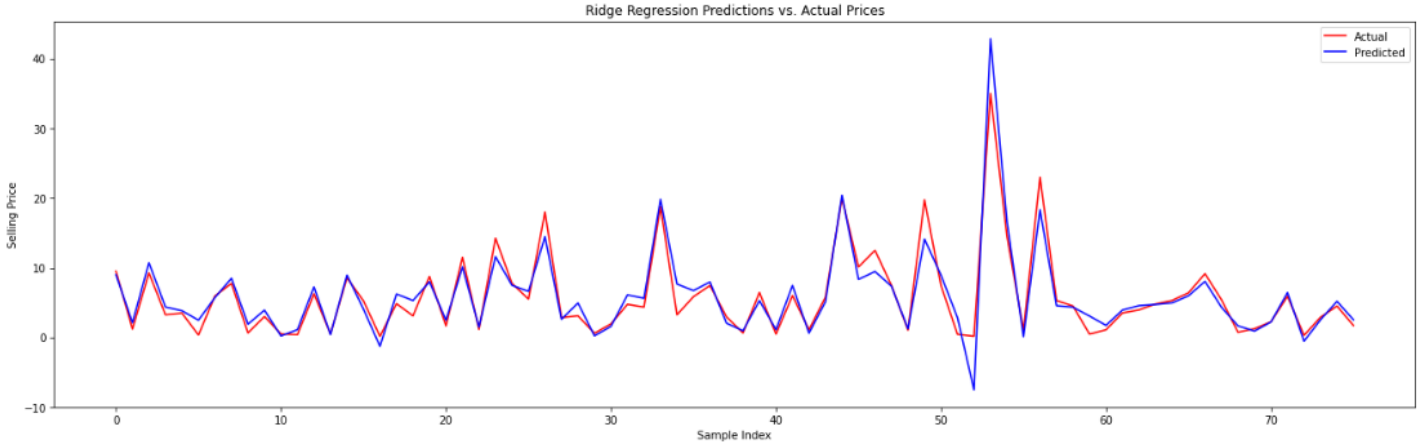
Για την γλώσσα προγραμματισμού Python:

```
>>> ##Ridge
>>> # define model evaluation method
>>> cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
>>> # define Ridge model
>>> ridge_model = RidgeCV(alphas=arange(0.01, 1000, 5), cv=cv)  # Adjust the alpha range as needed
>>> # fit model
>>> ridge_model.fit(x_train, y_train)
RidgeCV(alphas=array([1.0000e-02, 5.0100e+00, 1.0010e+01, 1.5010e+01, 2.0010e+01,
       2.5010e+01, 3.0010e+01, 3.5010e+01, 4.0010e+01, 4.5010e+01,
       5.0010e+01, 5.5010e+01, 6.0010e+01, 6.5010e+01, 7.0010e+01,
       7.5010e+01, 8.0010e+01, 8.5010e+01, 9.0010e+01, 9.5010e+01,
       1.0001e+02, 1.0501e+02, 1.1001e+02, 1.1501e+02, 1.2001e+02,
       1.2501e+02, 1.3001e+02, 1.3501e+02, 1.4001e+02, 1.4501e+0...
       8.5001e+02, 8.5501e+02, 8.6001e+02, 8.6501e+02, 8.7001e+02,
       8.7501e+02, 8.8001e+02, 8.8501e+02, 8.9001e+02, 8.9501e+02,
       9.0001e+02, 9.0501e+02, 9.1001e+02, 9.1501e+02, 9.2001e+02,
       9.2501e+02, 9.3001e+02, 9.3501e+02, 9.4001e+02, 9.4501e+02,
       9.5001e+02, 9.5501e+02, 9.6001e+02, 9.6501e+02, 9.7001e+02,
       9.7501e+02, 9.8001e+02, 9.8501e+02, 9.9001e+02, 9.9501e+02]),
        cv=RepeatedKFold(n_repeats=3, n_splits=10, random_state=1))
>>> # Train and test score for ridge regression
>>> R2_train_score_ridge = ridge_model.score(x_train, y_train)
>>> R2_test_score_ridge = ridge_model.score(x_test, y_test)
>>> # Summarize chosen configuration
>>> print('Ridge best alpha (lambda): %f' % ridge_model.alpha_)
Ridge best alpha (lambda): 15.010000
>>> print("The train score for ridge model is {:.4f}".format(R2_train_score_ridge))
The train score for ridge model is 0.8564
>>> print("The test score for ridge model is {:.4f}".format(R2_test_score_ridge))
The test score for ridge model is 0.8966
>>> # Retrieve coefficients after fitting with the best alpha
>>> print("Ridge coefficients:", ridge_model.coef_)
Ridge coefficients: [ 0.90164677  3.88138475 -0.61054073 -0.81919679 -0.46387598 -0.40235208
 -0.20633426]
>>> # Predictions on the test set
>>> y_pred = ridge_model.predict(x_test)
>>> # Scatter plot of predicted vs actual values
>>> plt.figure(figsize=(8, 6))
<Figure size 800x600 with 0 Axes>
>>> plt.scatter(y_test, y_pred, color='blue', alpha=0.6)
<matplotlib.collections.PathCollection object at 0x0000027AA2040AD0>
>>> plt.title('Actual vs Predicted Selling Price // Ridge')
Text(0.5, 1.0, 'Actual vs Predicted Selling Price // Ridge')
>>> plt.xlabel('Actual Selling Price')
Text(0.5, 0, 'Actual Selling Price')
>>> plt.ylabel('Predicted Selling Price')
Text(0, 0.5, 'Predicted Selling Price')
>>> plt.xlim(0,20)
(0.0, 20.0)
>>> plt.ylim(0,20)
(0.0, 20.0)
>>> plt.grid(True)
>>> plt.show()
>>> y_pred = ridge_model.predict(x_test)
>>> plt.figure(figsize=(8, 6))
<Figure size 800x600 with 0 Axes>
>>> plt.plot(y_test.values, label='Actual', color='red')  # Plotting y_test as red line
[<matplotlib.lines.Line2D object at 0x0000027AA21B5290>]
>>> plt.plot(y_pred, label='Predicted', color='blue')    # Plotting y_pred as blue line
[<matplotlib.lines.Line2D object at 0x0000027AA0D04890>]
>>> plt.xlabel('Sample Index')
Text(0.5, 0, 'Sample Index')
>>> plt.ylabel('Selling Price')
Text(0, 0.5, 'Selling Price')
>>> plt.title('Ridge Regression Predictions vs. Actual Prices')
Text(0.5, 1.0, 'Ridge Regression Predictions vs. Actual Prices')
>>> plt.legend()
<matplotlib.legend.Legend object at 0x0000027AA0C4A2D0>
>>> plt.show()
```

Εξίσου προχωρούμε σε μοντελοποίηση από την οποία έχουμε R^2 για το Train να ισούται με 0.856 και για το Test 0.896 με τα ακόλουθα διαγράμματα:



Όπου είναι ευδιάκριτη η υψηλή γραμμική συσχέτιση μεταξύ των υπολογισθέντων και των πραγματικών τιμών και στα δύο διαγράμματα. Εμφανείς γίνονται μερικές ενδείξεις για ακραίες τιμές



- Lasso

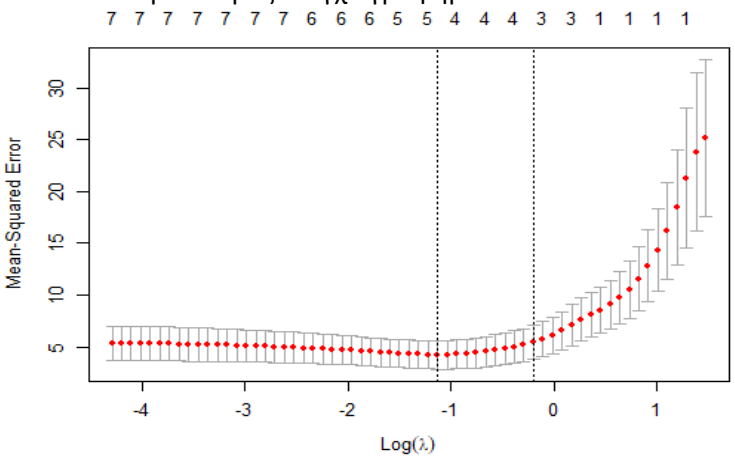
Για το στατιστικό πακέτο R

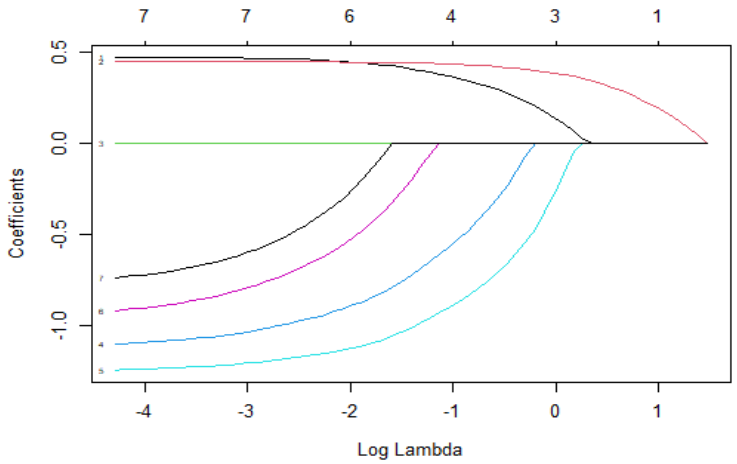
```
> # Reproducible random sampling
> set.seed(123)
> #perform k-fold cross-validation to find optimal lambda value
> cv_model_lasso <- cv.glmnet(x = train_x, y = train_y, alpha = 1,nfolds = 10)
> cv_model_lasso

Call:  cv.glmnet(x = train_x, y = train_y, nfolds = 10, alpha = 1)

Measure: Mean-Squared Error

    Lambda Index Measure    SE Nonzero
min 0.3242    29    4.24 1.379       4
1se 0.8219    19    5.43 1.633       3
> #find optimal lambda value that minimizes test MSE
> best_lambda_lasso <- cv_model_lasso$lambda.min
> best_lambda_lasso
[1] 0.3241874
```

Διαμορφώνουμε το μοντέλο και βρίσκουμε την καλύτερη τιμή του λάμδα.

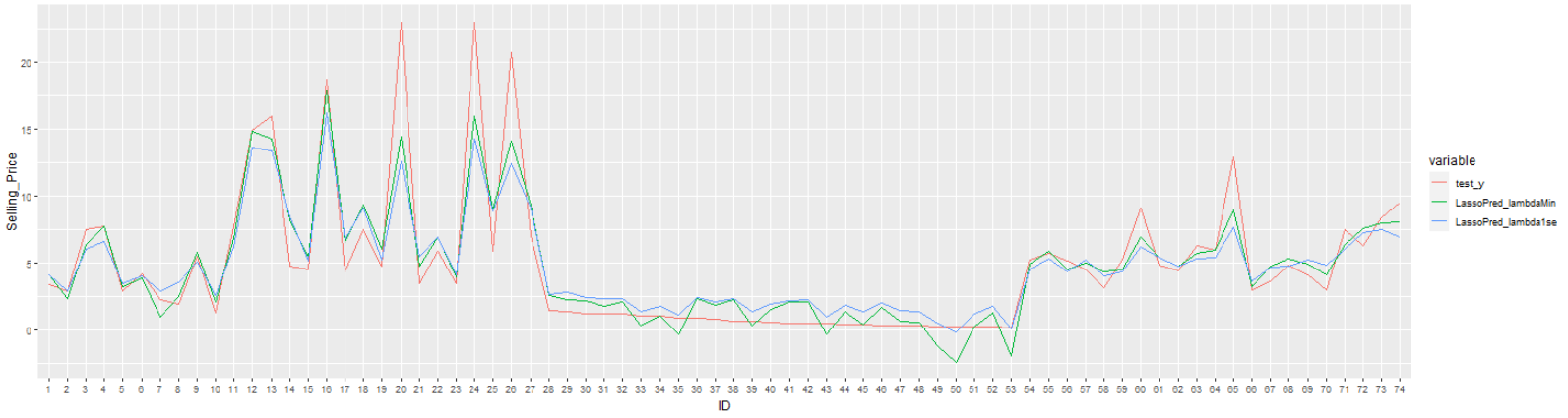
```
> #produce plot of test MSE by lambda value
> plot(cv_model_lasso)
> c(log(cv_model_lasso$lambda.min), log(cv_model_lasso$lambda.1se))
[1] -1.126433 -0.196096
> #produce Lasso trace plot
> plot(cv_model_lasso$glmnet.fit,xvar = "lambda",label = TRUE)
> t(colnames(train_x))
     [,1]   [,2]           [,3]         [,4]         [,5]          [,6]           [,7]
[1,] "Year" "Present_Price" "Kms_Driven" "Fuel_Type" "Seller_Type" "Transmission" "Owner"
> #find coefficients and R2 of best lasso model (lambda min)
> best_model_lasso <- glmnet(x = train_x, y = train_y, alpha = 1, lambda = best_lambda_lasso)
> coef(best_model_lasso)
8 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept)   -755.9455744
Year             0.3775338
Present_Price    0.4353092
Kms_Driven       .
Fuel_Type       -0.6059546
Seller_Type     -0.9253420
Transmission     .
Owner            .
> # 'Kms_Driven', 'Transmission' and 'Owner' were removed or not retained in the model
> # due to their coefficients being reduced to zero by the Lasso regularization.
> y_pred_lasso <- as.numeric( predict(object = best_model_lasso,newx = test_x) )
> R2(y_pred_lasso, test_y)
[1] 0.87621
> #find coefficients and R2 of best model (lambda 1se)
> best_model_1se_lasso <- glmnet(x = train_x, y = train_y, alpha = 1, lambda = cv_model_lasso$lambda.1se)
> coef(best_model_1se_lasso)
8 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept)   -400.5068886
Year             0.2000230
Present_Price    0.3986287
Kms_Driven       .
Fuel_Type        .
Seller_Type     -0.4744199
Transmission     .
Owner            .
> # 'Kms_Driven','Fuel_Type', 'Transmission', and 'Owner' were removed or not retained in the model
> # due to their coefficients being reduced to zero by the Lasso regularization.
> y_pred_1se_lasso <- as.numeric( predict(object = best_model_1se_lasso,newx = test_x) )
> R2(y_pred_1se_lasso, test_y)
[1] 0.8637573
> dataForPlotLasso <- data.frame(test_y,LassoPred_lambdaMin = y_pred_lasso,
+                                LassoPred_lambda1se = y_pred_1se_lasso)
> dataForPlotLasso$ID <- as.factor(1:nrow(dataForPlotLasso))
> dataForPlotLassoMelted <- reshape2::melt(data = dataForPlotLasso,value.name = "Selling_Price")
Using ID as id variables
> ggplot(data = dataForPlotLassoMelted,aes(x=ID,y=Selling_Price,group=variable,col=variable))+geom_line()
```

Με τον παραπάνω κώδικα μπορούμε να διακρίνουμε τα επιμέρους στοιχεία του μοντέλου μας, υπογραμμίζοντας το πολύ υψηλό R^2 να ισούται με περίπου 0.87 και για τους δύο ελέγχους και προχωρούμε στα επόμενα προς έλεγχο γραφήματα:





Όπου επαληθεύουμε το best lambda που εμφανίστηκε στον κώδικα στο πεδίο που ορίζεται το μικρότερο MSE στο διάγραμμα της εξέλιξής βάσει των διάφορων λάμδα, καθώς και της μέτριας απόδοσης των επιθυμητών για τον κώδικα coefficients. (σαν διαδικασία μηδενίζει τα coef των λιγότερο για το μοντέλο σημαντικών variables.)



Στις πραγματικές τιμές από τα δεδομένα μας σε σχέση με τις παραχθείσες παρατηρείται σχετική ομάλυνση κατά περιπτώσεις ενώ παράλληλα και δημιουργία θορύβου από τις περιπτώσεις 28 έως 53.

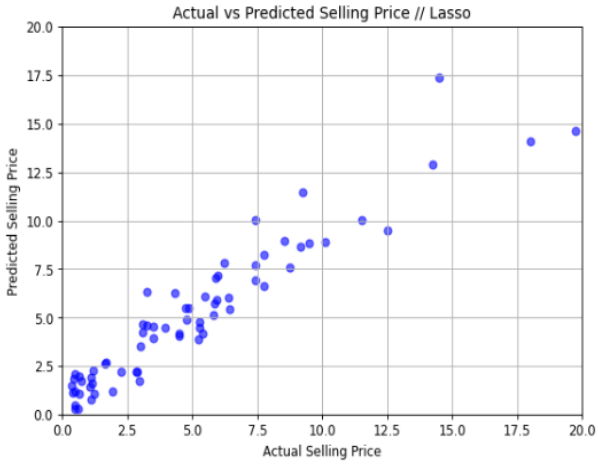
Για την γλώσσα προγραμματισμού Python:

```
>>> ##Lasso
>>>
>>> # define Lasso model
>>> lasso_model = LassoCV(alphas=arange(0, 10, 0.1), cv=cv, n_jobs=-1, max_iter=1000)
>>>
>>> # fit model
>>> lasso_model.fit(x_train, y_train)
LassoCV(alphas=array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2,
       1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5,
       2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8,
       3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5. , 5.1,
       5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6. , 6.1, 6.2, 6.3, 6.4,
       6.5, 6.6, 6.7, 6.8, 6.9, 7. , 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7,
       7.8, 7.9, 8. , 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8, 8.9, 9. ,
       9.1, 9.2, 9.3, 9.4, 9.5, 9.6, 9.7, 9.8, 9.9]),
        cv=RepeatedKFold(n_repeats=3, n_splits=10, random_state=1), n_jobs=-1)
>>>
>>> # Train and test score for Lasso regression
>>> R2_train_score_ls = lasso_model.score(x_train, y_train)
>>> R2_test_score_ls = lasso_model.score(x_test, y_test)
>>>
>>> # Summarize chosen configuration
>>> print('Lasso best alpha (lambda): %f' % lasso_model.alpha_)
Lasso best alpha (lambda): 0.100000
>>> print("The train score for Lasso model is {:.4f}".format(R2_train_score_ls))
The train score for Lasso model is 0.8648
>>> print("The test score for Lasso model is {:.4f}".format(R2_test_score_ls))
The test score for Lasso model is 0.8336
>>> print("Lasso coefficients:", lasso_model.coef_)
Lasso coefficients: [ 0.90524218  4.73962426 -0.5890576  -0.58679605 -0.10195663 -0.05311256
 -0.19469525]
```

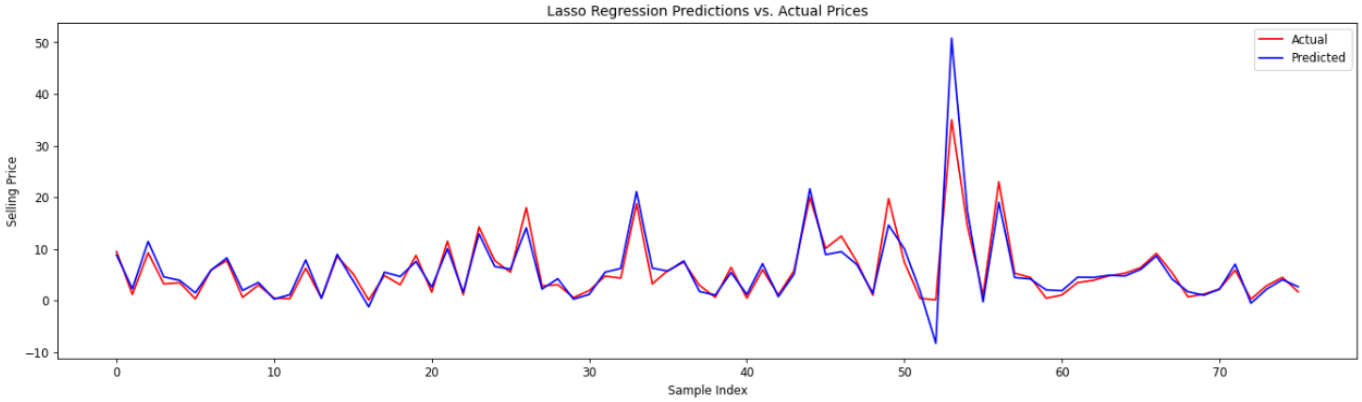
Αφού διαμορφώσουμε κατάλληλα τα δεδομένα εκπαίδευσης , ελέγχου καθώς και το έλεγχο Cross Validation. Μπορούμε εύκολα από τα output του κώδικα να λάβουμε πληροφορίες σχετικά με το καλύτερο λάμδα, τα υψηλά σκορ των ελέγχων X^2 για τα δεδομένα εκπαίδευσης και ελέγχου καθώς και τα σχετικά coefficients.

Στη συνέχεια θα δούμε σχετικά διαγράμματα:

```
# Predictions on the test set
y_pred = lasso_model.predict(x_test)
# Scatter plot of predicted vs actual values
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.6)
plt.title('Actual vs Predicted Selling Price // Lasso')
plt.xlabel('Actual Selling Price')
plt.ylabel('Predicted Selling Price')
plt.xlim(0,20)
plt.ylim(0,20)
plt.grid(True)
plt.show()

y_pred = lasso_model.predict(x_test)
plt.figure(figsize=(8, 6))
plt.plot(y_test.values, label='Actual', color='red')  # Plotting y_test as red line
plt.plot(y_pred, label='Predicted', color='blue')    # Plotting y_pred as blue line
plt.xlabel('Sample Index')
plt.ylabel('Selling Price')
plt.title('Lasso Regression Predictions vs. Actual Prices')
plt.legend()
plt.show()
```



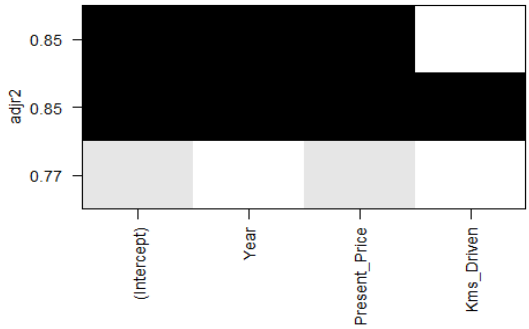
Παρατηρώντας μία εξίσου υψηλή γραμμική εξάρτηση για τα παραχθέντα δεδομένα σε σχέση με τα πραγματικά το οποίο επιβεβαιώνεται και στο ακόλουθο διάγραμμα με ελάχιστες αποκλίσεις σε ακραίες τιμές.

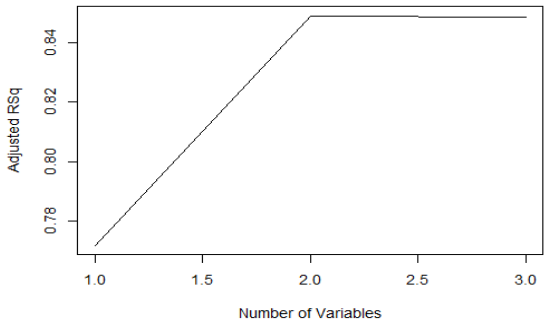


- Leaps *(Υλοποίηση μόνο στο στατιστικό πακέτο R)*

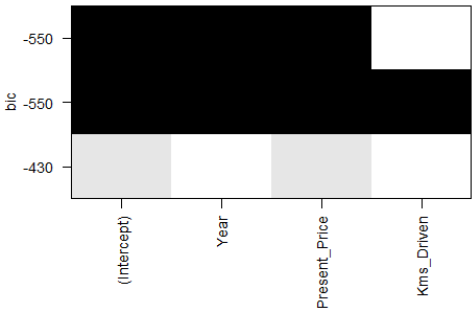
```
> #Keep only quantitative variables
> data_Leaps <- subset(data1,select=c(Year,Selling_Price,Present_Price,Kms_Driven))
> str(data_Leaps)
Classes 'data.table' and 'data.frame':  301 obs. of  4 variables:
 $ Year         : num  2014 2013 2017 2011 2014 ...
 $ Selling_Price: num  3.35 4.75 7.25 2.85 4.6 9.25 6.75 6.5 8.75 7.45 ...
 $ Present_Price: num  5.59 9.54 9.85 4.15 6.87 9.83 8.12 8.61 8.89 8.92 ...
 $ Kms_Driven   : num  27000 43000 6900 5200 42450 ...
 - attr(*, ".internal.selfref")=<externalptr>
>
> # Use all predictor variables
> regfit_full3 <- regsubsets(Selling_Price ~., data = data_Leaps ,nvmax = 3 ,nbest = 1)
> reg_summary <- summary(regfit_full3)
> reg_summary
Subset selection object
Call: regsubsets.formula(Selling_Price ~ ., data = data_Leaps, nvmax = 3,
    nbest = 1)
3 Variables  (and intercept)
             Forced in Forced out
Year             FALSE      FALSE
Present_Price    FALSE      FALSE
Kms_Driven       FALSE      FALSE
1 subsets of each size up to 3
Selection Algorithm: exhaustive
         Year Present_Price Kms_Driven
1  ( 1 ) " "  "*"           " "
2  ( 1 ) "*"  "*"           " "
3  ( 1 ) "*"  "*"           "*"
>
> #We can see that the Kms_Driven variable don't contribute that much to the model
> plot(reg_summary$adjr2, xlab = "Number of Variables", ylab = "Adjusted RSq", type = "l")
> reg_summary$adjr2
[1] 0.7718498 0.8490448 0.8485968
> #By the plot seems that addingthe 3rd variable isn't gaining that much for the model
> adj_r2_max <-  which.max(reg_summary$adjr2)
> #Highest R-square gained with the 2nd model
>
> #Seeing the coefficients for the to highest rated models
> print(c(coef(regfit_full3, 2),coef(regfit_full3, 3)))
  (Intercept)          Year Present_Price   (Intercept)          Year Present_Price    Kms_Driven
-9.854594e+02  4.897223e-01  5.246442e-01 -9.684713e+02  4.813043e-01  5.256219e-01 -1.213502e-06
>
> #Checking our model of choice (2) with plots for R-square and BIC
> plot(regfit_full3, scale = "adjr2")
> plot(regfit_full3, scale = "bic")
```

Με τον παραπάνω κώδικα προχωρούμε σε μοντελοποίηση με την μέθοδο Leaps, όπου αφού κρατήσουμε μόνο τις μετρήσιμες μεταβλητές κατασκευάζουμε το μοντέλο και παράγουμε σχετικά διαγράμματα για τους διάφορους συνδυασμούς μοντέλων όπου ανά βήμα προστίθεται μία μεταβλητή.





Από το output του κώδικα γνωρίζουμε πως η σειρά ένταξης είναι πρώτα η μεταβλητή Present_Price μετά η Year και τελικά η Kms_Driven. Προσθέτοντας την δεύτερη μεταβλητή στο μοντέλο βλέπουμε αύξηση στο 0.85 για το adjusted R^2 ενώ με την 3η παρατηρούμε πως το μοντέλο μας δεν λαμβάνει κάποια σημαντική προστιθέμενη αξία.



Όμοια μπορούν να παρατηρηθούν και στο διάγραμμα για τα BIC των μοντέλων όπου όπως και στις προηγούμενες μεθόδους μας υποδεικνύει πως για οικονομία δεδομένων η μεταβλητή Kms_Driven δεν προσθέτει ιδιαίτερη αξία στο μοντέλο.

- Elastic Net

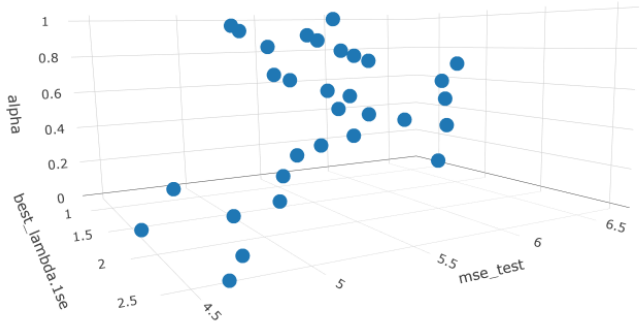
Για το στατιστικό πακέτο R

```
## Elastic net

# Reproducible random sampling
set.seed(123)

#Train x and y and Test x and y are the same as before

alphasToTest <- 0:30/30
models <- list()
glm_models <- list()
results <- data.frame()

for(curID in 1:length(alphasToTest)){
  curAlpha <- alphasToTest[curID]
  cat("Alpha =",curAlpha,"\n")
  models[[curID]] <-
    cv.glmnet(x = train_x, y = train_y, alpha = curAlpha, nfolds = 10,type.measure="mse")
  curModel <- glmnet(x = train_x, y = train_y, alpha = curAlpha,
                     lambda = models[[curID]]$lambda.1se,)
  glm_models[[curID]] <- curModel

  predicted <- predict(curModel, newx=test_x)

  ## Calculate the Mean Squared Error
  mse <- mean((test_y - predicted)^2)

  ## Store the results
  tempDF <- data.frame(alpha=curAlpha, best_lambda.1se=models[[curID]]$lambda.1se, mse_test=mse)
  results <- rbind(results, tempDF)
}
```

Ως τελευταία μέθοδο της πρώτης άσκησης θα προχωρήσουμε σε κατασκευή μοντέλου με την μέθοδο Elastic Net. Συγκεκριμένα στον διπλανό κώδικα παρατηρούμε την εύρεση καλύτερου Alpha για την συνολικά καλύτερη απόδοση του μοντέλου.

```
> plot(results$alpha, results$mse_test,type = "l")
> results <- results[order(results$mse),]
> # Best regularized model
> best_alpha_elastic_net <- results$alpha[1]
> best_lambda_elastic_net <- results$best_lambda.1se[1]
> cat("best_alpha_elastic_net =",best_alpha_elastic_net,"\nbest_lambda_elastic_net =",best_lambda_elastic_net,"\n")
best_alpha_elastic_net = 0.06666667
best_lambda_elastic_net = 1.917993
> best_ID <- as.numeric(rownames(results)[1])
> #To see which variable was the least contributing to the model
> coef(glm_models[[best_ID]])
8 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept)   -5.849815e+02
Year           2.958188e-01
Present_Price  3.028162e-01
Kms_Driven    -1.334227e-06
Fuel_Type     -1.347878e+00
Seller_Type   -1.698074e+00
Transmission  -1.049722e+00
Owner         -3.084322e-01
> #As we can see was the variable 'Kms_Driven"
> y_pred_1se_elastic_net <- predict(glm_models[[best_ID]],newx = test_x)
> # plot 3d scatter of alpha, lambda and mse
> plot_ly(data = results, x=~alpha, y=~best_lambda.1se, z=~mse_test) %>%
+   add_markers()
```

Μπορούμε με τον παραπάνω κώδικα να κρατήσουμε τα καλύτερα Alpha = 0.067- και Lambda = 1.918 όπου θα μας αποδώσουν τα ως άνω coefficients για το μοντέλο μας.



Με το διπλανό 3D plot εξίσου να παρατηρήσουμε πως τα alpha και lambda επηρεάζουν τις τιμές των σφαλμάτων

```
> data.frame( modelName = c("OLS","Ridge","Lasso","EL"),
+             R2 = c(R2(predictions, test_y), R2(y_pred_1se, test_y),
+                    R2(y_pred_1se_lasso, test_y),R2(y_pred_1se_elastic_net,test_y)),
+             RMSE = c( RMSE(predictions, test_y), RMSE(y_pred_1se, test_y),
+                       RMSE(y_pred_1se_lasso, test_y),RMSE(y_pred_1se_elastic_net,test_y)),
+             MAE = c( MAE(predictions, test_y), MAE(y_pred_1se, test_y),
+                      MAE(y_pred_1se_lasso, test_y) ,MAE(y_pred_1se_elastic_net,test_y)),
+             MSE=c(mean( (test_y - predictions)^2 ),mean( (test_y - y_pred_1se)^2 ),mean( (test_y - y_pred_1se_lasso)^2 ),
+                   mean( (test_y - y_pred_1se_elastic_net)^2 ) ))
  modelName        R2     RMSE      MAE      MSE
1       OLS 0.8974604 1.676636 1.178818 2.811109
2     Ridge 0.8921119 2.150960 1.294577 4.626627
3     Lasso 0.8637573 2.369970 1.541991 5.616758
4        EL 0.8904260 2.111049 1.287345 4.456528
```

Τελικά μπορούμε να κάνουμε έναν συγκεντρωτικό πίνακα αποτελεσμάτων για τα καλύτερα R^2 και μικρότερα σφάλματα. Συγκεκριμένα φαίνεται πως η μέθοδος Linear Regression για όλες τις μεταβλητές είχε το μεγαλύτερο R^2 και τα μικρότερα σφάλματα.

Για την γλώσσα προγραμματισμού Python

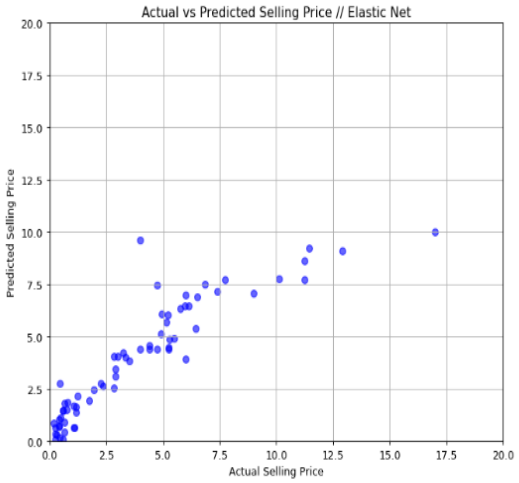
```
##Elastic Net

# Define Elastic Net model
ratios = arange(0, 1, 0.01)
alphas = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1.0, 10.0, 100.0, 1000.0]  # Removed 0.0 from alphas list
ElasticNetModel = ElasticNetCV(l1_ratio=ratios, alphas=alphas, cv=cv, n_jobs=-1)

# Fit model
ElasticNetModel.fit(x_train, y_train)

# Train and test score for Elastic Net regression
R2_train_score_en = ElasticNetModel.score(x_train, y_train)
R2_test_score_en = ElasticNetModel.score(x_test, y_test)

# Retrieve best alpha and l1_ratio
best_alpha_en = ElasticNetModel.alpha_
best_l1_ratio_en = ElasticNetModel.l1_ratio_
```

Προχωρούμε στην ως άνω διερεύνηση της καλύτερης τιμής alpha και κατόπιν στην κατασκευή του μοντέλου με τα παρακάτω αποτελέσματα:

```
>>> # Summarize chosen configuration
>>> print('ElasticNet best alpha (lambda): %f' % best_alpha_en)
ElasticNet best alpha (lambda): 0.100000
>>> print('ElasticNet best l1_ratio (alpha): %f' % best_l1_ratio_en)
ElasticNet best l1_ratio (alpha): 0.230000
>>> print("The train score for ElasticNet model is {:.4f}".format(R2_train_score_en))
The train score for ElasticNet model is 0.8901
>>> print("The test score for ElasticNet model is {:.4f}".format(R2_test_score_en))
The test score for ElasticNet model is 0.8111
```

Όπου με τους διπλανές τιμές το μοντέλο για το δείγμα εκπαίδευσης έχει score 0.89 και για το δείγμα ελέγχου 0.81

Τελικά θα κατασκευάσουμε και σχετικό διάγραμμα Actual vs Predicted



Όπου γίνεται αντιληπτή η γραμμική εξάρτηση των δύο δειγμάτων με ενδείξεις για πιθανές ακραίες τιμές. Το παρόν επιβεβαιώνει μία καλή εφαρμογή του μοντέλου.

Παρουσιάστηκαν ως άνω οι διάφορες μέθοδοι επιλογής βέλτιστων παραμέτρων που ζητήθηκαν από την υπόθεση και συγκρίθηκαν επιμέρους. Συνολικά τα μοντέλα μας προσέφεραν κοντινά αποτελέσματα με ικανοποιητικά έως εξαιρετικά προσαρμοσμένα μοντέλα με ελάχιστες διαφοροποιήσεις. Όλες οι μέθοδοι κατέληγαν στην ίδια θεώρηση για σχετικά ασήμαντες μεταβλητές ως προς την προστιθέμενη αξία του εκάστοτε μοντέλου με κατά περιπτώσεις διαφορετικές τιμές παραγόντων όπου έφεραν και τις αυξομειώσεις στις στα κριτήρια επιλογής των μοντέλων.

ΑΣΚΗΣΗ 2η

Προχωρώντας στην δεύτερη άσκηση της εργασίας, και μέσω διαγνωστικών ιατρικών δεδομένων θα κατασκευαστούν και αξιολογηθούν αντίστοιχα μοντέλα ταξινόμησης για την νόσηση ή μη μία γυναίκας από διαβήτη με βάση τα διαθέσιμα δεδομένα. Οι μέθοδοι θα γίνουν επιμέρους μέσω του στατιστικού πακέτου R και της γλώσσας προγραμματισμού Python. Θα χρησιμοποιηθούν οι μέθοδοι Logistic Regression και SVM.

Θα ξεκινήσουμε την ανάλυσή μας για όλες τις μεταβλητές του μοντέλου με το στατιστικό πακέτο R ως εξής:

- Logistic Regression

```
> #Load Data
> mydata <- fread(file = "diabetes.csv")
> #First Look
> str(mydata)
Classes 'data.table' and 'data.frame':  768 obs. of  9 variables:
 $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
 $ Glucose                 : int  148 85 183 89 137 116 78 115 197 125 ...
 $ BloodPressure           : int  72 66 64 66 40 74 50 0 70 96 ...
 $ SkinThickness           : int  35 29 0 23 35 0 32 0 45 0 ...
 $ Insulin                 : int  0 0 0 94 168 0 88 0 543 0 ...
 $ BMI                     : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
 $ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
 $ Age                     : int  50 31 32 21 33 30 26 29 53 54 ...
 $ Outcome                 : int  1 0 1 0 1 0 1 0 1 1 ...
 - attr(*, ".internal.selfref")=<externalptr>
> psych::describe(mydata)
                          vars   n   mean     sd median trimmed   mad   min    max  range  skew kurtosis   se
Pregnancies                  1 768   3.85   3.37   3.00    3.46  2.97  0.00  17.00  17.00  0.90     0.14 0.12
Glucose                      2 768 120.89  31.97 117.00  119.38 29.65  0.00 199.00 199.00  0.17     0.62 1.15
BloodPressure                3 768  69.11  19.36  72.00   71.36 11.86  0.00 122.00 122.00 -1.84     5.12 0.70
SkinThickness                4 768  20.54  15.95  23.00   19.94 17.79  0.00  99.00  99.00  0.11    -0.53 0.58
Insulin                      5 768  79.80 115.24  30.50   56.75 45.22  0.00 846.00 846.00  2.26     7.13 4.16
BMI                          6 768  31.99   7.88  32.00   31.96  6.82  0.00  67.10  67.10 -0.43     3.24 0.28
DiabetesPedigreeFunction     7 768   0.47   0.33   0.37    0.42  0.25  0.08   2.42   2.34  1.91     5.53 0.01
Age                          8 768  33.24  11.76  29.00   31.54 10.38 21.00  81.00  60.00  1.13     0.62 0.42
Outcome                      9 768   0.35   0.48   0.00    0.31  0.00  0.00   1.00   1.00  0.63    -1.60 0.02
> names(mydata)
[1] "Pregnancies"              "Glucose"                  "BloodPressure"            "SkinThickness"            "Insulin"
[6] "BMI"                      "DiabetesPedigreeFunction" "Age"                      "Outcome"
> #set all columns as numeric and factor
> mydata$Pregnancies <- as.numeric(mydata$Pregnancies)
> mydata$Glucose <- as.numeric(mydata$Glucose)
> mydata$BloodPressure <- as.numeric(mydata$BloodPressure)
> mydata$SkinThickness <- as.numeric(mydata$SkinThickness)
> mydata$Insulin <- as.numeric(mydata$Insulin)
> mydata$Age <- as.numeric(mydata$Age)
> mydata$Outcome <- factor(mydata$Outcome, levels = c(0, 1))
> str(mydata)
Classes 'data.table' and 'data.frame':  768 obs. of  9 variables:
 $ Pregnancies             : num  6 1 8 1 0 5 3 10 2 8 ...
 $ Glucose                 : num  148 85 183 89 137 116 78 115 197 125 ...
 $ BloodPressure           : num  72 66 64 66 40 74 50 0 70 96 ...
 $ SkinThickness           : num  35 29 0 23 35 0 32 0 45 0 ...
 $ Insulin                 : num  0 0 0 94 168 0 88 0 543 0 ...
 $ BMI                     : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
 $ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
 $ Age                     : num  50 31 32 21 33 30 26 29 53 54 ...
 $ Outcome                 : Factor w/ 2 levels "0","1": 2 1 2 1 2 1 2 1 2 2 ...
 - attr(*, ".internal.selfref")=<externalptr>
> # Setting seed to generate a reproducible random sampling
> set.seed(123)
> # Randomly creating training data as 75% of the dataset
> random_sample <- createDataPartition(mydata$Outcome, p = 0.75,list = F)
> # Generating training dataset from the random_sample
> training_dataset  <- mydata[random_sample,]
> # Generating testing dataset from rows which are not included in random_sample
> testing_dataset <- mydata[-random_sample, ]
```

Όπου παρουσιάζονται ονομαστικά τα δεδομένα και κατόπιν διαμορφώνονται ανάλογα με την οντότητά τους. Εν συνεχεία διαχωρίζουμε τυχαία τα δεδομένα εκπαίδευσης και ελέγχου.

```
> # Fit Logistic Regression model using 5-fold cross-validation
> glm_model <- train(Outcome ~ ., data = training_dataset, method = "glm",family = "binomial")
> summary(glm_model)

Call:
NULL

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.5322  -0.7419  -0.4193   0.7516   2.9546

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)             -8.5310268  0.8437413 -10.111  < 2e-16 ***
Pregnancies              0.0986620  0.0371840   2.653  0.00797 **
Glucose                  0.0352065  0.0042798   8.226  < 2e-16 ***
BloodPressure           -0.0130993  0.0058429  -2.242  0.02497 *
SkinThickness           -0.0018763  0.0079011  -0.237  0.81229
Insulin                 -0.0004813  0.0010424  -0.462  0.64427
BMI                      0.0919668  0.0175764   5.232 1.67e-07 ***
DiabetesPedigreeFunction 0.8609583  0.3572869   2.410  0.01597 *
Age                      0.0196020  0.0112372   1.744  0.08109 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 745.11  on 575  degrees of freedom
Residual deviance: 545.50  on 567  degrees of freedom
AIC: 563.5

Number of Fisher Scoring iterations: 5

> # Predicting the target variable
> predictions <- predict(glm_model, testing_dataset)
> # Κατασκευή data frame με τις προβλέψεις και τις πραγματικές τιμές Outcome
> results <- data.frame(
+   Predicted_Outcome = as.factor(predictions),
+   Actual_Outcome = testing_dataset$Outcome
+ )
> # Υπολογισμός απόδοσης του μοντέλου
> confusionMatrix(
+   data = results$Predicted_Outcome,
+   reference = results$Actual_Outcome
+ )
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 108  26
         1  17  41

               Accuracy : 0.776
                 95% CI : (0.7104, 0.8329)
    No Information Rate : 0.651
    P-Value [Acc > NIR] : 0.0001183

                  Kappa : 0.4912

 Mcnemar's Test P-Value : 0.2224692

            Sensitivity : 0.8640
            Specificity : 0.6119
         Pos Pred Value : 0.8060
         Neg Pred Value : 0.7069
             Prevalence : 0.6510
         Detection Rate : 0.5625
   Detection Prevalence : 0.6979
      Balanced Accuracy : 0.7380

       'Positive' Class : 0
```

Συνεχίζουμε με κατασκευή μοντέλου λογιστικής παλινδρόμησης το οποίο μας δίνει τα διπλανά αποτελέσματα.

Ο έλεγχος Wald αναδεικνύει ως σημαντικές τις μεταβλητές Pregnacies, Glucose, BloodPressure και BMI.

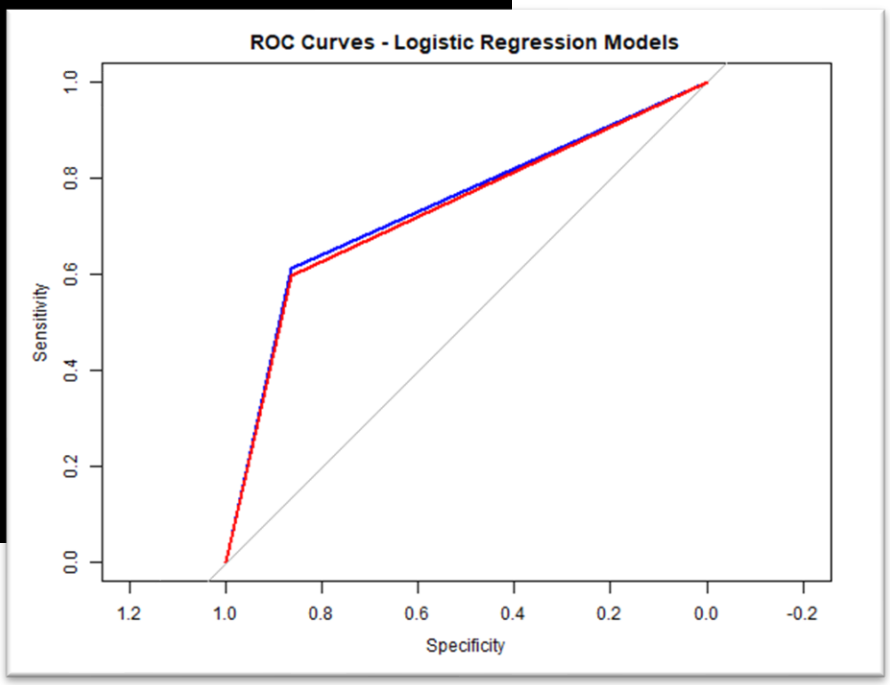
Παρατηρούμε εξίσου ικανοποιητικά αποτελέσματα στο Confusion matrix όσο και στο accuracy και τα λοιπά μέτρα όπως η ειδικότητα, η θετική και αρνητική προγνωστική, Kappa και ισορροπημένη ακρίβεια.
Για λόγους διερεύνησης θα παρατηρήσουμε και κατασκευή μοντέλου με τις πιο σημαντικές μεταβλητές σύμφωνα με τον έλεγχο του Wald.

```
> mydata2 <- subset(mydata,select=c(Pregnancies,Glucose,BloodPressure,BMI,Outcome))
> mydata2 <- subset(mydata,select=c(Pregnancies,Glucose,BloodPressure,BMI,Outcome))
> # Reproducible random sampling
> set.seed(123)
> random_sample1 <- createDataPartition(mydata2$Outcome, p = 0.75, list = FALSE)
> training_dataset2  <- mydata2[random_sample1, ]
> testing_dataset2 <- mydata2[-random_sample1, ]
> model2 <- train(Outcome ~., data = training_dataset2, "glm",family = "binomial")
> summary(model2)

Call:
NULL

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.1912  -0.7360  -0.4351   0.7630   2.9744

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  -7.817361   0.775802 -10.076  < 2e-16 ***
Pregnancies   0.131955   0.031386   4.204 2.62e-05 ***
Glucose       0.036146   0.003962   9.123  < 2e-16 ***
BloodPressure -0.011679   0.005602  -2.085   0.0371 *
BMI           0.089770   0.016307   5.505 3.70e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 745.11  on 575  degrees of freedom
Residual deviance: 555.62  on 571  degrees of freedom
AIC: 565.62

Number of Fisher Scoring iterations: 5

> predictions2 <- predict(model2, testing_dataset2)
> results2 <- data.frame(Predicted_Outcome2 = as.factor(predictions2),
+                        Actual_Outcome2 = testing_dataset2$Outcome)
> confusionMatrix(data = results2$Predicted_Outcome2, reference = results2$Actual_Outcome2)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 108  27
         1  17  40

               Accuracy : 0.7708
                 95% CI : (0.7048, 0.8283)
    No Information Rate : 0.651
    P-Value [Acc > NIR] : 0.0002216

                  Kappa : 0.4776

 Mcnemar's Test P-Value : 0.1748444

            Sensitivity : 0.8640
            Specificity : 0.5970
         Pos Pred Value : 0.8000
         Neg Pred Value : 0.7018
             Prevalence : 0.6510
         Detection Rate : 0.5625
   Detection Prevalence : 0.7031
      Balanced Accuracy : 0.7305

       'Positive' Class : 0
```

Όπου με τον ίδιο τρόπο θα κατασκευάσουμε το μοντέλο μας με τις πιο σημαντικές μεταβλητές σύμφωνα με τον έλεγχο του Wald.

Πράγματι μπορούμε να λάβουμε παρεμφερή δεδομένα σύμφωνα με το Confusion Matrix, το accuracy και τα λοιπά μέτρα σύγκρισης (ειδικότητα, η θετική και αρνητική προγνωστική, Kappa και ισορροπημένη ακρίβεια), με οικονομία του μοντέλου όπου είναι πάντα θεμιτή.

Τα παραπάνω μπορούν να επαληθευτούν και στο παρακάτω διάγραμμα.



Για την γλώσσα προγραμματισμού Python

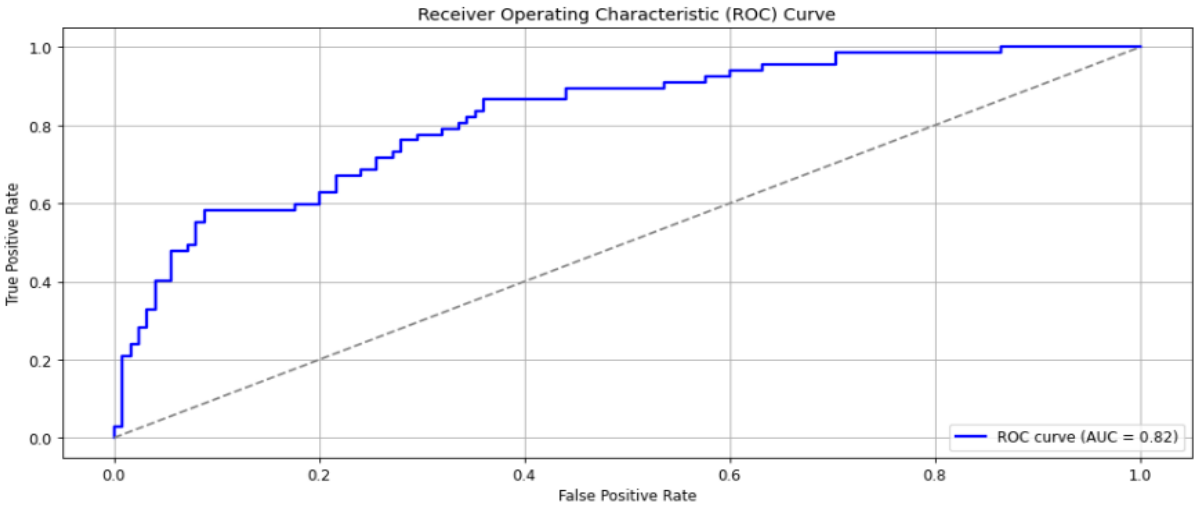
```
>>> ###Exercise 2
>>> ##For all data
>>> #Load Data
>>> data = pd.read_csv('diabetes.csv')
>>> data.head()
   Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age  Outcome
0            6      148             72  ...                     0.627   50        1
1            1       85             66  ...                     0.351   31        0
2            8      183             64  ...                     0.672   32        1
3            1       89             66  ...                     0.167   21        0
4            0      137             40  ...                     2.288   33        1

[5 rows x 9 columns]
>>> # Select Variables for x
>>> x = data.iloc[:,0:8]
>>> x.head()
   Pregnancies  Glucose  BloodPressure  ...   BMI  DiabetesPedigreeFunction  Age
0            6      148             72  ...  33.6                     0.627   50
1            1       85             66  ...  26.6                     0.351   31
2            8      183             64  ...  23.3                     0.672   32
3            1       89             66  ...  28.1                     0.167   21
4            0      137             40  ...  43.1                     2.288   33

[5 rows x 8 columns]
>>> # Select Outcome as target
>>> y = data.iloc[:,8]
>>> y.head()
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
>>> # standardize features
>>> scale = StandardScaler()
>>> x = scale.fit_transform(x)
>>> # Create a 75% random split of data for training/testing
>>> x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.75,stratify=y)
>>> ##Logistic Regression
>>> # Initialize Logistic Regression model
>>> log_reg_model = LogisticRegression()
>>> # Fit the model
>>> log_reg_model.fit(x_train, y_train)
LogisticRegression()
>>> # Retrieve coefficients of the features
>>> coefficients_log = log_reg_model.coef_
>>> # Evaluate model performance on training and testing set
>>> R2_train_score_log = log_reg_model.score(x_train, y_train)
>>> R2_test_score_log = log_reg_model.score(x_test, y_test)
>>> print("Train score for Logistic Regression: {:.4f}".format(R2_train_score_log))
Train score for Logistic Regression: 0.7743
>>> print("Train score for Logistic Regression: {:.4f}".format(R2_test_score_log))
Train score for Logistic Regression: 0.7552
>>> print("Coefficients:", coefficients_log)
Coefficients: [[ 0.27783463  1.2003717  -0.28988377 -0.06133545 -0.2144512   0.69875829
   0.18217472  0.22474678]]
```

Με την ίδια μέθοδο θα προχωρήσουμε σαν ανάγνωση των δεδομένων και τον διαχωρισμό τους σε target και δείγμα.

Κατόπιν κανονικοποίησης των δεδομένων και διαχωρισμό του δείγματος σε εκπαίδευσης και ελέγχου κατασκευάζουμε το μοντέλο στο οποίο και παρατηρούνται R^2 και για τα δύο δείγματα κοντά στο 0.76 αρκετά ικανοποιητικά.

Μαζί και με την αποτύπωση των coefficients μπορούμε να προχωρήσουμε σε απεικόνιση του μοντέλου με το επόμενο διάγραμμα.

```
# Get predicted probabilities
y_pred_prob = log_reg_model.predict_proba(x_test)[:, 1]

# Calculate fpr, tpr, thresholds and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



Στο οποίο παρατηρείται ένα πολύ καλό AUC να ισούται με 0.82 και μία αρκετά ικανοποιητική καμπύλη προς το άνω αριστερό άκρο του διαγράμματος

- Support Vector Machines

Για το στατιστικό πακέτο R

```r
## Support Vector Machines

# Setting seed to generate a reproducible random sampling
set.seed(123)

#Scale
mydata1 <- copy(mydata)  # Making a copy of the original data

numeric_columns <- names(mydata)[sapply(mydata, is.numeric) & names(mydata) != "Outcome"]
mydata1[, (numeric_columns) := lapply(.SD, scale), .SDcols = numeric_columns]
mydata1$Outcome <- as.factor(mydata1$Outcome)

# Setting seed to generate a reproducible random sampling
set.seed(123)

# Randomly creating training data as 75% of the dataset
random_sample <- createDataPartition(mydata1$Outcome, p = 0.75,list = F)

# Generating training dataset from the random_sample
training_dataset  <- mydata1[random_sample,]

# Generating testing dataset from rows which are not included in random_sample
testing_dataset <- mydata1[-random_sample, ]

### Identify the best kernel, cost C, and sigma
kernelValues <- c("linear", "radial")
costValuesToCheck <- 10^(-5:5)
sigmaValuesToCheck <- 10^(-3:3)

trainAccuracy <- array(NA, dim = c(length(kernelValues), length(costValuesToCheck), length(sigmaValuesToCheck)))
testAccuracy <- array(NA, dim = c(length(kernelValues), length(costValuesToCheck), length(sigmaValuesToCheck)))

for (k in 1:length(kernelValues)) {
  for (i in 1:length(costValuesToCheck)) {
    for (j in 1:length(sigmaValuesToCheck)) {
      cat("Calculate SVM accuracy for kernel =", kernelValues[k], ", cost C =", costValuesToCheck[i], "and sigma =", sigmaValuesToCheck[j], "\n")
      curSVM <- svm(formula = Outcome ~ .,
                    data = training_dataset,
                    type = 'C-classification',
                    kernel = kernelValues[k],
                    scale = FALSE,
                    cost = costValuesToCheck[i],
                    gamma = 1 / (2 * sigmaValuesToCheck[j]^2))
      trainAccuracy[k, i, j] <- confusionMatrix(data = curSVM$fitted,
                                                reference = training_dataset$Outcome)$overall[1]
      testAccuracy[k, i, j] <- confusionMatrix(data = predict(curSVM, testing_dataset),
                                               reference = testing_dataset$Outcome)$overall[1]
    }
  }
}
```

Κατόπιν μορφοποίησης των δεδομένων κατάλληλα μπορούμε να προχωρήσουμε στον διαχωρισμό των δεδομένων του δείγματος σε εκπαίδευσης και ελέγχου. Αφετέρου και βάσει της υπόθεσης προχωρούμε σε διερεύνηση καλύτερου κόστους C , sigma και kernel μέσω ενός for loop το οποίο εξετάζει το καλύτερο accuracy που δίνει ο κάθε πιθανός συνδυασμός των ανωτέρω με τα παρακάτω αποτελέσματα:

```
> # Finding the best parameters
> best_accuracy <- max(testAccuracy)
> best_indices <- which(testAccuracy == best_accuracy, arr.ind = TRUE)
> best_kernel <- kernelValues[best_indices[1,1]]
> best_cost <- costValuesToCheck[best_indices[1,2]]
> best_sigma <- sigmaValuesToCheck[best_indices[1,3]]
>
> best_accuracy
[1] 0.7916667
> best_kernel
[1] "radial"
> best_cost
[1] 1
> best_sigma
[1] 10
>
> # Defining training control as cross-validation and value of K equal to 5
> train_control <- trainControl(method = "cv",number = 5)
>
> # Fitting SVM to the Training set with the best parameters
> SVMclassifier <- svm(
+   formula = Outcome ~ .,
+   data = training_dataset,
+   type = 'C-classification',
+   kernel = best_kernel,  # Use the best kernel value
+   scale = FALSE,
+   cost = best_cost,  # Use the best 'C' parameter
+   gamma = 1 / (2 * best_sigma^2), # Use the best 'sigma' parameter
+   trControl = train_control  # Use the best 'sigma' parameter
+ )
>
> summary(SVMclassifier)

Call:
svm(formula = Outcome ~ ., data = training_dataset, type = "C-classification", kernel = best_kernel, cost = best_cost, gamma = 1/(2 *
    best_sigma^2), trControl = train_control, scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  373

 ( 187 186 )


Number of Classes:  2

Levels:
 0 1
```

Όπου βλέπουμε πως τα καλύτερα αποτελέσματα δίνονται με accuracy 0.79 με τις εξής παραμέτρους:

C = 1, kernel = radial και sigma = 10

Με τα παραπάνω προχωρούμε σε μοντελοποίηση για περαιτέρω ανάλυση.

```
> # Training confusion matrix
> cm1 <- confusionMatrix(data = SVMclassifier$fitted, reference = training_dataset$Outcome)
>
> cm1$overall
      Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull AccuracyPValue  McnemarPValue
  7.725694e-01   4.506378e-01   7.361181e-01   8.061979e-01   6.510417e-01   1.647382e-10   3.133623e-11
>
> # Predicting the Test set results
> y_pred <- predict(SVMclassifier, newdata = testing_dataset)
>
> # Making the Test Confusion Matrix
> cm <- table(testing_dataset$Outcome, y_pred)
> confusionMatrix(cm)
Confusion Matrix and Statistics

   y_pred
     0   1
  0 116   9
  1  31  36

               Accuracy : 0.7917
                 95% CI : (0.7273, 0.8468)
    No Information Rate : 0.7656
    P-Value [Acc > NIR] : 0.2235298

                  Kappa : 0.5037

 Mcnemar's Test P-Value : 0.0008989

            Sensitivity : 0.7891
            Specificity : 0.8000
         Pos Pred Value : 0.9280
         Neg Pred Value : 0.5373
             Prevalence : 0.7656
         Detection Rate : 0.6042
   Detection Prevalence : 0.6510
      Balanced Accuracy : 0.7946

       'Positive' Class : 0

>
> SVMclassifier

Call:
svm(formula = Outcome ~ ., data = training_dataset, type = "C-classification", kernel = best_kernel, cost = best_cost, gamma = 1/(2 *
    best_sigma^2), trControl = train_control, scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  373

> SVMclassifier$SV
    Pregnancies      Glucose BloodPressure SkinThickness      Insulin           BMI DiabetesPedigreeFunction         Age
1    0.63953049  0.847771321    0.14954330    0.90667906 -0.692439325  0.2038799073             0.4681868702  1.42506672
2    1.23307662  1.942458024   -0.26376935   -1.28737326 -0.692439325 -1.1025369566             0.6040037019 -0.10551539
5   -0.25078869 -1.341602086   -0.98706650    0.71861743  0.071157897 -0.1258952234            -0.6756926672 -0.61570943
341 -1.14110788 -0.528406249   -0.26376935    1.03205348 -0.137095891  0.2038799073             0.1150631080 -0.95583878
344 -0.54756176  0.409896639    0.04621514   -1.28737326 -0.692439325 -0.3922520597             0.2116439661 -0.87080644
348  3.01371499  1.692243920   -0.36709752    0.59324302 -0.692439325  0.2038799073            -0.7843461325  0.40467865
364 -0.54756176 -0.497129486    0.56285595    1.53355115  0.964913736  0.2165635661             0.7217116226 -0.36061241
 [ reached getOption("max.print") -- omitted 248 rows ]
> dim(SVMclassifier$SV)
[1] 373   8
```

Παραπάνω μπορούν να εντοπισθούν πολύ ικανοποιητικά αποτελέσματα όσο στο Confusion Matrix όσο και στις διάφορες τιμές των επιμέρους ελέγχων με Accuracy = 0.79, θετική προγνωστική 0.93, αρνητική 0.54 και ισορροπημένη ακρίβεια 0.7946. Επίσης εμφανίζεται Kappa = 0.5037, ευαισθησία 0.7891 και ειδικότητα 0.8 .

Για την γλώσσα προγραμματισμού Python

```
##Support Vector Machines

# Identify best hyperparameters by exhaustive grid search
# Defining parameter ranges
param_grid = {
    'C': [10 ** i for i in range(-5, 6)],  # Values of C from 10^-5 to 10^5
    'gamma': [10 ** i for i in range(-3, 4)],  # Values of gamma (sigma) from 10^-3 to 10^3
    'kernel': ['linear', 'rbf']
}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3, cv=5)

# fitting the model for grid search
grid.fit(x_train, y_train)
```

Με τις ίδιες παραμέτρους προχωρούμε σε διερεύνηση των καλύτερων C, Sigma (Gamma για την Python) και kernel. Κατόπιν κατασκευάζουμε το μοντέλο

```
>>> print(grid.best_params_)
{'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
>>>
>>> # print how our model looks after hyper-parameter tuning
>>> best_model = grid.best_estimator_
>>> print(best_model)
SVC(C=1, gamma=0.01)
>>>
>>> grid_predictions = grid.predict(x_test)
>>>
>>> # print classification report
>>> print(metrics.classification_report(y_train, grid.best_estimator_.predict(x_train),
...    target_names=["0", "1"]))
              precision    recall  f1-score   support

           0       0.65      0.74      0.70       375
           1       0.36      0.26      0.30       201

    accuracy                           0.58       576
   macro avg       0.50      0.50      0.50       576
weighted avg       0.55      0.58      0.56       576

>>>
>>> print(metrics.confusion_matrix(y_test, grid_predictions))
[[94 31]
 [55 12]]
>>> print(metrics.classification_report(y_test, grid_predictions,target_names=['0', '1']))
              precision    recall  f1-score   support

           0       0.63      0.75      0.69       125
           1       0.28      0.18      0.22        67

    accuracy                           0.55       192
   macro avg       0.45      0.47      0.45       192
weighted avg       0.51      0.55      0.52       192
```

Μετά από την σχετική διερεύνηση το καλύτερο accuracy επιτεύχθηκε με κόστος C = 1, Sigma (Gamma για την Python) = 0.01 και kernel = rbf .

Σύμφωνα με το Confusion Matrix και τα διάφορα μέτρα σύγκριση που παρουσιάζονται τα αποτελέσματα αρκούν για να προχωρήσουμε το μοντέλο μας όμως δεν είναι και τα πιο ικανοποιητικά.

Συνεχίζοντας την άσκηση βάσει της υπόθεσης θα δημιουργήσουμε δείγμα μόνο με τις μεταβλητές Glucose και BMI ώστε να προβούμε στην κατασκευή των παραπάνω μεθόδων μοντελοποίησης.

- Logistic Regression // *Για το στατιστικό πακέτο R*

```
## For variables: "Glucose" and "BMI"

mydata3 <- mydata[, c("Glucose", "BMI", "Outcome")]
mydata3$Outcome <- as.factor(mydata3$Outcome)

# Setting seed to generate a reproducible random sampling
set.seed(123)

# Randomly creating training data as 75% of the dataset
random_sample3 <- createDataPartition(mydata3$Outcome, p = 0.75,list = F)

# Generating training dataset from the random_sample
training_dataset3  <- mydata3[random_sample3,]

# Generating testing dataset from rows which are not included in random_sample
testing_dataset3 <- mydata3[-random_sample3, ]

##Logistic Regression

# Fit Logistic Regression model using 5-fold cross-validation
glm_model3 <- train(Outcome ~ ., data = training_dataset3, method = "glm",family = "binomial")
```

Ξεκινάμε δημιουργώντας εκ νέου δείγμα από το αρχείο csv με τις επιθυμητές μεταβλητές. Κατόπιν χωρίζεται κατάλληλα σε δείγμα εκπαίδευσης και ελέγχου.
Στο τέλος του διπλανού κώδικα κατασκευάζουμε το μοντέλο παλινδρόμησης.

```
> summary(glm_model3)

Call:
NULL

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.2257  -0.7714  -0.4624   0.7650   3.0015

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -7.78198    0.72217 -10.776  < 2e-16 ***
Glucose      0.03622    0.00389   9.312  < 2e-16 ***
BMI          0.08021    0.01560   5.141 2.73e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 745.11  on 575  degrees of freedom
Residual deviance: 576.33  on 573  degrees of freedom
AIC: 582.33

Number of Fisher Scoring iterations: 4
```

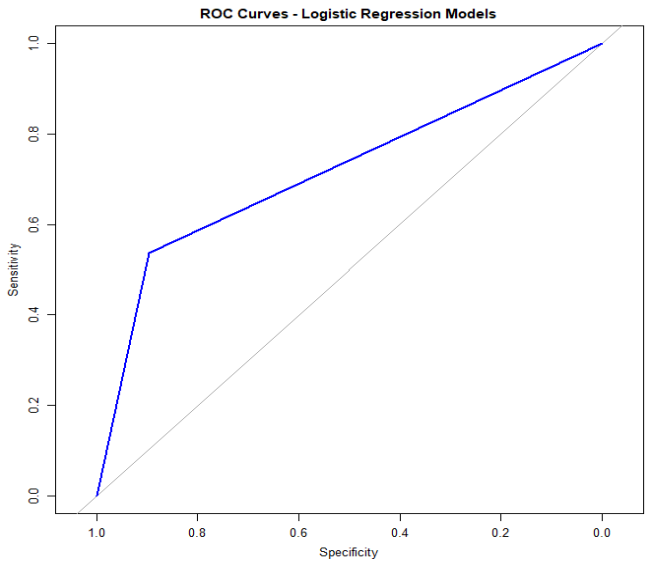
Από το summary του μοντέλου μπορούμε να παρατηρήσουμε την πολύ υψηλή σημαντικότητα των δύο παραμέτρων για το μοντέλο μας. Άξιο αναφοράς είναι πως το AIC που λάβαμε είναι το μεγαλύτερο της άσκησης έως τώρα.

```
> # Predicting the target variable
> predictions3 <- predict(glm_model3, testing_dataset3)
>
> # Κατασκευή data frame με τις προβλέψεις και τις πραγματικές τιμές Outcome
> results <- data.frame(
+   Predicted_Outcome = as.factor(predictions3),
+   Actual_Outcome = testing_dataset3$Outcome
+ )
> # Υπολογισμός απόδοσης του μοντέλου
> confusionMatrix(
+   data = results$Predicted_Outcome,
+   reference = results$Actual_Outcome
+ )
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 112  31
         1  13  36

               Accuracy : 0.7708
                 95% CI : (0.7048, 0.8283)
    No Information Rate : 0.651
    P-Value [Acc > NIR] : 0.0002216

                  Kappa : 0.4621

 Mcnemar's Test P-Value : 0.0103818

            Sensitivity : 0.8960
            Specificity : 0.5373
         Pos Pred Value : 0.7832
         Neg Pred Value : 0.7347
             Prevalence : 0.6510
         Detection Rate : 0.5833
   Detection Prevalence : 0.7448
      Balanced Accuracy : 0.7167

       'Positive' Class : 0

> # ROC curve for model
> roc_data3 <- roc(testing_dataset3$Outcome, as.numeric(predictions3))
Setting levels: control = 0, case = 1
Setting direction: controls < cases
>
> # Plotting
> plot(roc_data3, col = "blue", main = "ROC Curves - Logistic Regression Models")
>
> #We can assume a very efficient model
>
```

Συνεχίζοντας την ανάλυση κατασκευάζουμε Confusion Matrix ώστε να επιβεβαιώσουμε την ορθή αντιστοίχιση των περιπτώσεων, η οποί και δίνει πολύ καλά αποτελέσματα.

Επιπροσθέτως, έχουμε καταφέρει να λάβουμε ένα αρκετά ικανοποιητικό accuracy = 0.77, balanced = 0.72 και παρόμοιες τιμές για την θετική και αρνητική προγνωστική.

Τελικά θα απεικονίσουμε τα ανωτέρω και στο επόμενο διάγραμμα:



Από το οποίο είναι εμφανές πως το μοντέλο μας έχει ορθά αποτελέσματα με μία ικανοποιητική τιμή για το AUC.

Ιδανικά για ένα εξαιρετικό αποτέλεσμα θα θέλαμε η ράχη της θεωρητικής καμπύλης να προσεγγίζει το άνω αριστερό άκρο του διαγράμματος.

Για την γλώσσα προγραμματισμού Python

```
>>> ##For the Variables Glucose and BMI
>>>
>>> # Select Age and Estimated Salary as features (predictors)
>>> x2 = data.iloc[:, [1, 5]]
>>> x2.head()
   Glucose   BMI
0      148  33.6
1       85  26.6
2      183  23.3
3       89  28.1
4      137  43.1
>>> # Select Item Purchased as target
>>> y.head()
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

Όμοια δημιουργούμε εκ νέου δείγμα από το αρχείο csv με τις επιθυμητές μεταβλητές.

Παρακάτω προχωράμε σε κανονικοποίηση και διαχωρισμό σε δείγμα εκπαίδευσης και ελέγχου.

```
# standardize features
x2 = scale.fit_transform(x2)

# Create a 75% random split of data for training/testing
x2_train, x2_test, y_train, y_test = train_test_split(x2, y, train_size=0.75,stratify=y)


#Logistic Regression

# Initialize Logistic Regression model
log_reg_model1 = LogisticRegression()
# Fit the model
log_reg_model1.fit(x2_train, y_train)

# Retrieve coefficients of the features
coefficients_log1 = log_reg_model1.coef_

# Evaluate model performance on training and testing set
R2_train_score_log1 = log_reg_model1.score(x2_train, y_train)
R2_test_score_log1 = log_reg_model1.score(x2_test, y_test)
```

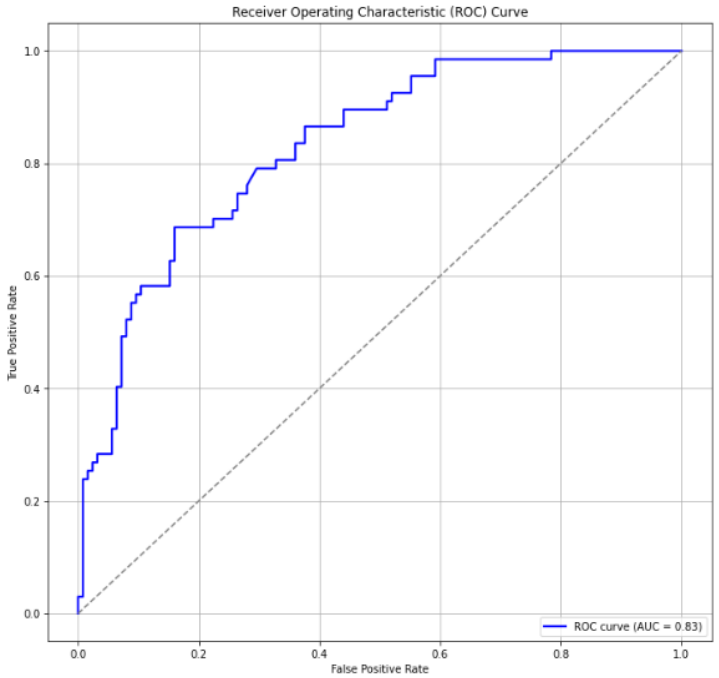
Με τελικό σκοπό την κατασκευή του μοντέλου λογιστικής παλινδρόμησης και στον υπολογισμό μέτρων σύγκριση όπως μπορεί να παρατηρηθεί στον διπλανό κώδικα.

```
>>> print("Train score for Logistic Regression: {:.4f}".format(R2_train_score_log1))
Train score for Logistic Regression: 0.7569
>>> print("Train score for Logistic Regression: {:.4f}".format(R2_test_score_log1))
Train score for Logistic Regression: 0.7708
>>> print("Coefficients:", coefficients_log1)
Coefficients: [[1.01748965 0.64247762]]
>>> # Get predicted probabilities
```

Παρατηρούνται εξίσου ικανοποιητικά R^2 για το δείγμα εκπαίδευσης και ελέγχου

Τελικά θα κατασκευάσουμε και το επόμενο διάγραμμα για να έχουμε μία ολοκληρωμένη εικόνα:

```python
# Get predicted probabilities
y_pred_prob1 = log_reg_model1.predict_proba(x2_test)[:, 1]

# Calculate fpr, tpr, thresholds and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob1)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



Όπου με AUC = 0.83 έχουμε μία πολύ καλή εικόνα για την αποτελεσματικότητα του μοντέλου βάσει και της αρκετά αντιπροσωπευτικής καμπύλης για ένα πολύ καλό αποτέλεσμα όπως προαναφέραμε.

- Support Vector Machines // *Για το στατιστικό πακέτο R*

```r
## Support Vector Machines

# Scale only the numeric columns: Glucose and BMI
numeric_cols <- names(mydata3)[sapply(mydata3, is.numeric)]
mydata3[, (numeric_cols) := lapply(.SD, scale), .SDcols = numeric_cols]

# Convert to data.table
setDT(mydata3)

# Ensure the 'Outcome' column remains unchanged
mydata3$Outcome <- as.factor(mydata3$Outcome)

# Setting seed to generate a reproducible random sampling
set.seed(400)

# Randomly creating training data as 75% of the dataset
random_sample <- createDataPartition(mydata3$Outcome, p = 0.75,list = F)

# Generating training dataset from the random_sample
training_dataset  <- mydata3[random_sample,]

# Generating testing dataset from rows which are not included in random_sample
testing_dataset <- mydata3[-random_sample, ]

### Identify the best kernel, cost C, and sigma as before

for (k in 1:length(kernelValues)) {
  for (i in 1:length(costValuesToCheck)) {
    for (j in 1:length(sigmaValuesToCheck)) {
      cat("Calculate SVM accuracy for kernel =", kernelValues[k], ", cost C =", costValuesToCheck[i], "and sigma =", sigmaValuesToCheck[j], "\n")
      curSVM <- svm(formula = Outcome ~ .,
                    data = training_dataset,
                    type = 'C-classification',
                    kernel = kernelValues[k],
                    scale = FALSE,
                    cost = costValuesToCheck[i],
                    gamma = 1 / (2 * sigmaValuesToCheck[j]^2))
      trainAccuracy[k, i, j] <- confusionMatrix(data = curSVM$fitted,
                                                reference = training_dataset$Outcome)$overall[1]
      testAccuracy[k, i, j] <- confusionMatrix(data = predict(curSVM, testing_dataset),
                                               reference = testing_dataset$Outcome)$overall[1]
    }
  }
}
```

Παραπάνω μπορούμε να δούμε πως μετά από ορθή κανονικοποίηση του δείγματος προχωρήσαμε για άλλη μία φορά στον διαχωρισμό του σε εκπαίδευσης και ελέγχου. Κατόπιν με ένα for loop θα αναγνωρίσουμε βάσει του accuracy ποιες θα είναι οι καλύτερες τιμές για το κόστος C, την μεταβλητή sigma και το kernel.

```
> best_accuracy
[1] 0.7291667
> best_kernel
[1] "radial"
> best_cost
[1] 1e+05
> best_sigma
[1] 1
> # Fitting SVM to the Training set with the best parameters
> SVMclassifier3 <- svm(
+   formula = Outcome ~ .,
+   data = training_dataset,
+   type = 'C-classification',
+   kernel = best_kernel,  # Use the best kernel value
+   scale = FALSE,
+   cost = best_cost,  # Use the best 'C' parameter
+   gamma = 1 / (2 * best_sigma^2), # Use the best 'sigma' parameter
+   trControl = train_control
+   )

WARNING: reaching max number of iterations
> summary(SVMclassifier3)

Call:
svm(formula = Outcome ~ ., data = training_dataset, type = "C-classification", kernel = best_kernel, cost = best_cost, gamma = 1/(2 * best_sigma^2),
    trControl = train_control, scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1e+05

Number of Support Vectors:  271

 ( 134 137 )


Number of Classes:  2

Levels:
 0 1
```

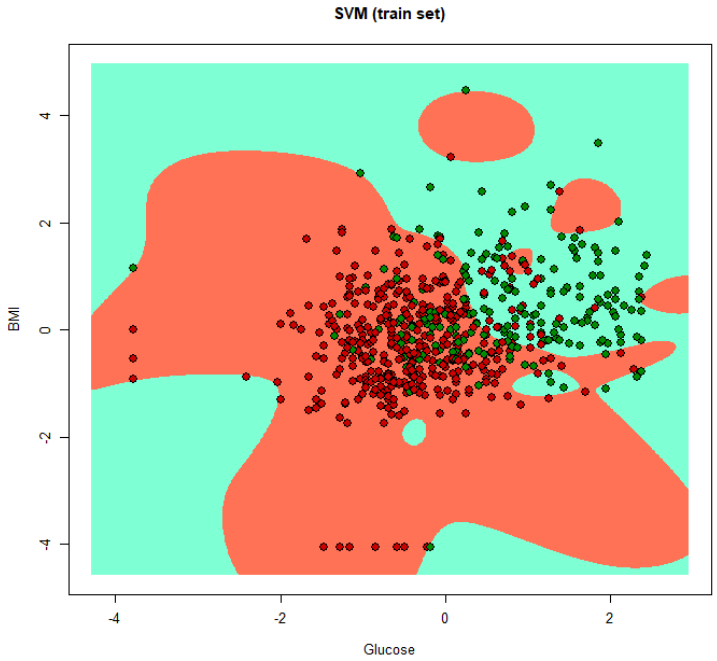
Παραπάνω θα δούμε τα αποθηκευμένα καλύτερα αποτελέσματα που μας δίνουν accuracy = 0.73 να είναι τα C= 100000, sigma = 1 και kernel = radial . Με τα οποία θα κατασκευάσουμε και το θεωρητικά ιδανικό μοντέλο.

```
> # Training confusion matrix
> cm3 <- confusionMatrix(data = SVMclassifier3$fitted, reference = training_dataset$Outcome)
> cm3$overall
      Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull AccuracyPValue  McnemarPValue
  8.090278e-01   5.539599e-01   7.745057e-01   8.403427e-01   6.510417e-01   5.052365e-17   2.983294e-06
> # Predicting the Test set results
> y_pred3 <- predict(SVMclassifier3, newdata = testing_dataset)
> # Making the Test Confusion Matrix
> cm <- table(testing_dataset$Outcome, y_pred3)
> cm
   y_pred3
      0   1
  0 110  15
  1  37  30
> confusionMatrix(cm)
Confusion Matrix and Statistics

   y_pred3
      0   1
  0 110  15
  1  37  30

               Accuracy : 0.7292
                 95% CI : (0.6605, 0.7906)
    No Information Rate : 0.7656
    P-Value [Acc > NIR] : 0.897846

                  Kappa : 0.3548

 Mcnemar's Test P-Value : 0.003589

            Sensitivity : 0.7483
            Specificity : 0.6667
         Pos Pred Value : 0.8800
         Neg Pred Value : 0.4478
             Prevalence : 0.7656
         Detection Rate : 0.5729
   Detection Prevalence : 0.6510
      Balanced Accuracy : 0.7075

       'Positive' Class : 0
```

Από το οποίο εκλαμβάνουμε τα διπλανά αποτελέσματα με ένα ικανοποιητικό Confusion Matrix καθώς και εκτός των υπολοίπων μετρήσεων ένα καλό accuracy = 0.73 , υψηλή θετική πρόγνωση στο 0.88 και καλή ισορροπημένη ακρίβεια στο 0.71.

```r
CreatePlot <- function(dataSet, SVMclassifierObject, plotTitle) {

  X1 <- seq(min(dataSet[, 1]) - 0.5, max(dataSet[, 1]) + 0.5, by = 0.01)
  X2 <- seq(min(dataSet[, 2]) - 0.5, max(dataSet[, 2]) + 0.5, by = 0.01)

  grid_set <- expand.grid(X1, X2)
  colnames(grid_set) <- c('Glucose', 'BMI')
  y_grid <- predict(SVMclassifierObject, newdata = grid_set)

  plot(dataSet[, -3],
       main = plotTitle,
       xlab = 'Glucose', ylab = 'BMI',
       xlim = range(X1), ylim = range(X2))

  contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)

  # Change 'Yes' and 'No' to '1' and '0' for coloring
  points(grid_set$Glucose, grid_set$BMI, pch = '.', col = ifelse(y_grid == 1, 'aquamarine', 'coral1'))

  points(dataSet, pch = 21, cex = 1.5, bg = ifelse(dataSet[, 3] == 1, 'green4', 'red3'))
}

CreatePlot(dataSet = training_dataset,SVMclassifierObject = SVMclassifier3,
           plotTitle = "SVM (train set)")
```

Συνεχίζοντας στην οπτικοποίηση δημιουργούμε την διπλανή συνάρτηση η οποία θα δημιουργήσει Scatter Plot χρωματικά χωρισμένο βάσει της ταξινόμησης καθώς και χρωματισμό των χωρίων επιλογής.



ως έχουν.

```
> dim(SVMclassifier3$SV)
[1] 271   2
```

Παρατηρείται αρκετά συγκεκριμένη οριοθέτηση στο κέντρο και γύρω του διαγράμματος. Γεγονός που οφείλεται στο υψηλό κόστος που επιλέχθηκε (ένδειξη overfitting). Οι νησίδες αποτελούν χαρακτηριστικό της radial καθώς και η πολυπλοκότητάς τους βάσει του sigma.

Βάσει της υπόθεσης καθώς και είναι άξιο παρατηρήσεων θα δούμε και τις περιπτώσεις με κόστος C = 1 και C = 1000 , διατηρώντας τα υπόλοιπα

```r
##At last we will try the abovementioned for costs: C=1 and C=1000

#C=1
# Fitting SVM to the Training set with the parameters
SVMclassifier4 <- svm(
  formula = Outcome ~ .,
  data = training_dataset,
  type = 'C-classification',
  kernel = best_kernel,  # Use the best kernel value
  scale = FALSE,
  cost = 1,
  gamma = 1 / (2 * best_sigma^2), # Use the best 'sigma' parameter
  trControl = train_control
)
```

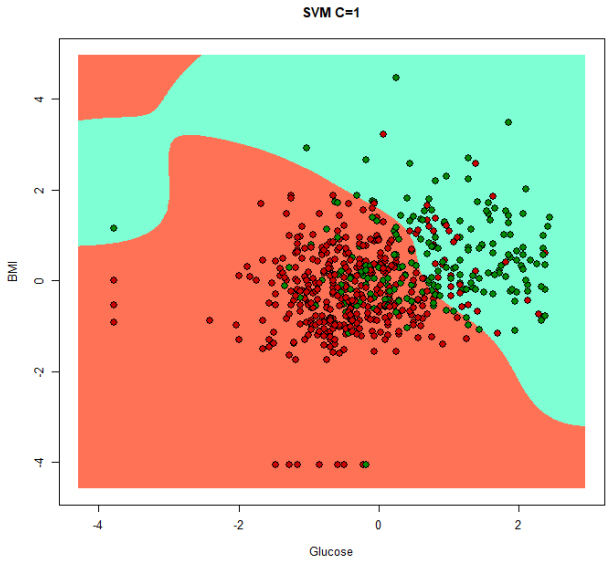
Ξεκινάμε με την κατασκευή του μοντέλου με κόστος C = 1 και παρακάτω θα δούμε την ανάλυσή του.

```
> summary(SVMclassifier4)

Call:
svm(formula = Outcome ~ ., data = training_dataset, type = "C-classification", kernel = best_kernel, cost = 1, gamma = 1/(2 * best_sigma^2), trControl = train_control,
    scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  290

 ( 141 149 )


Number of Classes:  2

Levels:
 0 1



> cm4 <- confusionMatrix(data = SVMclassifier4$fitted, reference = training_dataset$Outcome)
> cm4$overall
      Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull AccuracyPValue  McnemarPValue
  7.829861e-01   4.937563e-01   7.470457e-01   8.159942e-01   6.510417e-01   3.596189e-12   1.760761e-05
> y_pred4 <- predict(SVMclassifier4, newdata = testing_dataset)
> cm4 <- table(testing_dataset$Outcome, y_pred4)
> cm4
   y_pred4
     0   1
  0 112  13
  1  42  25
> confusionMatrix(cm4)
Confusion Matrix and Statistics

   y_pred4
     0   1
  0 112  13
  1  42  25

               Accuracy : 0.7135
                 95% CI : (0.644, 0.7763)
    No Information Rate : 0.8021
    P-Value [Acc > NIR] : 0.9987984

                  Kappa : 0.2992

 Mcnemar's Test P-Value : 0.0001597

            Sensitivity : 0.7273
            Specificity : 0.6579
         Pos Pred Value : 0.8960
         Neg Pred Value : 0.3731
             Prevalence : 0.8021
         Detection Rate : 0.5833
   Detection Prevalence : 0.6510
      Balanced Accuracy : 0.6926

       'Positive' Class : 0

> CreatePlot(dataSet = training_dataset,SVMclassifierObject = SVMclassifier4,
+            plotTitle = "SVM C=1")
>
> dim(SVMclassifier4$SV)
[1] 290   2
```

Μπορεί να παρατηρηθεί στα παραπάνω output μείωση των αποδόσεων τόσο σε ακρίβεια όσο και στις υπόλοιπες μεθόδους σύγκρισης που προαναφέραμε

```
CreatePlot(dataSet = training_dataset,SVMclassifierObject = SVMclassifier4,
           plotTitle = "SVM C=1")

dim(SVMclassifier4$SV)
```

Έχοντας μικρότερο κόστος μπορεί να παρατηρηθεί μία όχι και τόσο overfit κατανομή των δύο χωρίων ταξινόμησης με τα δύο σύνολα σχεδόν να ισομοιράζονται με μία βελτιωμένη εικόνα.



```r
#C=1000
# Fitting SVM to the Training set with the parameters
SVMclassifier5 <- svm(
  formula = Outcome ~ .,
  data = training_dataset,
  type = 'C-classification',
  kernel = best_kernel,  # Use the best kernel value
  scale = FALSE,
  cost = 1000,
  gamma = 1 / (2 * best_sigma^2), # Use the best 'sigma' parameter
  trControl = train_control
)
```

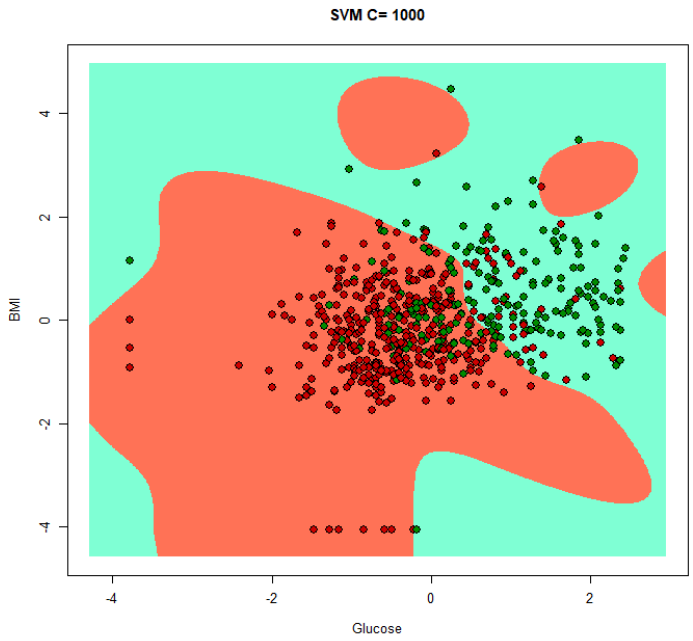
Τέλος για την διένεξη της άσκησης με το στατιστικό πακέτο R, κατασκευάζουμε το μοντέλο με κόστος C = 1000.

```
> summary(SVMclassifier5)

Call:
svm(formula = Outcome ~ ., data = training_dataset, type = "C-classification", kernel = best_kernel, cost = 1000, gamma = 1/(2 * best_sigma^2), trControl = train_control,
    scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1000

Number of Support Vectors:  278

 ( 137 141 )


Number of Classes:  2

Levels:
 0 1



>
> # Training confusion matrix
> cm5 <- confusionMatrix(data = SVMclassifier5$fitted, reference = training_dataset$Outcome)
> cm5$overall
      Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull AccuracyPValue  McnemarPValue
  7.934028e-01   5.261717e-01   7.580047e-01   8.257588e-01   6.510417e-01   5.470099e-14   1.828411e-03
>
> # Predicting the Test set results
> y_pred5 <- predict(SVMclassifier5, newdata = testing_dataset)
>
> # Making the Test Confusion Matrix
> cm5<- table(testing_dataset$Outcome, y_pred5)
> cm5
   y_pred5
     0   1
  0 108  17
  1  38  29
> confusionMatrix(cm4)
Confusion Matrix and Statistics

   y_pred4
     0   1
  0 112  13
  1  42  25

               Accuracy : 0.7135
                 95% CI : (0.644, 0.7763)
    No Information Rate : 0.8021
    P-Value [Acc > NIR] : 0.9987984

                  Kappa : 0.2992

 Mcnemar's Test P-Value : 0.0001597

            Sensitivity : 0.7273
            Specificity : 0.6579
         Pos Pred Value : 0.8960
         Neg Pred Value : 0.3731
             Prevalence : 0.8021
         Detection Rate : 0.5833
   Detection Prevalence : 0.6510
      Balanced Accuracy : 0.6926

       'Positive' Class : 0
```

Σε σύγκριση με τα προηγούμενα αποτελέσματα, παρουσιάζοντας τους ίδιους ελέγχους, ενώ πετυχαίνουμε το ίδιο accuracy υπάρχει ένας πιο ομαλός καταμερισμός των περιπτώσεων στην διαγώνιο των λάθος ταξινομήσεων του Confusion Matrix. Για την τελική σύγκριση θα κατασκευάσουμε το επόμενο διάγραμμα:

```r
CreatePlot(dataSet = training_dataset,SVMclassifierObject = SVMclassifier5,
           plotTitle = "SVM C= 1000")
```

```
> dim(SVMclassifier5$SV)
[1] 278   2
```



Όπου και όπως θα αναμέναμε με τα ίδια στοιχεία του Scatter plot έχουμε ένα διαχωρισμό στα χωρία ταξινόμησης μεταξύ του πρώτου αρκετά συγκεκριμένου διαγράμματος και του δεύτερου απλά χωρισμένου. Παρατηρούνται να έχουν σχηματιστεί μερικές νησίδες με τα δύο χωρία να αρχίζουν να διαχωρίζονται και από μέσα προς τα έξω.

Για την γλώσσα προγραμματισμού Python

```
##Support Vector Machines

# Identify best hyperparameters by exhaustive grid search
grid2 = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3, cv=5)
# fitting the model for grid search
grid2.fit(x2_train, y_train)
```

Έχοντας φτιάξει κατάλληλα τα δεδομένα παραπάνω, προχωρούμε σε κατασκευή του μοντέλου αναζήτησης καλύτερων παραμέτρων

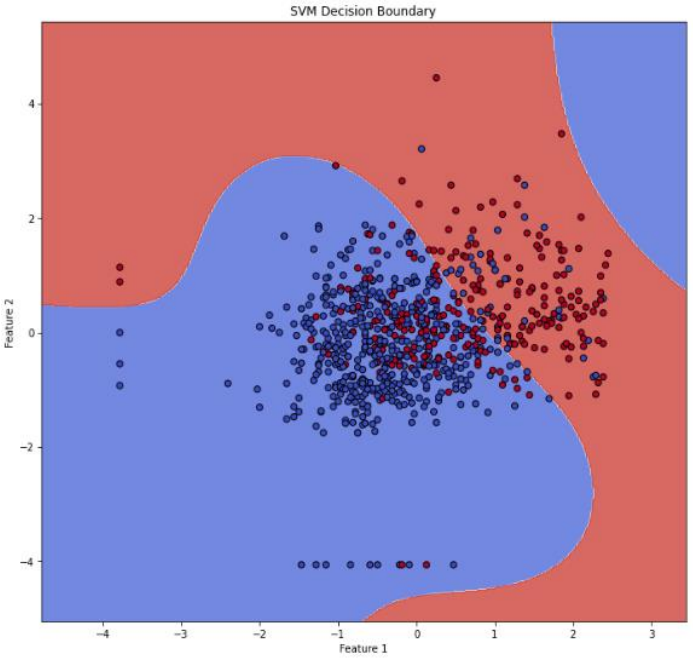
```
>>> print(grid2.best_params_)
{'C': 100000, 'gamma': 0.01, 'kernel': 'rbf'}
>>> # print how our model looks after hyper-parameter tuning
>>> best_model2 = grid2.best_estimator_
>>> print(best_model2)
SVC(C=100000, gamma=0.01)
>>> grid_predictions2 = grid2.predict(x2_test)
>>> # print classification report
>>> print(metrics.classification_report(y_train, grid2.best_estimator_.predict(x2_train),
...   target_names=["0", "1"]))
              precision    recall  f1-score   support

           0       0.77      0.91      0.84       375
           1       0.76      0.50      0.60       201

    accuracy                           0.77       576
   macro avg       0.77      0.71      0.72       576
weighted avg       0.77      0.77      0.76       576

>>> print(metrics.confusion_matrix(y_test, grid_predictions2))
[[116   9]
 [ 40  27]]
>>> print(metrics.classification_report(y_test, grid_predictions2,target_names=['0', '1']))
              precision    recall  f1-score   support

           0       0.74      0.93      0.83       125
           1       0.75      0.40      0.52        67

    accuracy                           0.74       192
   macro avg       0.75      0.67      0.67       192
weighted avg       0.75      0.74      0.72       192
```

Παρατηρούμε πως οι καλύτερες παράμετροι αναδείχθηκαν οι C = 100000, Sigma (Gamma) = 0.01 και Kernel = rbf .

Μπορούν στα διπλανά output να εντοπιστούν πολύ ισχυρά μέτρα σύγκρισης όσον αφορά την ακρίβεια το accuracy και το score.

```
# Define ranges for creating the meshgrid
x_min, x_max = x2[:, 0].min() - 1, x2[:, 0].max() + 1
y_min, y_max = x2[:, 1].min() - 1, x2[:, 1].max() + 1
h = 0.02  # Step size in the mesh

# Create a meshgrid
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Predict the class for each point in the meshgrid
Z = best_model2.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plotting the decision boundary and data points
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(x2[:, 0], x2[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolors='k')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('SVM Decision Boundary')
plt.show()
```

Για την οπτικοποίηση θα προχωρήσουμε αντίστοιχα σε δημιουργία Scatter Plot με εμφανή τα δύο χωρία ταξινόμησης.



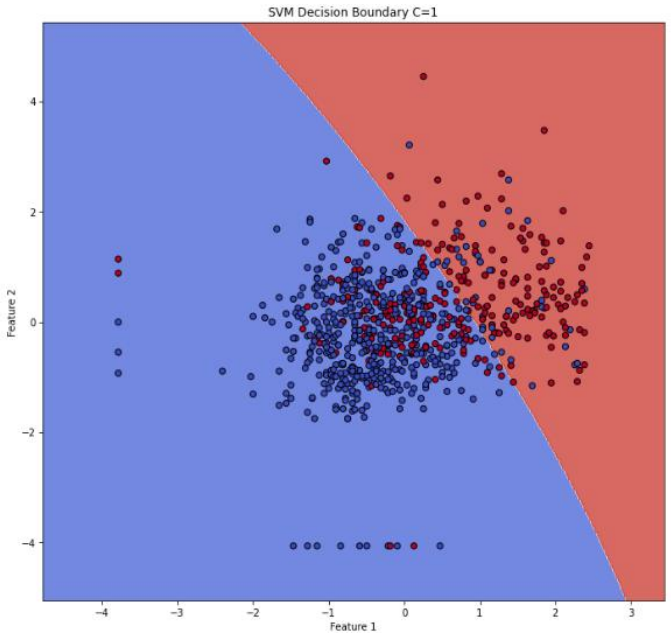
Είναι αρκετά ξεκάθαρος ο διαχωρισμός των χωρίων, χαρακτηριστικός rbf, όπου μετά από σχετική διερεύνηση είδαμε πως το sigma θα είχαμε και πολυπλοκότερο αποτέλεσμα.

Παρακάτω θα δούμε πως το κόστος θα επηρεάσει το διάγραμμα:

```
>>> #Plotting for C=1 and C=1000
>>>
>>> # Accessing the best parameters
>>> best_params = grid2.best_params_
>>>
>>> # Extracting individual best parameters
>>> best_gamma = best_params['gamma']
>>> best_kernel = best_params['kernel']
>>>
>>> #For C=1
>>>
>>> SVMclassifier1 = SVC(C=1, kernel=best_kernel, gamma= best_gamma)
>>> SVMclassifier1.fit(x2_train, y_train)
SVC(C=1, gamma=0.01)
```

Με τις ίδιες παραμέτρους κατασκευάζουμε εκ νέου SVM classifier με κόστος C = 1 .
Κατόπιν προχωρούμε σε διαμόρφωση του διαγράμματος ως εξής:

```
# Define ranges for creating the meshgrid
x_min, x_max = x2[:, 0].min() - 1, x2[:, 0].max() + 1
y_min, y_max = x2[:, 1].min() - 1, x2[:, 1].max() + 1
h = 0.02  # Step size in the mesh

# Create a meshgrid
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Predict the class for each point in the meshgrid
Z = SVMclassifier1.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plotting the decision boundary and data points
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(x2[:, 0], x2[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolors='k')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('SVM Decision Boundary C=1')
plt.show()
```



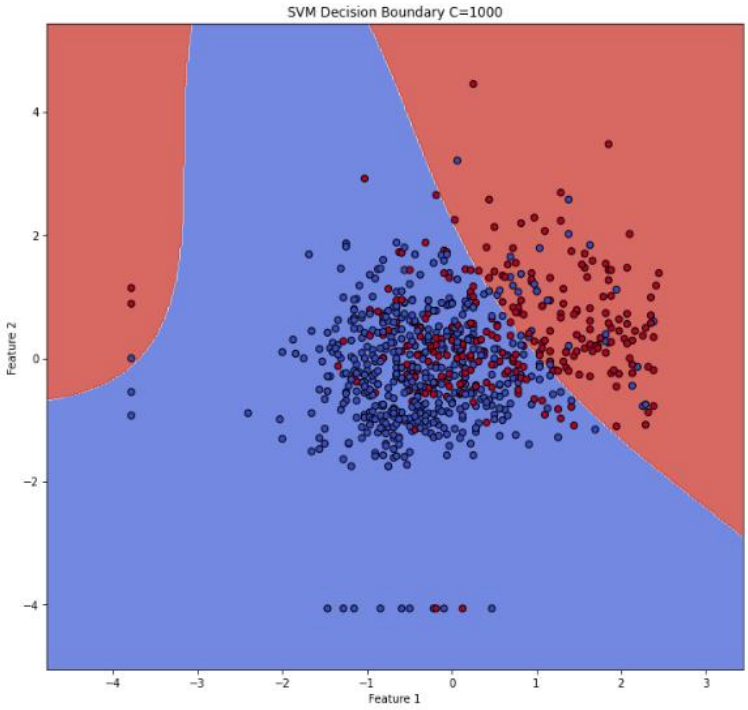
Αναμενόμενα το χαμηλότερο κόστος έχει δώσει αρκετά robust ταξινόμηση χωρίς επιρροές που θυμίζει αρκετά kernel = linear.

Για την τελική διερεύνηση βάση της υπόθεσης θα προχωρήσουμε στην εκ νέου μοντελοποίηση με τις ίδιες παραμέτρους χρησιμοποιώντας C = 1000 όπου θα περιμένουμε ένα σχήμα σε επίπεδο πολυπλοκότητας των χωρίων μεταξύ των δύο που προηγήθηκαν.

```
>>> #For C=1000
>>>
>>> SVMclassifier2 = SVC(C=1000, kernel=best_kernel, gamma= best_gamma)
>>> SVMclassifier2.fit(x2_train, y_train)
SVC(C=1000, gamma=0.01)
```

Κατασκευή του επιθυμητού μοντέλου και διαμόρφωση διαγράμματος ως άνωθεν για την πιστή σύγκριση:

```
# Define ranges for creating the meshgrid
x_min, x_max = x2[:, 0].min() - 1, x2[:, 0].max() + 1
y_min, y_max = x2[:, 1].min() - 1, x2[:, 1].max() + 1
h = 0.02  # Step size in the mesh

# Create a meshgrid
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Predict the class for each point in the meshgrid
Z = SVMclassifier2.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plotting the decision boundary and data points
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(x2[:, 0], x2[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolors='k')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('SVM Decision Boundary C=1000')
plt.show()
```



Όπου και βρισκόμαστε εντός λογικών ορίων σε ένα μεταίχμιο των προαναφερθέντων περιπτώσεων με ένα ξεκίνημα μίας πιο εξειδικευμένης διαμόρφωσης των χωρίων ταξινόμησης.

Παρουσιάστηκε για άλλη μία φορά η διαφορά στον χειρισμό των δύο συστημάτων (Στατιστικό Πακέτο R και Γλώσσα Προγραμματισμού Python) για προβλήματα ταξινόμησης με την χρήση των μεθόδων Logistic Regression και Support Vector Machines βάσει του προβλήματος της υπόθεσης που προαναφέραμε. Αναγνωρίσαμε αρκετές ομοιότητες τόσο στην συγκεκριμένη άσκηση όσο και στο σύνολο της εργασίας αναφορικά με τα διάφορα αποτελέσματα (πχ στο τελευταίο σκέλος βάση της διερεύνησης αναγνωρίσθηκε και στις δύο πλατφόρμες C = 100000) καθώς και στα μέτρα σύγκρισης, με εύλογες ελάχιστες αποκλίσεις μεταξύ των περιπτώσεων. Τα αποτελέσματα δεν έπαψαν να μας προσφέρουν από ικανοποιητικά έως και εξαιρετικά δεδομένα καθ' όλη τη διάρκεια της ανάλυσης. οι μέθοδοι κατέληγαν στην ίδια θεώρηση για σχετικά ασήμαντες μεταβλητές ως προς την προστιθέμενη αξία του εκάστοτε μοντέλου με κατά περιπτώσεις διαφορετικές τιμές παραγόντων όπου έφεραν και τις αυξομειώσεις στις στα κριτήρια επιλογής των μοντέλων. Τελικά παρατηρήσαμε πως τα διάφορα κόστη μπορούν να συμβάλλουν στην διαφοροποίηση των χωρίων ταξινόμησης των διαγραμμάτων όπως και από περεταίρω διερεύνηση μας έδειξαν και οι τιμές των Kernel και Sigma . Εν κατακλείδι, μπορέσαμε να αξιολογήσουμε και να αναλύσουμε τα ως άνω μοντέλα με έναν αρκετά user-friendly χειρισμό και ιδανικά αποτελέσματα και από τα δύο συστήματα διεξαγωγής των case study.