Homework III
Data Mining

113550802 - Elisabeth Le Leslé & 113550819 - Guillaume Drui

Question 1:

After removing all rows where any predictor (excluding the Outcome column) has a value of 0 or is missing (NA) and converting the target variable "Outcome" to a factor, we built the logistic regression model with "Outcome" as the dependant variable and all other variables as predictors.

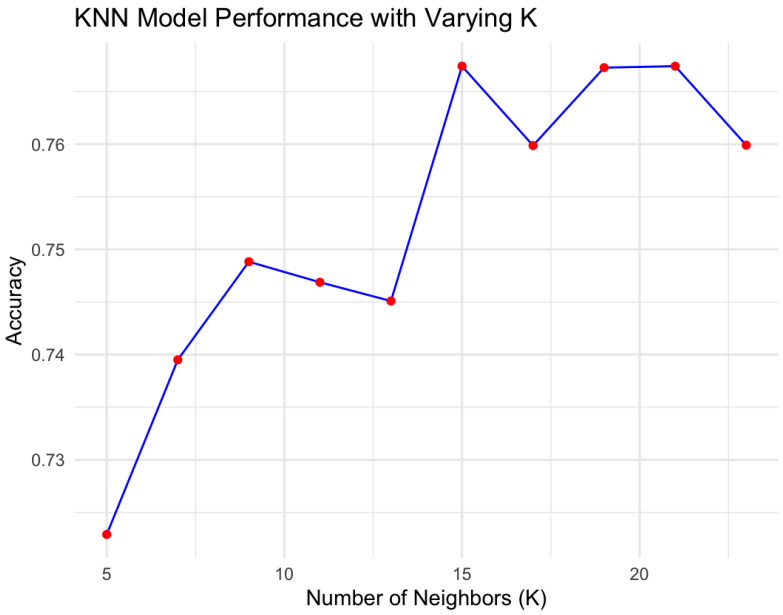
```
Call:
glm(formula = Outcome ~ ., family = "binomial", data = pima_clean)

Coefficients:
                           Estimate Std. Error z value Pr(>|z|)
(Intercept)              -1.083e+01  1.423e+00  -7.610 2.73e-14 ***
Pregnancies               7.364e-02  5.973e-02   1.233   0.2176
Glucose                   3.616e-02  6.249e-03   5.785 7.23e-09 ***
BloodPressure             5.993e-03  1.320e-02   0.454   0.6497
SkinThickness             1.110e-02  1.869e-02   0.594   0.5527
Insulin                   3.231e-05  1.445e-03   0.022   0.9822
BMI                       7.615e-02  3.174e-02   2.399   0.0164 *
DiabetesPedigreeFunction  1.097e+00  4.777e-01   2.297   0.0216 *
Age                       4.075e-02  1.919e-02   2.123   0.0337 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 426.34  on 335  degrees of freedom
Residual deviance: 288.92  on 327  degrees of freedom
AIC: 306.92

Number of Fisher Scoring iterations: 5
```

We then identify the following significant predictors with p-value < 0.05:
[1] "Glucose" "BMI" "DiabetesPedigreeFunction"
[4] "Age"

After creating a data subset depending only on the significant predictors, we conduct classification using KNN:

- We first start by defining Training (70%) and Test (30%) sets
- We train the KNN model 10-fold cross-validation to optimize the k value, which determines how many neighbors to consider.

- Predictions are made on the test dataset using the trained KNN model.



This plot shows that the optimal tradeoff between low k and best accuracy is k=15.

Next we train a Naive Bayes model using the training data.

Here are the performance metrics side to side for each model:

```
KNN Performance:
 Accuracy: 0.7652174
Precision: 0.7987421
Recall: 0.852349
F1 Score: 0.8246753
```

```
Naive Bayes Performance:
 Accuracy: 0.7478261
Precision: 0.7692308
Recall: 0.8724832
F1 Score: 0.8176101
```

Interpretation:

- **KNN** has a slightly higher accuracy and precision compared to Naive Bayes, indicating it is better at identifying true positives among its predictions.
- **Naive Bayes**, however, shows a better recall, meaning it is more effective at identifying actual positive cases.
- Overall, both models demonstrate strong predictive capabilities, with KNN being slightly preferred based on the provided metrics.

Question 2:

We first perform MLR and identify significant predictors ($p < 0.05$) for wine quality using lm(). We obtain the following results:

```
Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)          2.197e+01  2.119e+01   1.036   0.3002
fixed.acidity        2.499e-02  2.595e-02   0.963   0.3357
volatile.acidity    -1.084e+00  1.211e-01  -8.948  < 2e-16 ***
citric.acid         -1.826e-01  1.472e-01  -1.240   0.2150
residual.sugar       1.633e-02  1.500e-02   1.089   0.2765
chlorides           -1.874e+00  4.193e-01  -4.470 8.37e-06 ***
free.sulfur.dioxide  4.361e-03  2.171e-03   2.009   0.0447 *
total.sulfur.dioxide -3.265e-03  7.287e-04  -4.480 8.00e-06 ***
density             -1.788e+01  2.163e+01  -0.827   0.4086
pH                  -4.137e-01  1.916e-01  -2.159   0.0310 *
sulphates            9.163e-01  1.143e-01   8.014 2.13e-15 ***
alcohol              2.762e-01  2.648e-02  10.429  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.648 on 1587 degrees of freedom
Multiple R-squared:  0.3606,    Adjusted R-squared:  0.3561
F-statistic: 81.35 on 11 and 1587 DF,  p-value: < 2.2e-16
```

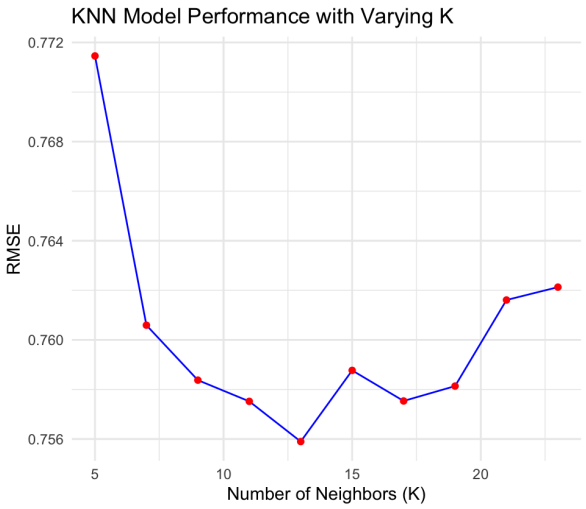
Therefore, the significant predictors with p<0.05 are:

```
> print(significant_predictors)
[1] "volatile.acidity"     "chlorides"            "free.sulfur.dioxide"
[4] "total.sulfur.dioxide" "pH"                   "sulphates"
[7] "alcohol"
```

We then divide data into training (70%) and testing (30%) sets to perform KNN and MARS regressions.

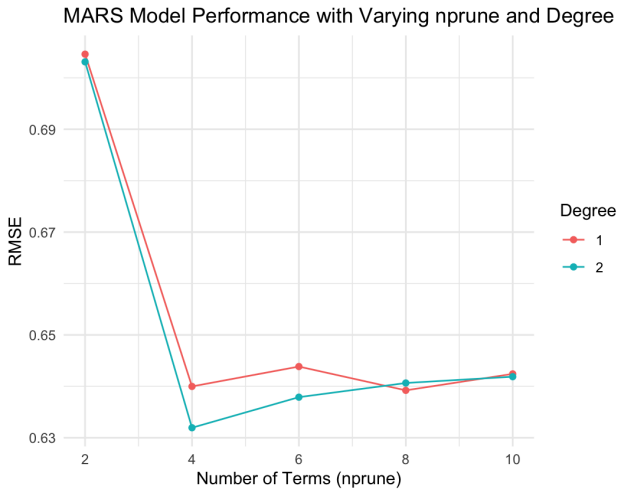
Our goal is to optimize k using cross-validation, then train and predict using KNN (taking only odd k).
The KNN regression yields the following results:

```
> print(knn_results)
    k      RMSE  Rsquared       MAE     RMSESD RsquaredSD      MAESD
1   5 0.7714564 0.1284316 0.5933985 0.03912623 0.04341915 0.02227233
2   7 0.7605921 0.1313780 0.5878115 0.04050656 0.03684590 0.02234439
3   9 0.7583751 0.1263650 0.5907267 0.03552254 0.04041692 0.01827652
4  11 0.7575191 0.1236522 0.5931093 0.03466400 0.03728218 0.01710792
5  13 0.7558972 0.1239439 0.5924063 0.03627344 0.03126960 0.01536684
6  15 0.7587684 0.1169628 0.5979759 0.03412757 0.03559373 0.01587886
7  17 0.7575351 0.1196857 0.5981839 0.03603702 0.04007343 0.01757967
8  19 0.7581348 0.1174285 0.6014430 0.03612046 0.03475617 0.01829191
9  21 0.7616088 0.1087576 0.6039892 0.03477266 0.02986489 0.01763116
10 23 0.7621283 0.1075579 0.6033682 0.03724854 0.03347720 0.01825706
```



From the graph, we can see that the RMSE is minimized for k=13, so we take it as our optimal value.
For the MARS regression, we aim to optimize degree and nprune using cross-validation. We obtain these results and plot:

```
> print(mars_results)
   degree nprune      RMSE  Rsquared       MAE     RMSESD RsquaredSD      MAESD
1       1      2 0.7046018 0.2383245 0.5552960 0.06723486 0.07794286 0.04531850
6       2      2 0.7031076 0.2412213 0.5542882 0.06553339 0.07286449 0.04373775
2       1      4 0.6399791 0.3685117 0.4995916 0.06318309 0.09991097 0.04981272
7       2      4 0.6319417 0.3840922 0.4906974 0.06816954 0.09825633 0.05518684
3       1      6 0.6438466 0.3602987 0.5036429 0.06032219 0.08955766 0.04725221
8       2      6 0.6378908 0.3721975 0.4991831 0.06666618 0.09715612 0.05341663
4       1      8 0.6392178 0.3694137 0.4991176 0.05989054 0.08766388 0.04742214
9       2      8 0.6406463 0.3663845 0.5012225 0.05809116 0.07933503 0.04890790
5       1     10 0.6424173 0.3640102 0.4983320 0.06523032 0.10127607 0.05033223
10      2     10 0.6418738 0.3652490 0.5013158 0.05876586 0.07998913 0.05014991
```



Our best values are thus nprune=7 and degree=2.

To evaluate our model performances, we then calculate RMSE, MAE, and MAPE for both KNN and MARS models.

```
KNN Performance:
 RMSE: 0.7522285
MAE: 0.5907821
MAPE: 0.1074631
```

```
MARS Performance:
 RMSE: 0.6436932
MAE: 0.5031413
MAPE: 0.09214728
```

Interpretation:

- **RMSE:** MARS has a lower RMSE compared to KNN, indicating it's generally making smaller prediction errors on the test set.
- **MAE:** MARS's lower MAE shows it has better average prediction accuracy than KNN, producing smaller deviations from true values.
- **MAPE:** MARS outperforms KNN with a lower MAPE, indicating that, on average, MARS's predictions deviate by a smaller percentage from the actual quality scores.

Therefore MARS is the better regression model in this case.

Question 3:

After removing the "job", "day", "month" columns and converting the target variable "y" to a factor, we fit the Logistic Regressor model using glm where y is the dependent variable, and all other variables are independent predictors. The family is specified as binomial for binary outcomes.

```
Call:
glm(formula = y ~ ., family = binomial(link = "logit"), data = bank_data)

Coefficients:
                     Estimate Std. Error z value Pr(>|z|)
(Intercept)        -2.907e+00  4.897e-01  -5.937 2.91e-09 ***
age                 9.891e-03  5.852e-03   1.690  0.09100 .
maritalmarried     -4.528e-01  1.673e-01  -2.707  0.00679 **
maritalsingle      -1.242e-01  1.941e-01  -0.640  0.52227
educationsecondary  1.441e-01  1.806e-01   0.798  0.42485
educationtertiary   4.469e-01  1.881e-01   2.376  0.01752 *
educationunknown   -2.046e-01  3.260e-01  -0.628  0.53022
defaultyes          3.971e-01  4.269e-01   0.930  0.35230
balance             5.942e-06  1.777e-05   0.334  0.73808
housingyes         -5.497e-01  1.207e-01  -4.553 5.28e-06 ***
loanyes            -7.847e-01  1.915e-01  -4.096 4.20e-05 ***
contacttelephone   -4.542e-03  2.185e-01  -0.021  0.98342
contactunknown     -1.120e+00  1.747e-01  -6.408 1.47e-10 ***
duration            3.989e-03  1.930e-04  20.670  < 2e-16 ***
campaign           -7.704e-02  2.687e-02  -2.868  0.00414 **
pdays              -2.795e-04  9.243e-04  -0.302  0.76235
previous           -1.788e-03  3.759e-02  -0.048  0.96206
poutcomeother       4.671e-01  2.585e-01   1.807  0.07073 .
poutcomesuccess     2.373e+00  2.600e-01   9.126  < 2e-16 ***
poutcomeunknown    -3.211e-01  3.006e-01  -1.068  0.28541
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 3231.0  on 4520  degrees of freedom
Residual deviance: 2302.6  on 4501  degrees of freedom
AIC: 2342.6

Number of Fisher Scoring iterations: 6
```

We then extract the names of predictors that are statistically significant (p-value < 0.05) and exclude the intercept.

```
> print(significant_predictors)
[1] "maritalmarried"    "educationtertiary" "housingyes"        "loanyes"
[5] "contactunknown"    "duration"          "campaign"          "poutcomesuccess"
```

After defining our get_mode() function to calculate the value that appears most frequently, we create a new sample where numeric features are replaced with their median and categorical features with their computed mode. We obtain the probability of the new sample using predict()

```
> cat("Probability for y=yes:", probability, "\n")
Probability for y=yes: 0.04253354
> cat("Probability for y=no:", 1 - probability, "\n")
Probability for y=no: 0.9574665
```

Next, we fit a Naïve Bayes model using all predictors.
We then fit it using partial predictors (in our case we kept only the significant ones).

```
Naïve Bayes with All Predictors:
 Overall Accuracy: 0.8652953
 Accuracy for y-yes: 0.91675
 Accuracy for y-no: 0.4702495
```

```
Naïve Bayes with Partial Predictors:
 Overall Accuracy: 0.8967043
 Accuracy for y-yes: 0.959
 Accuracy for y-no: 0.4184261
```

Interpretation:
Although NB with All predictors has a lower overall and y-yes accuracy, its accuracy for y-no is better than NB with partial predictors only.
However, the specificity for y-no is still low in both cases.
The results indicate a potential issue with class imbalance, where the model is very good at identifying positive outcomes but struggles with negative outcomes. This could be due to the nature of the dataset, where the positive class is more prevalent or better represented by the significant predictors.
However, we notice this:

```
> table(bank_data$y)

  no  yes
4000  521
```

The "no" class is far more frequent than the "yes" class, with a roughly 88.5% to 11.5% ratio.
This imbalance means that a model should achieve high accuracy simply by predicting the majority class ("no") more often, even if it doesn't effectively learn the underlying patterns for the minority class ("yes").
However, we notice a High Sensitivity for y-yes from the previous NB results. Indeed:

- Both models show high accuracy for predicting "yes" (95.9% with partial predictors and 91.68% with all predictors).
- This high sensitivity suggests the models are effective in capturing the minority class when it occurs.

This is also a case of Low Specificity for y-no:

- The models perform poorly for the "no" class, with relatively low specificity (41.84% with partial predictors and 47.02% with all predictors).
- This low specificity means the models incorrectly classify many instances of "no" as "yes", which could be a consequence of attempting to balance predictions for both classes.

We conclude that, due to the **Bias Toward Majority Class**, NB manages to capture patterns for the minority class ("yes"), but this comes at the cost of incorrectly predicting "no" as "yes."

To balance the results out, we can Oversample the Minority Class using SMOTE for example, or use a different algorithm than NB.

Question 4:

We start by fitting a Multiple Linear Regression (MLR) model using all available features in the training dataset. To identify key predictors, we examine the model's summary statistics and select those variables with p-values less than 0.05. This threshold indicates statistically significant predictors, suggesting these variables have a meaningful impact on predicting container values. We exclude the intercept from this list of significant variables

```
[1] "GCTI"                  "HSI"                   "CCFI"                  "Retail.sales.index_EU"
[5] "Retail.sales.index_US" "Container_SeasonIndex"
```

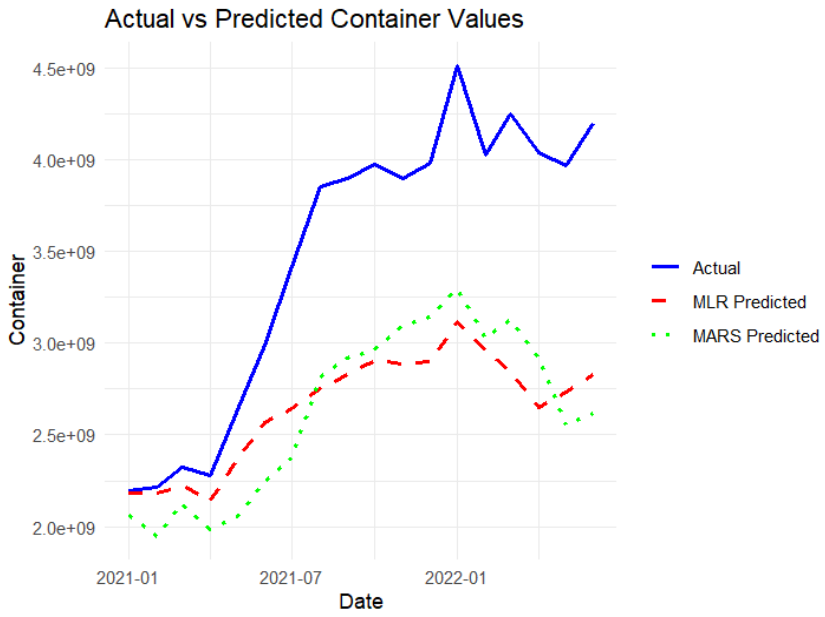
For the MARS model, we fit the model using the training data and then employ the evimp function to evaluate variable importance. The evimp output reveals the variables that contribute to the model's predictive power, allowing us to directly identify the significant features used by MARS.

```
[1] "GCTI"                  "CCFI"                  "Year"                  "Container_SeasonIndex"
[5] "IPI_TW"                "Date"
```

We then take the union of those variables to refit both models. We then evaluate our models using RMSE, MAE and MAPE and create the following performance matrix:

	Method	RMSE	MAE	MAPE
1	MLR	970333850	829833143	0.2132755
2	MARS	945283290	854481478	0.2306992

Below is the plotted prediction for the years 2021-2023



Question 5:

After splitting the data into training and testing, we create a Naïve Bayes and a Logit model.

```
# Build Naïve Bayes model
nb_model <- naiveBayes(survival ~ class + gender + age + fare, data = train_data)

# Build Logistic Regression model
logit_model <- glm(survival ~ class + gender + age + fare, data = train_data, family=binomial(link="logit"))
```

We then use our models on the three different scenarios, here is how the first scenario was implemented, the other scenarios are implemented analogously:

```
# Scenario 1: Impact of Gender (Male vs Female), given Age = 30, Class = 2, Fare = 20
scenario1 <- data.frame(
  class = factor(2, levels = levels(train_data$class)),
  age = 30,
  fare = 20,
  gender = factor(c("male", "female"), levels = levels(train_data$gender))
)

# Naïve Bayes predictions for Scenario 1
nb_pred1 <- predict(nb_model, scenario1, type = "raw")$posterior[, "survived"]

# Logistic Regression predictions for Scenario 1
logit_pred1 <- predict(logit_model, scenario1, type = "response")

# Display predictions for Scenario 1
scenario1_results <- data.frame(
  Gender = scenario1$gender,
  Naive_Bayes = nb_pred1,
  Logistic_Regression = logit_pred1
)
print("Scenario 1: Impact of Gender (Male vs Female), given Age = 30, Class = 2, Fare = 20")
print(scenario1_results)
```

Below are the results for the different scenarios:

Scenario 1:

	Gender	Naive_Bayes	Logistic_Regression
1	male	0.9688062	0.2237134
2	female	0.9961025	0.6978828

Here we see that being a female on the titanic gives a higher survival chance than being a male.
Scenario 2:

	Age	Naive_Bayes	Logistic_Regression
1	1	0.9284357	0.7667714
2	25	0.9880095	0.5898067

Here we see that that being a young child from first class gives a high survival rate compared to being a 25 year old male from first class.

Scenario 3:

	Class	Naive_Bayes	Logistic_Regression
1	1	0.9985274	0.9201626
2	2	0.9959633	0.7324089
3	3	0.9923147	0.5283373

Here we see that the higher the class, the better the better the chances of survival for females.

If we had to output if a certain person will survive or not using those two method, they would agree most of the time, i.e. logistic regression and Naive Bayes would both output 1 or 0. However there is a big difference in the probability. This might be due to the fact that Niave Bayes assumes that the features are independent even though they are not.

Question 6:

After cleaning and splitting the data on “Handysize” which we are trying to predict. We apply Ridge and Lasso and ElasticNet

```
ridge_model <- glmnet(x, y,family= "gaussian", alpha=0, lambda=1)
ridge_beta <- as.vector(ridge_model$beta)
names(ridge_beta) <- predictor_names
```

```
lasso_model <- glmnet(x, y,family="gaussian", alpha=1, lambda=1)
lasso_beta <- as.vector(lasso_model$beta)
names(lasso_beta) <- colnames(x)
```

```
elastic_model <- glmnet(x, y,family="gaussian", alpha=0.5, lambda=1)
elastic_beta <- as.vector(elastic_model$beta)
names(elastic_beta) <- colnames(x)
```

We run the predictions and save their MAPE,RMSE and MAE into a performance matrix

```
<- sqrt(mean((y - pred_elastic)^2))  # RMSE
<- mean(abs(y - pred_elastic))       # MAE
<- mean(abs((y - pred_elastic) / y)) * 100  # MAPE in percentage
```

Here is the resulting performance table:

```
             Ridge       Lasso   ElasticNet
RMSE 3.966920e+06 3.966920e+06 3.966920e+06
MAE  3.083434e+06 3.083434e+06 3.083434e+06
MAPE 6.153065e+00 6.153064e+00 6.153064e+00
```

To identify the most important predictors, we sort the obtained beta absolute coefficient values in decreasing order and choose the top 25%. This gives us the following most important predictors:
For Ridge:

```
[1] "PPI_CN"              "PPI_US"              "Seasonindex"         "Iron.Ore"
[5] "IMF_base.metal.index"
```

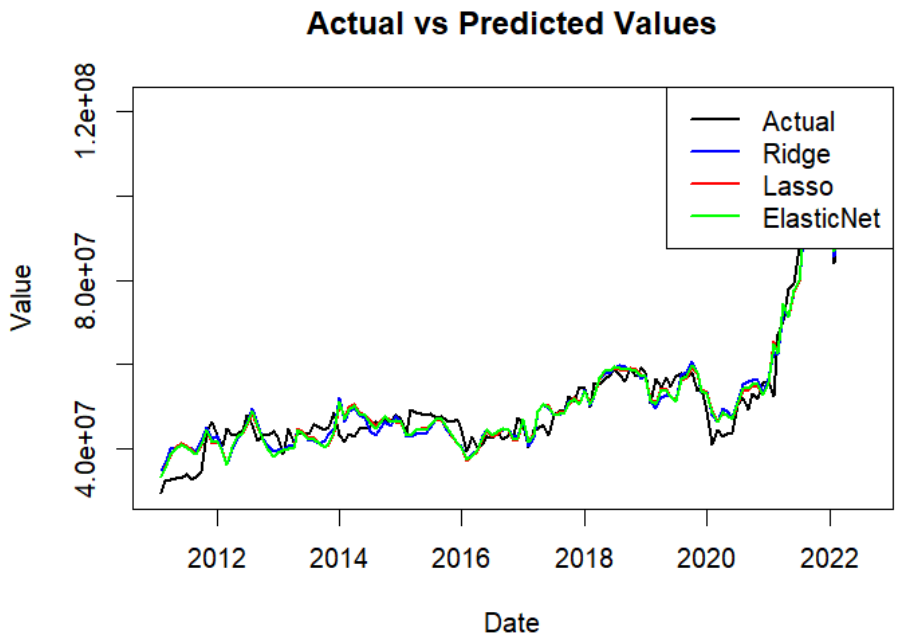
For Lasso:

```
[1] "PPI_CN"              "PPI_US"              "Seasonindex"         "Iron.Ore"
[5] "IMF_base.metal.index"
```

For ElasticNet:

```
[1] "PPI_CN"              "PPI_US"              "Seasonindex"         "Iron.Ore"
[5] "IMF_base.metal.index"
```

These results support the idea that Ridge, Lasso, and ElasticNet can find and agree on the same most important features.
Finally plotting our predictions we get:



As expected we can see that ElasticNet behaves more smoothly and is between the Ridge and Lasso predictions on the graph