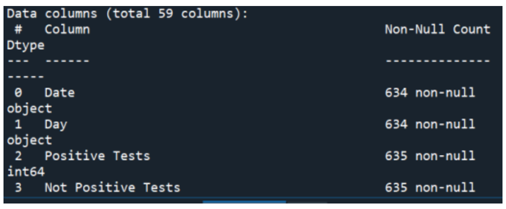
ML Regression Assignment – Scientific Report

Machine Learning – Dr. Chen Hajaj, Faculty of Engineering, IEM
By: Nadav Erez

In this assignment, we were given the task of predicting the number of positive and non-positive tests in a certain city.

First of all as part of my data exploration, I would like to get a basic idea about how the data looks like. After looking at the data, I observed:



```
Data columns (total 59 columns):
 #   Column                                          Non-Null Count
Dtype
---  ------                                          --------------
-----
 0   Date                                            634 non-null
object
 1   Day                                             634 non-null
object
 2   Positive Tests                                  635 non-null
int64
 3   Not Positive Tests                              635 non-null
```

Two total null values. I proceed with dropping these rows.

As for the features: I choose columns F-W out of the dataset.

Target variables: Two response variables – Positive & Non-positive tests. I execute the whole code for both.

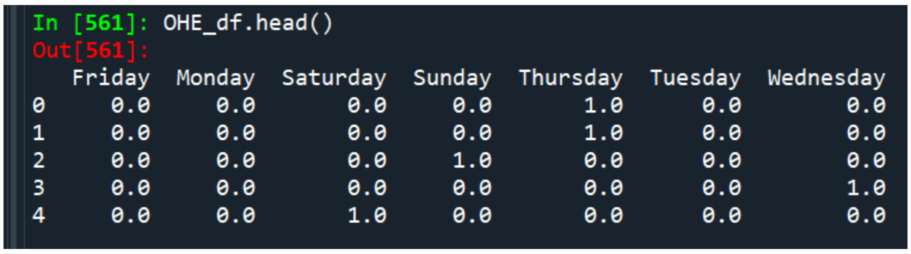
For convenience, I will call the results for the first target variable (Positive tests) "POS", and the second target variable will be "NEG".

In the code, variables associated with the Positive tests contain 'pos' and variables associated with the Non-positive tests contain 'neg' in them (For example: X_pos_train / X_ neg _train, y_pred_pos / y_pred_test, regressor_pos / regressor_neg). Everything is basically written twice for both target variables.

1. Pre-Processing

As part of pre-processing the data for the model, the following methods are executed:

- **One Hot Encoder (OHE)** – In an effort of making the models more accurate, I added binary categorical features based on what day it is. This is done by applying OHE to the non-ordinal categorical feature 'Day'.



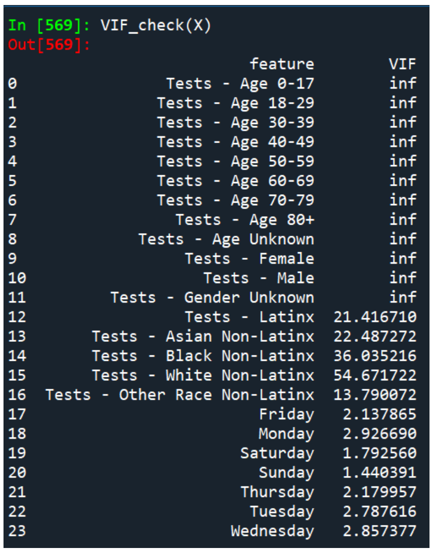
```
In [561]: OHE_df.head()
Out[561]:
   Friday  Monday  Saturday  Sunday  Thursday  Tuesday  Wednesday
0     0.0     0.0       0.0     0.0       1.0      0.0        0.0
1     0.0     0.0       0.0     0.0       1.0      0.0        0.0
2     0.0     0.0       0.0     1.0       0.0      0.0        0.0
3     0.0     0.0       0.0     0.0       0.0      0.0        1.0
4     0.0     0.0       1.0     0.0       0.0      0.0        0.0
```

- **Variance Inflation Factor (VIF)** – for getting an idea on the extent of multicollinearity in the features.
- **Correlation** – for checking correlation between features.
- **Feature Selection: Backward Elimination** – dropping features that can be "explained" by other features – this should lower the multicollinearity.

VIF shows that there is high multicollinearity in the features, and the percentage of correlated features in the data is 54%.

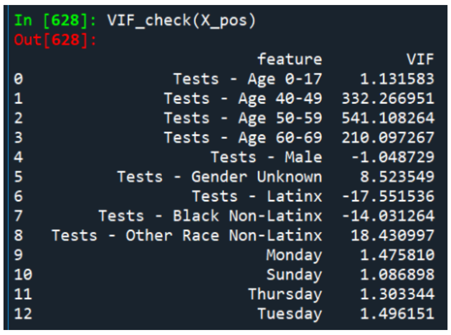
By using Backward Elimination, we lower the VIFs and the correlation (the data has extremely high multicollinearity, I would say this is problematic for the Linear modeling, but I understand that instructions were given to ignore this. I tried to deal with this anyway, at least lower it a bit).

VIF and correlation percentage before and after Backward Elimination:



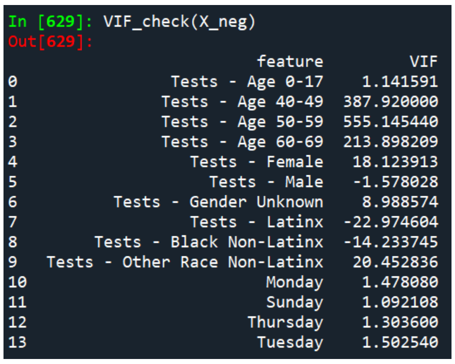
```
In [569]: VIF_check(X)
Out[569]:
                         feature        VIF
0                Tests - Age 0-17       inf
1               Tests - Age 18-29       inf
2               Tests - Age 30-39       inf
3               Tests - Age 40-49       inf
4               Tests - Age 50-59       inf
5               Tests - Age 60-69       inf
6               Tests - Age 70-79       inf
7                 Tests - Age 80+       inf
8             Tests - Age Unknown       inf
9                  Tests - Female       inf
10                   Tests - Male       inf
11         Tests - Gender Unknown       inf
12                 Tests - Latinx  21.416710
13      Tests - Asian Non-Latinx  22.487272
14      Tests - Black Non-Latinx  36.035216
15      Tests - White Non-Latinx  54.671722
16 Tests - Other Race Non-Latinx  13.790072
17                        Friday   2.137865
18                        Monday   2.926690
19                      Saturday   1.792560
20                        Sunday   1.440391
21                      Thursday   2.179957
22                       Tuesday   2.787616
23                     Wednesday   2.857377
```

Corr. % after B.E: **54%**



```
In [628]: VIF_check(X_pos)
Out[628]:
                         feature         VIF
0               Tests - Age 0-17    1.131583
1              Tests - Age 40-49  332.266951
2              Tests - Age 50-59  541.108264
3              Tests - Age 60-69  210.097267
4                   Tests - Male   -1.048729
5         Tests - Gender Unknown    8.523549
6                 Tests - Latinx  -17.551536
7       Tests - Black Non-Latinx  -14.031264
8  Tests - Other Race Non-Latinx   18.430997
9                         Monday    1.475810
10                        Sunday    1.086898
11                      Thursday    1.303344
12                       Tuesday    1.496151
```

POS: Corr. % after B.E: **46%**



```
In [629]: VIF_check(X_neg)
Out[629]:
                         feature         VIF
0               Tests - Age 0-17    1.141591
1              Tests - Age 40-49  387.920000
2              Tests - Age 50-59  555.145440
3              Tests - Age 60-69  213.898209
4                 Tests - Female   18.123913
5                   Tests - Male   -1.578028
6         Tests - Gender Unknown    8.988574
7                 Tests - Latinx  -22.974604
8       Tests - Black Non-Latinx  -14.233745
9  Tests - Other Race Non-Latinx   20.452836
10                        Monday    1.478080
11                        Sunday    1.092108
12                      Thursday    1.303600
13                       Tuesday    1.502540
```

NEG: Corr. % after B.E: **50%**

Before B.E, most of the features have > 5 VIF (INF!). We can see that the binary weekdays features have OK VIF, which indicates that it was a smart move to use OHE.

Backward Elimination dropped 11 features from the data when pre-processing for POS, and 10 features when pre-processing for NEG (Tests – Female feature was not dropped, the rest are the same).

2. Train-test split and Scaling

After these steps I go on to split the data into train and test sets, 80% to the training set and 20% to the test set.

I apply **Scaling** to the train and test features, with the scaling centering the data around zero mean:

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

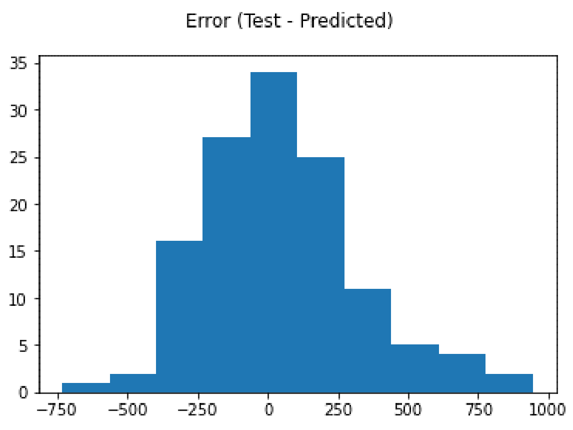
3. Models

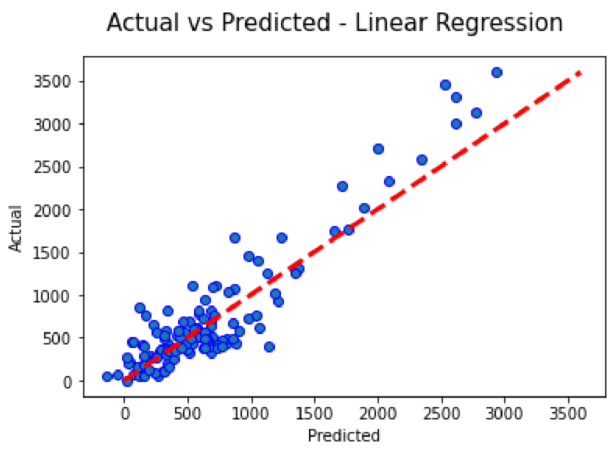
Linear Regression – Least Squares

A Linear model was fitted to the training set, predicting the test set. For evaluation, I use **10-fold Cross Validation** and select the mean score out of the scores as my metric. Furthermore, MSE, RMSE, R squared are calculated for evaluation.

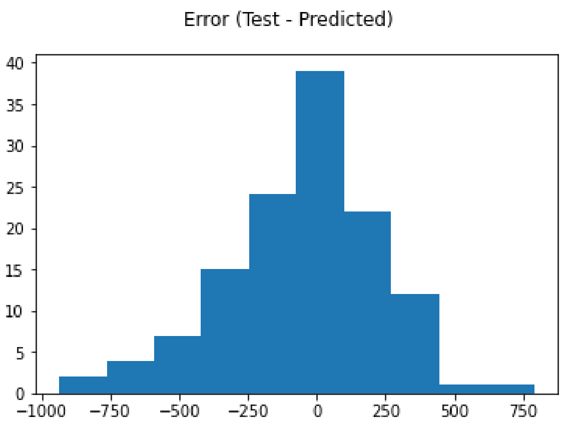
R squared: 0.84, which means that 16% of the variance cannot be explained by the model.

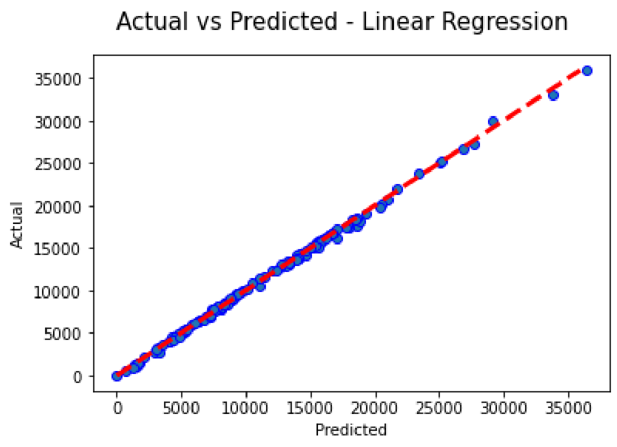
Positive tests (POS):





Non-Positive tests (NEG):



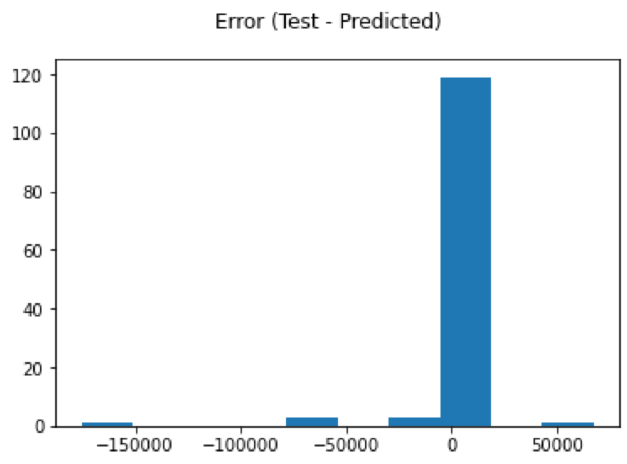


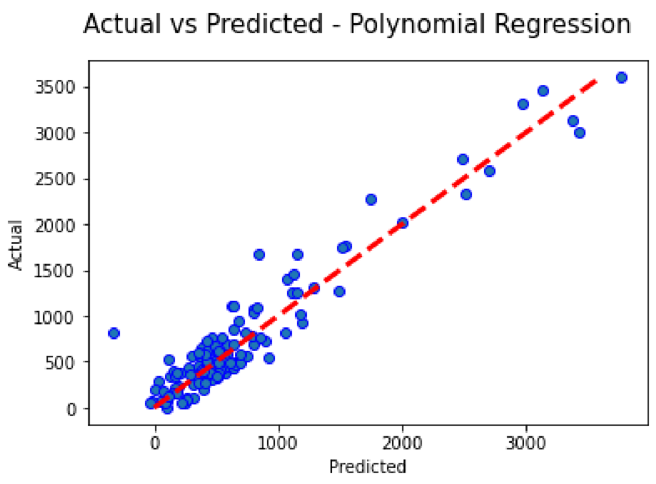
Polynomial Regression

First, X_train and X_test are transformed to Polynomial Regression. For tuning the parameters (not defined hyper parameters in Pol. Reg.), I run the model in iterations by increasing degree. In each iteration, 10-fold CV is executed on the train set and returns the highest score. I chose the best degree based on the model with the best mean CV score. Best degree being 1 or 2 for different executions of the code.

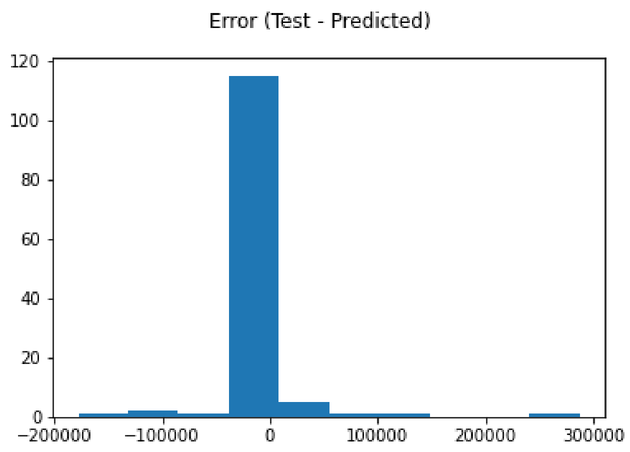
Side note: I didn't include more iterations for higher degrees basically because of run time, and it was clear that above X^2 the model performs worse.

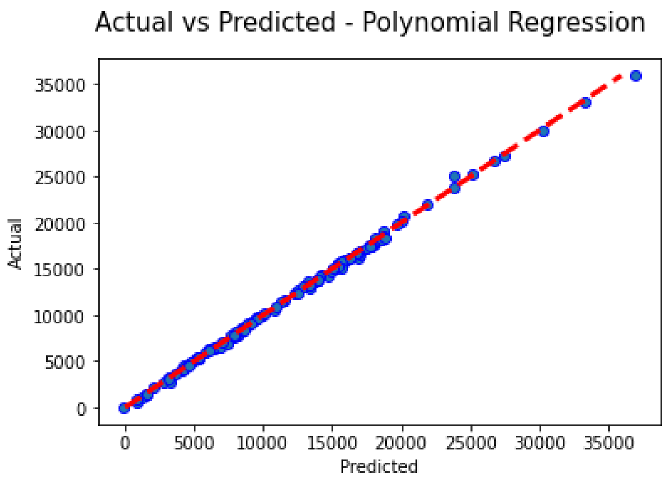
Positive tests (POS):





Negative tests (NEG):





From the visualization, this model appears to perform better than the OLS model. RMSE is lower as well.

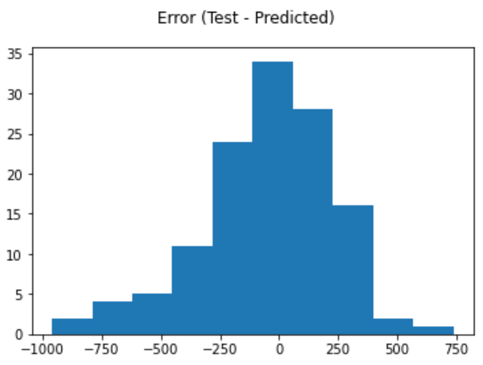
Ridge & Lasso Regression

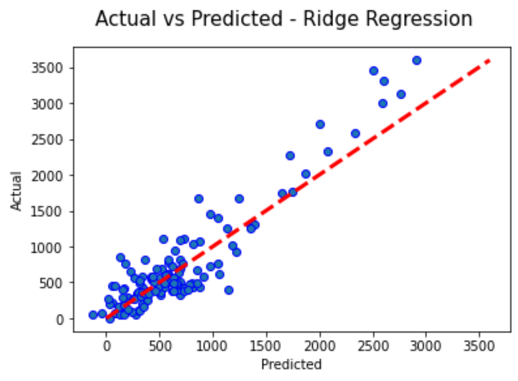
For the penalty parameter hyper tuning (alpha/lambda), I use Repeated K-fold CV (10 splits 3 times). The GridSearch function uses the CV for finding the optimal alpha. The hyper tuning is executed in two iterations, one for each model. Each iterations returns the optimal parameter.

After finding the best parameter for each model, a model is fit with the parameter and 10-fold CV is executed to find the best score.

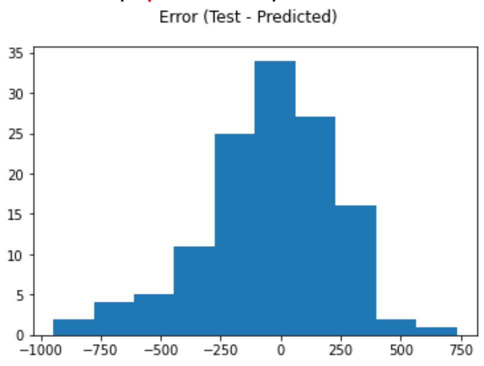
Positive tests (POS):

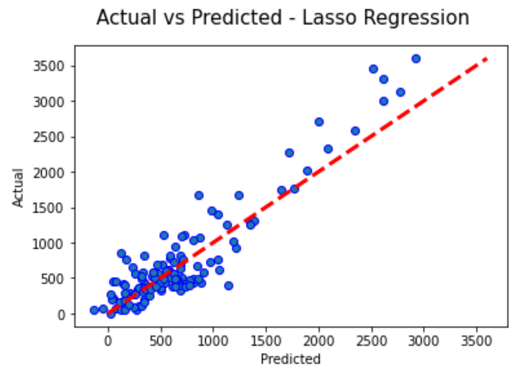
Ridge (alpha = 0.24):





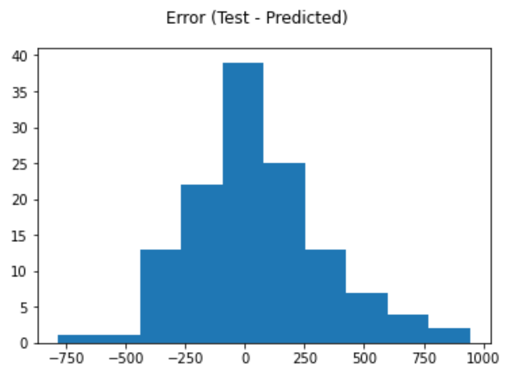
Lasso (alpha = 0.06):

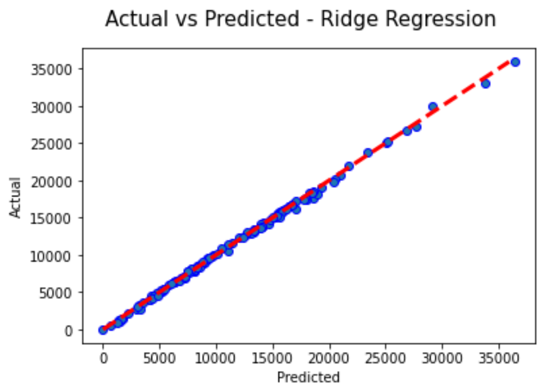




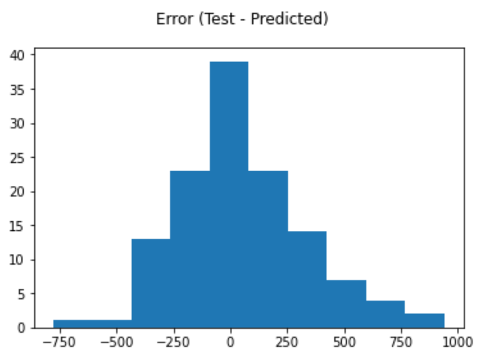
Negative tests (NEG):

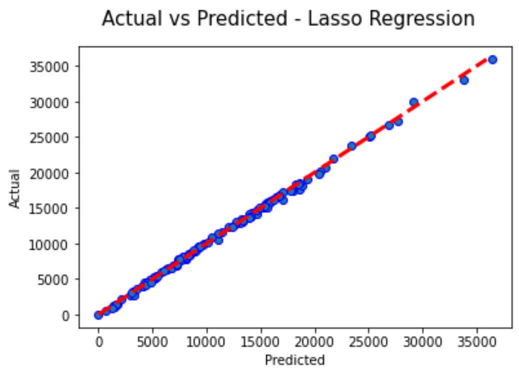
Ridge (alpha = 0.03):





Lasso (alpha = 0.02):





Both models put little weight on the penalty error, with Lasso being almost the same as the OLS model error-wise.

Generally, these model's predictions are almost identical to the OLS model.

Random Forest

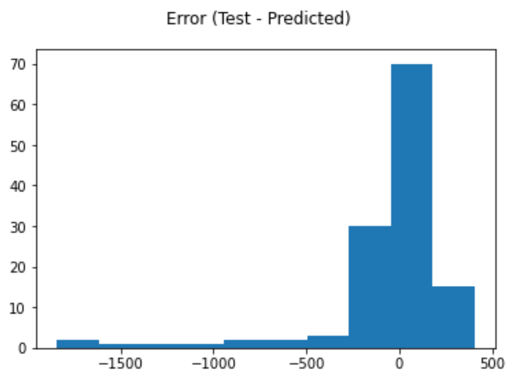
For this model, hyper tuning is executed with GridSearch and Repeated 10-fold CV, just like Ridge and Lasso. Parameters tuned are the number of trees in the forest and the depth. After running the hyper tuning at first, there was a very long run time when given n_estimators range of 50,100,500,1000 and max_depth range of 1-10.

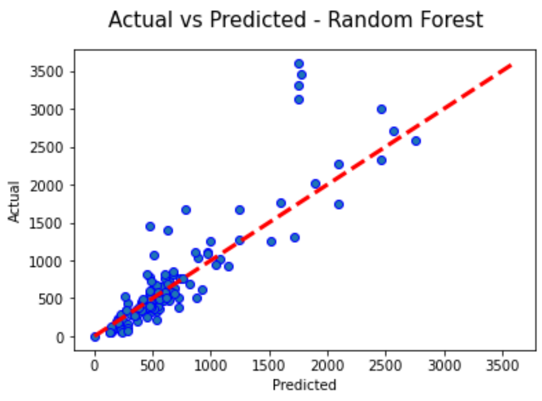
After running experiments, the grid search is given a pre-determined number of trees range (30-70 for POS and 70-120 for NEG) and max depth (ranging from 8 to 11 for POS and 8-10 for NEG), resulting in:

Positive tests RF: 70 trees and max depth of 10.
Negative tests RF: 100 trees and max depth of 9.

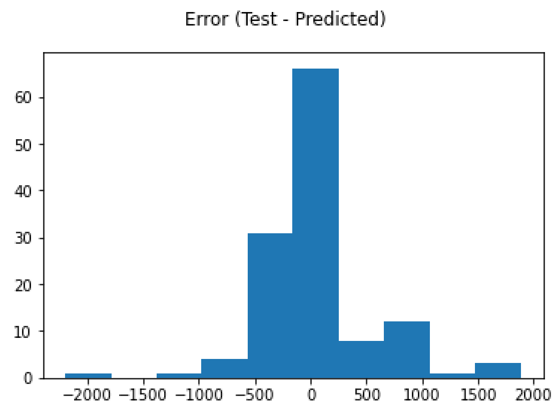
After finding the best parameters, a model is fit to the training data, and 10-fold CV is executed to find the best score for the Random Forest.

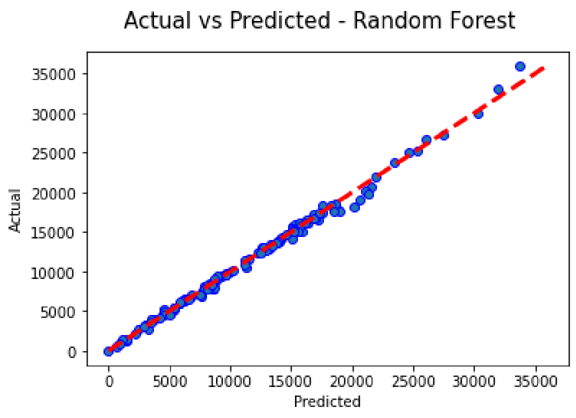
Positive tests (POS):





Negative tests (NEG):

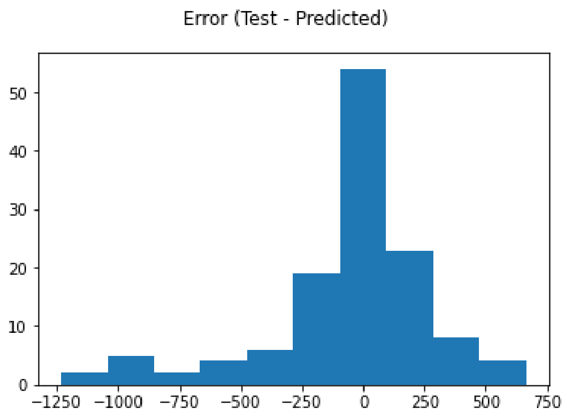


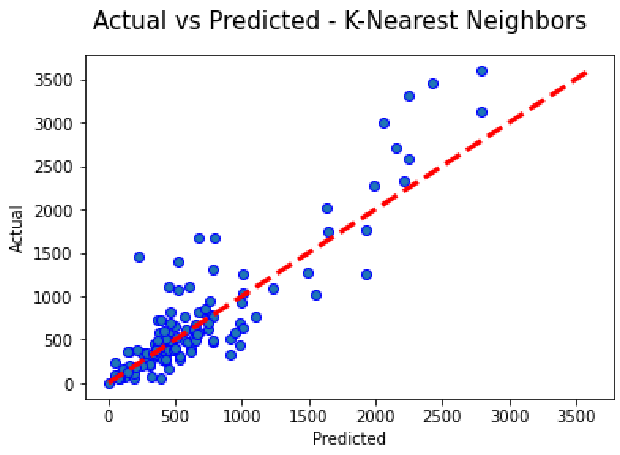


K-Nearest Neighbors

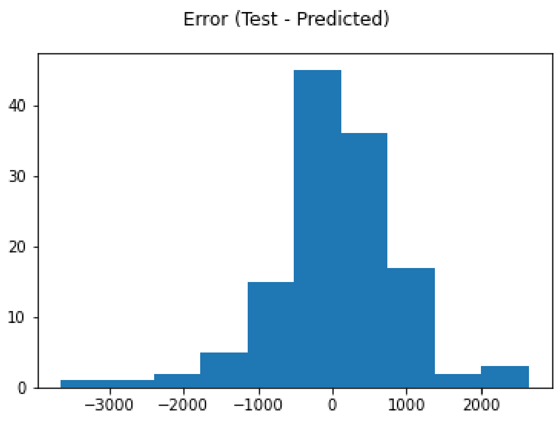
K-NN also uses Repeated 10-fold CV in a GridSearch for finding the best model parameter (K: how many neighbors will the model check). The model is fitted to the training set afterwards with the K that was found (K=1 for both models).

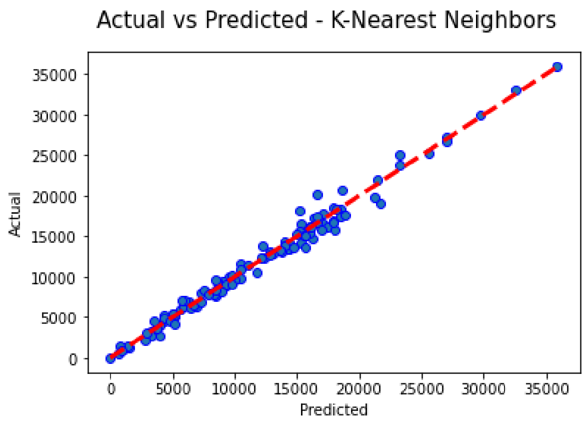
Positive tests (POS):





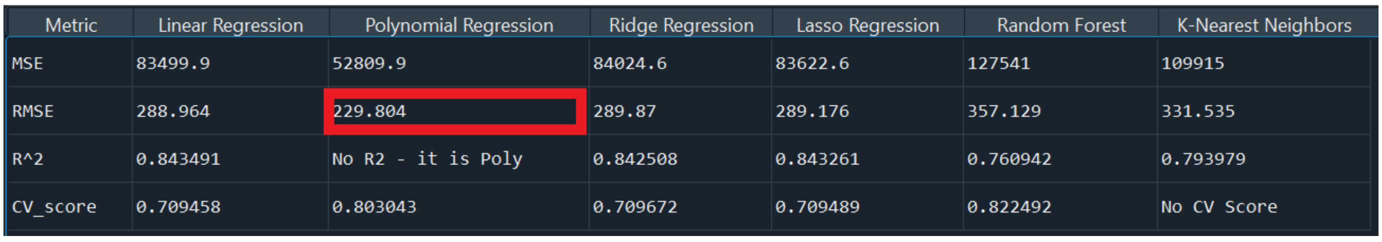
Negative tests (NEG):





Final results

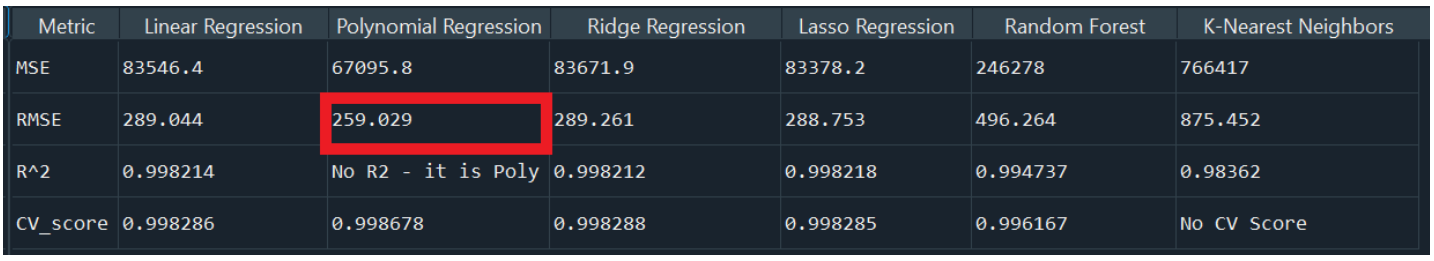
Positive tests:



Metric	Linear Regression	Polynomial Regression	Ridge Regression	Lasso Regression	Random Forest	K-Nearest Neighbors
MSE	83499.9	52809.9	84024.6	83622.6	127541	109915
RMSE	288.964	229.804	289.87	289.176	357.129	331.535
R^2	0.843491	No R2 - it is Poly	0.842508	0.843261	0.760942	0.793979
CV_score	0.709458	0.803043	0.709672	0.709489	0.822492	No CV Score

There were not very high differences between the models. However, it's clear that Polynomial Regression has the lowest RMSE.

Non-Positive tests:



Metric	Linear Regression	Polynomial Regression	Ridge Regression	Lasso Regression	Random Forest	K-Nearest Neighbors
MSE	83546.4	67095.8	83671.9	83378.2	246278	766417
RMSE	289.044	259.029	289.261	288.753	496.264	875.452
R^2	0.998214	No R2 - it is Poly	0.998212	0.998218	0.994737	0.98362
CV_score	0.998286	0.998678	0.998288	0.998285	0.996167	No CV Score

The models perform way better in predicting Non-positive tests. Here Polynomial Regression wins as well, and we can also see that Random Forest and K-NN's RMSE is higher than the rest, which indicates that the other models are better.