Flight Delay Forecasting

in this report I will show the steps of my try to solve the Machine Learning problem of Flight Delay Forecasting to estimate the flight delays, and discuss the used Machine Learning algorithms with the following main steps:

- Preprocess, visualize and splitting the data
- Applying Machine Learning Methods ( )
- Comparing the selected machine learning models performance
- Outlier detection and removal

The task is to predict the Delay time of travel using the data set which has the following Data:

Name of the airport where the flight departed. The name is given as airport international code	**<u>Departure Airport</u>**
Time scheduled for the flight take-off from origin airport	**<u>Scheduled departure time</u>**
Flight destination airport. The name is given as airport international code	**<u>Destination Airport</u>**
Time scheduled for the flight touch-down at the destination airport	**<u>Scheduled arrival time</u>**
Flight delay in minutes	**<u>Delay (in minutes)</u>**

- **Preprocessing:**

_ New Meaningful arguments:
Now from the above variables I was able to deduce the following variables which will possibly help to predict more accurately:

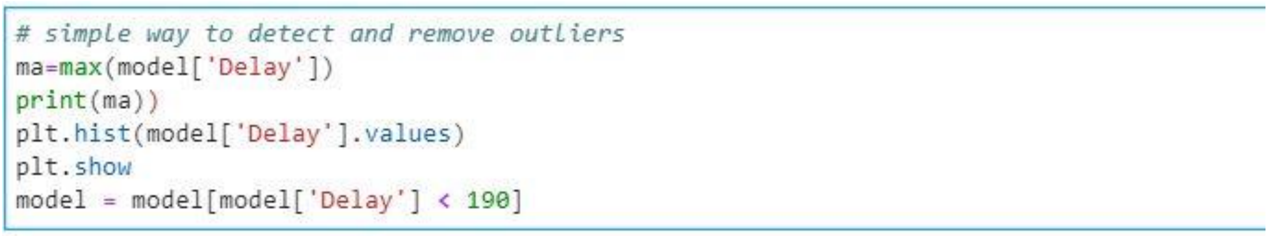
Duration of flight: the time difference between departure and arrival. (in minutes)
Day of the week: using pandas library from 0 (Monday) to 6 (Saturday)
Week of the year, month of the year ; what time of the year we are in might affect on traffic and delay (holidays for example)
Departure Hour of the flight: also the time of takeoff might play a part in delay

_ Label encoding:

Using Label encoder LabelEncoder from sklearn we can convert the string representation into the machine-readable format and this is applied for Destination Airport and Departure Airport; it gives every airport a unique index to be able to use it in detecting

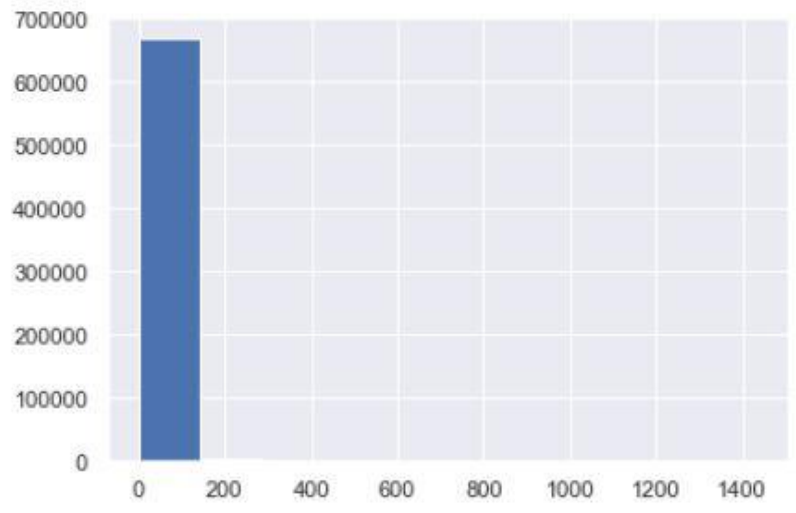
_ Outliers:

A very simple way (not the most sufficient one; we will still have outliers.) to deal with outliers is to look at the most important variable "Delay" and try to use our observations:



```
# simple way to detect and remove outliers
ma=max(model['Delay'])
print(ma))
plt.hist(model['Delay'].values)
plt.show
model = model[model['Delay'] < 190]
```

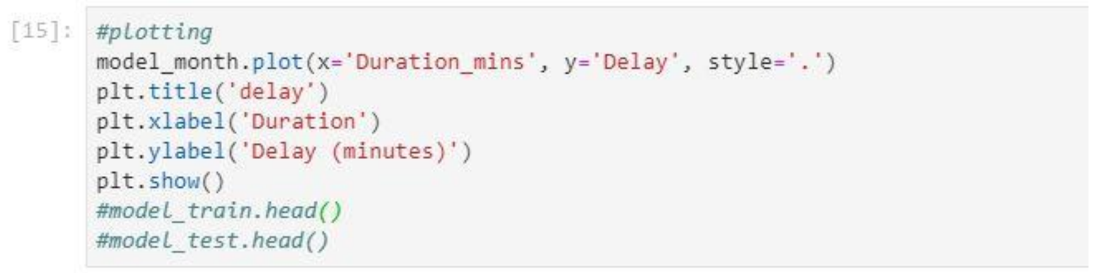
```
1436.0
```



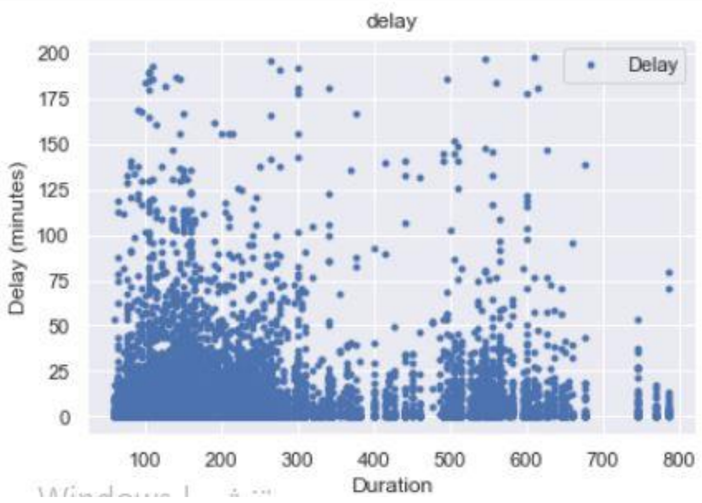
we see that the biggest value of 'Delay' is 1436 mins and most of the values are less than 200 so we can remove all values bigger than 200 which will make more than 95% of the Data.

_ Visualization:

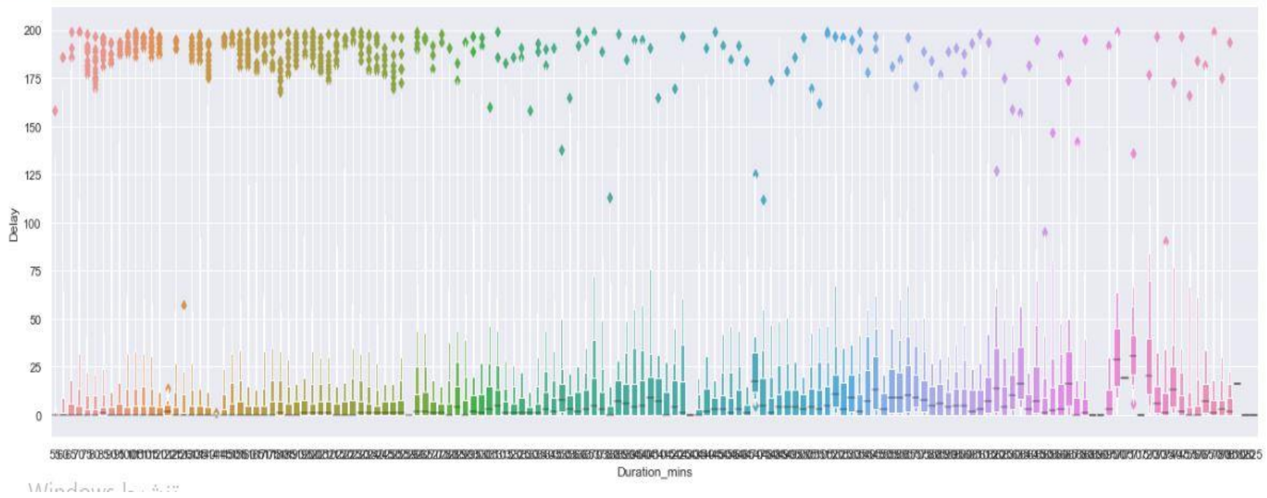
It's not very easy to visualize such big and complex Data, we can visualize using PCA or simply with Delay and flight duration for example:



```
[15]: #plotting
      model_month.plot(x='Duration_mins', y='Delay', style='.')
      plt.title('delay')
      plt.xlabel('Duration')
      plt.ylabel('Delay (minutes)')
      plt.show()
      #model_train.head()
      #model_test.head()
```



```
[17]: sns.catplot(y = "Delay", x = "Duration_mins", data = model_train.sort_values("Delay", ascending = False), kind="boxen", height = 6, aspect = 3)
      plt.show()
```



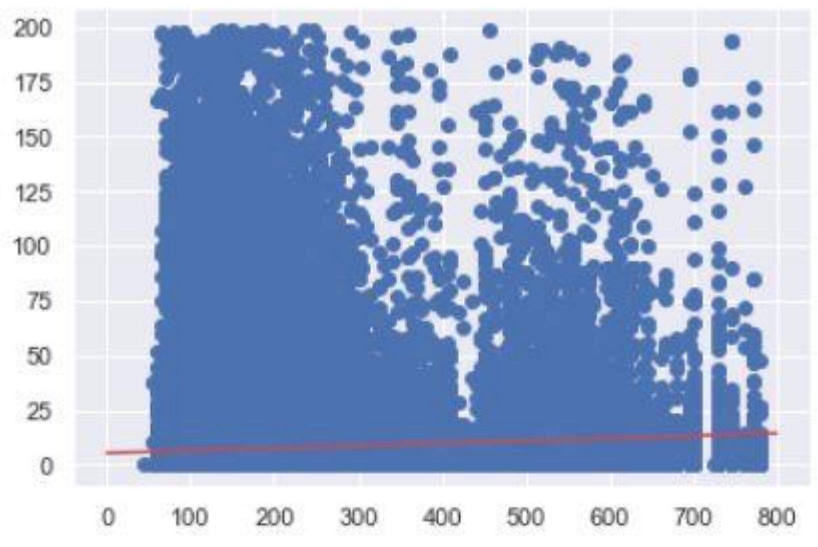
Above we can see Delay (y – axis) and Duration (x – axis).

- ## ML methods and algorithms used:

_ Polynomial Regression and Linear Regression:
Using one variable 'Duration_mins' which is the duration of the flight in minutes and 'Delay' as Target we are able to predict the Delay of the year 2018 flights but with very poor results because it's not a Linear Regression problem, I think it's too complicated for polynomial Regression.

And the results below confirm my opinion:
we see a very far prediction from a lot of the values with a bad r2score of -80

```
[31]: <function matplotlib.pyplot.show(*args, **kw)>
```



```
[32]: from sklearn.metrics import r2_score
      prediction = lin.predict(x_test_poly)
      r2_score(prediction, y_test)

[32]: -80.09937598653926
```

_ Regularization with Lasso Regression:

Ridge and Lasso regression are powerful techniques generally used for creating parsimonious models in presence of a 'large' number of features.

And after searching for the right value of alpha with 1 predictor 'Duration_mins', We find the following results:

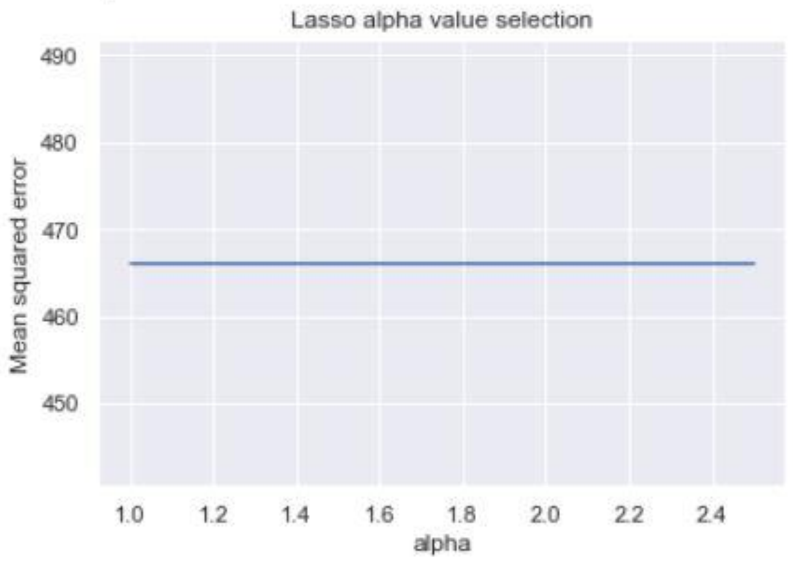
Losses are not changing as we change alpha!

which means alpha here has no real effect on the simplicity of the model, maybe because the data is too complicated or the outliers are not removed well or because we are using only one parameter.

However we have a mean absolute error of 9.6 min , and root mean squared error is 16.3 min which is not that bad.

```
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred_L)))
print('Coefficient of Determination:', metrics.r2_score(y_test, y_pred_L))
```

```
Best alpha: 1
```



```
MAE: 9.559458403455313
RMSE: 16.327490039804015
Coefficient of Determination: -0.05386686482505443
```

_ Random Forest Regressor

As for my last algorithm or method I decided to put more predictors into the decision process, so I used the variables that I deduced before (which I think should have an influence on the 'Delay') and put them into ' ExtraTreesRegressor' to know what features has the bigger effect on the Target (delay).

' ExtraTreesRegressor ' is a class that implements a **meta estimator** (An estimator which takes another estimator as a parameter. Examples include pipeline. Pipeline , model_selection.)
and that meta estimator that it fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

A **random forest** is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

(So basically, we are trying to take a lot of estimators or algorithms that choose among themselves the best method).

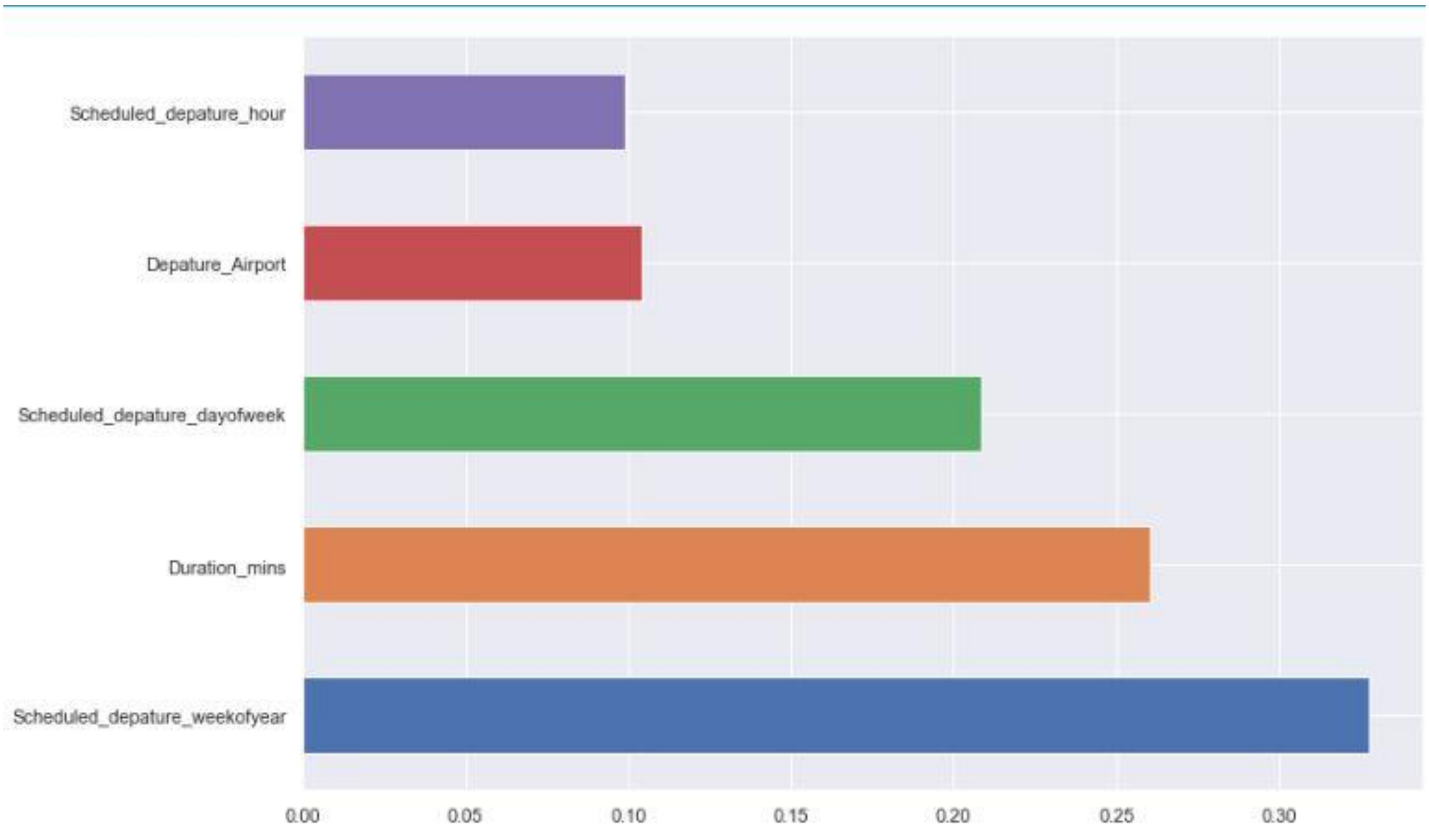
Another important idea is **Feature importance.**

Feature importance scores can be calculated for problems that involve predicting a numerical value, called regression, and those problems that involve predicting a class label, called classification.

The scores are useful and can be used in a range of situations in a predictive modeling problem, such as:

- Better understanding the data.
- Better understanding a model.
- Reducing the number of input feature

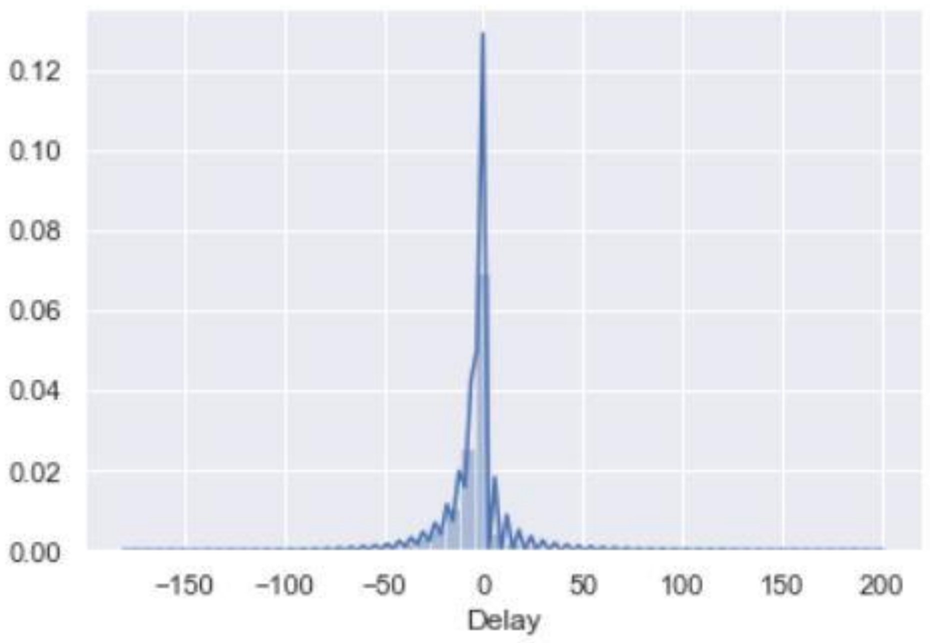
Results of **ExtraTreesRegressor** using Feature importance:



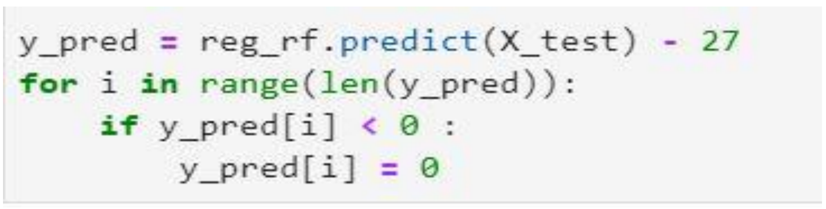
As we see the 'Week Of Year' , 'Duration mins' and 'Day Of Week' are the most important to our delay.

(I took some variables out because they were not really effective like 'Destination Airport).

And after applying Random Forrest Regression using all above features and variables (best results are when we take the most important 3 features) and after fitting and predicting we have the following results:

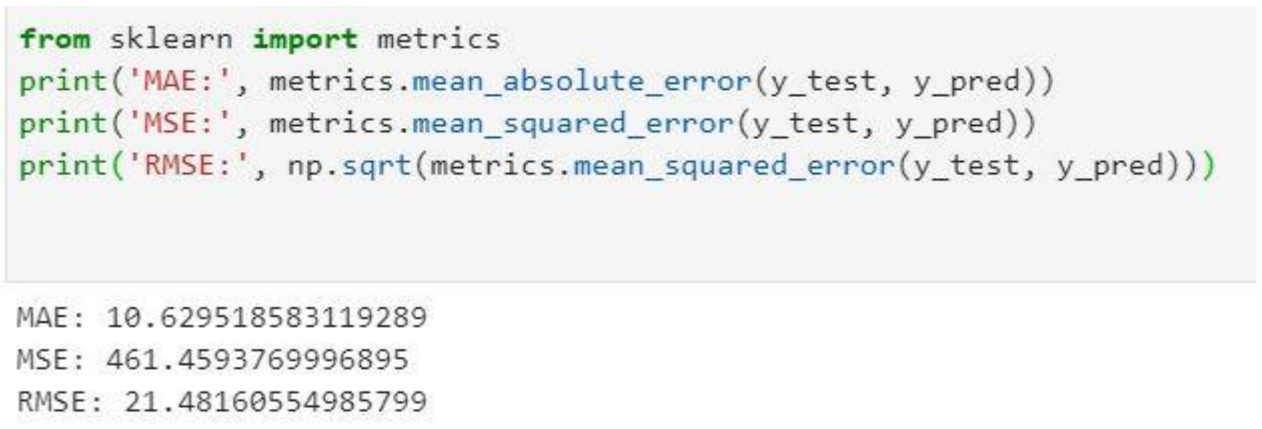


This graph above shows us Y_real-Y_predict that means we are guessing a bit less than we should so I shifted the prediction a bit up hoping to get a better result.



```
y_pred = reg_rf.predict(X_test) - 27
for i in range(len(y_pred)):
    if y_pred[i] < 0 :
        y_pred[i] = 0
```

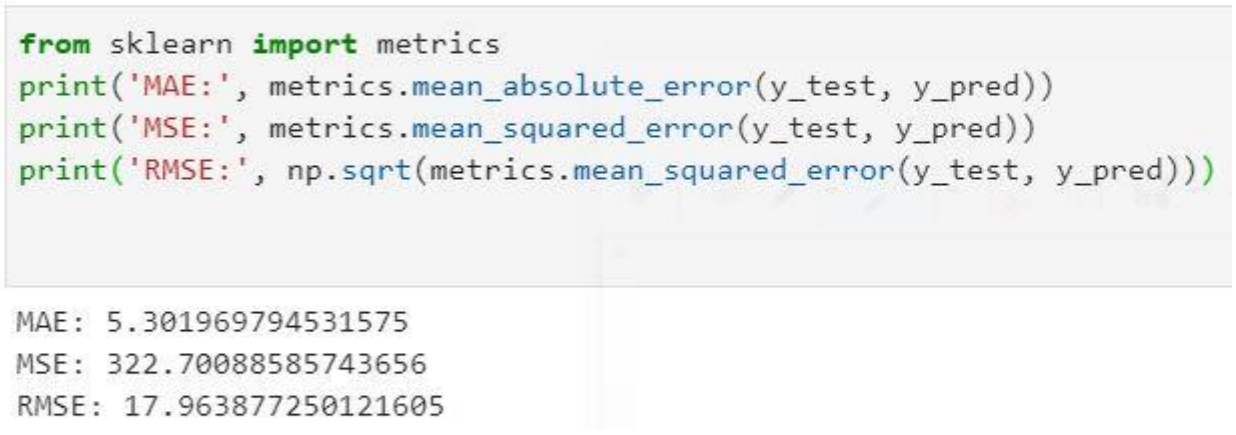
I subtracted an amount between 50 and 0 (after reading the Y_real-Y_predict graph) and if the value is less than 0 then it's not a Delay and it's Zero, notice if we put all the predicted values to 0 we will still get a result even better or similar to polynomial regression (with 1 predictor).
results before shifting :



```
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
MAE: 10.629518583119289
MSE: 461.4593769996895
RMSE: 21.48160554985799
```

Results after shifting:



```
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
MAE: 5.301969794531575
MSE: 322.70088585743656
RMSE: 17.963877250121605
```

The Results aren’t great as expected but they are better than the previous ones.

P.S: I Tried to use local outliner factor and some other ML algorithms but did not succeed.

Thank you for your time.

Sinan Ibrahim