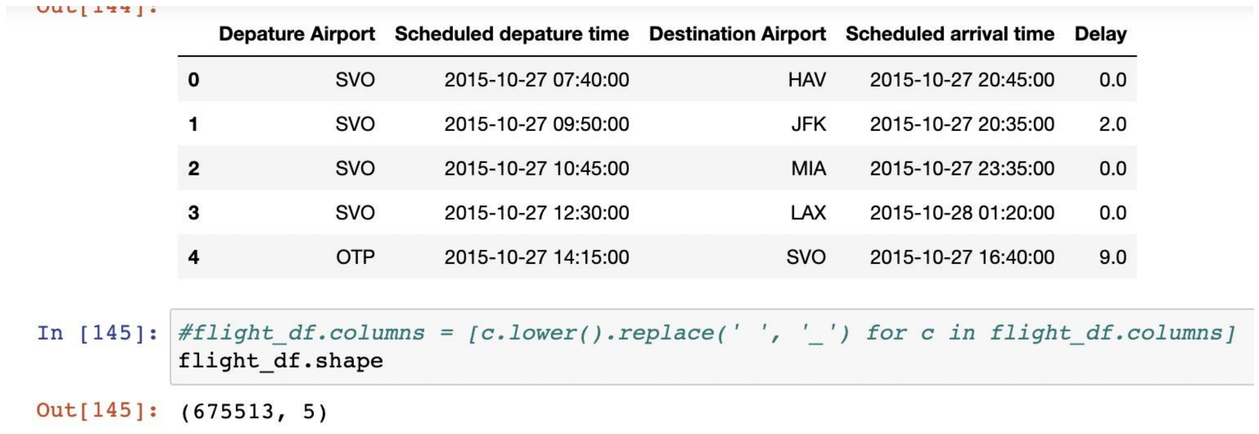
Machine Learning Assignment-1 Documentation and Report

Motivation:

The following report consists of a detailed report of how I aim to make predictions based on a table(csv file) consisting of flight travel information like Airport destination etc.. The process involved using 3 Machine learning models -- Linear Regression, Multiple Linear Regression(Ridge) and Polynomial Regression. Estimation is made on flight delay based on Data received from an Innopolis flight partner company.

Brief Data description and pre-processing :

The data consists of 675513 data values, and 5 columns consisting of Destination Airport, Departure Airport, Scheduled departure Time, Scheduled Arrival Time and Delay. The figures below is a snapshot of the data set, showing the first 5 rows, along with the dimension of the data set, and summary statistics

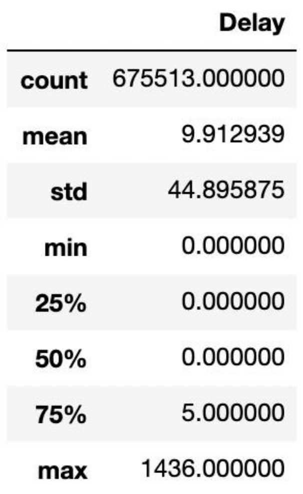


Out[144]:

	Depature Airport	Scheduled depature time	Destination Airport	Scheduled arrival time	Delay
0	SVO	2015-10-27 07:40:00	HAV	2015-10-27 20:45:00	0.0
1	SVO	2015-10-27 09:50:00	JFK	2015-10-27 20:35:00	2.0
2	SVO	2015-10-27 10:45:00	MIA	2015-10-27 23:35:00	0.0
3	SVO	2015-10-27 12:30:00	LAX	2015-10-28 01:20:00	0.0
4	OTP	2015-10-27 14:15:00	SVO	2015-10-27 16:40:00	9.0

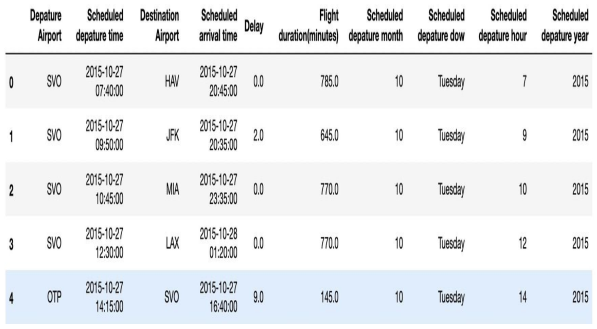
```
In [145]: #flight_df.columns = [c.lower().replace(' ', '_') for c in flight_df.columns]
          flight_df.shape

Out[145]: (675513, 5)
```

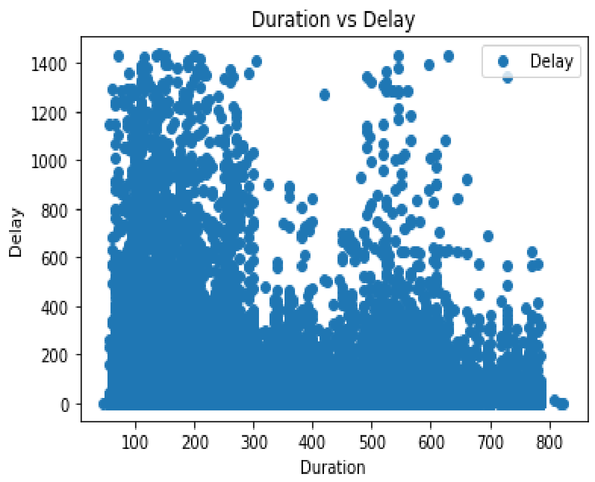


	Delay
count	675513.000000
mean	9.912939
std	44.895875
min	0.000000
25%	0.000000
50%	0.000000
75%	5.000000
max	1436.000000

For the preprocessing step, i separated the date time column for scheduled departure and made them into separate columns, ie. I made the Scheduled departure Hour, Year, Month, day of week as new columns(to be used as features) and saved it into a dataframe, dropping the Scheduled departure time(as the new columns capture more data into separate features) and also the Scheduled Arrival time, because departure makes more sense to take into account. A new column was also made based on flight duration. And plotted against flight delay. A table of the new dataframe and the plot of flight duration vs flight delay is given below

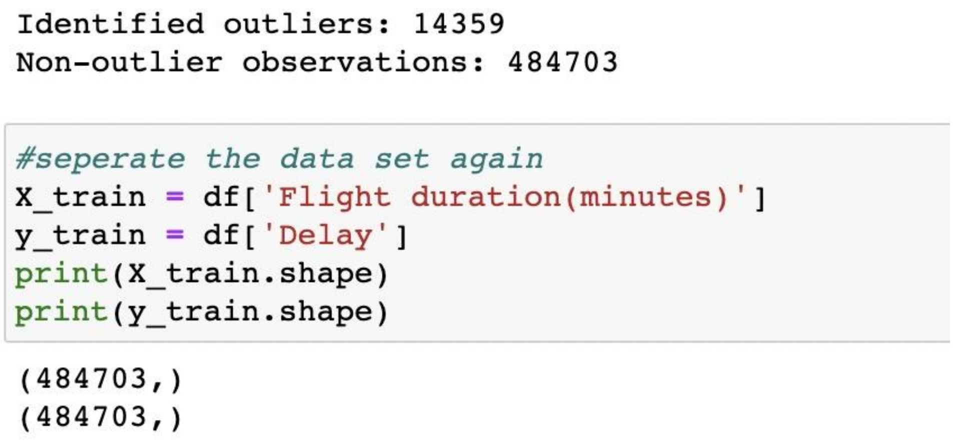


	Depature Airport	Scheduled depature time	Destination Airport	Scheduled arrival time	Delay	Flight duration(minutes)	Scheduled depature month	Scheduled depature dow	Scheduled depature hour	Scheduled depature year
0	SVO	2015-10-27 07:40:00	HAV	2015-10-27 20:45:00	0.0	785.0	10	Tuesday	7	2015
1	SVO	2015-10-27 09:50:00	JFK	2015-10-27 20:35:00	2.0	645.0	10	Tuesday	9	2015
2	SVO	2015-10-27 10:45:00	MIA	2015-10-27 23:35:00	0.0	770.0	10	Tuesday	10	2015
3	SVO	2015-10-27 12:30:00	LAX	2015-10-28 01:20:00	0.0	770.0	10	Tuesday	12	2015
4	OTP	2015-10-27 14:15:00	SVO	2015-10-27 16:40:00	9.0	145.0	10	Tuesday	14	2015



Splitting the Data :
The Data was split on the Scheduled departure Time, and Train data was taken for the year between(and inclusive) of 2015 & 2017, while the test data was taken for the year 2018. This was done for all models.

Outlier Detection:
For Simple linear regression and Polynomial Regression, The Flight duration was used as a single feature for estimating flight delay. **The Outlier Method for Simple Linear Regression and Polynomial regression**, was taking the standard deviation on both the Train putting an upper and lower cutoff based on adding and subtracting the mean(of the train data) from the train data respectively, and hence removing the outliers . The following is the number of outliers detected.

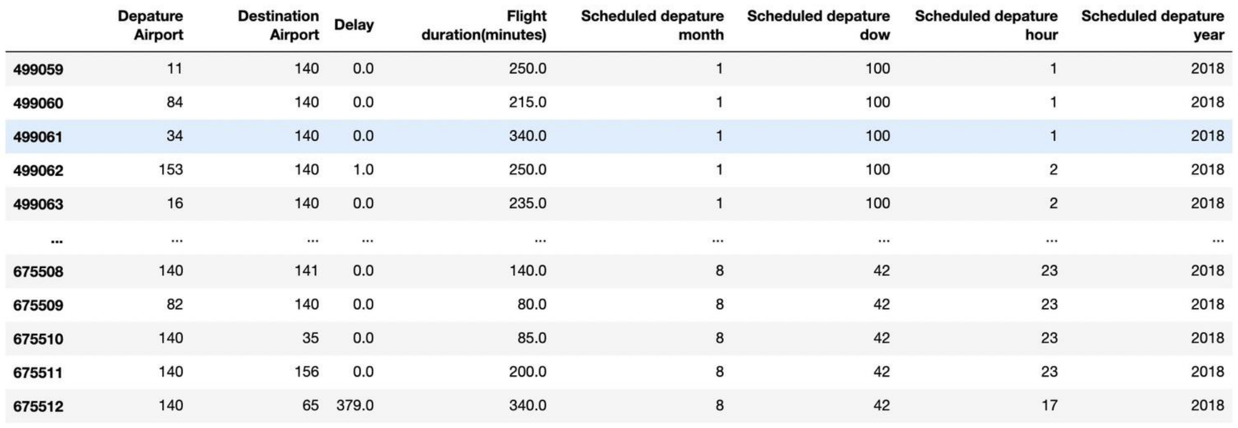


```
Identified outliers: 14359
Non-outlier observations: 484703
```

```
#seperate the data set again
X_train = df['Flight duration(minutes)']
y_train = df['Delay']
print(X_train.shape)
print(y_train.shape)
```

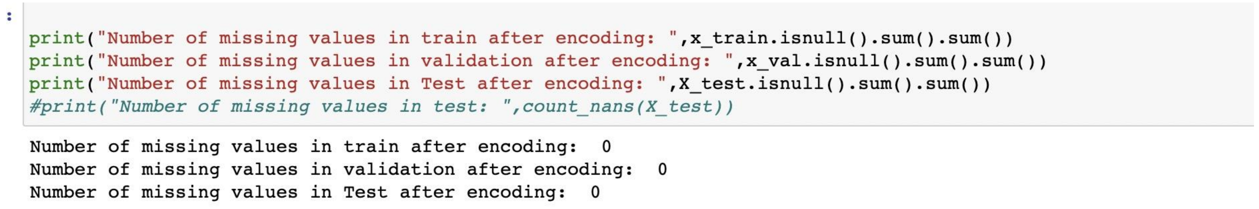
```
(484703,)
(484703,)
```

For Multiple Linear Regression, certain other preprocessing methods were used before outlier detection and removal, which included the following: Scheduled departure Airport ,Scheduled Destination Airport and Scheduled day of the week were encoded using Label Encoder. Since LabelEncoder can encode only a single column, both these columns were stacked and then applied the encoding . To deal with new data , we store the transformed class labels as a dictionary , and apply the encoding to the test data, if it exists , use the same numerical encoding , if not, use -1. I used -1, as the likelihood of too many new airports existing in the test data is quite small, and day of week is already taken care of as the probability of new value is zero. I have used LabelEncoder for the scheduled departure airports and day of the week, as certain airports have more flights, and possibly could lead to more delays, and the same for the days of the week, eg, Friday could have more influx of flights and hence more delays, and so on so forth, and the months have the same reason, eg : December is a festive month, might result in more flight delays. The following is the table snapshot, after encoding(of the test data).



	Depature Airport	Destination Airport	Delay	Flight duration(minutes)	Scheduled depature month	Scheduled depature dow	Scheduled depature hour	Scheduled depature year
499059	11	140	0.0	250.0	1	100	1	2018
499060	84	140	0.0	215.0	1	100	1	2018
499061	34	140	0.0	340.0	1	100	1	2018
499062	153	140	1.0	250.0	1	100	2	2018
499063	16	140	0.0	235.0	1	100	2	2018
...	...	...	...	...	...	...	...	...
675508	140	141	0.0	140.0	8	42	23	2018
675509	82	140	0.0	80.0	8	42	23	2018
675510	140	35	0.0	85.0	8	42	23	2018
675511	140	156	0.0	200.0	8	42	23	2018
675512	140	65	379.0	340.0	8	42	17	2018

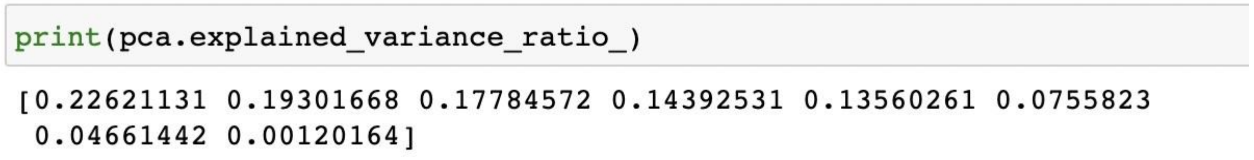
Further, more as mentioned seen in the code below, there exists no empty values , so imputing could have been avoided, but i used it for safety measure, by using SimpleImputer class, with a strategy equal to ‘mean’, and used MinMaxScaler(), for rescaling all the features , for proper processing and using PCA.



```
print("Number of missing values in train after encoding: ",x_train.isnull().sum().sum())
print("Number of missing values in validation after encoding: ",x_val.isnull().sum().sum())
print("Number of missing values in Test after encoding: ",X_test.isnull().sum().sum())
#print("Number of missing values in test: ",count_nans(X_test))
```

```
Number of missing values in train after encoding:  0
Number of missing values in validation after encoding:  0
Number of missing values in Test after encoding:  0
```

Local Outlier Factor detection was used after all the processing to remove possible outliers. 6045 outliers were removed from the training data, and PCA(n_components = 8) along with variance ratio was calculated as shown below:



```
print(pca.explained_variance_ratio_)
```

```
[0.22621131 0.19301668 0.17784572 0.14392531 0.13560261 0.0755823
 0.04661442 0.00120164]
```

Machine Learning Models used and Results :

Simple Linear Regression:

Using flight duration as the single feature, the following the result for this model along with the various metrics calculated on train data and test data:

```
y_train_pred  = linreg.predict(X_train)
print('Mean Absolute Error training :', metrics.mean_absolute_error(y_train_pred, y_train))
print('Mean Squared Error training :', metrics.mean_squared_error(y_train, y_train_pred))
print('Root Mean Squared Error training :', np.sqrt(metrics.mean_squared_error(y_train, y_train_p
print('R2 score training :', metrics.r2_score(y_train,y_train_pred))
```

```
Mean Absolute Error training : 15.25866212144142
Mean Squared Error training : 2143.64488975645
Root Mean Squared Error training : 46.29951284577895
R2 score training : 0.0012500638690523536
```

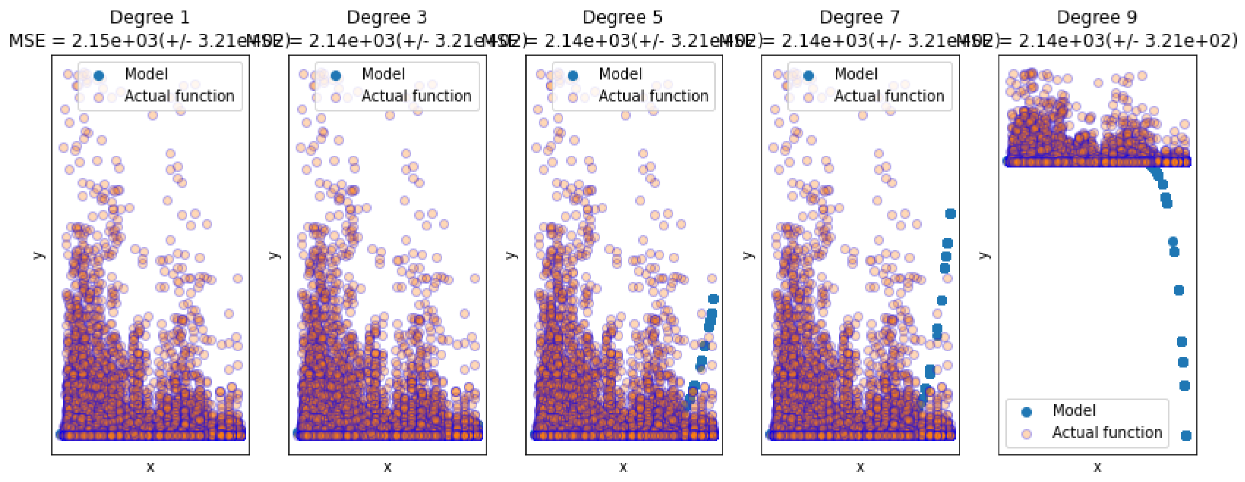
```
y_pred = linreg.predict(X_test)
eval_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
```

```
print('Mean Absolute Error for test :', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error for test :', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error for test :', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R2 score  for test :', metrics.r2_score(y_test,y_pred))
```

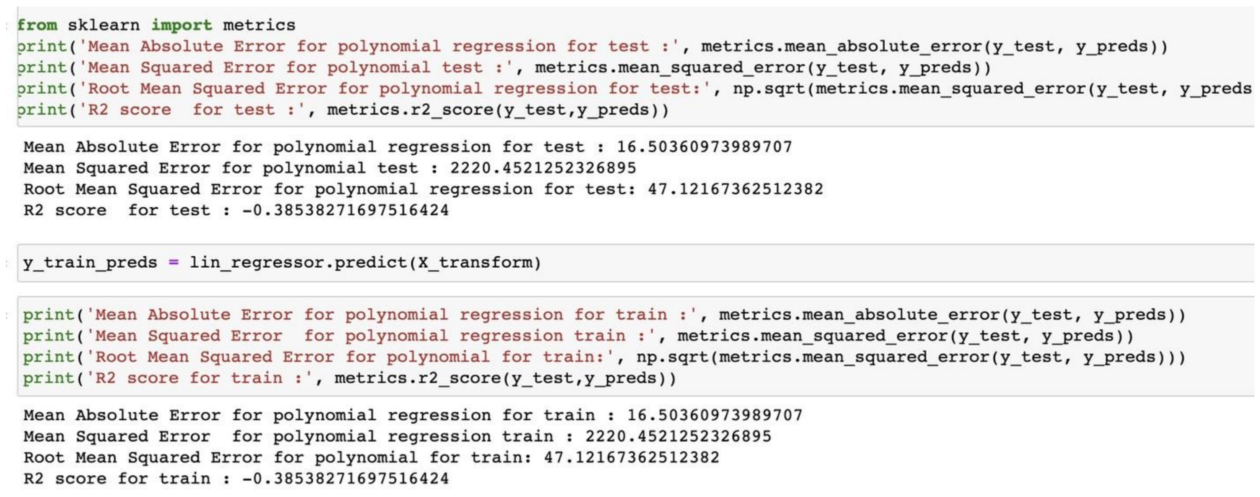
```
Mean Absolute Error for test : 14.410128100731614
Mean Squared Error for test : 1619.3014771835187
Root Mean Squared Error for test : 40.24054518993895
R2 score  for test : -0.010313284654723587
```

For Polynomial Regression :

The same feature was used for polynomial regression and cross validation set was used for finding a good value of degree for transforming the variable into a polynomial feature. The following graphs display polynomial regression for various degrees like 1, 3,5,7 and 9. As seen below when degree goes to about 9, the model goes for a downward trend.



The MSE seems a little better for degree 5 polynomial. The following was train and test error after collecting best degree polynomial :



```
from sklearn import metrics
print('Mean Absolute Error for polynomial regression for test :', metrics.mean_absolute_error(y_test, y_preds))
print('Mean Squared Error for polynomial test :', metrics.mean_squared_error(y_test, y_preds))
print('Root Mean Squared Error for polynomial regression for test:', np.sqrt(metrics.mean_squared_error(y_test, y_preds
print('R2 score  for test :', metrics.r2_score(y_test,y_preds))
```

```
Mean Absolute Error for polynomial regression for test : 16.50360973989707
Mean Squared Error for polynomial test : 2220.4521252326895
Root Mean Squared Error for polynomial regression for test: 47.12167362512382
R2 score  for test : -0.38538271697516424
```

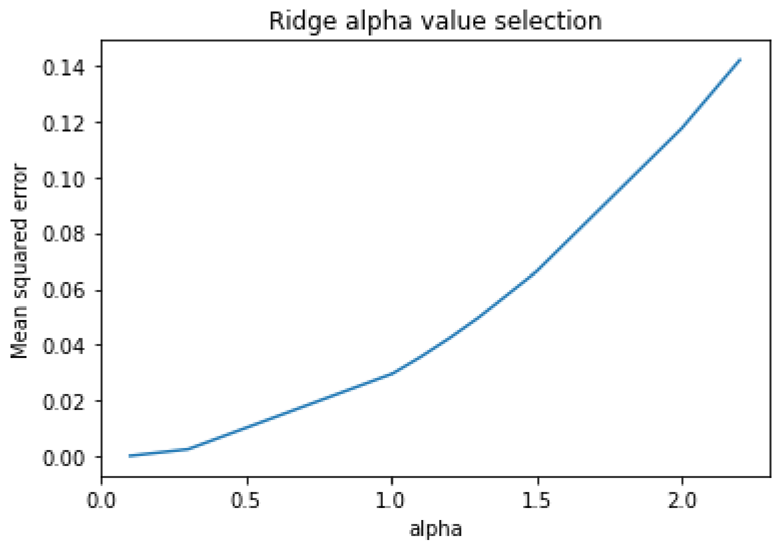
```
y_train_preds = lin_regressor.predict(X_transform)
```

```
print('Mean Absolute Error for polynomial regression for train :', metrics.mean_absolute_error(y_test, y_preds))
print('Mean Squared Error  for polynomial regression train :', metrics.mean_squared_error(y_test, y_preds))
print('Root Mean Squared Error for polynomial for train:', np.sqrt(metrics.mean_squared_error(y_test, y_preds)))
print('R2 score for train :', metrics.r2_score(y_test,y_preds))
```

```
Mean Absolute Error for polynomial regression for train : 16.50360973989707
Mean Squared Error  for polynomial regression train : 2220.4521252326895
Root Mean Squared Error for polynomial for train: 47.12167362512382
R2 score for train : -0.38538271697516424
```

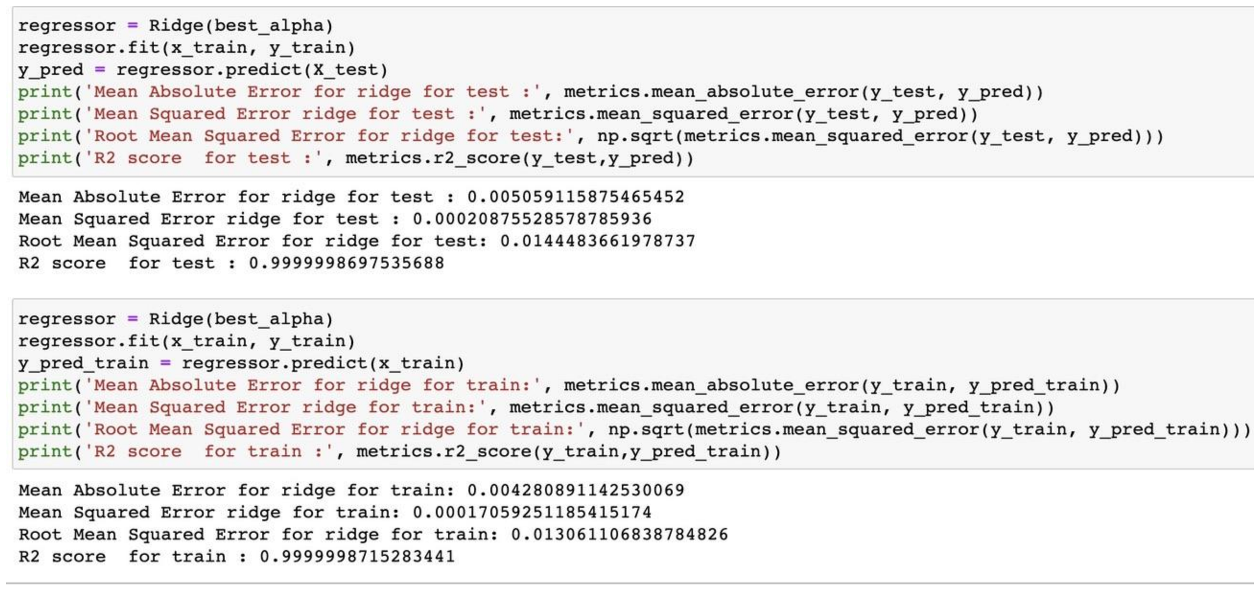
For Multiple Linear Regression:

Before data preprocessing and outlier detection, a validation set of size ⅛ th of the training data was created, as we had used Ridge regression to the variance calculated from PCA, and I felt this was appropriate as each feature contributed to the data variance pretty evenly. So the validation set was used to find best alpha value from the following list : [2.2, 2, 1.5, 1.3, 1.2, 1.1, 1, 0.3, 0.1], and the



following plot was observed :

Best alpha value was taken into consideration and the following errors were observed for both train and test data :



```
regressor = Ridge(best_alpha)
regressor.fit(x_train, y_train)
y_pred = regressor.predict(X_test)
print('Mean Absolute Error for ridge for test :', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error ridge for test :', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error for ridge for test:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R2 score  for test :', metrics.r2_score(y_test,y_pred))
```

```
Mean Absolute Error for ridge for test : 0.005059115875465452
Mean Squared Error ridge for test : 0.00020875528578785936
Root Mean Squared Error for ridge for test: 0.0144483661978737
R2 score  for test : 0.9999998697535688
```

```
regressor = Ridge(best_alpha)
regressor.fit(x_train, y_train)
y_pred_train = regressor.predict(x_train)
print('Mean Absolute Error for ridge for train:', metrics.mean_absolute_error(y_train, y_pred_train))
print('Mean Squared Error ridge for train:', metrics.mean_squared_error(y_train, y_pred_train))
print('Root Mean Squared Error for ridge for train:', np.sqrt(metrics.mean_squared_error(y_train, y_pred_train)))
print('R2 score  for train :', metrics.r2_score(y_train,y_pred_train))
```

```
Mean Absolute Error for ridge for train: 0.004280891142530069
Mean Squared Error ridge for train: 0.00017059251185415174
Root Mean Squared Error for ridge for train: 0.013061106838784826
R2 score  for train : 0.9999998715283441
```

As we can see , the training error is slightly less than the test error for all metrics , and it should be for a good regression model.

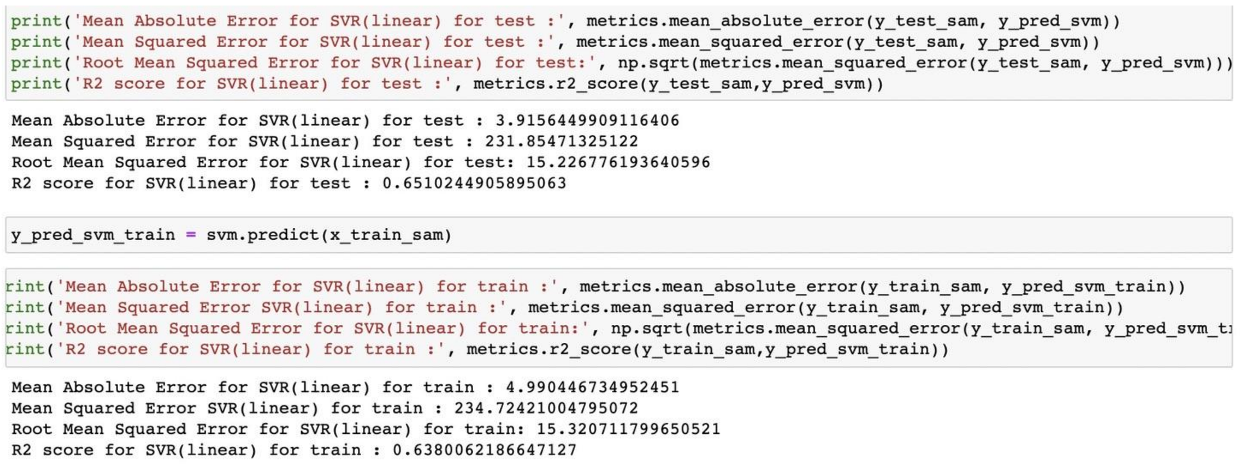
Conclusions :

From the observed results , it would seem that polynomial regression overcomplicates the modelling process, and simple regression actually resulted in slightly larger training error than test error, and this is a case of bias-ness in the data, or due to fact that simple linear regression underfit the data as it was too simple or it could be that the fit is unknown, we can see from the graph of plotting flight duration and delay, that it is not simple linear correlation . And the case with polynomial regression seems to overfit once we take the degree higher.

Multiple linear regression with Ridge regularization seems to work best for prediction in this case, as per my opinion, as can be seen from the metrics evaluated, and goodness of fit value is almost 1, which is a good sign that fit is almost perfect.

SVR :

I have run a trial with a support vector regressor from sklearn. For the kernel I used linear , due to the fact that multiple Linear Regression showed more promising results, MSE values were also lower with the ‘linear’ kernel compared to polynomials or Radial Basis functions. The following are the train and test errors using various metrics after finding best C and gamma parameters using cross Validation set. (I used 1000 training samples , 100 validation and 400 test samples). Linear kernel:



```
print('Mean Absolute Error for SVR(linear) for test :', metrics.mean_absolute_error(y_test_sam, y_pred_svm))
print('Mean Squared Error for SVR(linear) for test :', metrics.mean_squared_error(y_test_sam, y_pred_svm))
print('Root Mean Squared Error for SVR(linear) for test:', np.sqrt(metrics.mean_squared_error(y_test_sam, y_pred_svm)))
print('R2 score for SVR(linear) for test :', metrics.r2_score(y_test_sam,y_pred_svm))
```

```
Mean Absolute Error for SVR(linear) for test : 3.9156449909116406
Mean Squared Error for SVR(linear) for test : 231.85471325122
Root Mean Squared Error for SVR(linear) for test: 15.226776193640596
R2 score for SVR(linear) for test : 0.6510244905895063
```

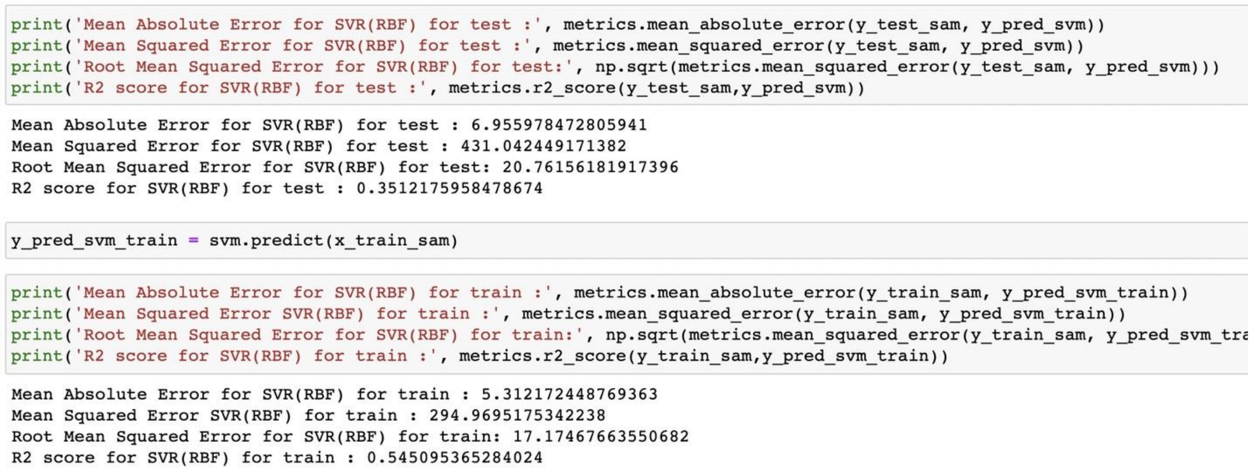
```
y_pred_svm_train = svm.predict(x_train_sam)
```

```
rint('Mean Absolute Error for SVR(linear) for train :', metrics.mean_absolute_error(y_train_sam, y_pred_svm_train))
rint('Mean Squared Error SVR(linear) for train :', metrics.mean_squared_error(y_train_sam, y_pred_svm_train))
rint('Root Mean Squared Error for SVR(linear) for train:', np.sqrt(metrics.mean_squared_error(y_train_sam, y_pred_svm_t
rint('R2 score for SVR(linear) for train :', metrics.r2_score(y_train_sam,y_pred_svm_train))
```

```
Mean Absolute Error for SVR(linear) for train : 4.990446734952451
Mean Squared Error SVR(linear) for train : 234.72421004795072
Root Mean Squared Error for SVR(linear) for train: 15.320711799650521
R2 score for SVR(linear) for train : 0.6380062186647127
```

Linear kernel showed a difference in train and test i.e test seems to be lower than train, could be due to imbalance in data splitting and also due to small training set.

Model for rbf with same C and gamma parameters seems to be appropriate with test and train error, although slightly higher than using Linear kernel .RBF kernel:



```
print('Mean Absolute Error for SVR(RBF) for test :', metrics.mean_absolute_error(y_test_sam, y_pred_svm))
print('Mean Squared Error for SVR(RBF) for test :', metrics.mean_squared_error(y_test_sam, y_pred_svm))
print('Root Mean Squared Error for SVR(RBF) for test:', np.sqrt(metrics.mean_squared_error(y_test_sam, y_pred_svm)))
print('R2 score for SVR(RBF) for test :', metrics.r2_score(y_test_sam,y_pred_svm))
```

```
Mean Absolute Error for SVR(RBF) for test : 6.955978472805941
Mean Squared Error for SVR(RBF) for test : 431.042449171382
Root Mean Squared Error for SVR(RBF) for test: 20.76156181917396
R2 score for SVR(RBF) for test : 0.3512175958478674
```

```
y_pred_svm_train = svm.predict(x_train_sam)
```

```
print('Mean Absolute Error for SVR(RBF) for train :', metrics.mean_absolute_error(y_train_sam, y_pred_svm_train))
print('Mean Squared Error SVR(RBF) for train :', metrics.mean_squared_error(y_train_sam, y_pred_svm_train))
print('Root Mean Squared Error for SVR(RBF) for train:', np.sqrt(metrics.mean_squared_error(y_train_sam, y_pred_svm_tra
print('R2 score for SVR(RBF) for train :', metrics.r2_score(y_train_sam,y_pred_svm_train))
```

```
Mean Absolute Error for SVR(RBF) for train : 5.312172448769363
Mean Squared Error SVR(RBF) for train : 294.9695175342238
Root Mean Squared Error for SVR(RBF) for train: 17.17467663550682
R2 score for SVR(RBF) for train : 0.545095365284024
```

Link to github repo: