Learning From Data

ECMM445

COURSEWORK REPORT

A. INTRODUCTION

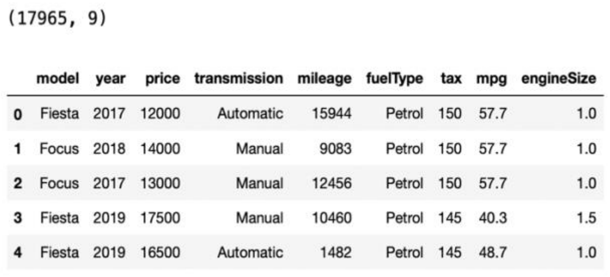
The dataset “Car price prediction” deals with features 9 associated features and 17811 rows of data. The main objective of this analysis is the prediction of used car prices of Ford models and the right period for selling certain cars. Ford has huge variety of models from Mustang to sports car where the price range is also varied.

In this assignment, I will be performing data cleaning, preprocessing, visualization and will be using 4 regression models for prediction along with validating and analyzing the results obtained. The further snags which can be incorporated into preliminary hypothesis is classifying models of Ford, also we can remove any potential outliers from the data to improve the overall performance.

B. DATASET DESCRIPTION

I chose the dataset containing used cars listings in UK. The data contains all Ford models as it is one of the common cars in the British market. The data has 9 attributes and 17965 data entries. The dataset has the following columns/attributes:

- Model: Gives the model of the Ford car.
- Year: Gives the year of purchase
- Price: gives the resale price of that model
- Transmission: Refers to the whole drivetrain (clutch, gearbox, prop shaft.
- Mileage: Mileage provided by the car
- FuelType: Kind of fuel which goes into the car. (Petrol, diesel, electric, other)
- Tax: Vehicle tax to be paid for car
- Mpg: Miles per gallon
- engineSize: Engine size of the car model

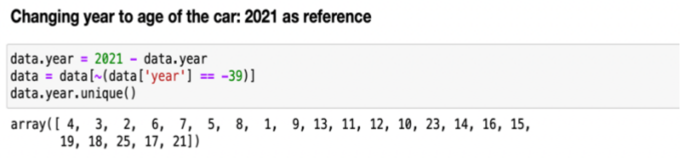


```
(17965, 9)
```

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
0	Fiesta	2017	12000	Automatic	15944	Petrol	150	57.7	1.0
1	Focus	2018	14000	Manual	9083	Petrol	150	57.7	1.0
2	Focus	2017	13000	Manual	12456	Petrol	150	57.7	1.0
3	Fiesta	2019	17500	Manual	10460	Petrol	145	40.3	1.5
4	Fiesta	2019	16500	Automatic	1482	Petrol	145	48.7	1.0

C. DATA EXPLORATION

I loaded the ford.csv file as a pandas dataframe. checked for null values using isna() function and removed the duplicates from the dataframe, using the drop_duplicates(keep=False, inplace=True). The entries were reduced to 17811. I also converted the year to age of the car by taking reference year as 2021 and subtracting the year from it, resulting in the age of the car.



Changing year to age of the car: 2021 as reference

```
data.year = 2021 - data.year
data = data[~(data['year'] == -39)]
data.year.unique()
```

```
array([ 4,  3,  2,  6,  7,  5,  8,  1,  9, 13, 11, 12, 10, 23, 14, 16, 15,
       19, 18, 25, 17, 21])
```

The following plot demonstrates the relationship between mileage and age of the car:

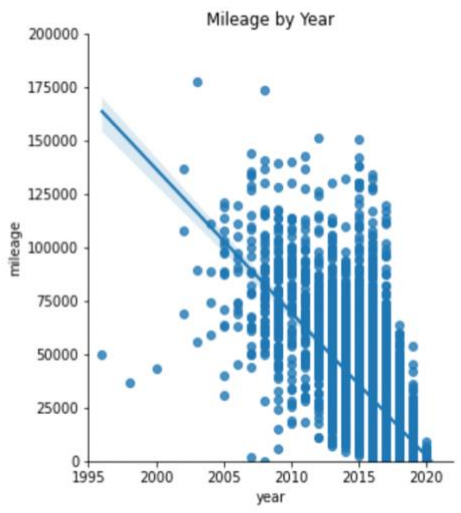


Fig. Plot between mileage and year.

It is observed that as the age of the car increases, the mileage also increases. However, there were few old cars which were not used by owners as much.

The graph below spans the prices of each model. Mustang has the highest price range, which might influence our results.

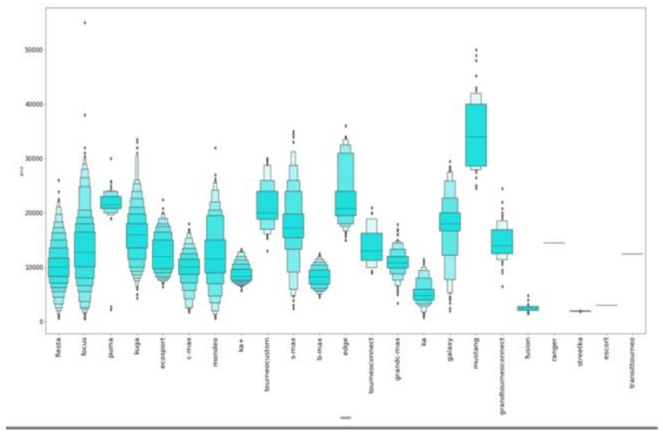


Fig. Plot showing prices of all models

The following graph shows relation between mpg and fueltype

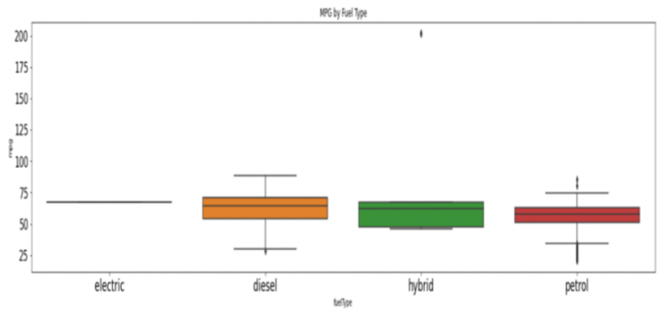


Fig. Plot between mpg and fueltype

The below plot describes the relationship between prices and fueltype:

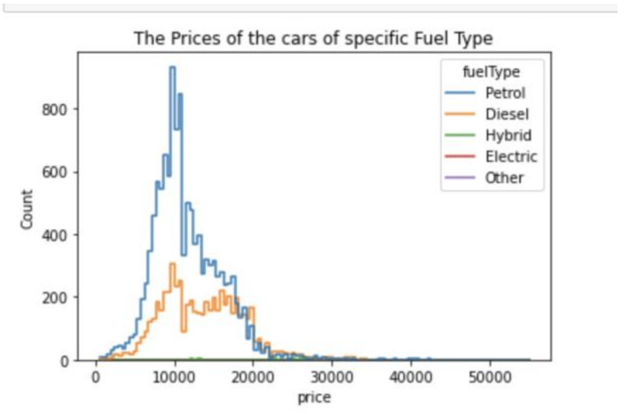


Fig. Plot between prices and fueltype.

The graph above shows that most of the petrol models have a price from 900-1000 Pounds

The following plot demonstrates the relationship between the different features.

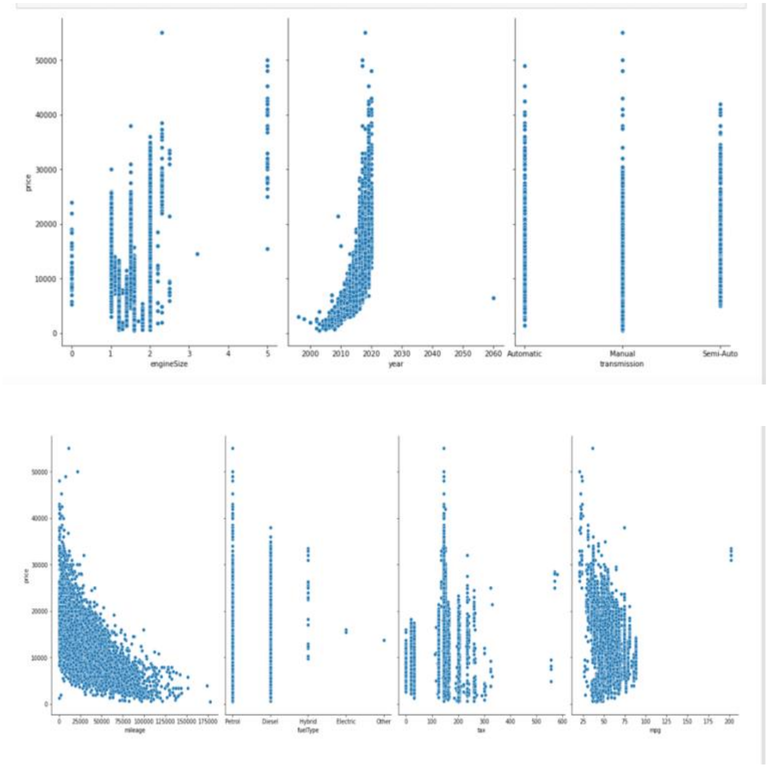


Fig. relationship between the different features.

The above plots show varied negative and positive correlation between different features with respect to price.

Correlation matrix for variables

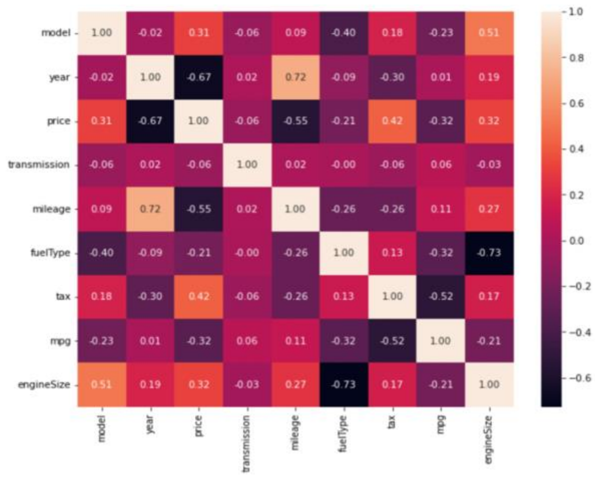
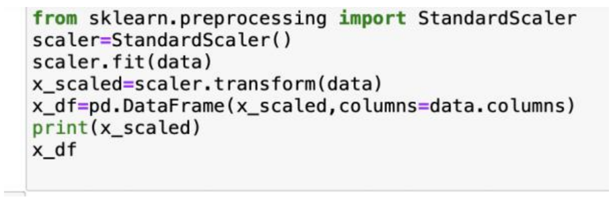


Fig. Matrix for Correlation (feature against feature)

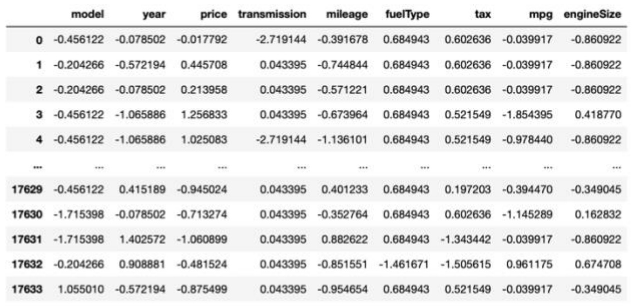
- Model: 0.31 correlation to price
- Year: -0.67 correlation to price
- Transmission: -0.06 correlation to price
- Mileage: -0.55 correlation to price
- FuelType: -0.21 correlation to price

- Tax: 0.42 correlation to price
- Mpg: -0.32 correlation to price
- engineSize: 0.32 correlation to price

I also normalized the features using the StandardScaler function from sklearn.



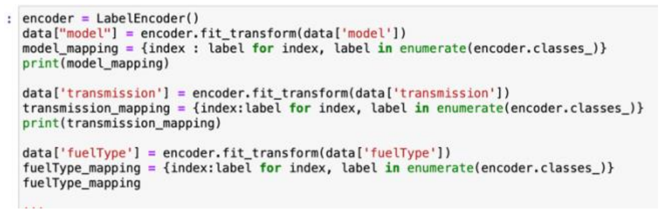
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(data)
x_scaled=scaler.transform(data)
x_df=pd.DataFrame(x_scaled,columns=data.columns)
print(x_scaled)
x_df
```



	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
0	-0.456122	-0.078502	-0.017792	-2.719144	-0.391678	0.684943	0.602636	-0.039917	-0.860922
1	-0.204266	-0.572194	0.445708	0.043395	-0.744844	0.684943	0.602636	-0.039917	-0.860922
2	-0.204266	-0.078502	0.213958	0.043395	-0.571221	0.684943	0.602636	-0.039917	-0.860922
3	-0.456122	-1.065886	1.256833	0.043395	-0.673964	0.684943	0.521549	-1.854395	0.418770
4	-0.456122	-1.065886	1.025083	-2.719144	-1.136101	0.684943	0.521549	-0.978440	-0.860922
...	...	...	...	...	...	...	...	...	...
17629	-0.456122	0.415189	-0.945024	0.043395	0.401233	0.684943	0.197203	-0.394470	-0.349045
17630	-1.715398	-0.078502	-0.713274	0.043395	-0.352764	0.684943	0.602636	-1.145289	0.162832
17631	-1.715398	1.402572	-1.060899	0.043395	0.882622	0.684943	-1.343442	-0.039917	-0.860922
17632	-0.204266	0.908881	-0.481524	0.043395	-0.851551	-1.461671	-1.505615	0.961175	0.674708
17633	1.055010	-0.572194	-0.875499	0.043395	-0.954654	0.684943	0.521549	-0.039917	-0.349045

Fig. Table shows Scaled Features

Finally, I converted the categorical data into numerical/indicator data using the function labelencoder().

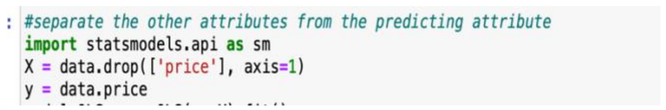


```
encoder = LabelEncoder()
data["model"] = encoder.fit_transform(data['model'])
model_mapping = {index : label for index, label in enumerate(encoder.classes_)}
print(model_mapping)

data['transmission'] = encoder.fit_transform(data['transmission'])
transmission_mapping = {index:label for index, label in enumerate(encoder.classes_)}
print(transmission_mapping)

data['fuelType'] = encoder.fit_transform(data['fuelType'])
fuelType_mapping = {index:label for index, label in enumerate(encoder.classes_)}
fuelType_mapping
```

The attributes were divided into dependent and independent variables.



```
#separate the other attributes from the predicting attribute
import statsmodels.api as sm
X = data.drop(['price'], axis=1)
y = data.price
```

y: dependent variable i.e 'price'

X: independent variable i.e all other variables except price.

Example: fueltype, mpg,model,tax etc.

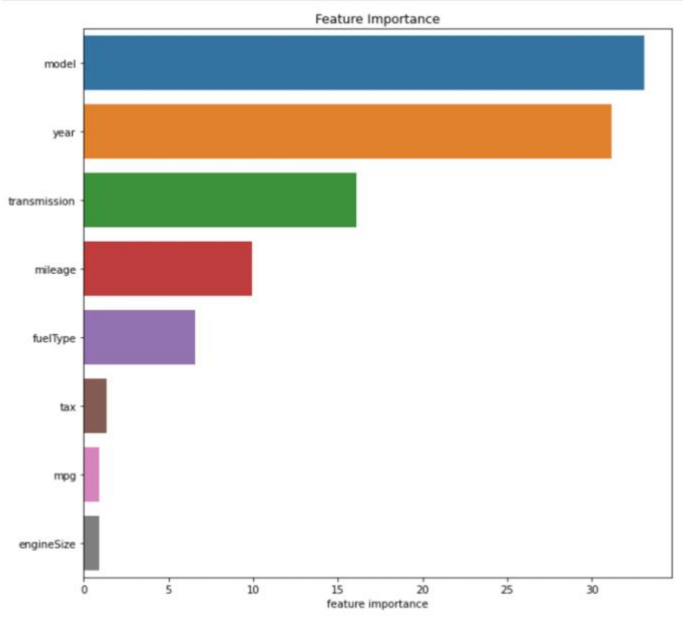
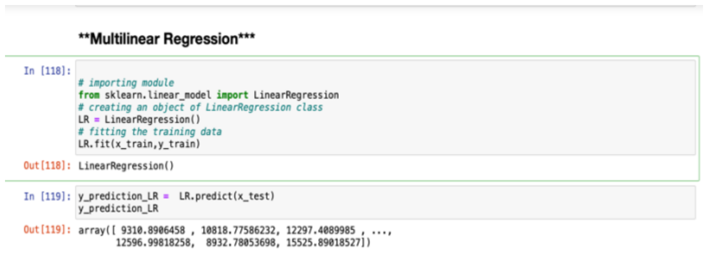


Fig. Plot showing Feature Importance

D. SUMMARY OF MODELS

The following algorithms were implemented to analyze the r2 score, mean squared error and root mean squared error parameters for efficiency:

1. *Multilinear Regression:*



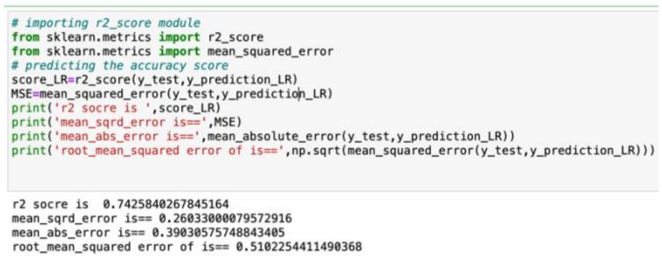
Multilinear Regression*

```
In [118]: # importing module
          from sklearn.linear_model import LinearRegression
          # creating an object of LinearRegression class
          LR = LinearRegression()
          # fitting the training data
          LR.fit(x_train,y_train)
Out[118]: LinearRegression()

In [119]: y_prediction_LR =  LR.predict(x_test)
          y_prediction_LR
Out[119]: array([ 9310.8906458 , 10818.77586232, 12297.4089985 , ...,
                 12596.99818258,  8932.78053698, 15525.89018527])
```

For implementing the model, the data was divided into x_train,y_train,x_test, y_test for training and testing where 80% of the data was in training the model and the rest 20% in testing the model.

In the snippet above, I created an object called LR of class LinearRegression which is a module in sklearn. LR object fit the training data into our model. Using the predict function, we predicted the y values, which are seen as an array in the above snippet.

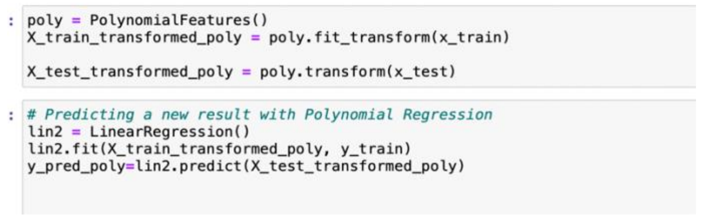


```
# importing r2_score module
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
# predicting the accuracy score
score_LR=r2_score(y_test,y_prediction_LR)
MSE=mean_squared_error(y_test,y_prediction_LR)
print('r2 socre is ',score_LR)
print('mean_sqrd_error is==',MSE)
print('mean_abs_error is==',mean_absolute_error(y_test,y_prediction_LR))
print('root_mean_squared error of is==',np.sqrt(mean_squared_error(y_test,y_prediction_LR)))
```

```
r2 socre is  0.7425840267845164
mean_sqrd_error is== 0.26033000079572916
mean_abs_error is== 0.39030575748843405
root_mean_squared error of is== 0.5102254411490368
```

PARAMETERS	SCORE
R2	0.7425840267845164
Mean Squared Error	0.26033000079572916
Mean Absolute Error	0.39030575748843405
Root mean Squared error	0.5102254411490368

2. *Polynomial Regression*



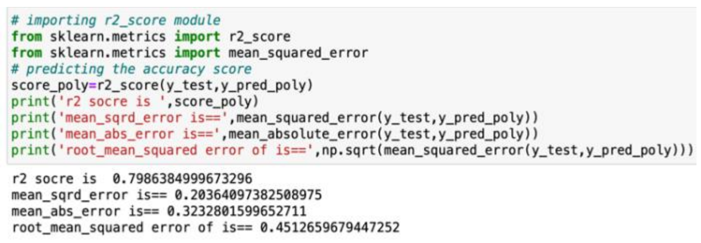
```
poly = PolynomialFeatures()
X_train_transformed_poly = poly.fit_transform(x_train)

X_test_transformed_poly = poly.transform(x_test)
```

```
# Predicting a new result with Polynomial Regression
lin2 = LinearRegression()
lin2.fit(X_train_transformed_poly, y_train)
y_pred_poly=lin2.predict(X_test_transformed_poly)
```

The data was divided into x_train,y_train,x_test, y_test for training and testing where 80% of the data was in training the model and the rest 20% in testing the model.

In the snippet above, I created an object called poly of class PolynomialFeatures which is a module in sklearn. poly object transforms the training data into polynomial data with enlarged inputs and fits it to our model. I created another object of LinearRegression called lin2 and fit the transformed x_train and y_train , later lin2 predicts the values with transformed x_test.

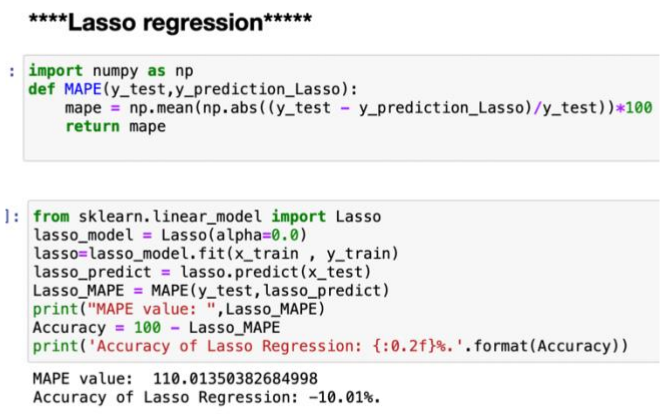


```
# importing r2_score module
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
# predicting the accuracy score
score_poly=r2_score(y_test,y_pred_poly)
print('r2 socre is ',score_poly)
print('mean_sqrd_error is==',mean_squared_error(y_test,y_pred_poly))
print('mean_abs_error is==',mean_absolute_error(y_test,y_pred_poly))
print('root_mean_squared error of is==',np.sqrt(mean_squared_error(y_test,y_pred_poly)))
```

```
r2 socre is  0.7986384999673296
mean_sqrd_error is== 0.20364097382508975
mean_abs_error is== 0.3232801599652711
root_mean_squared error of is== 0.4512659679447252
```

PARAMETERS	SCORES
R2	0.7986384999673296
Mean squared error	0.20364097382508975
Mean Absolute error	0.3232801599652711
Root mean squared error	0.4512659679447252

3. *Lasso regression:*



******Lasso regression*******

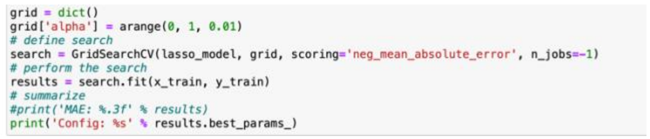
```
import numpy as np
def MAPE(y_test,y_prediction_Lasso):
    mape = np.mean(np.abs((y_test - y_prediction_Lasso)/y_test))*100
    return mape
```

```
from sklearn.linear_model import Lasso
lasso_model = Lasso(alpha=0.0)
lasso=lasso_model.fit(x_train , y_train)
lasso_predict = lasso.predict(x_test)
Lasso_MAPE = MAPE(y_test,lasso_predict)
print("MAPE value: ",Lasso_MAPE)
Accuracy = 100 - Lasso_MAPE
print('Accuracy of Lasso Regression: {:0.2f}%.'.format(Accuracy))
```

```
MAPE value:  110.01350382684998
Accuracy of Lasso Regression: -10.01%.
```

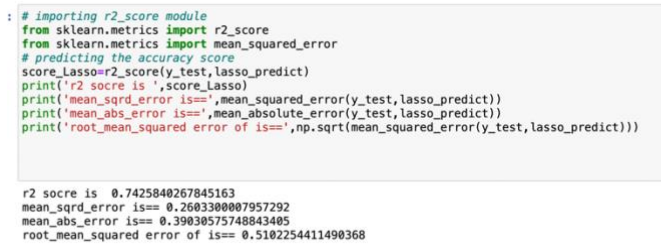
The data was divided into x_train,y_train,x_test, y_test for training and testing where 80% of the data was in training the model and the rest 20% in testing the model.

In order to tune the hyperparameters like lamda, I chose the lamda value through gridsearch that tests values between 0.0 and 1.0 with a grid separation of 0.01.



```
grid = dict()
grid['alpha'] = arange(0, 1, 0.01)
# define search
search = GridSearchCV(lasso_model, grid, scoring='neg_mean_absolute_error', n_jobs=-1)
# perform the search
results = search.fit(x_train, y_train)
# summarize
#print('MAE: %.3f' % results)
print('Config: %s' % results.best_params_)
```

The lamda value chosen by the method was 0.00 which means no penalty on the loss function. (Weight to the penalty is 0). As Lasso is a regularization algorithm is helpful when the estimated coefficients of the model become large making the model sensitive to inputs (example: number of observation>number of inputs). Thus, Lasso penalizes the model based on absolute coefficient values by minimising them and making some of them 0. This is L1 penalty.

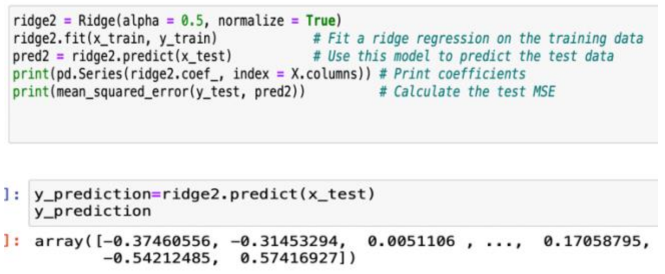


```
: # importing r2_score module
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
# predicting the accuracy score
score_Lasso=r2_score(y_test,lasso_predict)
print('r2 socre is ',score_Lasso)
print('mean_sqrd_error is==',mean_squared_error(y_test,lasso_predict))
print('mean_abs_error is==',mean_absolute_error(y_test,lasso_predict))
print('root_mean_squared error of is==',np.sqrt(mean_squared_error(y_test,lasso_predict)))

r2 socre is  0.7425840267845163
mean_sqrd_error is== 0.2603300007957292
mean_abs_error is== 0.39030575748843405
root_mean_squared error of is== 0.5102254411490368
```

PARAMETER	SCORES
R2	0.7425840267845163
Mean squared error	0.2603300007957292
Mean absolute error	0.39030575748843405
Root mean square error	0.5102254411490368

4. *Ridge Regression*



```
ridge2 = Ridge(alpha = 0.5, normalize = True)
ridge2.fit(x_train, y_train)             # Fit a ridge regression on the training data
pred2 = ridge2.predict(x_test)           # Use this model to predict the test data
print(pd.Series(ridge2.coef_, index = X.columns)) # Print coefficients
print(mean_squared_error(y_test, pred2))          # Calculate the test MSE
```

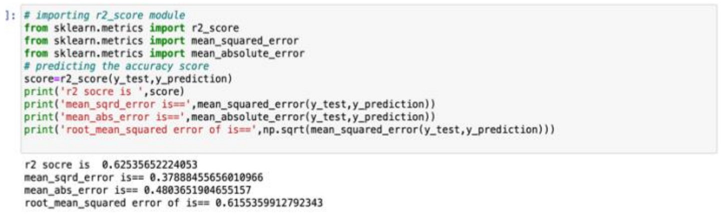
```
]: y_prediction=ridge2.predict(x_test)
   y_prediction

]: array([-0.37460556, -0.31453294,  0.0051106 , ...,  0.17058795,
          -0.54212485,  0.57416927])
```

The data was divided into x_train,y_train,x_test, y_test for training and testing where 80% of the data was in training the model and the rest 20% in testing the model.

In order to tune the hyperparameters like lamda, I chose the lamda value through gridsearch that tests values between 0.0 and 1.0 with a grid separation of 0.01.

The lamda value chosen by the method was 0.99 which means L2 penalty on the loss function. The L2 minimizes the large coefficients but not so much that we lose those from the model by allowing them to be 0.



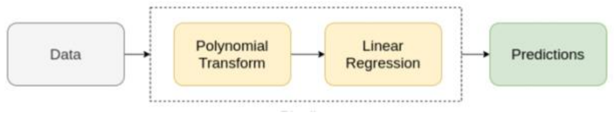
```
]: # importing r2_score module
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
# predicting the accuracy score
score=r2_score(y_test,y_prediction)
print('r2 socre is ',score)
print('mean_sqrd_error is==',mean_squared_error(y_test,y_prediction))
print('mean_abs_error is==',mean_absolute_error(y_test,y_prediction))
print('root_mean_squared error of is==',np.sqrt(mean_squared_error(y_test,y_prediction)))

r2 socre is  0.62535652224053
mean_sqrd_error is== 0.37888455656010966
mean_abs_error is== 0.4803651904655157
root_mean_squared error of is== 0.6155359912792343
```

PARAMETER	SCORES
R2	0.62535652224053
Mean squared error	0.3788845565601096
Mean absolute error	0.4803651904655157
Root mean square error	0.6155359912792343

E. EVALUATION RESULTS: FINAL MODEL

According to the analysis, I concluded that Polynomial regression performed the best with R2 of 0.79. This is due to the curvilinear nature of the independent and dependent variables.

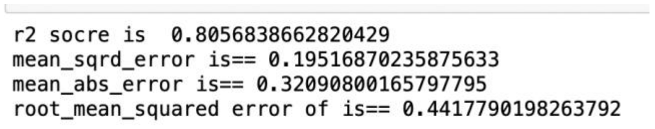


PIPELINE of the Algorithm

It also has a low Mean squared error of approx. 0.203 which implies that the model is accurate and data is closely distributed on the regression line and the difference between predicted values and actual values is very less.

I took the degree of polynomial to be 2, as we increase the degree, the accuracy will increase and will come to a point, after that the accuracy wont be affected.

Also when the Testing split was changed to the ratio of 60:40, the R2 score of polynomial model increased to 0.8056.



```
r2 socre is  0.8056838662820429
mean_sqrd_error is== 0.19516870235875633
mean_abs_error is== 0.32090800165797795
root_mean_squared error of is== 0.4417790198263792
```

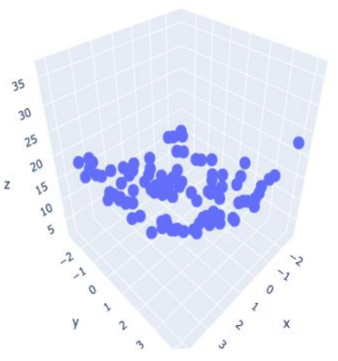
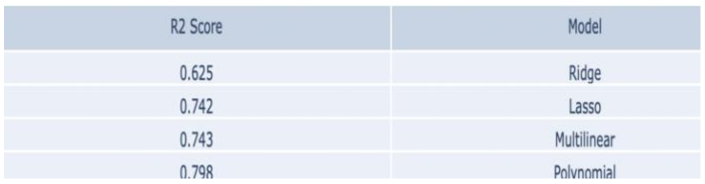


Fig. A 3D representation of data and its variables

In the above plot we can see that the data point forms a curvilinear plot which fits the polynomial model well.

F. KEY FINDINGS AND INSIGHTS

The following table summarizes the R2 scores, Mean squared errors, RMSE from all 4 models.



R2 Score	Model
0.625	Ridge
0.742	Lasso
0.743	Multilinear
0.798	Polynomial

I tested the Polynomial model for the following properties as well and below are my findings:

A. *Check for Homoscedasticity*

Homoscedasticity means that the residuals have equal or almost equal variance across the regression line. By plotting the error terms with predicted terms, we can check that there should not be any pattern in the error terms.

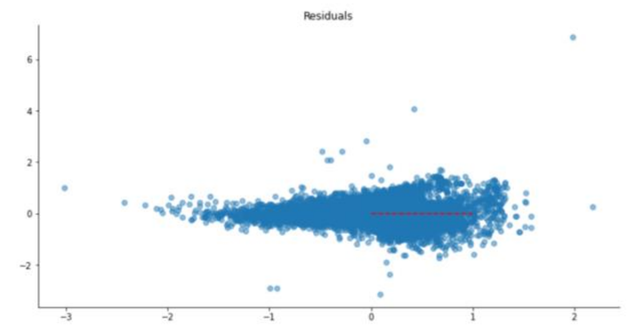


Fig. The residual points are distributed over the line

B. *Check for Normality of error terms/residuals*

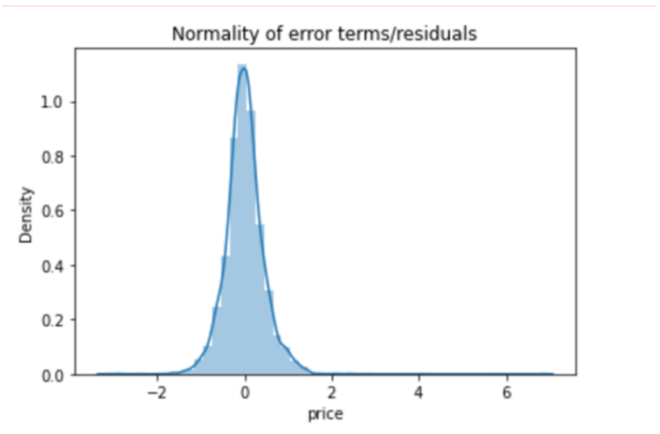
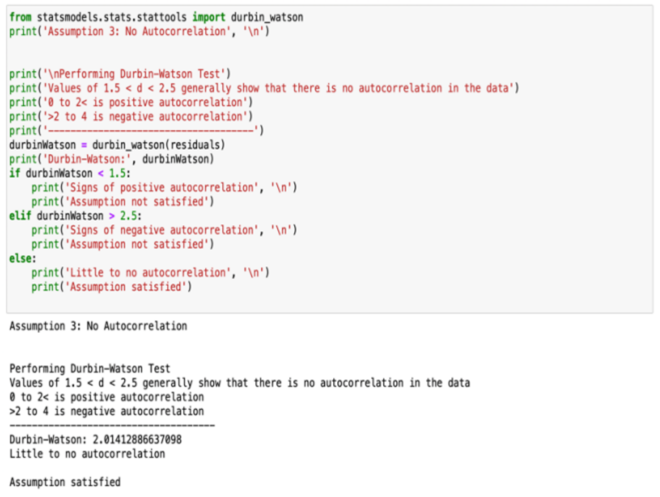


Fig. Distribution of residuals

C. *No Autocorrelation*



```
from statsmodels.stats.stattools import durbin_watson
print('Assumption 3: No Autocorrelation', '\n')


print('\nPerforming Durbin-Watson Test')
print('Values of 1.5 < d < 2.5 generally show that there is no autocorrelation in the data')
print('0 to 2< is positive autocorrelation')
print('>2 to 4 is negative autocorrelation')
print('-------------------------------------')
durbinWatson = durbin_watson(residuals)
print('Durbin-Watson:', durbinWatson)
if durbinWatson < 1.5:
    print('Signs of positive autocorrelation', '\n')
    print('Assumption not satisfied')
elif durbinWatson > 2.5:
    print('Signs of negative autocorrelation', '\n')
    print('Assumption not satisfied')
else:
    print('Little to no autocorrelation', '\n')
    print('Assumption satisfied')
```

```
Assumption 3: No Autocorrelation


Performing Durbin-Watson Test
Values of 1.5 < d < 2.5 generally show that there is no autocorrelation in the data
0 to 2< is positive autocorrelation
>2 to 4 is negative autocorrelation
-------------------------------------
Durbin-Watson: 2.01412886637098
Little to no autocorrelation

Assumption satisfied
```

G. IMPROVING PREDICTION

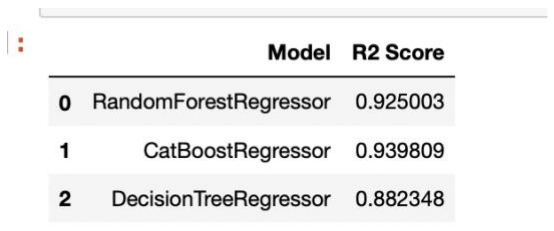
Currently, we tested our model with degree 2 which might not have covered all data points.

As part of improving the Polynomial model, we can increase the complexity of the model by increasing the degree to3 and 4 and evaluating our R2 score, MSE, RMSE to improve the results. A higher degree will cover more varied data points.

We can also change the ratio of training and test split from 80-20 to 60-40.

Polynomial regression models are also sensitive to outliers and can seriously affect the performance. We can eliminate outliers wherever possible.

I conducted further experiments by implementing Catboost, DecisionTree, Random Forest algorithms and got the below results:



	Model	R2 Score
0	RandomForestRegressor	0.925003
1	CatBoostRegressor	0.939809
2	DecisionTreeRegressor	0.882348

Therefore, for higher accuracy and R2 score, we can tune the above three algorithms for this dataset as they proved to be better in performance. Catboost performed the best, as it works well with categorical data and uses gradient boosting. It converts categorical data to numerical values using a variety of statistics based on categorical features. It is robust in the sense that it does not require tuning of hyperparameter and lowers the chances of overfitting.