Determining a Model for Diamond Dataset

Kiranmai Arva, Email id: arvakiranmai@gmail.com, Student Id: 200416487

Problem Statement:

Important Features that are required to predict the Price of the Diamonds.

1.Data Cleaning

Importing and Cleaning the data as discussed in the Assignment 3

```
from google.colab import drive
```

```
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee649

Enter your authorization code:
··········
Mounted at /content/gdrive
```

```
cd gdrive
```

```
/content/gdrive
```

```
cd My\ Drive
```

```
/content/gdrive/My Drive
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
import sklearn
from sklearn import linear_model
from sklearn import svm, preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.linear_model import Lasso, ElasticNet
from sklearn.feature_selection import SelectFromModel
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV , KFold , cross_val_score
%matplotlib inline
```

```
df=pd.read_csv('diamonds1.csv')
```

```
df.replace('Null',np.NaN, inplace = True)
```

```
df.head()
df = df.drop('Unnamed: 0', axis = 1)
df.head()
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58	334	4.2	4.23	2.63
4	0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75

```
df.drop([df.index[1203] , df.index[1223],df.index[1486]],inplace=True)
```

```
df['depth']=pd.to_numeric(df['depth'])
```

```
df['depth'] = df.apply(
    lambda row: float(((row['z']*2)/float(row['x'])+row['y'])*10) if pd.isnull(row['depth']) else row['depth'],
    axis=1
)
```

```
df["table"]= pd.to_numeric(df["table"])
```

```
df['table'] = df['table'].fillna(57)
```

```
df["price"]= pd.to_numeric(df["price"])
```

```
df.at[125,'price']=949
df.at[144,'price']=949
df.at[165,'price']=2401
df.at[449,'price']=2401
df.at[653,'price']=5325
df.at[881,'price']=5325
```

```
d_cut = {'Fair' : 5, 'Good' : 4, 'Very Good' : 3, 'Premium' : 2, 'Ideal' : 1}
d_clarity ={ 'I1' : 8, 'SI2' : 7, 'SI1' : 6, 'VS2' : 5, 'VS1' : 4, 'VVS2' : 3, 'VVS1' : 2 , 'IF' : 1}
d_color = {'D':1, 'E':2, 'F':3, 'G':4, 'H':5, 'I':6, 'J':7}
```

```
df['cut'] = df['cut'].map(d_cut)
df['clarity'] = df['clarity'].map(d_clarity)
df['color'] = df['color'].map(d_color)
```

```
df["x"]= pd.to_numeric(df["x"])
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53937 entries, 0 to 53939
Data columns (total 10 columns):
carat      53937 non-null float64
cut        53937 non-null int64
color      53937 non-null int64
clarity    53937 non-null int64
depth      53937 non-null float64
table      53937 non-null float64
price      53937 non-null float64
x          53937 non-null float64
y          53937 non-null float64
z          53937 non-null float64
dtypes: float64(7), int64(3)
memory usage: 7.0 MB
```

2. Determining a Model

From the Exploratory Data Analysis done from Ass 3, the two most important features required in predicting the Price of a diamond are Carat and Length of a diamond. To check if these features are truly related to Price some of the modeling techniques which I came up are.

1. Lasso Regression
2. Linear Regression
3. Elastic Net Regression

2.1. Feature selection using Lasso Model:

In order to select the most important features in the dataset and use the important feature to train the model by eliminating the least important features Lasso is used. Lasso Regression can perform both Feature Selection and Regularization [3].

In Lasso feature selection Some of the features which are not important in predicting the Price, their coefficients are reduced to zero.

so those features with coefficient zero can be eleminated.

Lasso regression automatically penalizes the extra features, that is it automatically sets coefficient of a least required feature to zero.

For example there are features X1, X2, X3 to know using which feature the outcome is more predictable lasso regression is used.

so n1, n2,n3 are the coefficient of the regressions for the features x1 to x3 . In the example above, of the three feature x3 is not very important feature in predicting the result so it will automatically penalize the coefficient of a x3 to zero. This process is done automatically by lasso regression.

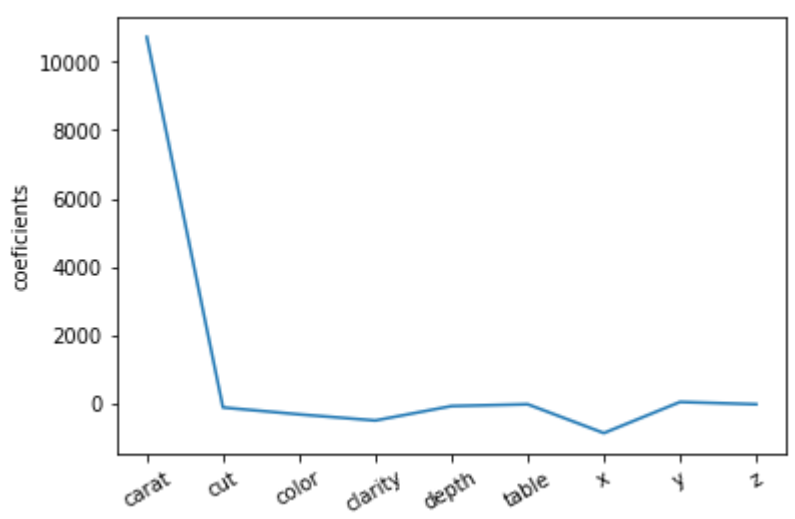
Lasso regression helps in selecting the important feature out of many features available.

```
# Price is the feature we are predicting so drop that column from the dataframe
x=df.drop('price',axis=1).values
y=df['price']
```

```
B=df.drop('price',axis=1).columns

print(B)
```

```
Index(['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'x', 'y', 'z'], dtype='object')
```

```
from sklearn.linear_model import Lasso
lasso=Lasso(alpha=0.1)
# Fitting the model
lasso_coef=lasso.fit(x,y).coef_
# Plotting the Graph
_=plt.plot(range(len(B)),lasso_coef)
_=plt.xticks(range(len(B)),B,rotation=30)
_=plt.ylabel("coeficients")
plt.show()
```



From the above plot, Carat is the most important feature from the available features. Using the feature carat the price of the diamond is easily predictable, rest all the features are not important to predict the price. But from the Exploratory Data Analysis done from the Assingment 3 the two important features that we came up are 'Carat' and 'Length'. In contradiction to the EDA,Lasso feature selection shows only one important feature. so lets try applying Lasso on both 'Carat' and 'Length' and come up with best feature among both.

Lasso Regression:

Least absolute shrinkage selection operator. It performs L1 regularization. The penalty in L1 Regularization equal to absolute value of magnitude of coefficients. In doing so some coefficients are reduced to zero and they can be eliminated from the model. The loss function of lasso is [6]

$L = \sum(\hat{Y}i - Yi)2 + \lambda \sum |\beta|$

coefficients of β is set to zero if they are not relevant

Applying Lasso Model using feature 'Carat' and 'Price'

```
train_df, test_df = train_test_split(df, test_size=0.2, random_state=12)
```

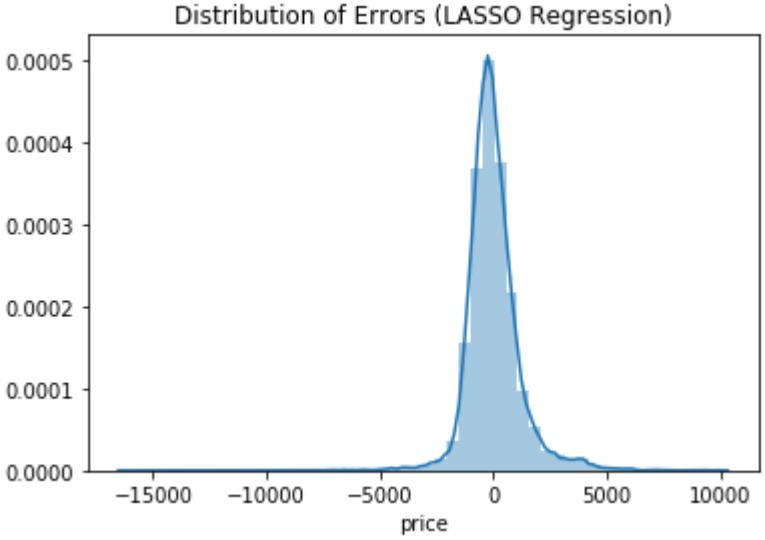
```
X_train = df.drop('price', axis = 1)
Y_train = df['price'].copy()
```

```
# Initialization

Y_test = test_df['price']
X_test = test_df.drop('price', axis = 1)
x=np.array(df['carat']).reshape(-1,1)
y=np.array(df['price']).reshape((-1,1))
```

```
# Spliting the train and test dataset

train_x, test_x, train_y, test_y = train_test_split(x, y,random_state = 2,test_size=0.3)

# Initializing the Lasso Model

las_reg = linear_model.Lasso()

# Fit the data(train the model)
las_reg.fit(train_x,train_y)

b=las_reg.fit(train_x,train_y)
```

```
#X_train_mapped = X_train.copy()

lasso_fit = lasso.fit(X_train, Y_train)

y_lasso=lasso_fit.predict(X_test)

# Predict

y_pred = np.array(las_reg.predict(test_x)).reshape((-1,1))

print(y_pred)

# model evaluation and printing values

print("accuracy: "+ str(las_reg.score(test_x,test_y)*100) + "%")

print("Mean absolute error: {}".format(mean_absolute_error(test_y,y_pred)))

print("Mean squared error: {}".format(mean_squared_error(test_y,y_pred)))

R2 = r2_score(test_y,y_pred)

print('R Squared: {}'.format(R2))

n=test_x.shape[0]
p=test_x.shape[1] - 1

adj_rsquared = 1 - (1 - R2) * ((n - 1)/(n-p-1))

print('Adjusted R Squared: {}'.format(adj_rsquared))

las_reg.score( x, y, sample_weight=None)

# Plotting a graph

sns.distplot(Y_test - y_lasso)

plt.title("Distribution of Errors (LASSO Regression)")

plt.show()
```

```
[[9442.64341098]
 [3634.91545854]
 [1853.87888646]
 ...
 [6035.44301222]
 [7119.55223001]
 [ 305.15143248]]
accuracy: 84.8746868913909%
Mean absolute error: 997.567299255898
Mean squared error: 2373007.12980434
R Squared: 0.8487468689139089
Adjusted R Squared: 0.8487468689139089
```



Above plot shows the distribution of Error Price between the test values and its corresponding predicted values. The range of distibution of error ranges from +5000 to -5000 those are price ranges.

Applying Lasso Model using feature 'Length' and 'Price'

```
train_df, test_df = train_test_split(df, test_size=0.2, random_state=12)
X_train = df.drop('price', axis = 1)
Y_train = df['price'].copy()
# Initialization

Y_test = test_df['price']
X_test = test_df.drop('price', axis = 1)
x=np.array(df['x']).reshape(-1,1)
y=np.array(df['price']).reshape((-1,1))
# Spliting the train and test dataset

train_x, test_x, train_y, test_y = train_test_split(x, y,random_state = 2,test_size=0.3)

# Initializing the Lasso Model

las_reg = linear_model.Lasso()

# Fit the data(train the model)
las_reg.fit(train_x,train_y)
b=las_reg.fit(train_x,train_y)

# Predict
y_pred = np.array(las_reg.predict(test_x)).reshape((-1,1))
print(y_pred)
# model evaluation and printing values
print("accuracy: "+ str(las_reg.score(test_x,test_y)*100) + "%")
print("Mean absolute error: {}".format(mean_absolute_error(test_y,y_pred)))
print("Mean squared error: {}".format(mean_squared_error(test_y,y_pred)))
R2 = r2_score(test_y,y_pred)
print('R Squared: {}'.format(R2))
n=test_x.shape[0]
p=test_x.shape[1] - 1
adj_rsquared = 1 - (1 - R2) * ((n - 1)/(n-p-1))
print('Adjusted R Squared: {}'.format(adj_rsquared))
las_reg.score( x, y, sample_weight=None)
```

```
[[9544.3319492 ]
 [4337.07414055]
 [2096.37532592]
 ...
 [6640.89123165]
 [7398.31054927]
 [ -49.646074  ]]
accuracy: 77.40891624747482%
Mean absolute error: 1364.7651798035984
Mean squared error: 3544310.2850040128
R Squared: 0.7740891624747481
Adjusted R Squared: 0.7740891624747481
0.7822211109038466
```

Lasso Results: From the Lasso Feature selection Algorithm the most important feature that is retrived from the algorithm is "Carat"
But from the EDA done in Assingment 3 we found 'Carat' and "Length" are two important features. So tested two feature against Price using Lasso Regression. When Carat is tested against Price the Accuracy of the algorithm is 84% similarly when tested against Length the accuracy is 77%
So from the Lasso Regression the highest priority in predicting the Price of a Diamond is Carat.

2.2 Linear Regression:

It is a predictive Modelling technique which shows the relationship between the dependent and independent variables. Here dependent refers to target and independent refers to predictors. A best fit line is drawn from the data points based on the difference between the distance of data points.
Linear Regression is represented by
Y =B+mX+e
where B is the intercept, m is the slope and e is the error.
Linear regression with one with one independent variable is called as simple regression. Regression with multiple independent variables is called as Multiple linear regression. To obtain the best fit line among the various data points it is calculated by minimizing the sum of the squares of the vertical deviations from each data point to the line [7]
Min||X-Y||2^2.

Considering the Multiple features into consideration and checking the Accuracy of Model

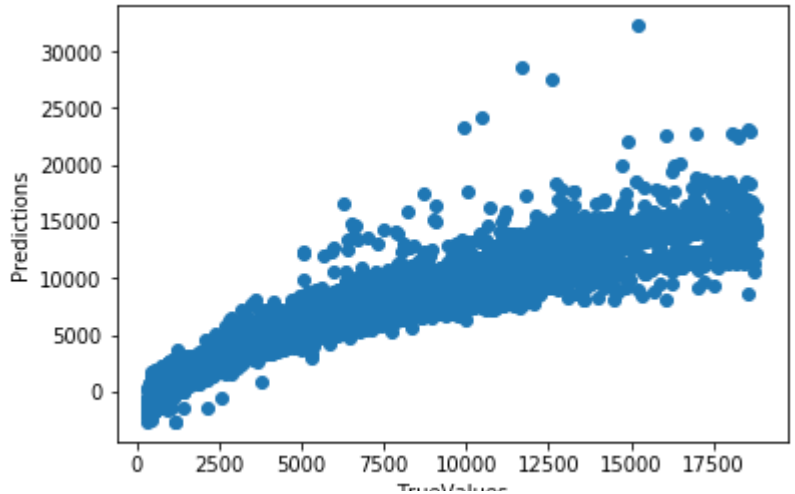
```
# Declaring X and Y from the data sets which are independent and dependent variables respectively
x = (df.drop(["price"],axis=1))

y = df.price
```

```
# sckit-learn implementation
from sklearn.model_selection import train_test_split

train_x, test_x, train_y, test_y = train_test_split(x, y,random_state = 2,test_size=0.3)
```

```
# imports
from sklearn.metrics import mean_absolute_error
```

```
from sklearn.metrics import mean_squared_error

from sklearn.metrics import r2_score

from sklearn import linear_model

# Model initialization

regr = linear_model.LinearRegression()

# Fit the data(train the model)

regr.fit(train_x,train_y)

# Predict

y_pred = regr.predict(test_x)




# model evaluation and printing the values

print("accuracy: "+ str(regr.score(test_x,test_y)*100) + "%")

print("Mean absolute error: {}".format(mean_absolute_error(test_y,y_pred)))

print("Mean squared error: {}".format(mean_squared_error(test_y,y_pred)))

R2 = r2_score(test_y,y_pred)

print('R Squared: {}'.format(R2))

n=test_x.shape[0]
p=test_x.shape[1] - 1

adj_rsquared = 1 - (1 - R2) * ((n - 1)/(n-p-1))

print('Adjusted R Squared: {}'.format(adj_rsquared))

plt.scatter(test_y, y_pred)

plt.xlabel('TrueValues')

plt.ylabel('Predictions')
```

```
accuracy: 90.79271162786956%
Mean absolute error: 803.4105540802199
Mean squared error: 1444529.4980897964
R Squared: 0.9079271162786956
Adjusted R Squared: 0.9078815722813067
Text(0, 0.5, 'Predictions')
```



In the above plot is plotted against the true values in from the test data set agianst the predicted values. Intresting point to be noted from the plot is if a best fit line is drawn most of the scatter points falls around the best fit line. From the Multiple Feature Linear regresssion model Correlation among the features is not addressed even though the accuracy is 90%.

Simple Linear Regression:

A) Considering Only one Independent variable "Carat" and Dependent variable "Price"

```
# Declaring X and Y from the data sets which are independent and dependent variables respectively
x = np.array(df.carat).reshape((-1,1))
y = np.array(df.price).reshape((-1,1))
```

```
# sckit-learn implementation
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(x, y,random_state = 2,test_size=0.3)
```

```
# imports
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn import linear_model
# Model initialization
regr = linear_model.LinearRegression()
# Fit the data(train the model)
regr.fit(train_x,train_y)
# Predict
y_pred = regr.predict(test_x)
print("accuracy: "+ str(regr.score(test_x,test_y)*100) + "%")
print("Mean absolute error: {}".format(mean_absolute_error(test_y,y_pred)))
print("Mean squared error: {}".format(mean_squared_error(test_y,y_pred)))
R2 = r2_score(test_y,y_pred)
print('R Squared: {}'.format(R2))
n=test_x.shape[0]
p=test_x.shape[1] - 1

adj_rsquared = 1 - (1 - R2) * ((n - 1)/(n-p-1))
print('Adjusted R Squared: {}'.format(adj_rsquared))
```

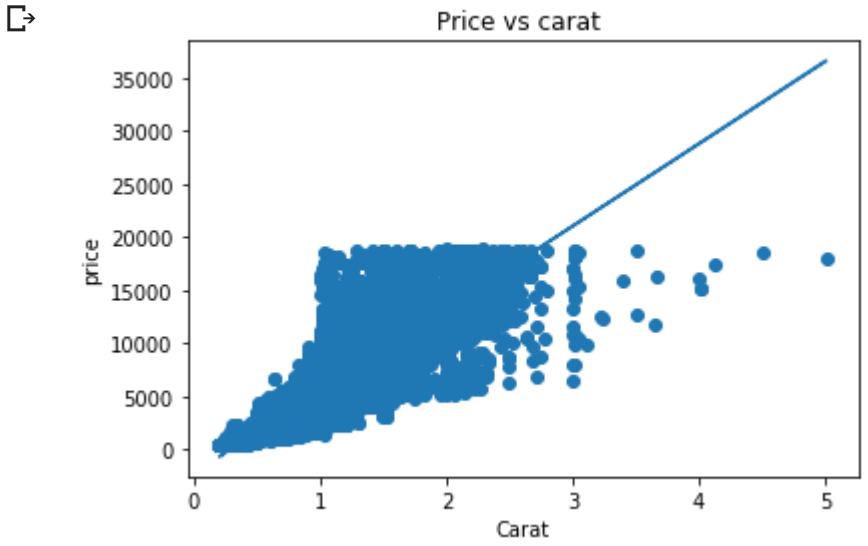
```
accuracy: 84.87506949799308%
Mean absolute error: 997.9939484312051
Mean squared error: 2372947.1027365774
R Squared: 0.8487506949799308
Adjusted R Squared: 0.8487506949799308
```

```
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
x = np.array(df.carat).reshape((-1,1))
y = np.array(df.price).reshape((-1,1))
model = LinearRegression().fit(x,y)
```

```
r_sq = model.score(x,y)
```

```
plt.scatter(x,y)
plt.title("Price vs carat ")
plt.ylabel('price')
```

```
plt.xlabel('Carat')
plt.plot(x,model.coef_*x+model.intercept_)
plt.show()
```



From the above plot one can observe the best fit line between the Price and the Carat. it is clearly seen that some data points doesn't fit the best fit line and the Co-efficient of Determination or R^2 is 0.84.

```
df['price1']= df['price']/1000
print(df['price1'])
```

B) Considering the other set of Feature like 'Length' and 'Price'.

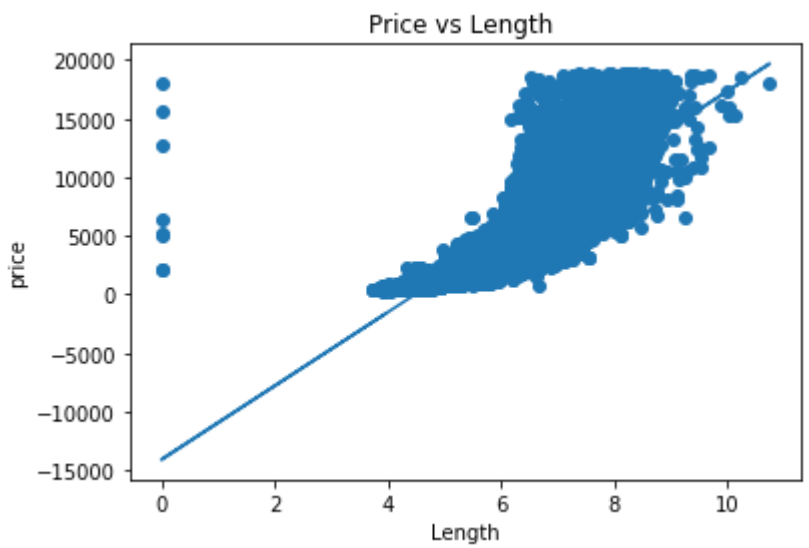
```
x = np.array(df.x).reshape((-1,1))
y = np.array(df.price).reshape((-1,1))
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(x, y,random_state = 2,test_size=0.3)
# imports
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn import linear_model
# Model initialization
regr = linear_model.LinearRegression()
# Fit the data(train the model)
```

```
regr.fit(train_x,train_y)
# Predict
y_pred = regr.predict(test_x)
print("accuracy: "+ str(regr.score(test_x,test_y)*100) + "%")
print("Mean absolute error: {}".format(mean_absolute_error(test_y,y_pred)))
print("Mean squared error: {}".format(mean_squared_error(test_y,y_pred)))
R2 = r2_score(test_y,y_pred)
print('R Squared: {}'.format(R2))
n=test_x.shape[0]
p=test_x.shape[1] - 1

adj_rsquared = 1 - (1 - R2) * ((n - 1)/(n-p-1))
print('Adjusted R Squared: {}'.format(adj_rsquared))
```

```
accuracy: 77.40844428477385%
Mean absolute error: 1364.97969767515
Mean squared error: 3544384.3311309475
R Squared: 0.7740844428477385
Adjusted R Squared: 0.7740844428477385
```

```
import numpy as np

from sklearn.linear_model import LinearRegression

import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures

x = np.array(df.x).reshape((-1,1))
y = np.array(df.price).reshape((-1,1))

model = LinearRegression().fit(x,y)
r_sq = model.score(x,y)

plt.scatter(x,y)

plt.title("Price vs Length ")

plt.ylabel('price')

plt.xlabel('Length')

plt.plot(x,model.coef_*x+model.intercept_)
```

```
plt.show()
```



In the above plot most of the scatter points distribution are between 4 to 9. Eventhough the data points are close most of the points are away from the best fit line.

Testing the Linear Regression Model:

```
### STATSMODELS ###
import statsmodels.formula.api as smf

# create a fitted model
lm1 = smf.ols(formula='price1 ~ carat', data=df).fit()

# print the coefficients
lm1.params
```

```
Intercept   -2.256425
carat        7.756522
dtype: float64
```

```
from sklearn.linear_model import LinearRegression
feature_cols = ['carat']
X = df[feature_cols]
```

```
y = df.price1

# instantiate and fit
lm2 = LinearRegression()
lm2.fit(X, y)

# print the coefficients
print(lm2.intercept_)
print(lm2.coef_)
```

```
-2.256424619038991
[7.75652228]
```

```
-2.25642+7.7565*25
```

```
191.65608
```

```
### STATSMODELS ###

# you have to create a DataFrame since the Statsmodels formula interface expects it
X_new = pd.DataFrame({'carat': [25]})

# predict for a new observation
lm1.predict(X_new)
```

```
0    191.656632
dtype: float64
```

```
lm2.predict(np.array([25]).reshape(-1,1))
```

```
array([191.65663231])
```

```
lm1.rsquared
```

```
0.8493323404162356
```

2.3. Elastic Net Regression

When working with high dimensional data, correlations between the variables are high which results in multicollinearity. If there is a situation that one variable need to be considered due to high probability of outcome but by eliminating other variables which are corelated there may be a chance of losing the information. In Lasso, it fails to perform the grouped selection as lasso selects one variable and ignore the other variables. Elastic Net performs the both L1 and L2 Regularization. Apart from that, alpha values can be tuned if alpha tends to zero then it is Ridge regression, if it tends to one it is lasso Regression.

Regularization: General regularization is adding the penalty term to the Loss Function.

Loss Function: The difference between the actual value and predicted value is considered as Loss Function

L1 Regularization: In L1 adds absolute value of magnitude of coefficient as penalty term

L2 Regularization: In L2 it adds squared magnitude of coefficients as penalty term to the loss function.

Elastic Net combines both L1 regularization penalty and L2 regularization penality.

Applying Elastic Net Regression model by supplying all the features

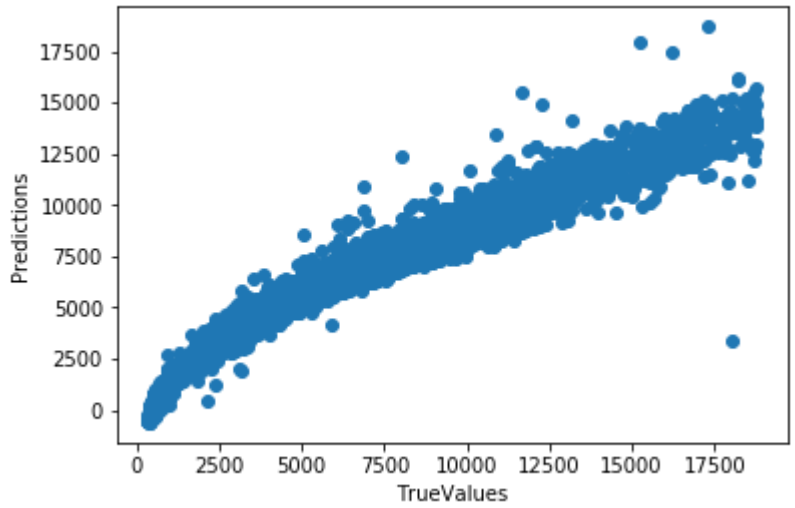
```
X = df.drop(['price'],1)
y = df['price']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

classifier = ElasticNet()
classifier.fit(X_train,y_train)

accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 5, scoring = 'r2',verbose = 1)

print('Elastic Net accuracy: ', classifier.score(X_test,y_test))

print(accuracies)

print("mean = {0}, std = {1}".format(np.mean(accuracies), np.std(accuracies)))
```

```
Elastic Net accuracy:  0.9315474708621928
[0.9341052  0.93134217 0.92571827 0.92804056 0.92811383]
mean = 0.9294640063262716, std = 0.002930799302409164
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.1s finished
```

Applying the Elastic Net Regression agaist Price vs Carat

```
x=np.array(df['carat']).reshape(-1,1)
y=np.array(df['price']).reshape((-1,1))
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


classifier = ElasticNet()
classifier.fit(X_train,y_train)
# Predict

y_pred = classifier.predict(X_test)

# For Cross Validation the whole data set is given with out splitting, it automatically splits the data.
#Here the number of cross vadilation folds are 5
# data is divided in to 5 folds and each time one fold is given for training.

accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 5, scoring = 'r2',verbose = 1)

print('Elastic Net accuracy: ', classifier.score(X_test,y_test))

print("Mean absolute error: {}".format(mean_absolute_error(y_test,y_pred)))

print("Mean squared error: {}".format(mean_squared_error(y_test,y_pred)))

print(accuracies)

print("mean = {0}, std = {1}".format(np.mean(accuracies), np.std(accuracies)))

plt.scatter(y_test, y_pred)

plt.xlabel('TrueValues')

plt.ylabel('Predictions')
```

```
Elastic Net accuracy:  0.9330257563897384
Mean absolute error: 686.8450109039009
Mean squared error: 1076154.4497949071
[0.93151481 0.92559615 0.93100306 0.93079782 0.92802327]
mean = 0.929387022316569, std = 0.002252107232565912
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.1s finished
Text(0, 0.5, 'Predictions')
```



In the above plot true values are plotted against predicted values from the algorithm. So if a best fit line is drawn from the above graph almost all the scatter points fall very near to the line reducing the mean squared error and producing the accurate R^2 value which is 0.93.

Metrics in Evaluating the Model

Mean Absolute Error: MAE is the absolute difference between the actual and forecasted values and average of the obtained difference [4].

Root Mean Squared Error: RMSE is the standard deviation of Residuals, Residual is defined as how far the data points are scattered from the

regression line. RMSE tells how these data points are scattered around the regression line.[5]

R^2 or Coefficient of determination: measure of fit that explains the variation of a dependent variable by independent variable. The strength of relationship changes

between the model and dependent variable. Larger the R^2 value better the regression model.

Model	Accuracy	MAE
Lasso Regression	84%	997.5
Linear Regression	90%	803.4
Elastic Net Regression	93%	686

From the above the highest accuracy is acheived by "Elastic Net Regression"

The best Model in terms of Mean Absolute Error is also "Elastic Net Regression"

It can be considered as the best Model as this model performs L3 regularization which will address the Correlation issues and Multi collinearity between the features.

As Elastic Net Regression is the best model let's Evaluate the model using Stats Model Library

Elastic Model Evaluation using Stats Library: Stats library contains advanced functions which are used for testing and modeling [8].

In this case I am cross validating the Elastic Net Regression model against stats library. This Library contains the OLS function which is Ordinary Least Square. ols is used in estimating the unknown parameters here the unknow parameter is Price that is for a given Carat value the price needs to be predicted.

```
### STATSMODELS ###
import statsmodels.formula.api as smf

# create a fitted model
lm1 = smf.ols(formula='price1 ~ carat', data=df).fit()

# print the coefficients
lm1.params
```

```
Intercept   -2.256425
carat        7.756522
dtype: float64
```

```
feature_cols = ['carat']
X = df[feature_cols]
y = df.price1

# instantiate and fit
lm2 =  ElasticNet()
lm2.fit(X, y)

# print the coefficients
print(lm2.intercept_)
print(lm2.coef_)
```

```
2.564392429979114
[1.71499166]
```

```
2.564392429979114+1.71499166*5
```

```
11.139350729979112
```

```
lm2.predict(np.array([5]).reshape(-1,1))
```

```
array([11.13935073])
```

Fictional Instance Description

1. Calculate the Intercept and coefficient of the given features.
2. Substitute the single carat value for which price needs to be predicted.
3. In the Elastic Net regression algorithm a single carat value is supplied.
4. From the both models the price predicted for the given carat value are same.
5. Hence the Elastic Net Model which is used in predicting the price gives greatest accuracy.

Finally the Model has the power to predict the accurate outcome for the future instances.

Interpolation: Picking a data point in between the actual points from the data set and checking the predicted value against the data set is known as interpolation.Here in this case a carat value which is equal to 0.21 is taken and fitted the Carat value in the best fit equation, it will come up with one price.Once the price is generated compare the value with the data set. As our model is not 100% accurate, the generated

price will be approximately equal to the dataset.

```
# For testing Interpolation particular carat value is selected from the dataset.
#instead of taking in between values here i am experimenting with the exact 'Carat' value
df.iloc[1]
```

```
carat        0.210
cut          2.000
color        2.000
clarity      6.000
depth       59.800
table       61.000
price      326.000
x            3.890
y            3.840
z            2.310
price1       0.326
Name: 1, dtype: float64
```

```
#Substituting the carat value from in the below equation with the above generated coefficient and intercept
(2.564392429979114+1.71499166*0.21)*100
```

```
292.45406785791135
```

```
# Predicted value from the Elastic Net Regression Model.
(lm2.predict(np.array([0.21]).reshape(-1,1)))*100
```

```
array([292.45406786])
```

```
# Difference between the Actual value and predicted value is
326-293
```

```
33
```

As our model is not 100% accurate the difference between the actual and the predicted value price is 33. The error rate is 33 this is becaues the model is only 93% accurate.Finally Elastic Net Regression Model is cross validated against the Stats Library and the model is tested by using Interpolation, The results produced are very acturate from both validation techniques.

Summary

The problem statement which I came up after Exploratory Data Analysis is Important features required to predict the price of the diamonds.
Initial Models that have been used in this Notebook are Lasso Regression, Linear Regression, Elastic Net Regression. The reason behind selecting the regression Models is because the final output is to predict the quantity which is Price.
Using Lasso Regression Feature selection, Important features in the data set can be identified. The important feature to Predict the Price of the diamond after applying the algorithm is Carat. Out of n number of features, this model can give the important features required in predicting the price. Finally, the accuracy of the algorithm is tested. The drawback of the algorithm is it eliminates the correlated features which may impact the outcome of the dataset.
In the Linear Regression Model, Both Simple Linear regression and Multiple Regression are tested. In the Simple Linear regression model is tested against the single feature, Here it is tested on both Carat and Length against the Price. It is difficult to test n number of features against Price and check accuracy. In Multiple Regression, this model is sensitive to outliers and all the features should be independent so the correlation problem cannot be addressed in this model. Correlation, in this case, is the depth of the diamond that has an impact on the carat and clarity. Multiple Regression cannot address this problem so this is not considered as the best model even though the accuracy good.
All the problems that are addressed in the above model are solved by the Elastic Net Regression Model. This model is very good at addressing the correlated issues because this model uses both L1 and L2 Regularization. This model has the capability of selecting the important features out of n number of features by addressing the correlation issue. Finally, of all the available features the most important feature in predicting the Price is ‘Carat’. The next priority goes to 'Lenght'.As the Carat weight increases, the price of the diamond also increases. This model is cross verified and tested against the stats library. For a given carat value the diamond price is the same for both models.

References

1. Medium. (2019). How to evaluate the performance of a machine learning model. [online] Available at: https://medium.com/datadriveninvestor/how-to-evaluate-the-performance-of-a-machine-learning-model-45063a7a38a7
2. ritchieng.github.io. (2019). Evaluating a Linear Regression Model. [online] Available at: https://www.ritchieng.com/machine-learning-evaluate-linear-regression-model/ [Accessed 16 Aug. 2019].
3. En.m.wikipedia.org. (2019). Lasso (statistics). [online] Available at: https://en.m.wikipedia.org/wiki/Lasso_(statistics) .
4. Acheron Analytics. (2019). How To Measure The Accuracy Of Predictive Models. [online] Available at: http://www.acheronanalytics.com/acheron-blog/how-to-measure-the-accuracy-of-predictive-models.

5. Statistics How To. (2019). RMSE: Root Mean Square Error - Statistics How To. [online] Available at: https://www.statisticshowto.datasciencecentral.com/rmse/.
6. Hackernoon.com. (2019). Practical machine learning: Ridge Regression vs. Lasso. [online] Available at: https://hackernoon.com/practical-machine-learning-ridge-regression-vs-lasso-a00326371ece.
7. Science, D. and know!, 7. (2019). 7 Regression Types and Techniques in Data Science. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/.
8. Mode Community. (2019). Statsmodels. [online] Available at: https://mode.com/python-tutorial/libraries/statsmodels/.