# Multiple Linear Regression

**Instructions**

Please share your answers filled in this word document. Submit code files wherever applicable.

Please ensure you update all the details:

**Name:Anandakrishnan k v**

**Batch ID:** 19042021

**Topic: Multilinear Regression**

**Grading Guidelines:**

**1. An assignment submission is considered complete only when correct and executable code(s) are submitted along with the documentation explaining the method and results. Failing to submit either of those will be considered an invalid submission and will not be considered for evaluation.**

**2. Assignments submitted after the deadline will affect your grades.**

**Grading:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ans** | **Date** |  |  | **Ans** | **Date** |
| Correct | On time | A | 100 |  |  |
| 80% & above | On time | B | 85 | Correct | Late |
| 50% & above | On time | C | 75 | 80% & above | Late |
| 50% & below | On time | D | 65 | 50% & above | Late |
|  |  | E | 55 | 50% & below |  |
| Copied/No Submission |  | F | 45 |  |  |

* **Grade A: (>= 90):** When all assignments are submitted on or before the given deadline.
* **Grade B: (>= 80 and < 90):** 
  + When assignments are submitted on time but less than 80% of problems are completed.

(OR)

* + All assignments are submitted after the deadline.
* **Grade C: (>= 70 and < 80):** 
  + When assignments are submitted on time but less than 50% of the problems are completed.

(OR)

* + Less than 80% of problems in the assignments are submitted after the deadline.
* **Grade D: (>= 60 and < 70):**
  + Assignments submitted after the deadline and with 50% or less problems.
* **Grade E: (>= 50 and < 60):** 
  + Less than 30% of problems in the assignments are submitted after the deadline.

(OR)

* + Less than 30% of problems in the assignments are submitted before the deadline.
* **Grade F: (< 50):** No submission (or) malpractice.

1. **Business Problem**
   1. **What is the business objective?**
   2. **Are there any constraints?**
2. **Work on each feature of the dataset to create a data dictionary as displayed in the below image:**



**2.1 Make a table as shown above and provide information about the features such as its data type and its relevance to the model building. And if not relevant, provide reasons and a description of the feature.**

1. **Data Pre-processing**

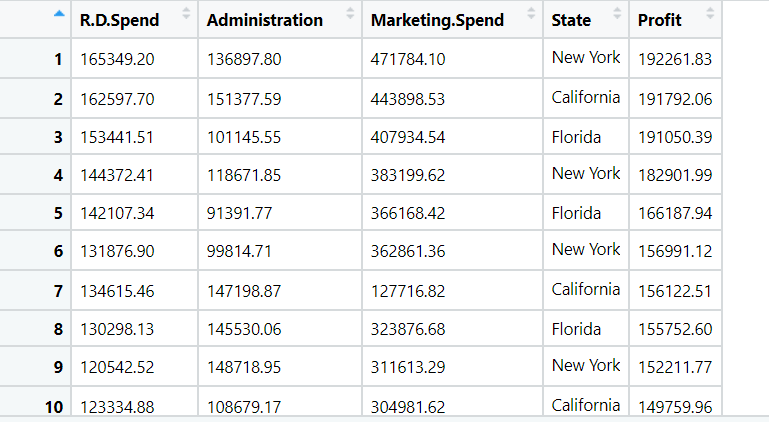
**3.1 Data Cleaning, Feature Engineering, etc.**

**3.2 Outlier Treatment.**

1. **Exploratory Data Analysis (EDA):**
   1. **Summary.**
   2. **Univariate analysis.**
   3. **Bivariate analysis.**
2. **Model Building**
   1. **Build the model on the scaled data (try multiple options).**
   2. **Perform Multi linear regression model and check for VIF, AvPlots, Influence Index Plots.**
   3. **Train and Test the data and compare RMSE values. Tabulate R-Squared and RMSE values for different models in the documentation and provide an explanation.**
   4. **Briefly explain the model output in the documentation.**
   5. **Tune the model and improve its accuracy.**
3. **Write about the benefits/impact of the solution - in what way does the business (client) benefit from the solution provided?**

**Problem Statements: -**

1. An analytics company has been tasked with the crucial job of finding out what factors affect a startup company and if it will be profitable or not. For this, they have collected some historical data and would like to apply multilinear regression to derive brief insights into their data. Predict profit, given different attributes for various startup companies.



**Solution:**

**What is the business objective?**

finding out what factors affect a startup company and Predict profit acccording to the availed data

**Are there any constraints?**

Minimize : Complexity of the model

Miniimize: Response time

Maximize : Accuracy of the model

**Python Code:**

**# Multilinear Regression**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import pandas as pd**

**import seaborn as sns**

**import os**

**import sklearn**

**# loading the data**

**df = pd.read\_csv("C://Users//user//Downloads//mlr//50\_Startups.csv")**

**###### Null value Treatment ########**

**df.isna().sum()**

**df.dropna(axis = 0, inplace = True) ## drop na values**

**#changing column names**

**df.columns**

**df.rename({'R&D Spend':'RD' ,'Marketing Spend':'Marketing'}, axis=1, inplace =True)**

**# Exploratory data analysis:**

**# 1. Measures of central tendency**

**# 2. Measures of dispersion**

**# 3. Third moment business decision**

**# 4. Fourth moment business decision**

**# 5. Probability distributions of variables**

**# 6. Graphical representations (Histogram, Box plot, Dot plot, Stem & Leaf plot, Bar plot, etc.)**

**df.describe()**

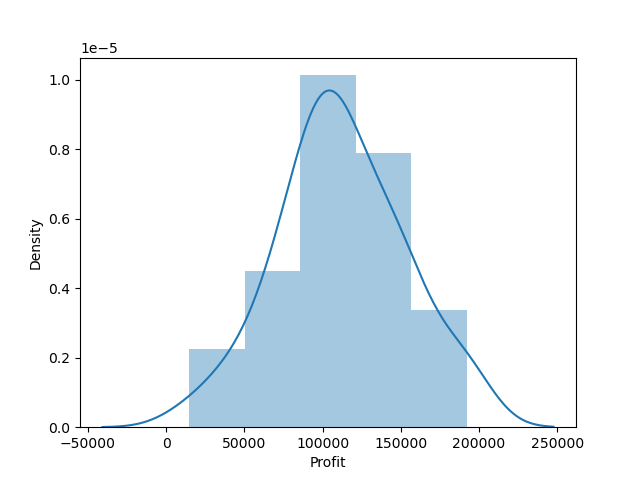
**#Graphical Representation**

**import matplotlib.pyplot as plt # mostly used for visualization purposes**

**## EDA on Dataset -**

**#Histgram on Profit**

**sns.distplot(df['Profit'],bins=5,kde=True)**

****

**# boxplot**

**#Check any outlier on features having numeric values**

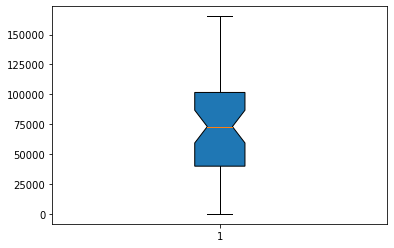
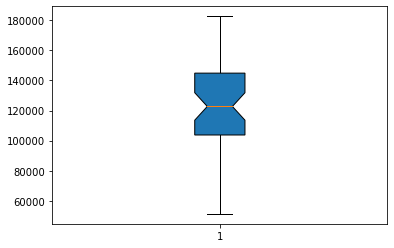
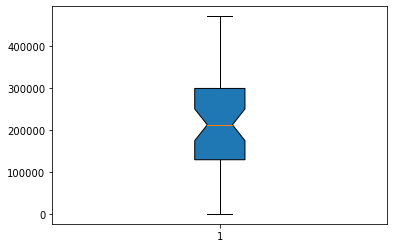
**import matplotlib.pyplot as plt**

**%matplotlib inline**

**for i in df.iloc[:,0:3]:**

**plt.boxplot(df[i],notch=True,patch\_artist=True)**

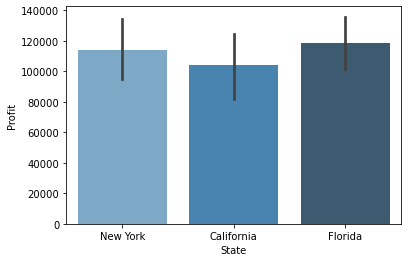
**plt.show()**

****

**# profit split in State level - Looks Florida has the maximum Profit**

**sns.barplot(x='State',y='Profit',data=df, palette="Blues\_d")**

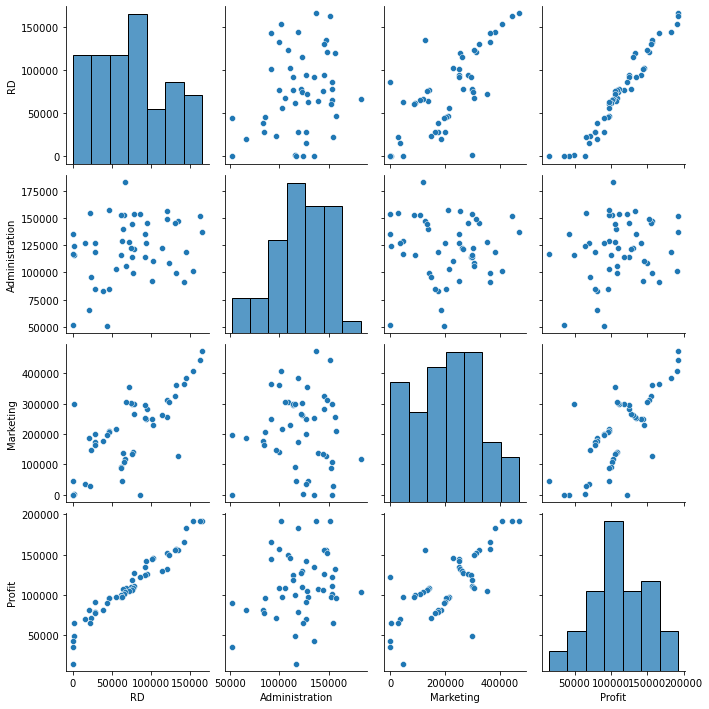
**#sns.lineplot(x='State',y='Profit',data=dataset)**

****

**# Scatter plot between the variables along with histograms**

**import seaborn as sns**

**sns.pairplot(df.iloc[:, :])**



**# Encoding categorical data**

**from sklearn.preprocessing import LabelEncoder, OneHotEncoder**

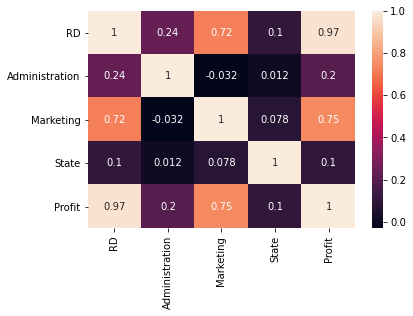
**labelencoder = LabelEncoder()**

**df["State"] = labelencoder.fit\_transform(df.iloc[:, 3])**

**# Correlation matrix**

**df.corr()**

**sns.heatmap(df.corr(), annot=True)**



**# we see there collinearity between input variables in comparitively high between**

**# [marketing & R&D] so there exists collinearity problem**

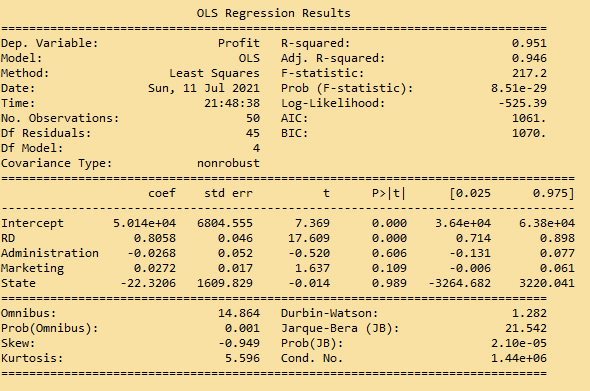
**# preparing model considering all the variables**

**import statsmodels.formula.api as smf # for regression model**

**ml1 = smf.ols('Profit ~ RD + Administration + Marketing + State', data = df).fit() # regression model**

**# Summary**

**ml1.summary()**

****

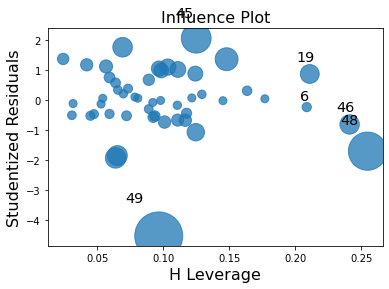
**# p-values for Administration, Marketing & State are more than 0.05**

**# Checking whether data has any influential values**

**# Influence Index Plots**

**import statsmodels.api as sm**

**sm.graphics.influence\_plot(ml1)**



**# Studentized Residuals = Residual/standard deviation of residuals**

**# index 49 is showing high influence so we can exclude that entire row**

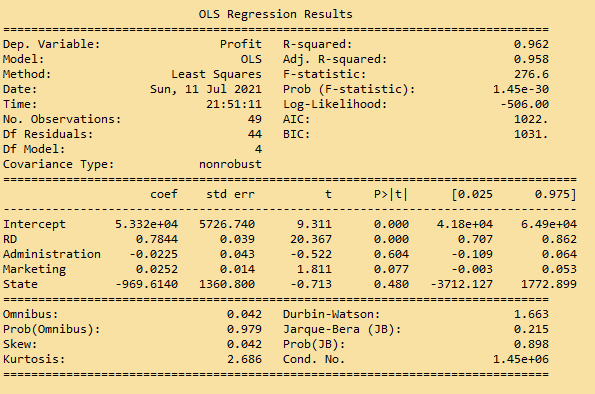
**df\_new = df.drop(df.index[[49]])**

**# Preparing model**

**ml\_new = smf.ols('Profit ~ RD + Administration + Marketing + State', data = df\_new).fit() # regression model**

**# Summary**

**ml\_new.summary()**



# Check for Colinearity to decide to remove a variable using VIF

# Assumption: VIF > 10 = colinearity

# calculating VIF's values of independent variables

rsq\_hp = smf.ols('RD ~ Administration + Marketing + State', data = df\_new).fit().rsquared

vif\_hp = 1/(1 - rsq\_hp)

rsq\_wt = smf.ols('Administration ~ RD + Marketing + State', data = df\_new).fit().rsquared

vif\_wt = 1/(1 - rsq\_wt)

rsq\_vol = smf.ols('Marketing ~ RD + Administration + State', data = df\_new).fit().rsquared

vif\_vol = 1/(1 - rsq\_vol)

rsq\_sp = smf.ols('State ~ RD + Administration + Marketing ', data = df\_new).fit().rsquared

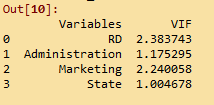
vif\_sp = 1/(1 - rsq\_sp)

# Storing vif values in a data frame

d1 = {'Variables':['RD', 'Administration', 'Marketing', 'State'], 'VIF':[vif\_hp, vif\_wt, vif\_vol, vif\_sp]}

Vif\_frame = pd.DataFrame(d1)

Vif\_frame



# another simple way of VIF calculation for whole data

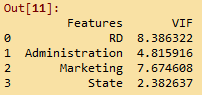
from statsmodels.stats.outliers\_influence import variance\_inflation\_factor as VIF

#Split Dataset into X and y

X=df.drop(columns='Profit')

y=df['Profit']

pd.DataFrame({'Features':X.columns,'VIF':[ VIF(X.values,i) for i in range(len(X.columns))]})

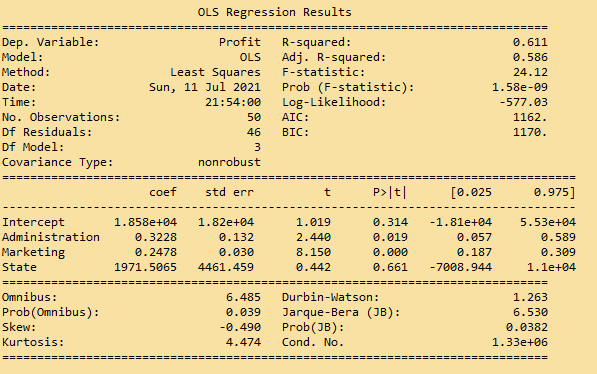


# As RD is having highest VIF value, we are going to drop this from the prediction model

# Final model

final\_ml = smf.ols('Profit ~ Administration + Marketing + State', data = df).fit()

final\_ml.summary()

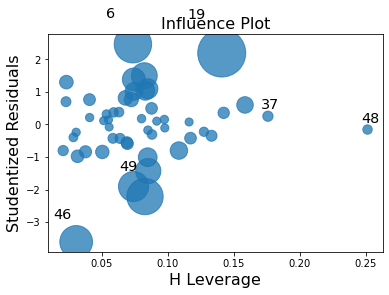


# p >0.05 for state column

sm.graphics.influence\_plot(final\_ml)

# Studentized Residuals = Residual/standard deviation of residuals

# index 19 is showing high influence so we can exclude that entire row

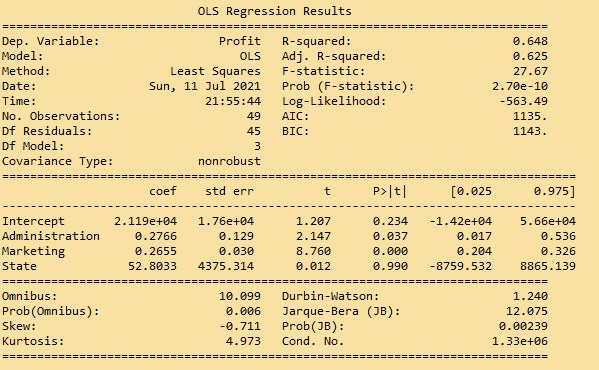


df\_new = df.drop(df.index[[19]])

# Final model

final\_ml = smf.ols('Profit ~ Administration + Marketing + State', data = df\_new).fit()

final\_ml.summary()



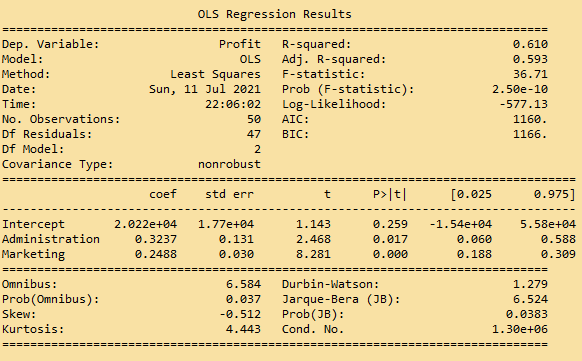
# p >0.05 for state column

# eliminate state column too

# Final model

final\_ml = smf.ols('Profit ~ Administration + Marketing ', data = df).fit()

final\_ml.summary()



# Prediction

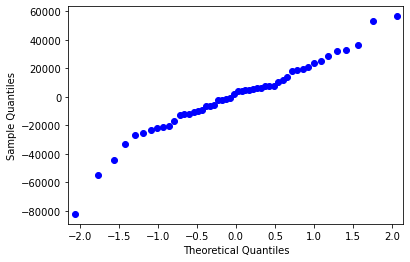
pred = final\_ml.predict(df)

# Q-Q plot

res = final\_ml.resid

sm.qqplot(res)

plt.show()



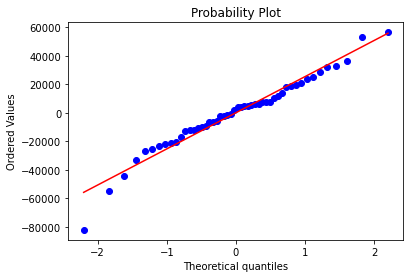
# Q-Q plot

from scipy import stats

from matplotlib import pylab

stats.probplot(res, dist = "norm", plot = pylab)

plt.show()



# Residuals vs Fitted plot

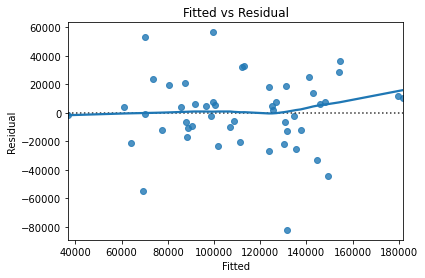
sns.residplot(x = pred, y = df.Profit, lowess = True)

plt.xlabel('Fitted')

plt.ylabel('Residual')

plt.title('Fitted vs Residual')

plt.show()



### Splitting the data into train and test data

from sklearn.model\_selection import train\_test\_split

df\_train, df\_test = train\_test\_split(df, test\_size = 0.2) # 20% test data

from sklearn.model\_selection import train\_test\_split

df\_train, df\_test = train\_test\_split(df, test\_size = 0.2) # 20% test data

# preparing the model on train data

model\_train = smf.ols("Profit ~ Administration + Marketing", data = df\_train).fit()

# prediction on test data set

test\_pred = model\_train.predict(df\_test)

# test residual values

test\_resid = test\_pred - df\_test.Profit

# RMSE value for test data

test\_rmse = np.sqrt(np.mean(test\_resid \* test\_resid))

test\_rmse

C:\Users\user\Documents\matplot\Untitled.png

# train\_data prediction

train\_pred = model\_train.predict(df\_train)

# train residual values

train\_resid = train\_pred - df\_train.Profit

# RMSE value for train data

train\_rmse = np.sqrt(np.mean(train\_resid \* train\_resid))

train\_rmse

C:\Users\user\Documents\matplot\Untitled.png

**Summary:**

* Plots explain the residuals thereby errors are identically normally, independatly & identically distributed
* The variance of errors are constant around the regression model line
* RMSE value is considerably low in comparisson with the range of output variable range
* So model showing better performance

1. Perform multilinear regression with price as the output variable and document the different RMSE values.



**Solution:**

**Are there any constraints?**

Minimize : Complexity of the model

Miniimize: Response time

Maximize : Accuracy of the model

**Python Code:**

**# Multilinear Regression**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import pandas as pd**

**import seaborn as sns**

**import os**

**import sklearn**

**# loading the data**

**df = pd.read\_csv("C://Users//user//Downloads//mlr//Computer\_Data.csv")**

**#changing column names**

**df.columns**

**df = df.iloc[:,1:] # eliminate unneccesary column**

**# Exploratory data analysis:**

**# 1. Measures of central tendency**

**# 2. Measures of dispersion**

**# 3. Third moment business decision**

**# 4. Fourth moment business decision**

**# 5. Probability distributions of variables**

**# 6. Graphical representations (Histogram, Box plot, Dot plot, Stem & Leaf plot, Bar plot, etc.)**

**###### Null value Treatment ########**

**df.isna().sum()**

**df.dropna(axis = 0, inplace = True) ## drop na values**

**# summary**

**df.columns**

**df.info()**

**df.describe()**

**# Encoding categorical data**

**#converting into numerical**

**from sklearn.preprocessing import LabelEncoder**

**lb = LabelEncoder()**

**df['cd'] = lb.fit\_transform(df['cd'])**

**df['multi'] = lb.fit\_transform(df['multi'])**

**df['premium'] = lb.fit\_transform(df['premium'])**

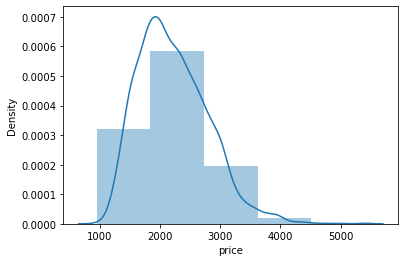
**#Graphical Representation**

**import matplotlib.pyplot as plt # mostly used for visualization purposes**

**## EDA on Dataset -**

**#Histgram on Profit**

**sns.distplot(df['price'],bins=5,kde=True)**



**# boxplot**

**#Check any outlier on features having numeric values**

**import matplotlib.pyplot as plt**

**%matplotlib inline**

**X= df[['speed', 'hd', 'ram', 'screen','ads', 'trend']]**

**for i in X.iloc[:,0:]:**

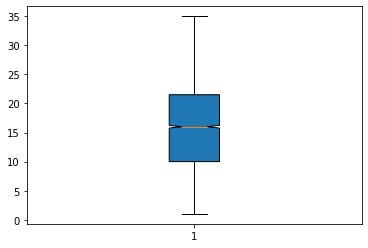
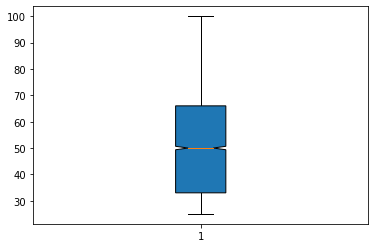
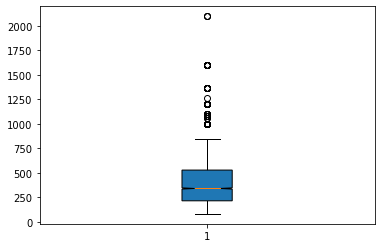
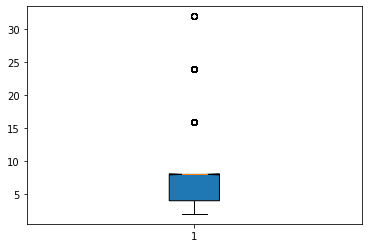
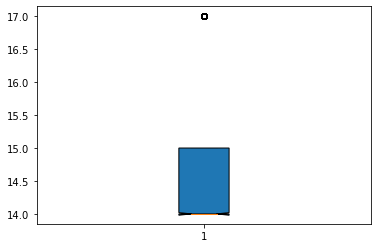
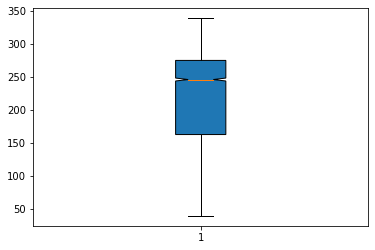
**plt.boxplot(df[i],notch=True,patch\_artist=True)**

**plt.show()**

**df.iloc[:,0:5].describe()**

**df.iloc[:,5:].describe()**

**# screen,ram,hd columns datas have outliers**

****

**#outlier treatment for df.hd**

**q1 = df['hd'].quantile(0.25)**

**q1**

**q3 = df['hd'].quantile(0.75)**

**q3**

**IQR = q3 - q1**

**IQR**

**lower\_limit = q1-(1.5\*IQR)**

**lower\_limit**

**upper\_limit = q3+(1.5\*IQR)**

**upper\_limit**

**#Replacing by pulling outliers to lower and upper limit**

**df['hd'] = np.where(df['hd']<q1, lower\_limit, np.where(df['hd']>q3, upper\_limit, df['hd']))**

**#outlier treatment for df.ram**

**q1 = df['ram'].quantile(0.25)**

**q1**

**q3 = df['ram'].quantile(0.75)**

**q3**

**IQR = q3 - q1**

**IQR**

**lower\_limit = q1-(1.5\*IQR)**

**lower\_limit**

**upper\_limit = q3+(1.5\*IQR)**

**upper\_limit**

**#Replacing by pulling outliers to lower and upper limit**

**df['ram'] = np.where(df['ram']<q1, lower\_limit, np.where(df['ram']>q3, upper\_limit, df['ram']))**

**#outlier treatment for df.screen**

**q1 = df['screen'].quantile(0.25)**

**q1**

**q3 = df['screen'].quantile(0.75)**

**q3**

**IQR = q3 - q1**

**IQR**

**lower\_limit = q1-(1.5\*IQR)**

**lower\_limit**

**upper\_limit = q3+(1.5\*IQR)**

**upper\_limit**

**#Replacing by pulling outliers to lower and upper limit**

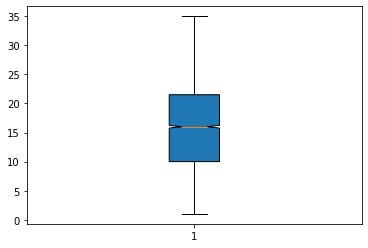
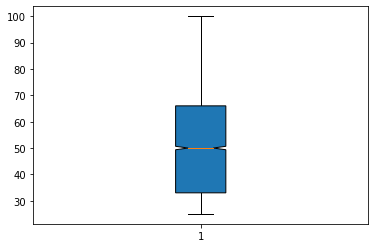
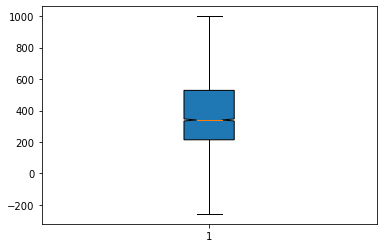
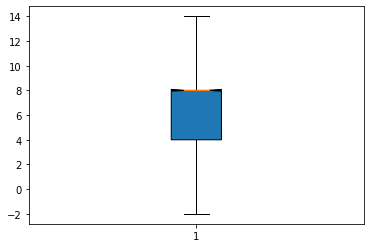
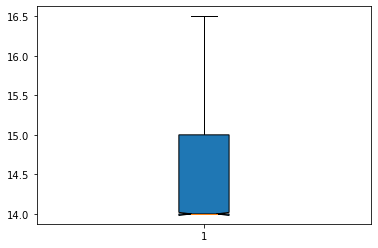
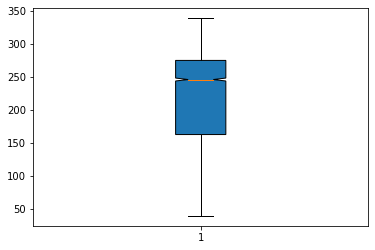
**df['screen'] = np.where(df['screen']<q1, lower\_limit, np.where(df['screen']>q3, upper\_limit,df['screen']))**

**# new boxplot after outlier treatment**

**for i in X.iloc[:,0:]:**

**plt.boxplot(df[i],notch=True,patch\_artist=True)**

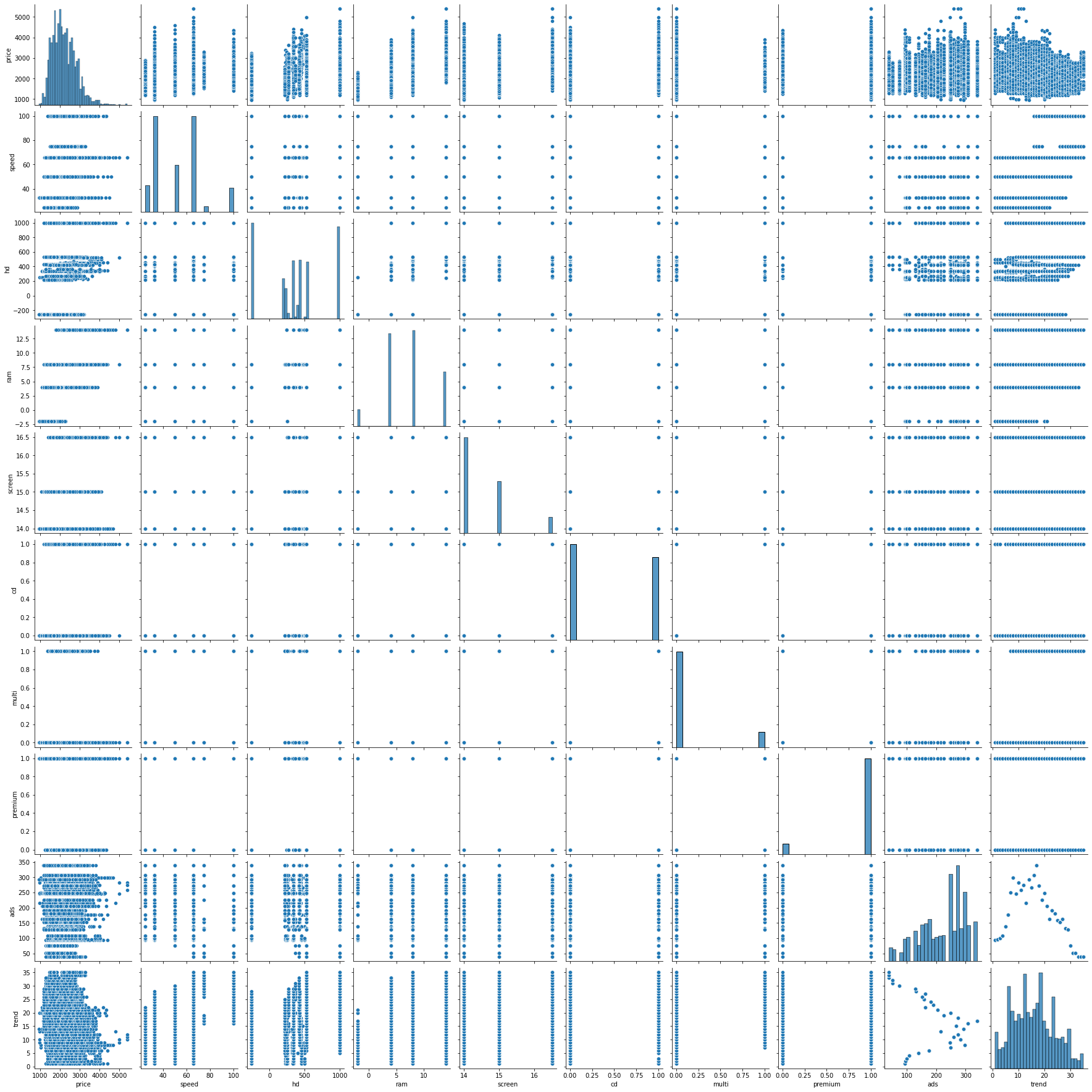
**plt.show()**

****

**# Scatter plot between the variables along with histograms**

**import seaborn as sns**

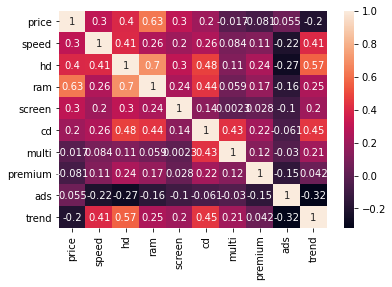
**sns.pairplot(df.iloc[:, :])**



**# Correlation matrix**

**df.corr()**

**sns.heatmap(df.corr(), annot=True)**



**# we see the collinearity between input variables are comparitively less**

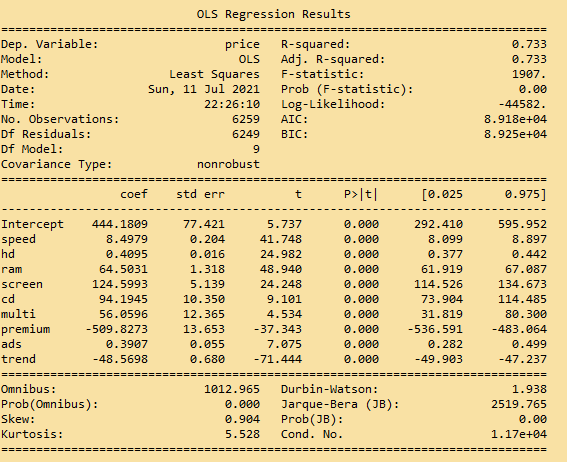
**# preparing model considering all the variables**

**import statsmodels.formula.api as smf # for regression model**

**final\_ml = smf.ols('price ~ speed + hd + ram + screen + cd + multi + premium + ads + trend', data = df).fit() # regression model**

**# Summary**

**final\_ml.summary()**

****

**# p-values < 0.05 for all the features**

**# Prediction**

**pred = final\_ml.predict(df)**

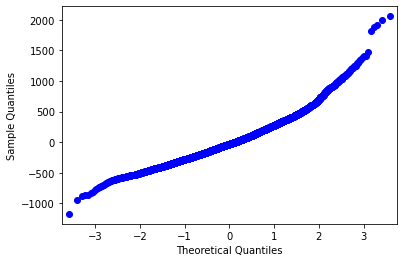
**# Q-Q plot**

**import statsmodels.api as sm**

**res = final\_ml.resid**

**sm.qqplot(res)**

**plt.show()**



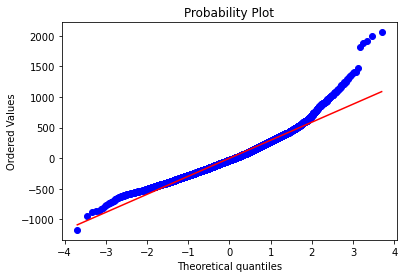
**# Q-Q plot**

**from scipy import stats**

**from matplotlib import pylab**

**stats.probplot(res, dist = "norm", plot = pylab)**

**plt.show()**



**# Residuals vs Fitted plot**

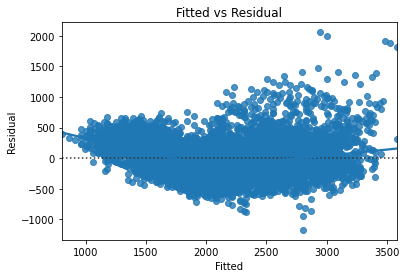
**sns.residplot(x = pred, y = df.price, lowess = True)**

**plt.xlabel('Fitted')**

**plt.ylabel('Residual')**

**plt.title('Fitted vs Residual')**

**plt.show()**



**### Splitting the data into train and test data**

**from sklearn.model\_selection import train\_test\_split**

**df\_train, df\_test = train\_test\_split(df, test\_size = 0.2) # 20% test data**

**# preparing the model on train data**

**model\_train = smf.ols("price ~ speed + hd + ram + screen + cd + multi + premium + ads + trend", data = df\_train).fit()**

**# prediction on test data set**

**test\_pred = model\_train.predict(df\_test)**

**# test residual values**

**test\_resid = test\_pred - df\_test.price**

**# RMSE value for test data**

**test\_rmse = np.sqrt(np.mean(test\_resid \* test\_resid))**

**test\_rmse**

**C:\Users\user\Documents\matplot\Untitled.png**

**# train\_data prediction**

**train\_pred = model\_train.predict(df\_train)**

**# train residual values**

**train\_resid = train\_pred - df\_train.price**

**# RMSE value for train data**

**train\_rmse = np.sqrt(np.mean(train\_resid \* train\_resid))**

**train\_rmse**

**C:\Users\user\Documents\matplot\Untitled.png**

**Summary:**

* Plots explain the residuals thereby errors are identically normally, independatly & identically distributed
* The variance of errors are constant around the regression model line
* RMSE value is considerably low in comparisson with the range of output variable range
* So model showing better performance

1. An online car sales platform would like to improve its customer base and their experience by providing them an easy way to buy and sell cars. For this, they would like an automated model which can predict the price of the car once the user inputs the required factors. Help the business achieve their objective by applying multilinear regression on the given dataset. Please use the below columns for the analysis purpose: price, age\_08\_04, KM, HP, cc, Doors, Gears, Quarterly\_Tax, and Weight.



**Solution:**

**What is the business objective?**

Builtan automated model which can predict the price of the car once the user inputs the required factors

**Are there any constraints?**

Minimize : Complexity of the model

Miniimize: Response time

Maximize : Accuracy of the model

**Python Code:**

**# Multilinear Regression**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import pandas as pd**

**import seaborn as sns**

**import os**

**import sklearn**

**# loading the data**

**df = pd.read\_csv("C://Users//user//Downloads//mlr//ToyotaCorolla.csv",encoding=('ISO-8859-1'),low\_memory=False)**

**#changing column names**

**df.columns**

**df = df[['Price', 'Age\_08\_04', 'KM', 'HP', 'cc', 'Doors', 'Gears', 'Quarterly\_Tax', 'Weight']] # eliminate unneccesary column**

**# Exploratory data analysis:**

**# 1. Measures of central tendency**

**# 2. Measures of dispersion**

**# 3. Third moment business decision**

**# 4. Fourth moment business decision**

**# 5. Probability distributions of variables**

**# 6. Graphical representations (Histogram, Box plot, Dot plot, Stem & Leaf plot, Bar plot, etc.)**

**###### Null value Treatment ########**

**df.isna().sum()**

**df.dropna(axis = 0, inplace = True) ## drop na values**

**# summary**

**df.columns**

**df.info()**

**df.describe()**

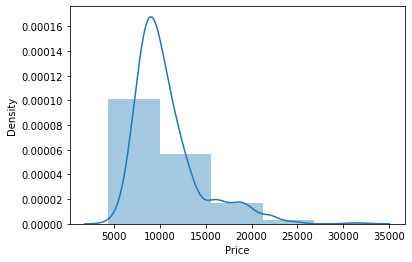
**#Graphical Representation**

**import matplotlib.pyplot as plt # mostly used for visualization purposes**

**## EDA on Dataset -**

**#Histgram on Profit**

**sns.distplot(df['Price'],bins=5,kde=True)**



**# boxplot**

**#Check any outlier on features having numeric values**

**import matplotlib.pyplot as plt**

**df.columns**

**%matplotlib inline**

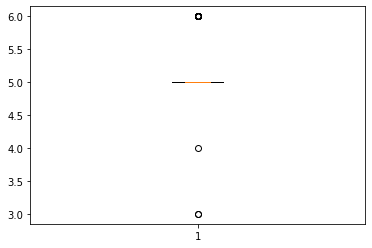
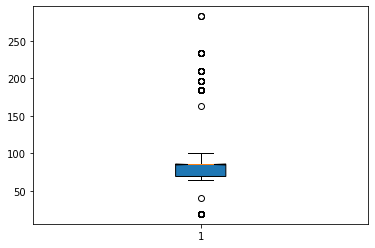
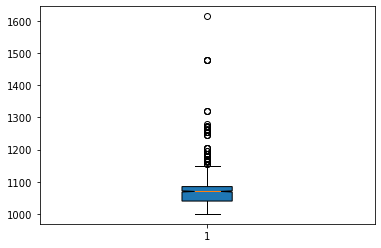
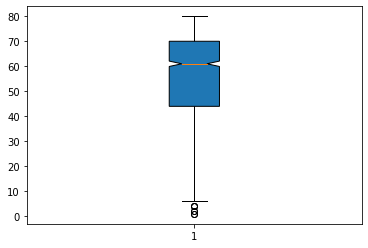
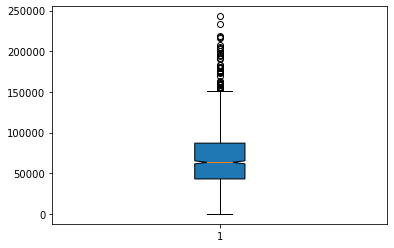
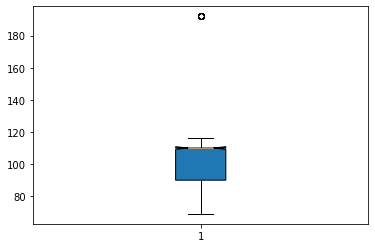
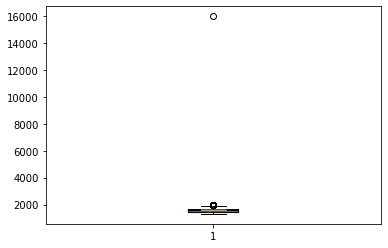
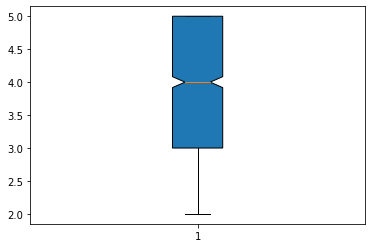
**for i in df.iloc[:,1:]:**

**plt.boxplot(df[i],notch=True,patch\_artist=True)**

**plt.show()**

**df.iloc[:,0:5].describe()**

**df.iloc[:,5:10].describe()**

****

**# 'Age\_08\_04', KM ,HP, CC, Gears, Quarterly\_Tax, Weight columns have outliers**

**####### outlier treatment ########**

**#outlier treatment for df.'Age\_08\_04'**

**q1 = df['Age\_08\_04'].quantile(0.25)**

**q1**

**q3 = df['Age\_08\_04'].quantile(0.75)**

**q3**

**IQR = q3 - q1**

**IQR**

**lower\_limit = q1-(1.5\*IQR)**

**lower\_limit**

**upper\_limit = q3+(1.5\*IQR)**

**upper\_limit**

**#Replacing by pulling outliers to lower and upper limit**

**df['Age\_08\_04'] = np.where(df['Age\_08\_04']<q1, lower\_limit, np.where(df['Age\_08\_04']>q3, upper\_limit, df['Age\_08\_04']))**

**#outlier treatment for df.KM**

**q1 = df['KM'].quantile(0.25)**

**q1**

**q3 = df['KM'].quantile(0.75)**

**q3**

**IQR = q3 - q1**

**IQR**

**lower\_limit = q1-(1.5\*IQR)**

**lower\_limit**

**upper\_limit = q3+(1.5\*IQR)**

**upper\_limit**

**#Replacing by pulling outliers to lower and upper limit**

**df['KM'] = np.where(df['KM']<q1, lower\_limit, np.where(df['KM']>q3, upper\_limit, df['KM']))**

**#outlier treatment for df.HP**

**q1 = df['HP'].quantile(0.25)**

**q1**

**q3 = df['HP'].quantile(0.75)**

**q3**

**IQR = q3 - q1**

**IQR**

**lower\_limit = q1-(1.5\*IQR)**

**lower\_limit**

**upper\_limit = q3+(1.5\*IQR)**

**upper\_limit**

**#Replacing by pulling outliers to lower and upper limit**

**df['HP'] = np.where(df['HP']<q1, lower\_limit, np.where(df['HP']>q3, upper\_limit,df['HP']))**

**#outlier treatment for df.cc**

**q1 = df['cc'].quantile(0.25)**

**q1**

**q3 = df['cc'].quantile(0.75)**

**q3**

**IQR = q3 - q1**

**IQR**

**lower\_limit = q1-(1.5\*IQR)**

**lower\_limit**

**upper\_limit = q3+(1.5\*IQR)**

**upper\_limit**

**#Replacing by pulling outliers to lower and upper limit**

**df['cc'] = np.where(df['cc']<q1, lower\_limit, np.where(df['cc']>q3, upper\_limit,df['cc']))**

**#outlier treatment for df.'Gears'**

**q1 = df['Gears'].quantile(0.25)**

**q1**

**q3 = df['Gears'].quantile(0.75)**

**q3**

**IQR = q3 - q1**

**IQR**

**lower\_limit = q1-(1.5\*IQR)**

**lower\_limit**

**upper\_limit = q3+(1.5\*IQR)**

**upper\_limit**

**#Replacing by pulling outliers to lower and upper limit**

**df['Gears'] = np.where(df['Gears']<q1, lower\_limit, np.where(df['Gears']>q3, upper\_limit, df['Gears']))**

**#outlier treatment for df.'Quarterly\_Tax'**

**q1 = df['Quarterly\_Tax'].quantile(0.25)**

**q1**

**q3 = df['Quarterly\_Tax'].quantile(0.75)**

**q3**

**IQR = q3 - q1**

**IQR**

**lower\_limit = q1-(1.5\*IQR)**

**lower\_limit**

**upper\_limit = q3+(1.5\*IQR)**

**upper\_limit**

**#Replacing by pulling outliers to lower and upper limit**

**df['Quarterly\_Tax'] = np.where(df['Quarterly\_Tax']<q1, lower\_limit, np.where(df['Quarterly\_Tax']>q3, upper\_limit, df['Quarterly\_Tax']))**

**#outlier treatment for df.'Weight'**

**q1 = df['Weight'].quantile(0.25)**

**q1**

**q3 = df['Weight'].quantile(0.75)**

**q3**

**IQR = q3 - q1**

**IQR**

**lower\_limit = q1-(1.5\*IQR)**

**lower\_limit**

**upper\_limit = q3+(1.5\*IQR)**

**upper\_limit**

**#Replacing by pulling outliers to lower and upper limit**

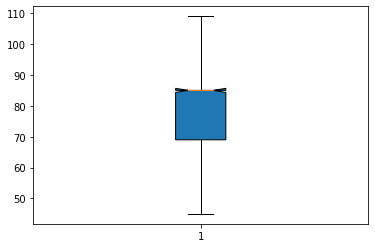
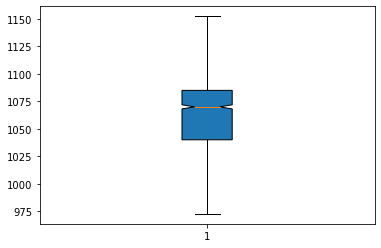
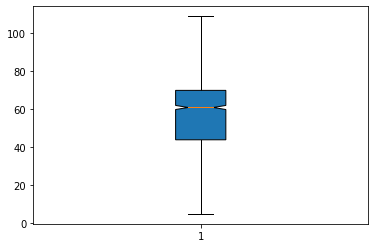
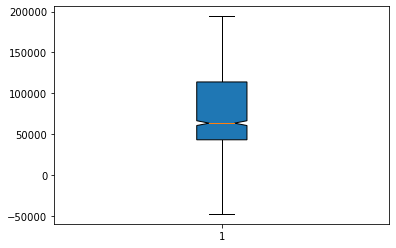
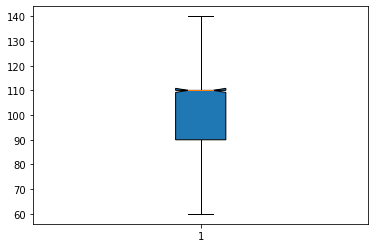
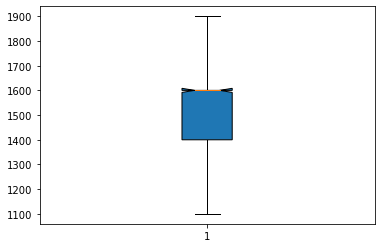
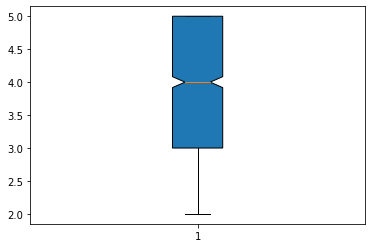
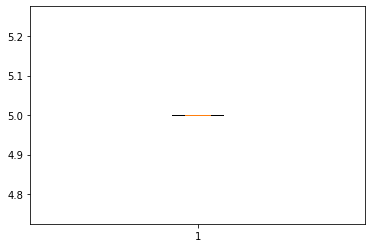
**df['Weight'] = np.where(df['Weight']<q1, lower\_limit, np.where(df['Weight']>q3, upper\_limit, df['Weight']))**

**# new boxplot after outlier treatment**

**for i in df.iloc[:,1:]:**

**plt.boxplot(df[i],notch=True,patch\_artist=True)**

**plt.show()**

****

**# unique value count**

**df["Gears"].value\_counts()**

**# droping Gears column since it have zero variance**

**df.drop("Gears", axis=1, inplace = True)**

**###### zero variance operation ###**

**df.shape**

**## importing ###**

**from sklearn.feature\_selection import VarianceThreshold**

**# Feature selector that removes all low-variance features that meets the variance threshold limit**

**var\_thres = VarianceThreshold(threshold=0.2) # Threshold is subjective.**

**var\_thres.fit(df) ### fit the var\_thres to data set df**

**# Generally we remove the columns with zero variance, but i took thresold value 0.2 (Near Zero Variance)**

**var\_thres.get\_support() ### it giving an array out, where zero variant column treat as False value. we already fit var\_thres to df1. so it gives corresponding information on df1**

**df.columns[var\_thres.get\_support()] ## non-zero variant column names**

**constant\_columns = [column for column in df.columns if column not in df.columns[var\_thres.get\_support()]]**

**print(len(constant\_columns)) ### number of zero variant variables**

**for feature in constant\_columns:**

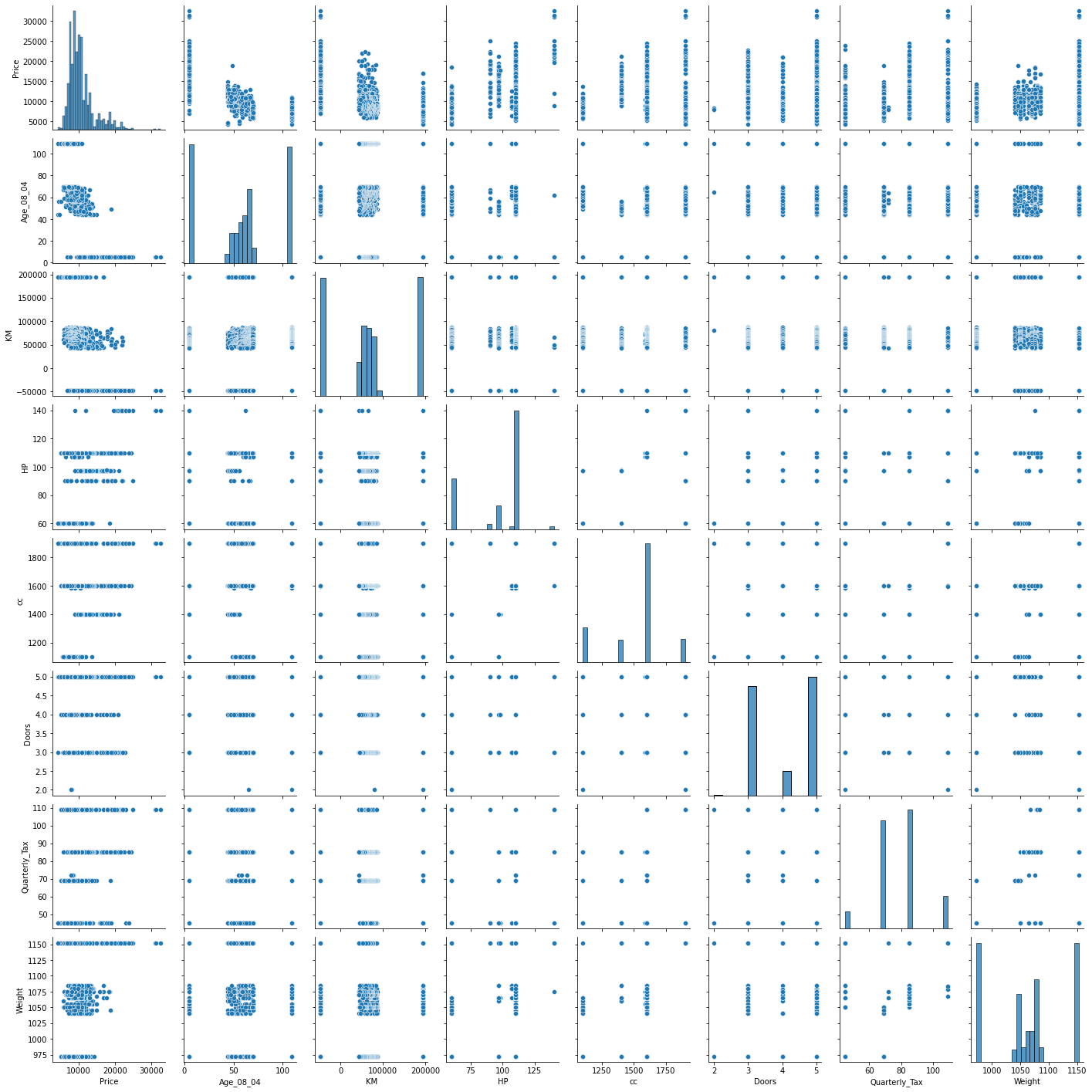
**print(feature) ### names of corresponding zero variant columns : no zero variant columns**

**# no zero variant columns**

**# Scatter plot between the variables along with histograms**

**import seaborn as sns**

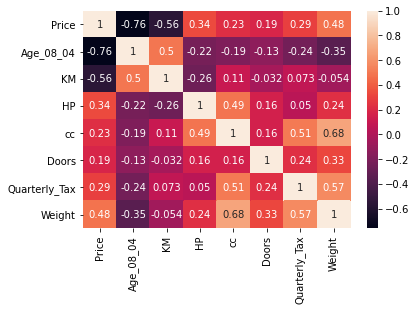
**sns.pairplot(df.iloc[:, :])**



**# Correlation matrix**

**df.corr()**

**sns.heatmap(df.corr(), annot=True)**



**# we see the collinearity between input variables are comparitively less**

**# preparing model considering all the variables**

**import statsmodels.formula.api as smf # for regression model**

**ml1 = smf.ols('Price ~ Age\_08\_04 + KM + HP + cc + Doors + Quarterly\_Tax + Weight', data = df).fit() # regression model**

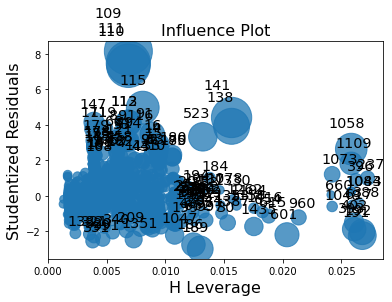
**# p\_vaue of Doors column is > 0.05**

**# Checking whether data has any influential values**

**# Influence Index Plots**

**import statsmodels.api as sm**

**sm.graphics.influence\_plot(ml1)**



**# Studentized Residuals = Residual/standard deviation of residuals**

**# index 109 is showing high influence so we can exclude that entire row**

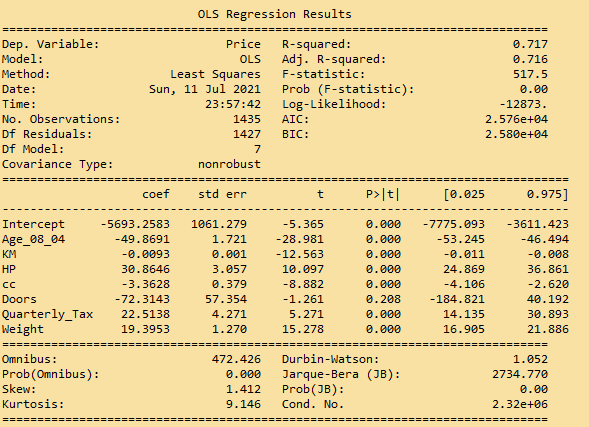
**df\_new = df.drop(df.index[[109]])**

**# Preparing model**

**ml\_new = smf.ols('Price ~ Age\_08\_04 + KM + HP + cc + Doors + Quarterly\_Tax + Weight', data = df\_new).fit() # regression model**

**# Summary**

**ml\_new.summary()**

****

**# Check for Colinearity to decide to remove a variable using VIF**

**# Assumption: VIF > 10 = colinearity**

**# calculating VIF's values of independent variables**

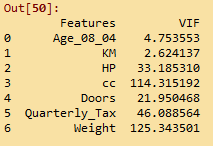
**from statsmodels.stats.outliers\_influence import variance\_inflation\_factor as VIF**

**#Split Dataset into X and y**

**X=df.drop(columns='Price')**

**y=df['Price']**

**pd.DataFrame({'Features':X.columns,'VIF':[ VIF(X.values,i) for i in range(len(X.columns))]})**

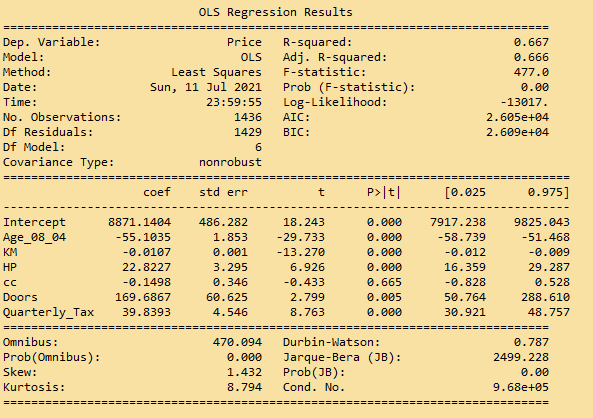
****

**# As Weight is having highest VIF value, we are going to drop this from the prediction model**

**# new model**

**ml\_nw = smf.ols('Price ~ Age\_08\_04 + KM + HP + cc + Doors + Quarterly\_Tax', data = df).fit()**

**ml\_nw .summary()**

****

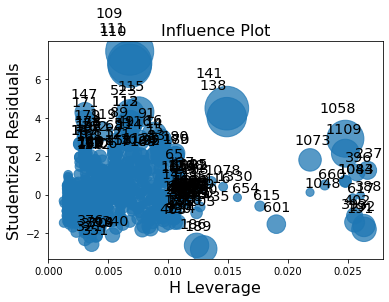
**# p >0.05 for cc column**

**# Checking whether data has any influential values**

**# Influence Index Plots**

**import statsmodels.api as sm**

**sm.graphics.influence\_plot(ml\_nw )**



**# Studentized Residuals = Residual/standard deviation of residuals**

**# index 109,111 is showing high influence so we can exclude that entire row**

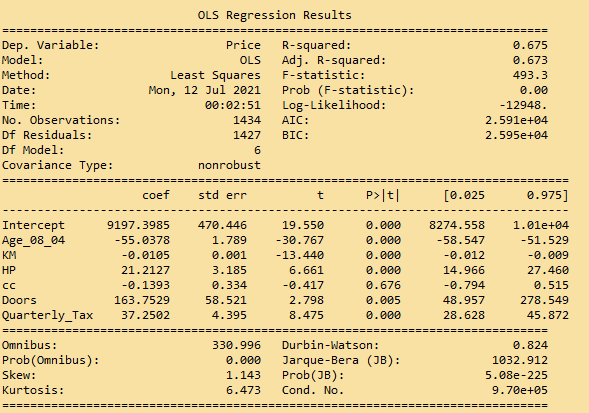
**df\_new = df.drop(df.index[[109,111]])**

**# Preparing model**

**ml\_nw1 = smf.ols('Price ~ Age\_08\_04 + KM + HP + cc + Doors + Quarterly\_Tax', data = df\_new).fit() # regression model**

**# Summary**

**ml\_nw1.summary()**

****

**# p\_value is still high for cc**

**from statsmodels.stats.outliers\_influence import variance\_inflation\_factor as VIF**

**#Split Dataset into X and y**

**X=df.drop(columns='Price')**

**X1= X.drop(columns='Weight')**

**y=df['Price']**

**pd.DataFrame({'Features':X1.columns,'VIF':[ VIF(X1.values,i) for i in range(len(X1.columns))]})**

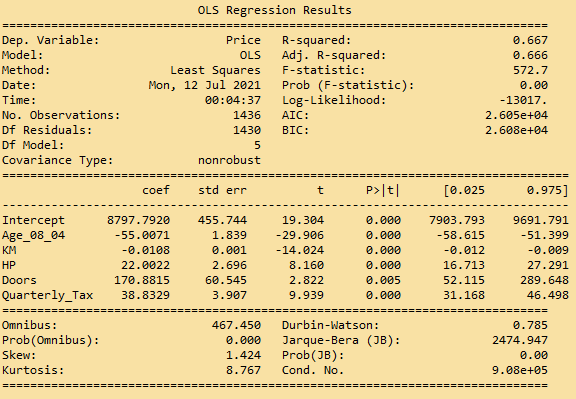
**# As cc is having highest VIF value, we are going to drop this from the prediction model**

**# new model**

**final\_ml = smf.ols('Price ~ Age\_08\_04 + KM + HP + Doors + Quarterly\_Tax', data = df).fit()**

**# Summary**

**final\_ml.summary()**

****

**# p-values < 0.05 for all the features**

**# Prediction**

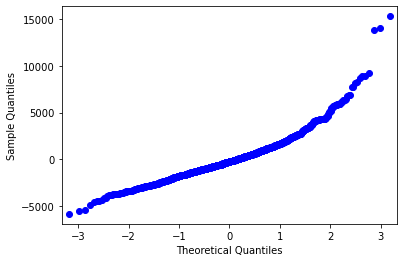
**pred = final\_ml.predict(df)**

**# Q-Q plot**

**res = final\_ml.resid**

**sm.qqplot(res)**

**plt.show()**



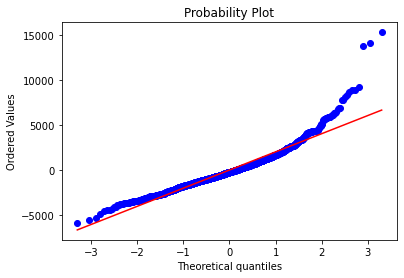
**# Q-Q plot**

**from scipy import stats**

**from matplotlib import pylab**

**stats.probplot(res, dist = "norm", plot = pylab)**

**plt.show()**



**# Residuals vs Fitted plot**

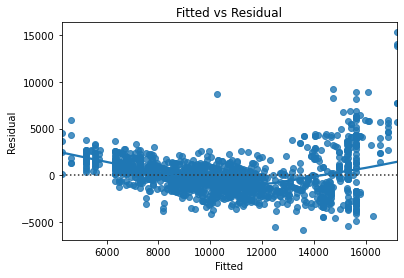
**sns.residplot(x = pred, y = df.Price, lowess = True)**

**plt.xlabel('Fitted')**

**plt.ylabel('Residual')**

**plt.title('Fitted vs Residual')**

**plt.show()**



**### Splitting the data into train and test data**

**from sklearn.model\_selection import train\_test\_split**

**df\_train, df\_test = train\_test\_split(df, test\_size = 0.2) # 20% test data**

**# preparing the model on train data**

**model\_train = smf.ols("Price ~ Age\_08\_04 + KM + HP + Doors + Quarterly\_Tax", data = df\_train).fit()**

**# prediction on test data set**

**test\_pred = model\_train.predict(df\_test)**

**# test residual values**

**test\_resid = test\_pred - df\_test.Price**

**# RMSE value for test data**

**test\_rmse = np.sqrt(np.mean(test\_resid \* test\_resid))**

**test\_rmse**

**C:\Users\user\Documents\matplot\Untitled.png**

**# train\_data prediction**

**train\_pred = model\_train.predict(df\_train)**

**# train residual values**

**train\_resid = train\_pred - df\_train.Price**

**# RMSE value for train data**

**train\_rmse = np.sqrt(np.mean(train\_resid \* train\_resid))**

**train\_rmse**

**C:\Users\user\Documents\matplot\Untitled.png**

**Summary:**

* Plots explain the residuals thereby errors are identically normally, independatly & identically distributed
* The variance of errors are constant around the regression model line
* RMSE value is considerably low in comparisson with the range of output variable range
* So model showing better performance