About Dataset

50 Startups' expenditures & profits

Aim of the project : We have to analysis the data expenditures vs Profit and ML prediction

Columns name :

1) R&D Spend 2) Administration 3) Marketing Spend 4) State 5) Profit

```
#Normal Library
#----------------------
import numpy as np
import pandas as pd


#Visulization Library
#--------------------
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()

#-------------------------------
#Warning library
import warnings
warnings.filterwarnings('ignore')


#anova testing
import scipy.stats as stats

#scaling the data
from sklearn.preprocessing import StandardScaler

#Train Test split
from sklearn.model_selection import train_test_split

#moving dependent variale in last column.
import movecolumn as mc
#ML model - regeression
from sklearn.linear_model import LinearRegression

#ML model - regeression - Performance matrix
from sklearn import metrics
from sklearn.metrics import r2_score,mean_absolute_percentage_error, mean_squared_error



#ML model - Classification

#ML model - Classification - Performance matrix
from sklearn import metrics
```

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score


#Improve the accuracy
from sklearn.model_selection import cross_val_score
```

Importing Dataset

Double-click (or enter) to edit

```
df = pd.read_csv(r"/content/50_Startups.csv")
```

```
df.head(10)
```

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94
5	131876.90	99814.71	362861.36	New York	156991.12
6	134615.46	147198.87	127716.82	California	156122.51
7	130298.13	145530.06	323876.68	Florida	155752.60
8	120542.52	148718.95	311613.29	New York	152211.77
9	123334.88	108679.17	304981.62	California	149759.96

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   R&D Spend        50 non-null     float64
 1   Administration   50 non-null     float64
 2   Marketing Spend  50 non-null     float64
 3   State            50 non-null     object
 4   Profit           50 non-null     float64
dtypes: float64(4), object(1)
memory usage: 2.1+ KB
```

Removing Duplicate Rows

```
def drop_dup(df):
    if df.duplicated().any() == True:
        df.drop_duplicates( inplace = True, Keep = "Last",reset_index = True)

        print("data after removig duplicate rows",df.duplicated().sum())
    else:
        return "No action required(No duplicate rows)"

drop_dup(df)
```

```
'No action required(No duplicate rows)'
```

Checking Null Values

```
print(df.isnull().sum())
print("****************************************************")
print(df.isnull().sum()/len(df)*100)
```

```
R&D Spend          0
Administration     0
Marketing Spend    0
State              0
Profit             0
dtype: int64
```

```
**************************************************
R&D Spend          0.0
Administration     0.0
Marketing Spend    0.0
State              0.0
Profit             0.0
dtype: float64
```

Check unique counts

```
def check_unique_count(df):
    unique_counts = df.nunique()
    print(unique_counts)
```

```
check_unique_count(df)
```

```
R&D Spend          49
Administration     50
Marketing Spend    48
State               3
Profit             50
dtype: int64
```

Check unique counts data entry in columns

```
for i in df.columns:
    print(i)
    print()
    print(set(df[i].tolist()))
    print()
```

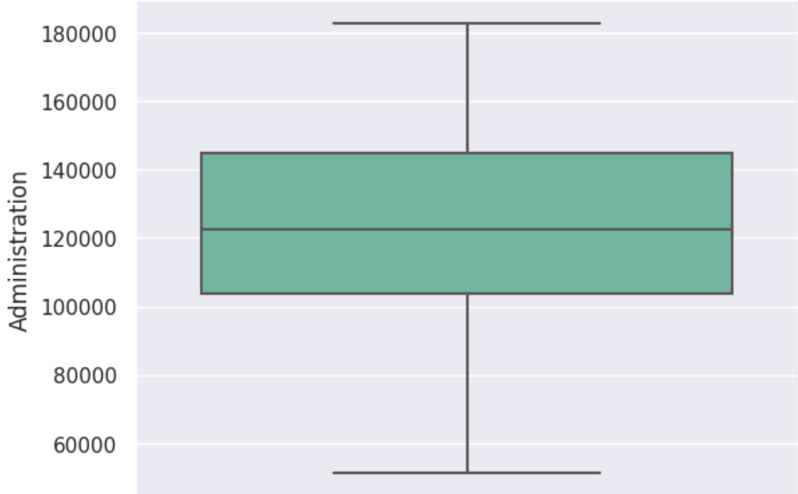
```
R&D Spend

{0.0, 66051.52, 20229.59, 119943.24, 73994.56, 15505.73, 86419.7, 64664.71, 101913.08, 14:

Administration

{110594.11, 156547.42, 135426.92, 108679.17, 144135.98, 91790.61, 118671.85, 103057.49, 1:

Marketing Spend

{256512.92, 0.0, 304768.73, 107138.38, 118148.2, 46085.25, 294919.57, 249744.55, 261776.2:

State

{'New York', 'California', 'Florida'}

Profit

{144259.4, 192261.83, 105733.54, 108552.04, 96778.92, 107404.34, 101004.64, 126992.93, 14:
```

Outliers Check -

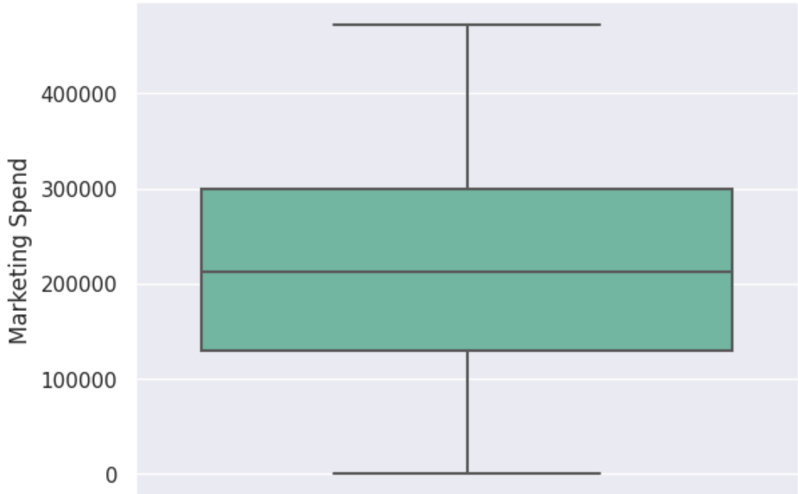
No outlier found as below boxplot analysis

```
#  Column(A) - R&D Spend - No Outlier found

sns.boxplot(y = "R&D Spend", data = df, palette ='Set2' )
plt.show()
```


```
# Column(B) - Administration - No Outlier found

sns.boxplot(y = "Administration", data = df, palette ='Set2' )
plt.show()
```



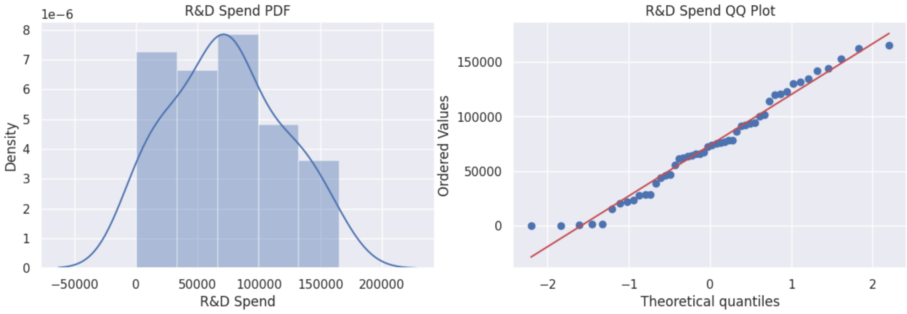

```
#Column(C) - Marketing Spend - No Outlier found

sns.boxplot(y = "Marketing Spend", data = df, palette ='Set2' )
plt.show()
```



Normal Distribution check -

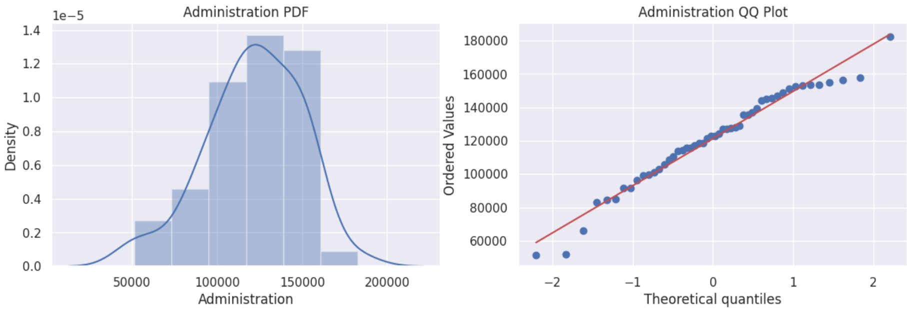
Analysis result... Skewness is close to 0, no action required.

```
# column(A) - R&D Spend

plt.figure(figsize=(14,4))
plt.subplot(121)
sns.distplot(df["R&D Spend"])
plt.title('R&D Spend PDF')



plt.subplot(122)
stats.probplot(df["R&D Spend"], dist='norm', plot=plt)
plt.title("R&D Spend QQ Plot")
plt.show()

#**************************************************
print("skewness of the column of area income :  ",df["R&D Spend"].skew())
#*************************************
```



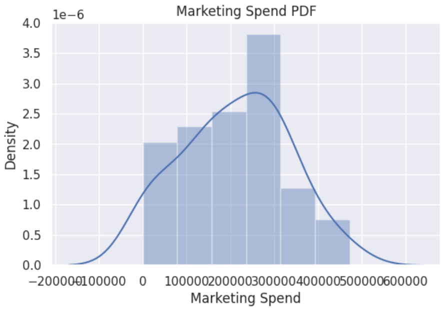

```
skewness of the column of area income :   0.164002172321177
```

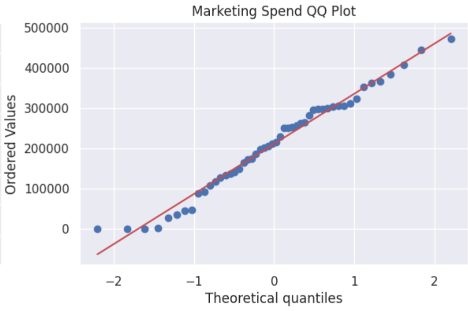
```
#column(B) -Administration

plt.figure(figsize=(14,4))
plt.subplot(121)
sns.distplot(df["Administration"])
plt.title('Administration PDF')



plt.subplot(122)
stats.probplot(df["Administration"], dist='norm', plot=plt)
plt.title("Administration QQ Plot")
plt.show()

#**************************************************
print("skewness of the column of Administration :  ",df["Administration"].skew())
#*************************************
```




```
skewness of the column of Administration :   -0.4890248099671768
```

```
#column(C) - Marketing Spend

plt.figure(figsize=(14,4))
plt.subplot(121)
sns.distplot(df["Marketing Spend"])
plt.title('Marketing Spend PDF')



plt.subplot(122)
stats.probplot(df["Marketing Spend"], dist='norm', plot=plt)
plt.title("Marketing Spend QQ Plot")
plt.show()

#**************************************************
print("skewness of the column of Marketing Spend :  ",df["Marketing Spend"].skew())
#*************************************
```

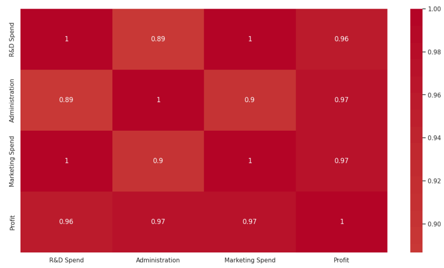





```
skewness of the column of Marketing Spend :   -0.04647226758360412
```

Finding correlation

```
plt.figure(figsize=(15,8))
corr = df.describe().corr()
sns.heatmap(corr, annot=True, cmap='coolwarm',center = 0)
plt.show()
```



Visulization Part -A

Profit state wise

```
sns.barplot(x="State", y= "Profit", data = df)
plt.xlabel("State")
plt.ylabel("Profit")
plt.title("Sum of profit vs state")
plt.show()
```

Visulization part - B

Analyse for best model through Pair plot

```
sns.pairplot(df, size = 5, kind = 'scatter')
plt.show()
```

it is given picture to judge,which model is best for this dataset

Cheat sheat

aisoma.de

Top Machine Learning Algorithms for Predictions

Name	Type	Description	Advantages	Disadvantages
Linear Regression		-The best fit line through all data points	-Easy to understand -you can clearly see what the biggest drivers odf the model are.	-sometimes to simple to capture cpmöex relationships between variables, -Tendency für the model to overfit.
Logistic Regression		-The adoption for linear regression to problembs of classification	-Easy to understand	-sometimes to simple to capture cpmöex relationships between variables, -Tendency für the model to overfit.
Decision Tree		-A graph that uses branching method to match all possible outcomes of a decision	-Easy to understand and implement.	-Not often use of ist own for prediction because it's also often too simple and not powerful enough for complex data.
Random Forest		- Takes the average of many decision trees. Each tree is weaker than the full decision tree, but combining them we get better overall performance.	-A sort of „wisdom of the crowd“, Tend to result in very high quality results. -Fast to train	-Can be slow to output predictions relative to other algorithms. -Not easy to understand predictions.
Gradient Boosting		-Uses even weaker decision trees that increasingly focused on „hard examples“	-High-performing	-A small change in the future set or training set can create radical changes in the model. -Not easy to understand predictions.
Neural Networks		-Mimics the behaviour of the brain. NNs are interconnected Neurons that pass messages to each other. Deep Learning uses severak layers of NNs to put one after the other.	-Can handle extremely complex tasks. No other alsgorithm comes close in image recognition.	-very very slow to train. Because they have so many layers. Require a lot of power. -Almost impossible to understand predictions.

One hot Encoding concept

```
df["State"].value_counts()
```

```
New York      17
California    17
Florida       16
Name: State, dtype: int64
```

```
df = pd.get_dummies(df, columns = ["State"],drop_first = True)
```

```
df
```

	R&D Spend	Administration	Marketing Spend	Profit	State_Florida	State_New York
0	165349.20	136897.80	471784.10	192261.83	0	1
1	162597.70	151377.59	443898.53	191792.06	0	0
2	153441.51	101145.55	407934.54	191050.39	1	0
3	144372.41	118671.85	383199.62	182901.99	0	1
4	142107.34	91391.77	366168.42	166187.94	1	0
5	131876.90	99814.71	362861.36	156991.12	0	1
6	134615.46	147198.87	127716.82	156122.51	0	0
7	130298.13	145530.06	323876.68	155752.60	1	0
8	120542.52	148718.95	311613.29	152211.77	0	1
9	123334.88	108679.17	304981.62	149759.96	0	0
10	101913.08	110594.11	229160.95	146121.95	1	0
11	100671.96	91790.61	249744.55	144259.40	0	0
12	93863.75	127320.38	249839.44	141585.52	1	0
13	91992.39	135495.07	252664.93	134307.35	0	0
14	119943.24	156547.42	256512.92	132602.65	1	0
15	114523.61	122616.84	261776.23	129917.04	0	1
16	78013.11	121597.55	264346.06	126992.93	0	0
17	94657.16	145077.58	282574.31	125370.37	0	1
18	91749.16	114175.79	294919.57	124266.90	1	0
19	86419.70	153514.11	0.00	122776.86	0	1
20	76253.86	113867.30	298664.47	118474.03	0	0
21	78389.47	153773.43	299737.29	111313.02	0	1
22	73994.56	122782.75	303319.26	110352.25	1	0
23	67532.53	105751.03	304768.73	108733.99	1	0
24	77044.01	99281.34	140574.81	108552.04	0	1
25	64664.71	139553.16	137962.62	107404.34	0	0
26	75328.87	144135.98	134050.07	105733.54	1	0
27	72107.60	127864.55	353183.81	105008.31	0	1
28	66051.52	182645.56	118148.20	103282.38	1	0
29	65605.48	153032.06	107138.38	101004.64	0	1
30	61994.48	115641.28	91131.24	99937.59	1	0
31	61136.38	152701.92	88218.23	97483.56	0	1
32	63408.86	129219.61	46085.25	97427.84	0	0
33	55493.95	103057.49	214634.81	96778.92	1	0
34	46426.07	157693.92	210797.67	96712.80	0	0
35	46014.02	85047.44	205517.64	96479.51	0	1
36	28663.76	127056.21	201126.82	90708.19	1	0
37	44069.95	51283.14	197029.42	89949.14	0	0
38	20229.59	65947.93	185265.10	81229.06	0	1
39	38558.51	82982.09	174999.30	81005.76	0	0
40	28754.33	118546.05	172795.67	78239.91	0	0
41	27892.92	84710.77	164470.71	77798.83	1	0
42	23640.93	96189.63	148001.11	71498.49	0	0

Spliting in Dep and In-Dependent variables

```
import movecolumn as mc
mc.MoveToLast(df,'Profit')
```

	R&D Spend	Administration	Marketing Spend	State_Florida	State_New York	Profit
0	165349.20	136897.80	471784.10	0	1	192261.83
1	162597.70	151377.59	443898.53	0	0	191792.06
2	153441.51	101145.55	407934.54	1	0	191050.39
3	144372.41	118671.85	383199.62	0	1	182901.99
4	142107.34	91391.77	366168.42	1	0	166187.94
5	131876.90	99814.71	362861.36	0	1	156991.12
6	134615.46	147198.87	127716.82	0	0	156122.51
7	130298.13	145530.06	323876.68	1	0	155752.60
8	120542.52	148718.95	311613.29	0	1	152211.77
9	123334.88	108679.17	304981.62	0	0	149759.96
10	101913.08	110594.11	229160.95	1	0	146121.95
11	100671.96	91790.61	249744.55	0	0	144259.40
12	93863.75	127320.38	249839.44	1	0	141585.52
13	91992.39	135495.07	252664.93	0	0	134307.35
14	119943.24	156547.42	256512.92	1	0	132602.65
15	114523.61	122616.84	261776.23	0	1	129917.04
16	78013.11	121597.55	264346.06	0	0	126992.93
17	94657.16	145077.58	282574.31	0	1	125370.37
18	91749.16	114175.79	294919.57	1	0	124266.90
19	86419.70	153514.11	0.00	0	1	122776.86
20	76253.86	113867.30	298664.47	0	0	118474.03
21	78389.47	153773.43	299737.29	0	1	111313.02
22	73994.56	122782.75	303319.26	1	0	110352.25
23	67532.53	105751.03	304768.73	1	0	108733.99
24	77044.01	99281.34	140574.81	0	1	108552.04
25	64664.71	139553.16	137962.62	0	0	107404.34
26	75328.87	144135.98	134050.07	1	0	105733.54
27	72107.60	127864.55	353183.81	0	1	105008.31
28	66051.52	182645.56	118148.20	1	0	103282.38
29	65605.48	153032.06	107138.38	0	1	101004.64
30	61994.48	115641.28	91131.24	1	0	99937.59
31	61136.38	152701.92	88218.23	0	1	97483.56
32	63408.86	129219.61	46085.25	0	0	97427.84
33	55493.95	103057.49	214634.81	1	0	96778.92
34	46426.07	157693.92	210797.67	0	0	96712.80
35	46014.02	85047.44	205517.64	0	1	96479.51
36	28663.76	127056.21	201126.82	1	0	90708.19
37	44069.95	51283.14	197029.42	0	0	89949.14
38	20229.59	65947.93	185265.10	0	1	81229.06
39	38558.51	82982.09	174999.30	0	0	81005.76
40	28754.33	118546.05	172795.67	0	0	78239.91
41	27892.92	84710.77	164470.71	1	0	77798.83
42	23640.93	96189.63	148001.11	0	0	71498.49
43	15505.73	127382.30	35534.17	0	1	69758.98
44	22177.74	154806.14	28334.72	0	0	65200.33
45	1000.23	124153.04	1903.93	0	1	64926.08
46	1315.46	115816.21	297114.46	1	0	49490.75
47	0.00	135426.92	0.00	0	0	42559.73
48	542.05	51743.15	0.00	0	1	35673.41

49	0.00	116983.80	45173.06	0	0	14681.40

```
x = df.iloc[:,0:-1]
y = df['Profit']
```

Scailing the data

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc_x = sc.fit_transform(x)
pd.DataFrame(sc_x)

variable = sc_x
variable.shape
```

```
(50, 5)
```

Split the data into train and test

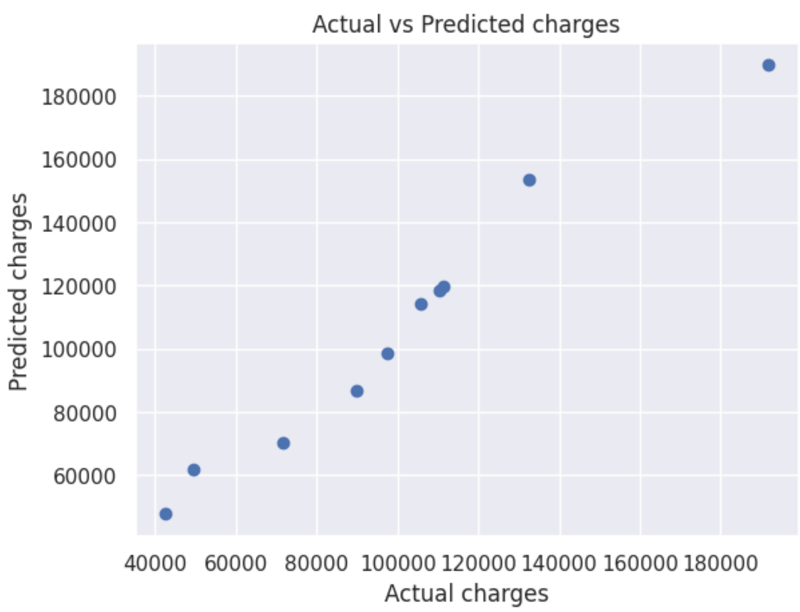
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(sc_x, y, test_size=0.2, random_state=101)
```

Linear regeression model

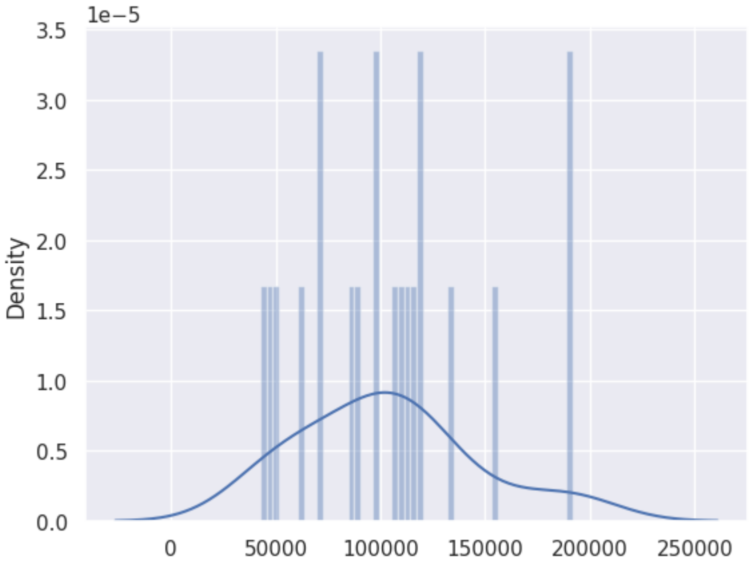
```
lr = LinearRegression()
lr.fit(x_train, y_train)

y_train_pred= lr.predict(x_train)
y_test_pred = lr.predict(x_test)


print("Train Prediction :  ", r2_score(y_train, y_train_pred))
print("Test Prediction :", r2_score(y_test, y_test_pred))
```

```
Train Prediction :   0.945849310601959
Test Prediction : 0.9493973303776394
```

```
# Check linearity

plt.scatter(y_test, y_test_pred)

plt.xlabel("Actual charges")
plt.ylabel("Predicted charges")
plt.title("Actual vs Predicted charges")

plt.show()
```



```
# Normality of Residual

sns.distplot((y_test, y_test_pred), bins=50)
plt.show()
```



Performance Matrix check

```
#Mean absolutr error ( MAE)

print("MAE :", metrics.mean_absolute_error(y_test, y_test_pred))
```

```
MAE : 7068.317931596443
```

```
# Mean Absolute Percent Error (MAPE)

print("MAPE :", metrics.mean_absolute_error(y_test, y_test_pred)/100)
```

```
MAPE : 70.68317931596442
```

```
## Mean Squared Error (MSE)

print("MSE :", metrics.mean_squared_error(y_test, y_test_pred))
```

```
MSE : 83628721.14289564
```

```
## Root Mean Squared Error (RMSE)
print("RMSE :", np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
RMSE : 9144.874036469591
```

```
ACC = 100-MAPE*100
print(f'accuracy of the model = {ACC}')
```

```
accuracy of the model = 80.83386851915917
```