CONTENTS

MULTIPLE LINEAR REGRESSION ON 50 START UPS

IMPORTING REQUIRED LIBRARIES

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
pd.set_option ( "display.max_columns" , None)
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV, train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

IMPORTING DATA (CSV)

In [2]:
```
DF = pd.read_csv ( r"A:\PANDAS DATA\DATA SETS\50 STARTUPS\50_Startups.csv" )
Start_Ups = DF
```

DATA CLEANING

CHECKING FOR NULL VALUES

In [3]:
```
def Null_Values ( DF ) :
    Total = dict( DF.isnull ( ).sum ( ) [ DF.isnull ( ).sum ( ) > 0 ] )
    if len ( Total ) > 0 :
        List = list ( dict(DF.isnull ( ).sum ( ) [ DF.isnull ( ).sum ( ) > 0 ] ) )
        Categorical = { i : DF [ i ].isnull ( ).sum ( ) for i in list ( DF.select_dtypes ( 'object' ) ) }
        Numeric = { i : DF [ i ].isnull ( ).sum ( ) for i in list ( DF.select_dtypes ( np.number ) ) }
        print ("The Categorical Null Values are : \n\n{}\n\nThe Numeric Null Values are : \n\n{}\n\n".format ( 
    else :
        print ( "This data set has no null values" )
Null_Values ( DF )
```

```
This data set has no null values
```

NON-NUMERIC TO NUMERIC

The following function is used for the data frame, where all columns are nominal.

In [4]:
```
LE = LabelEncoder ( )
def Encoding ( DF ) :
    Columns = list ( DF.select_dtypes ( 'object' ) )
    DF.loc [ : , Columns ] = DF.loc [ : , Columns ].apply ( LE.fit_transform )
    display (DF)
Encoding ( DF )
```

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	2	192261.83
1	162597.70	151377.59	443898.53	0	191792.06
2	153441.51	101145.55	407934.54	1	191050.39
3	144372.41	118671.85	383199.62	2	182901.99
4	142107.34	91391.77	366168.42	1	166187.94
5	131876.90	99814.71	362861.36	2	156991.12
6	134615.46	147198.87	127716.82	0	156122.51
7	130298.13	145530.06	323876.68	1	155752.60
8	120542.52	148718.95	311613.29	2	152211.77
9	123334.88	108679.17	304981.62	0	149759.96
10	101913.08	110594.11	229160.95	1	146121.95
11	100671.96	91790.61	249744.55	0	144259.40
12	93863.75	127320.38	249839.44	1	141585.52
13	91992.39	135495.07	252664.93	0	134307.35
14	119943.24	156547.42	256512.92	1	132602.65
15	114523.61	122616.84	261776.23	2	129917.04
16	78013.11	121597.55	264346.06	0	126992.93
17	94657.16	145077.58	282574.31	2	125370.37
18	91749.16	114175.79	294919.57	1	124266.90
19	86419.70	153514.11	0.00	2	122776.86
20	76253.86	113867.30	298664.47	0	118474.03
21	78389.47	153773.43	299737.29	2	111313.02
22	73994.56	122782.75	303319.26	1	110352.25
23	67532.53	105751.03	304768.73	1	108733.99
24	77044.01	99281.34	140574.81	2	108552.04
25	64664.71	139553.16	137962.62	0	107404.34
26	75328.87	144135.98	134050.07	1	105733.54
27	72107.60	127864.55	353183.81	2	105008.31
28	66051.52	182645.56	118148.20	1	103282.38
29	65605.48	153032.06	107138.38	2	101004.64
30	61994.48	115641.28	91131.24	1	99937.59
31	61136.38	152701.92	88218.23	2	97483.56
32	63408.86	129219.61	46085.25	0	97427.84
33	55493.95	103057.49	214634.81	1	96778.92
34	46426.07	157693.92	210797.67	0	96712.80
35	46014.02	85047.44	205517.64	2	96479.51
36	28663.76	127056.21	201126.82	1	90708.19
37	44069.95	51283.14	197029.42	0	89949.14
38	20229.59	65947.93	185265.10	2	81229.06
39	38558.51	82982.09	174999.30	0	81005.76
40	28754.33	118546.05	172795.67	0	78239.91
41	27892.92	84710.77	164470.71	1	77798.83
42	23640.93	96189.63	148001.11	0	71498.49
43	15505.73	127382.30	35534.17	2	69758.98
44	22177.74	154806.14	28334.72	0	65200.33
45	1000.23	124153.04	1903.93	2	64926.08
46	1315.46	115816.21	297114.46	1	49490.75
47	0.00	135426.92	0.00	0	42559.73
48	542.05	51743.15	0.00	2	35673.41
49	0.00	116983.80	45173.06	0	14681.40

RANDOM SAMPLING

In [5]:
```
def Sampling ( DF , Y ) : # Y is the target variable , accepted in string format
    global Train , Test , Train_X , Test_X , Train_Y , Test_Y
    Train , Test = train_test_split( DF , test_size = 0.2 , random_state = None ) # Random State = None ( small
    Train_X = Train.drop ( columns = [ Y ] )
    Train_Y = Train [ Y ].reset_index ( drop = True  )
    Test_X = Test.drop ( columns = [ Y ] )
    Test_Y = Test [ Y ].reset_index ( drop = True  )
    print ( "The shapes of sampled data sets are (after standard sampling ratio = 0.2 test size ) : \n" )
    print ( "Train Data = {}\n\nTest Data = {}\n\nTrain_X Data = {}\n\nTrain_Y Data = {}\n\nTest_X Data = {}\n\
Sampling ( DF , 'Profit' )
```

```
The shapes of sampled data sets are (after standard sampling ratio = 0.2 test size ) :

Train Data = (40, 5)

Test Data = (10, 5)

Train_X Data = (40, 4)

Train_Y Data = (40,)

Test_X Data = (10, 4)

Test_Y Data = (10,)
```

HYPER PARAMETER TUNING

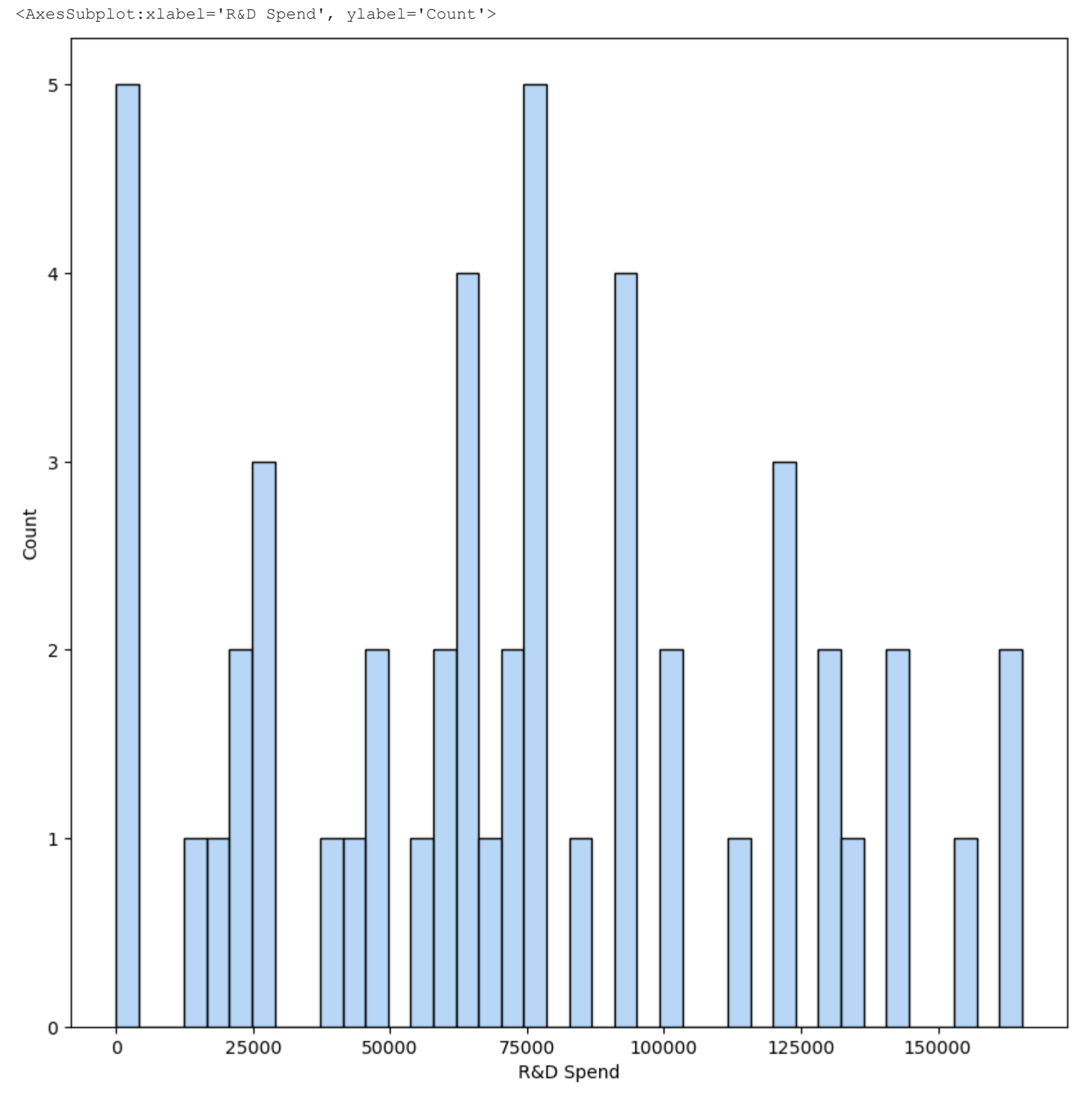
In [6]:
```
Parameters = { 'fit_intercept': [ True , False ],
               'copy_X': [True , False ] }
LR = LinearRegression (  )
grid_search = GridSearchCV ( LR , Parameters )
grid_search.fit ( Train_X , Train_Y )
print( 'Best hyperparameters : ' , grid_search.best_params_ )
```

```
Best hyperparameters :  {'copy_X': True, 'fit_intercept': True}
```

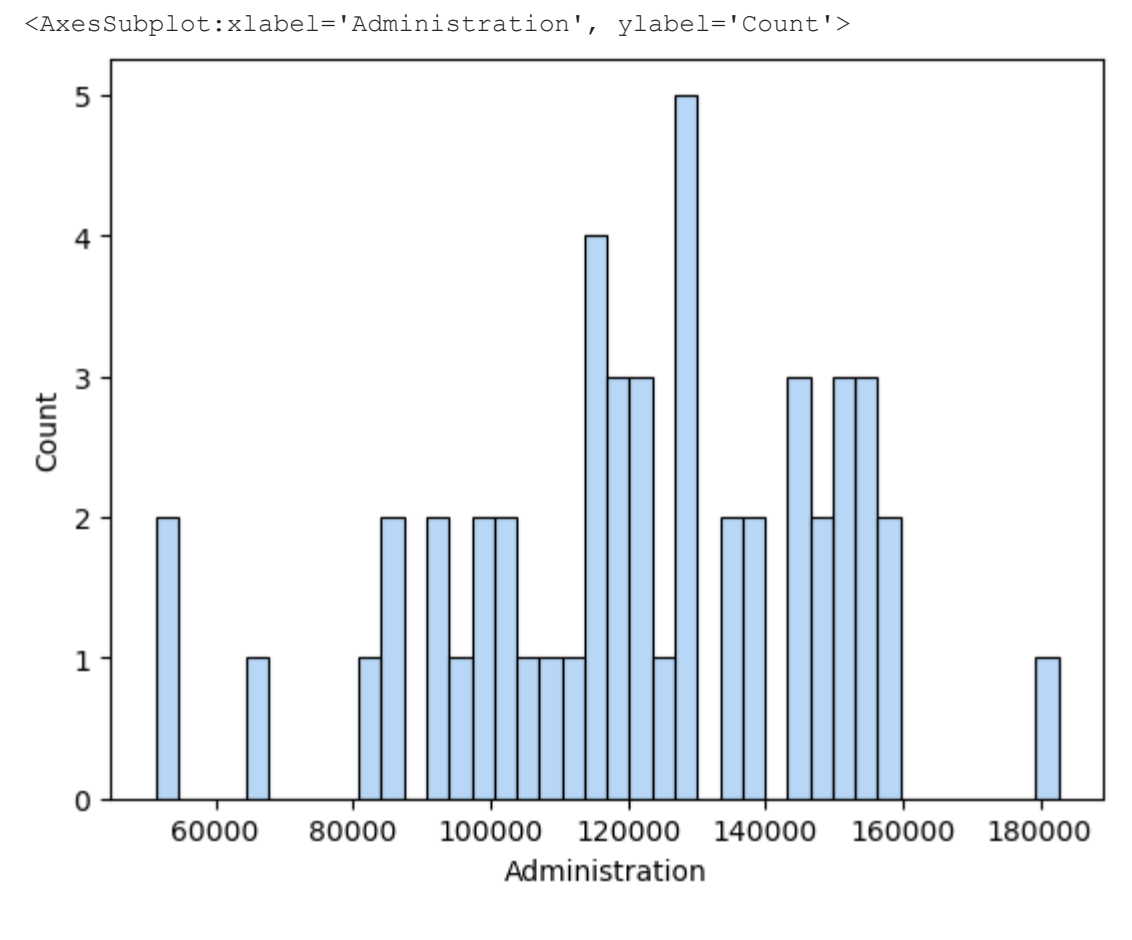
DESCRIBING DATA

In [7]:
```
def Plots ( DF ) :
    sns.set_palette( "pastel" )
    plt.figure ( figsize =  ( 10 , 10 ) )
    for i in list ( DF.select_dtypes ( np.number ).columns ) :
        display ( sns.histplot ( x = i , data = DF , bins = 40  )  )
        plt.show ( )
        print ( " " )
Plots(Start_Ups)
```

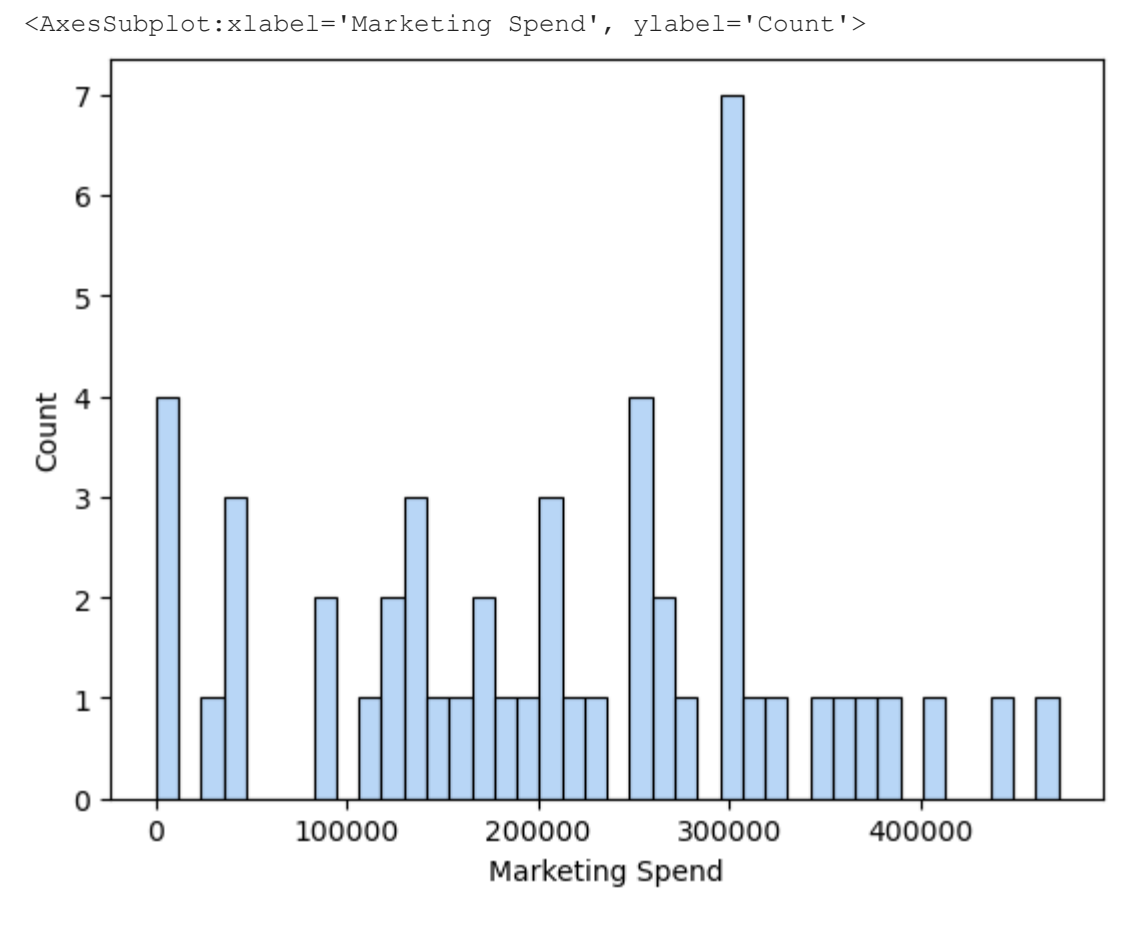
<AxesSubplot:xlabel='R&D Spend', ylabel='Count'>



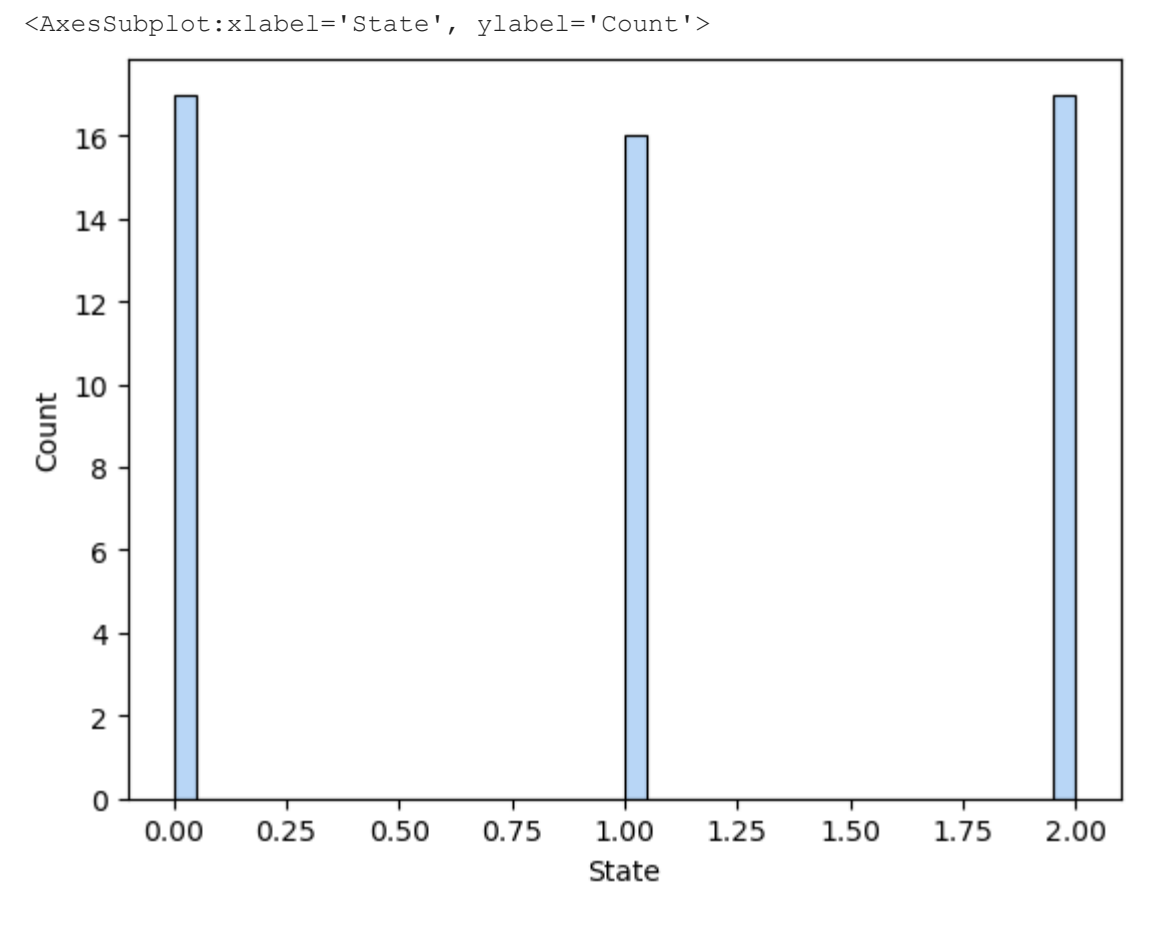
<AxesSubplot:xlabel='Administration', ylabel='Count'>



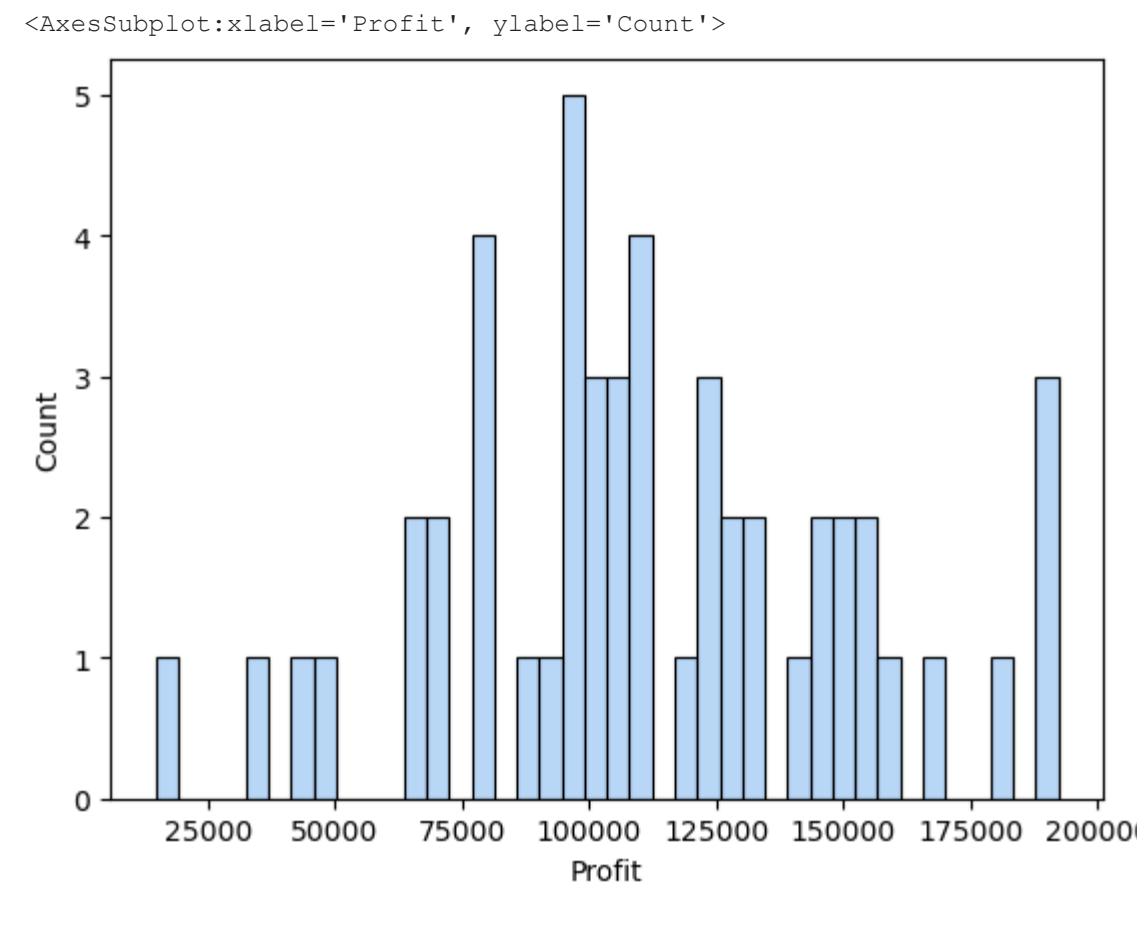
<AxesSubplot:xlabel='Marketing Spend', ylabel='Count'>



<AxesSubplot:xlabel='State', ylabel='Count'>



<AxesSubplot:xlabel='Profit', ylabel='Count'>



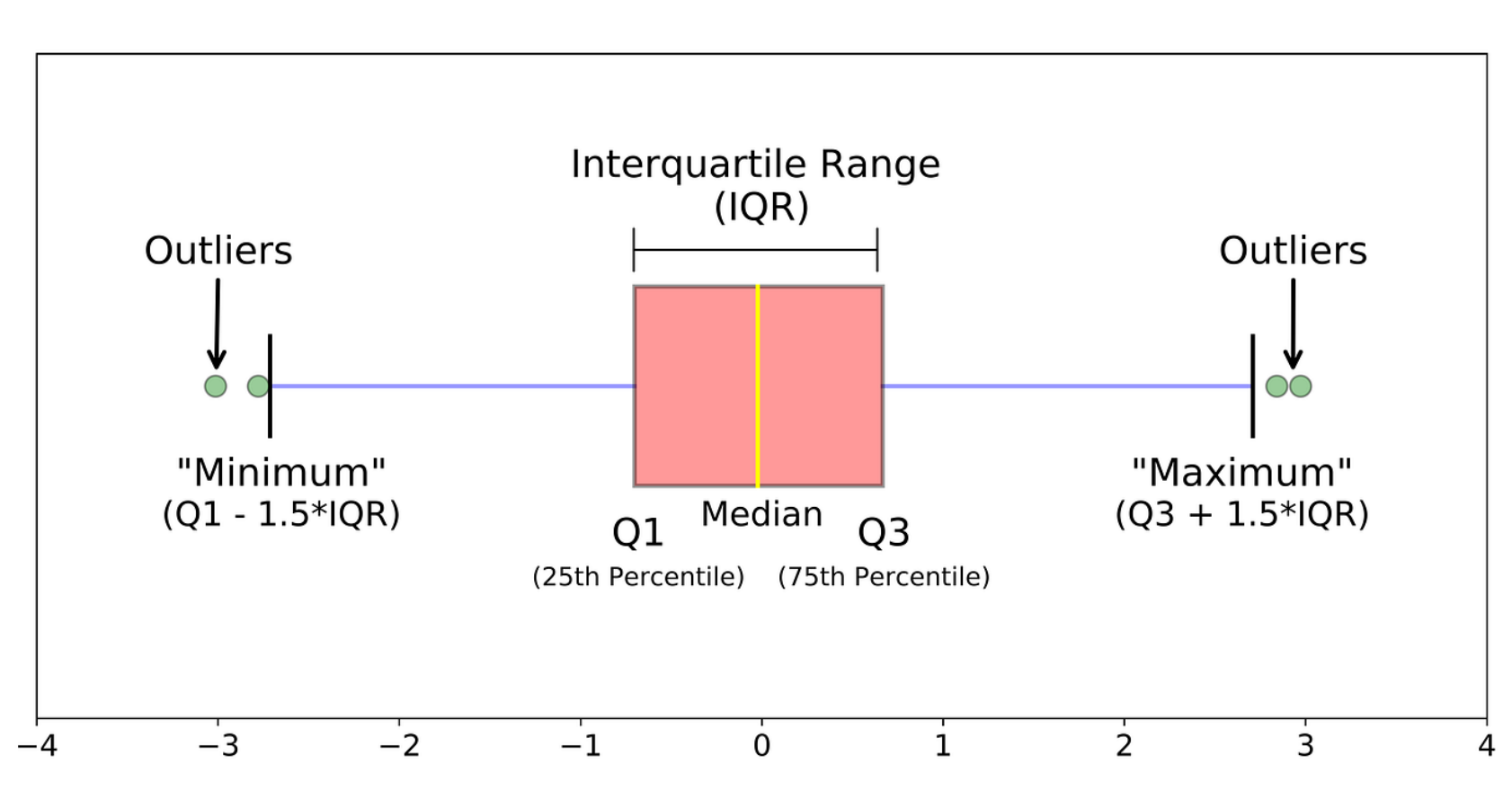
In [8]:
```
def Statistics ( DF ) :
    for i in list ( DF.select_dtypes ( np.number ).columns ) :
        global Mean , Med , SD , Kur
        Mean = np.mean ( DF [ i ] )
        Med = np.median ( DF [ i ] )
        SD = np.std ( DF [ i ] )
        Kur = DF [ i ].kurt ( ) + 3
        print ( '\n\nColumn ----> ', i )
        print ( "\n\nMean = {}\n\nMedian = {}\n\nStandard Deviation = {}\n\nKurtosis = {}".format ( Mean , Med
        if Mean != Med :
            if Mean > Med :
                print ( "\nRight Skewed" )
            else :
                print ( "\nLeft Skewed" )
        if Kur > 3 :
            print ( "\nThe distribution is Leptokurtic" )
        else :
            print ( "\nThe distribution is Platykurtic\n" )
        print ( "----------------------------------------------------------------" )

Statistics ( Start_Ups )
```

```
Column ---->  R&D Spend

Mean = 73721.61559999999
Median = 73051.08
Standard Deviation = 45440.915562565344
Kurtosis = 2.2385354431575326
Right Skewed
The distribution is Platykurtic
----------------------------------------------------------------

Column ---->  Administration

Mean = 121344.63959999995
Median = 122699.795
Standard Deviation = 27736.20965129446
Kurtosis = 3.225071135368654
Left Skewed
The distribution is Leptokurtic
----------------------------------------------------------------

Column ---->  Marketing Spend

Mean = 211025.09780000005
Median = 212716.24
Standard Deviation = 121061.23118271727
Kurtosis = 2.3282988718702486
Left Skewed
The distribution is Platykurtic
----------------------------------------------------------------

Column ---->  State

Mean = 1.0
Median = 1.0
Standard Deviation = 0.8246211251235321
Kurtosis = 1.4361702127659575
The distribution is Platykurtic
----------------------------------------------------------------

Column ---->  Profit

Mean = 112012.63920000002
Median = 107978.19
Standard Deviation = 39901.08281667283
Kurtosis = 2.936141114531469
Right Skewed
The distribution is Platykurtic
----------------------------------------------------------------
```

We can see that the distributions are platykurtic / leptokurtic, therefore the data is not normally distributed.

REMOVING OUTLIERS

DEFINING UPPER AND LOWER LIMIT OF QUARTILES



ASSUMPTIONS CHECK

MULTICOLLINEARITY

Multicollinearity occurs when two or more independent variables in a linear regression model are highly correlated with each other.

This can cause problems in the estimation of the model coefficients and the interpretation of the results.

METHOD 1 : CORRELATION MATRIX

In [9]:
```
Correlation = DF.corr ( )
Set_Corr = set ( Correlation.values [ ( Correlation > 0.5) | ( Correlation < -0.5)  ] )
Set_Corr.remove ( 1.0 )
sns.color_palette ( 'pastel' )
sns.heatmap ( DF.corr ( ) , cmap = "BuPu" , annot = True )
```

Out[9]: <AxesSubplot:>

METHOD 2 : VIF

The VIF measures how much the variance of the estimated regression coefficient is inflated due to the presence of multicollinearity.

A high VIF value indicates that the variance of the coefficient estimate is significantly larger than it would be if the predictor variable were not correlated with the other predictor variables in the model.

1. We can see that in the presence of multicollinearity, the coefficient of Research & Development is varying the most.

In [10]:
```
A = DF.iloc [ : , : - 1 ]
VIF = pd.DataFrame ( )
VIF [ 'Variables' ] = A.columns
VIF [ 'VIF_Values' ] = [ variance_inflation_factor ( A.values, i ) for i in range( A.shape[1] ) ]
VIF.sort_values ( 'VIF_Values' , ascending = False )
```

Out[10]:

	Variables	VIF_Values
0	R&D Spend	8.386322
2	Marketing Spend	7.674608
1	Administration	4.815916
3	State	2.382637

LINEAR REGRESSION MODELING

Best hyperparameters : {'copy_X': True, 'fit_intercept': True}

In [11]:
```
LR = LinearRegression ( copy_X = True , fit_intercept = True )
LR.fit ( Train_X , Train_Y )
```

Out[11]: LinearRegression()

In [12]:
```
Model = pd.DataFrame ( )
Model [ 'Predicted_Values' ] = LR.predict ( Train_X )
Model [ 'Actual_Values' ] = Train_Y
Model [ 'Error' ] = Train_Y - Model [ 'Predicted_Values' ]
Model [ 'Absolute_Error' ] = np.abs ( Model.Error )
Model [ 'Error_Percentage' ] = np.abs ( ( Model [ 'Error' ] * 100  ) / Train_Y )
Model = Model.sort_values ( 'Error_Percentage' , ascending = False )
Model
```

Out[12]:

	Predicted_Values	Actual_Values	Error	Absolute_Error	Error_Percentage
36	46828.384581	14681.40	-32146.984581	32146.984581	218.964026
34	47542.034814	35673.41	-11868.624814	11868.624814	33.270228
25	48697.538683	64926.08	16228.541317	16228.541317	24.995412
0	75230.381370	90708.19	15477.808630	15477.808630	17.063298
28	68513.680337	81229.06	12715.379663	12715.379663	15.653732
23	146657.223872	129917.04	-16740.183872	16740.183872	12.885287
1	55852.610677	49490.75	-6361.860677	6361.860677	12.854646
11	61260.858318	69758.98	8498.121682	8498.121682	12.182119
14	115261.582435	105008.31	-10253.272435	10253.272435	9.764249
6	128752.417997	141585.52	12833.102003	12833.102003	9.063852
20	88933.998355	96712.80	7778.801645	7778.801645	8.043198
3	45780.027285	42559.73	-3220.297285	3220.297285	7.566536
27	119090.858263	111313.02	-7777.838263	7777.838263	6.987357
35	89900.159997	96479.51	6579.350003	6579.350003	6.819427
17	126325.283030	134307.35	7982.066970	7982.066970	5.943135
15	73179.626099	77798.83	4619.203901	4619.203901	5.937369
7	101354.908357	107404.34	6049.431643	6049.431643	5.632390
31	173830.567919	182901.99	9071.422081	9071.422081	4.959718
24	131553.555967	125370.37	-6183.185967	6183.185967	4.931936
4	110919.907679	105733.54	-5186.367679	5186.367679	4.905130
19	85588.831281	89949.14	4360.308719	4360.308719	4.847527
38	117314.473361	122776.86	5462.386639	5462.386639	4.449036
29	113104.665035	108552.04	-4552.625035	4552.625035	4.193956
8	114266.096833	110352.25	-3913.846833	3913.846833	3.546685
32	128159.606470	124266.90	-3892.706470	3892.706470	3.132537
12	114766.629102	118474.03	3707.400898	3707.400898	3.129294
39	103581.171279	101004.64	-2576.531279	2576.531279	2.550904
30	170184.292636	166187.94	-3996.352636	3996.352636	2.404719
26	99483.035341	97483.56	-1999.475341	1999.475341	2.051090
2	152570.655904	149759.96	-2810.695904	2810.695904	1.876801
22	188232.227868	191792.06	3559.832132	3559.832132	1.856089
10	98782.760535	99937.59	1154.829465	1154.829465	1.155551
16	64503.672586	65200.33	696.657414	696.657414	1.068488
33	157141.405065	156122.51	-1018.895065	1018.895065	0.652625
18	153103.961809	152211.77	-892.191809	892.191809	0.586152
37	193229.742403	192261.83	-967.912403	967.912403	0.503435
9	97730.704066	97427.84	-302.864066	302.864066	0.310860
5	103450.945505	103282.38	-168.565505	168.565505	0.163208
13	80897.162356	81005.76	108.597644	108.597644	0.134062
21	96830.884528	96778.92	-51.964528	51.964528	0.053694

COEFFICIENTS OF VARIABLES

High coefficients of variables mean that the corresponding independent variable has a strong relationship with the dependent variable.

In [13]:
```
Coefficients = pd.DataFrame ()
Columns = list(DF.columns)
Columns.remove ( 'Profit' )
Coefficients [ 'Values' ]  = LR.coef_
Coefficients [ 'Columns' ] = Columns
Coefficients
```

Out[13]:

	Values	Columns
0	0.800407	R&D Spend
1	0.010174	Administration
2	0.027361	Marketing Spend
3	1089.762090	State

COEFFICIENT OF DETERMINANT

R2 value tells us how much variance the model is able to capture (SSR)

$$Adjusted\ R^2 = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

Where
R^2 Sample R-Squared
N Total Sample Size
p Number of independent variable

In [14]:
```
R2 = LR.score ( Train_X , Train_Y )

P = Train_X.shape [ 1 ]

N = Train_X.shape [ 0 ]

ADJ_R2 = 1 - ( ( ( 1 - R2) * ( N - 1 ) ) /  ( N - P - 1 ) )

print ( "The coefficient of determinant is : {} and the adjusted value is {}".format ( R2 * 100 , ADJ_R2 * 100
```
```
The coefficient of determinant is : 95.05035861084991 and the adjusted value is 94.48468530923276
```

CHECKING FOR ASSUMPTIONS

MEAN OF ERROR

In [15]:
```
print ( "The mean of the error is : " , np.mean ( Model.Error ) )
```
```
The mean of the error is :  -1.7280399333685637e-11
```

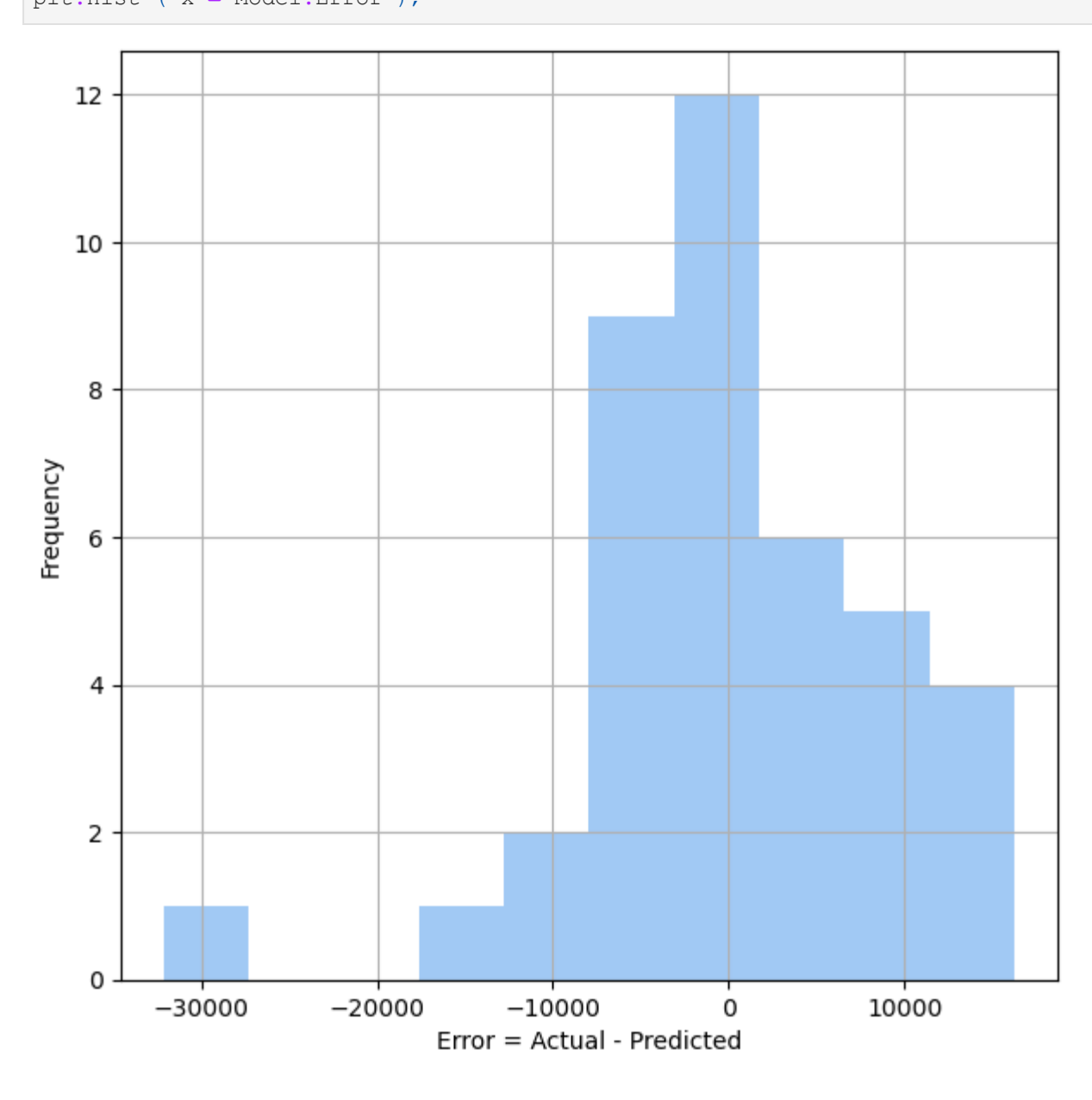
In [16]:
```
print ( "The mean of absolute error is : " , np.mean ( Model.Absolute_Error ) )
```
```
The mean of absolute error is :  6344.162122366675
```

NORMAL DISTRIBUTION

The Errors are violating the assumption of Normal Distribution.

The graph suggests that the kurtosis is higher than 3

In [17]:
```
plt.figure ( figsize = ( 7 , 7 ) )

plt.xlabel ( 'Error = Actual - Predicted' )

plt.ylabel ( 'Frequency' )

plt.grid ( )

plt.hist ( x = Model.Error );
```



HETEROSCEDASTICITY

In [18]:
```
sns.regplot ( x = Model.Actual_Values , y = Model.Predicted_Values , data = Model );
```

METRICS

MSE

In [19]:
```
Predicted_Test = LR.predict ( Test_X )

Error_Test = Test_Y - Predicted_Test

Error_test_Percentage = abs(Error_Test) * 100 / Test_Y

MSE = np.mean ( np.square ( Error_Test ) )

print ( "The mean square value of error test is " , MSE )

print ( "\nThe mean square value of error train is " , np.mean ( np.square (Model.Error) ) )
```
```
The mean square value of error test is  91270989.63188198

The mean square value of error train is  77776787.94248608
```

RMSE

In [20]:
```
RMSE = np.sqrt ( MSE )

print ( "The root mean square value of error test is " , RMSE )

print ( "\nroot mean square value of error train is " , np.sqrt ( np.mean ( np.square ( Model.Error) ) ) )
```
```
The root mean square value of error test is  9553.585171645354

root mean square value of error train is  8819.114918317262
```

MAPE

In [21]:
```
MAPE = np.mean ( Error_test_Percentage )

print ( "The mean absolute percentage error (of test) is " , MAPE )

print ( "\nThe mean absolute percentage error (of train) is " , np.mean ( Model.Error_Percentage ) )
```
```
The mean absolute percentage error (of test) is  6.142368258032201

The mean absolute percentage error (of train) is  11.677243912327064
```

ACCURACY

In [22]:
```
print ( "Accuracy on train data is : " , 100 - MAPE  )

print ( '\nAccuracy on test data is : ' , 100 - np.mean ( Model.Error_Percentage ) )
```
```
Accuracy on train data is :  93.8576317419678

Accuracy on test data is :  88.32275608767293
```

REMOVING OUTLIERS

We can see that the model is performing well on Test data and poor on train data, this is due to less records
in Test Data

Another reason could be that the Train data is influenced by outliers

In [23]:
```
def Quartiles ( DF , Y , K ) :

    global Lower , Upper , Lower_Outliers , Upper_Outliers

    Lower = DF [ Y ].quantile ( q = 0.25 )

    Upper = DF [ Y ].quantile ( q = 0.75 )

    IQR = Upper - Lower

    Lower_Outliers =  Lower - K * IQR

    Upper_Outliers =  Upper + K * IQR

    DF.drop ( labels = list ( DF [ DF [ Y ] >= Upper_Outliers ].index ) , inplace = True )

    DF.drop ( labels = list ( DF [ DF [ Y ] <= Lower_Outliers ].index ) , inplace = True )

    DF.reset_index ( drop = True , inplace = True )

Quartiles ( DF , 'Profit' , 0.75 ) ## K = 1 becasue the data is not vast
```

In [24]:
```
LR = LinearRegression ( copy_X = True , fit_intercept = True )

LR.fit ( Train_X , Train_Y )

Model = pd.DataFrame ( )

Model [ 'Predicted_Values' ] = LR.predict ( Train_X )

Model [ 'Actual_Values' ] = Train_Y

Model [ 'Error' ] = Train_Y - Model [ 'Predicted_Values' ]

Model [ 'Absolute_Error' ] = np.abs ( Model.Error )

Model [ 'Error_Percentage' ] = np.abs ( ( Model [ 'Error' ] * 100  ) / Train_Y )

Model = Model.sort_values ( 'Error_Percentage' , ascending = False )

Model
```

Out[24]:

	Predicted_Values	Actual_Values	Error	Absolute_Error	Error_Percentage
36	46828.384581	14681.40	-32146.984581	32146.984581	218.964026
34	47542.034814	35673.41	-11868.624814	11868.624814	33.270228
25	48697.538683	64926.08	16228.541317	16228.541317	24.995412
0	75230.381370	90708.19	15477.808630	15477.808630	17.063298
28	68513.680337	81229.06	12715.379663	12715.379663	15.653732
23	146657.223872	129917.04	-16740.183872	16740.183872	12.885287
1	55852.610677	49490.75	-6361.860677	6361.860677	12.854646
11	61260.858318	69758.98	8498.121682	8498.121682	12.182119
14	115261.582435	105008.31	-10253.272435	10253.272435	9.764249
6	128752.417997	141585.52	12833.102003	12833.102003	9.063852
20	88933.998355	96712.80	7778.801645	7778.801645	8.043198
3	45780.027285	42559.73	-3220.297285	3220.297285	7.566536
27	119090.858263	111313.02	-7777.838263	7777.838263	6.987357
35	89900.159997	96479.51	6579.350003	6579.350003	6.819427
17	126325.283030	134307.35	7982.066970	7982.066970	5.943135
15	73179.626099	77798.83	4619.203901	4619.203901	5.937369
7	101354.908357	107404.34	6049.431643	6049.431643	5.632390
31	173830.567919	182901.99	9071.422081	9071.422081	4.959718
24	131553.555967	125370.37	-6183.185967	6183.185967	4.931936
4	110919.907679	105733.54	-5186.367679	5186.367679	4.905130
19	85588.831281	89949.14	4360.308719	4360.308719	4.847527
38	117314.473361	122776.86	5462.386639	5462.386639	4.449036
29	113104.665035	108552.04	-4552.625035	4552.625035	4.193956
8	114266.096833	110352.25	-3913.846833	3913.846833	3.546685
32	128159.606470	124266.90	-3892.706470	3892.706470	3.132537
12	114766.629102	118474.03	3707.400898	3707.400898	3.129294
39	103581.171279	101004.64	-2576.531279	2576.531279	2.550904
30	170184.292636	166187.94	-3996.352636	3996.352636	2.404719
26	99483.035341	97483.56	-1999.475341	1999.475341	2.051090
2	152570.655904	149759.96	-2810.695904	2810.695904	1.876801
22	188232.227868	191792.06	3559.832132	3559.832132	1.856089
10	98782.760535	99937.59	1154.829465	1154.829465	1.155551
16	64503.672586	65200.33	696.657414	696.657414	1.068488
33	157141.405065	156122.51	-1018.895065	1018.895065	0.652625
18	153103.961809	152211.77	-892.191809	892.191809	0.586152
37	193229.742403	192261.83	-967.912403	967.912403	0.503435
9	97730.704066	97427.84	-302.864066	302.864066	0.310860
5	103450.945505	103282.38	-168.565505	168.565505	0.163208
13	80897.162356	81005.76	108.597644	108.597644	0.134062
21	96830.884528	96778.92	-51.964528	51.964528	0.053694

COFFEICIENTS OF DETERMINANT

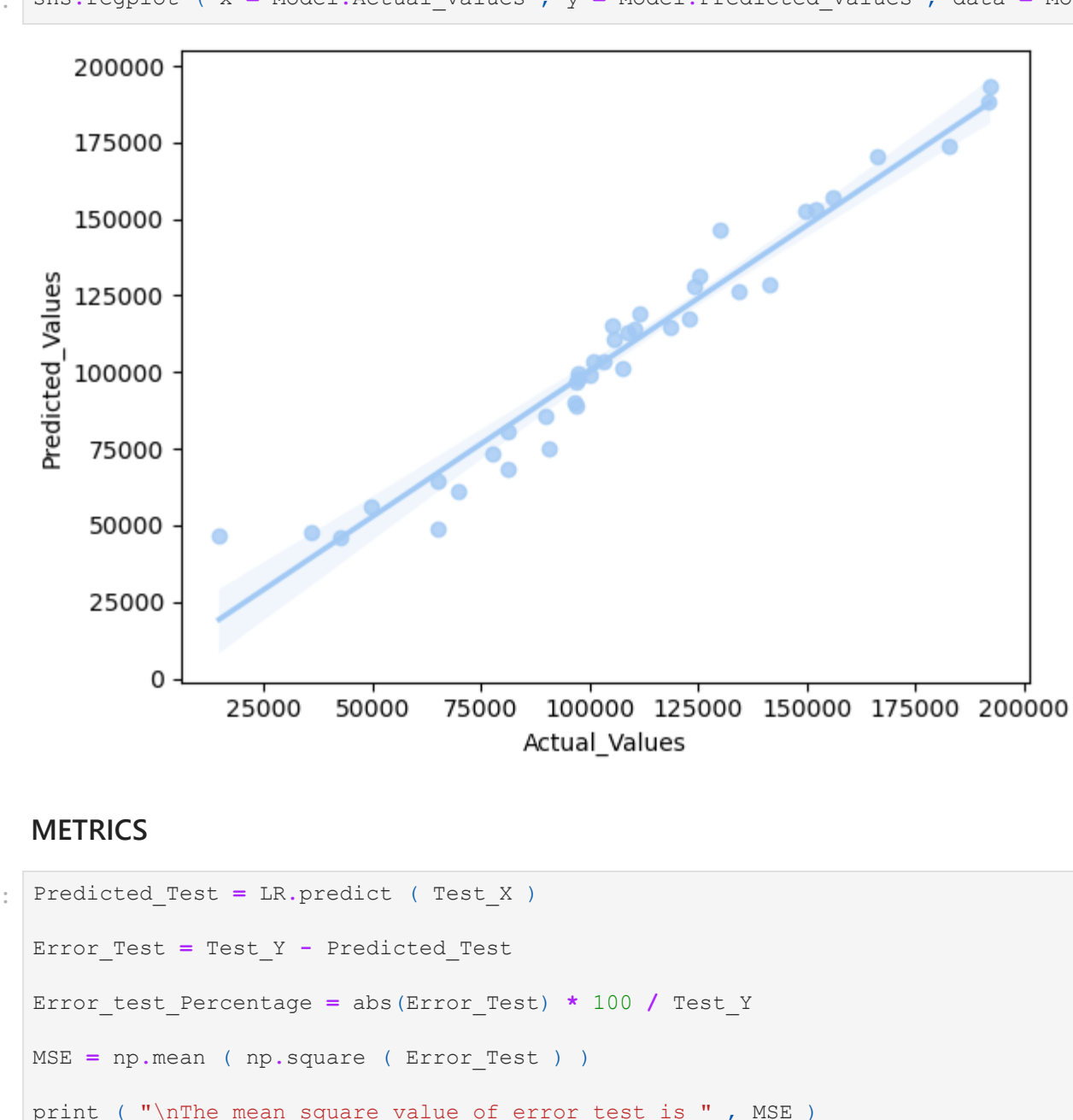
In [25]:
```
R2 = LR.score ( Train_X , Train_Y )

P = Train_X.shape [ 1 ]

N = Train_X.shape [ 0 ]

ADJ_R2 = 1 - ( ( ( 1 - R2) * ( N - 1 ) ) /  ( N - P - 1 ) )

print ( "The coefficient of determinant is : {} and the adjusted value is {}".format ( R2 * 100 , ADJ_R2 * 100
```
```
The coefficient of determinant is : 95.05035861084991 and the adjusted value is 94.48468530923276
```

MEAN

In [26]:
```
print ( "The mean of the error is : " , np.mean ( Model.Error ) )

print ( "\nThe mean of absolute error is : " , np.mean ( Model.Absolute_Error ) )
```
```
The mean of the error is :  -1.7280399333685637e-11

The mean of absolute error is :  6344.162122366675
```

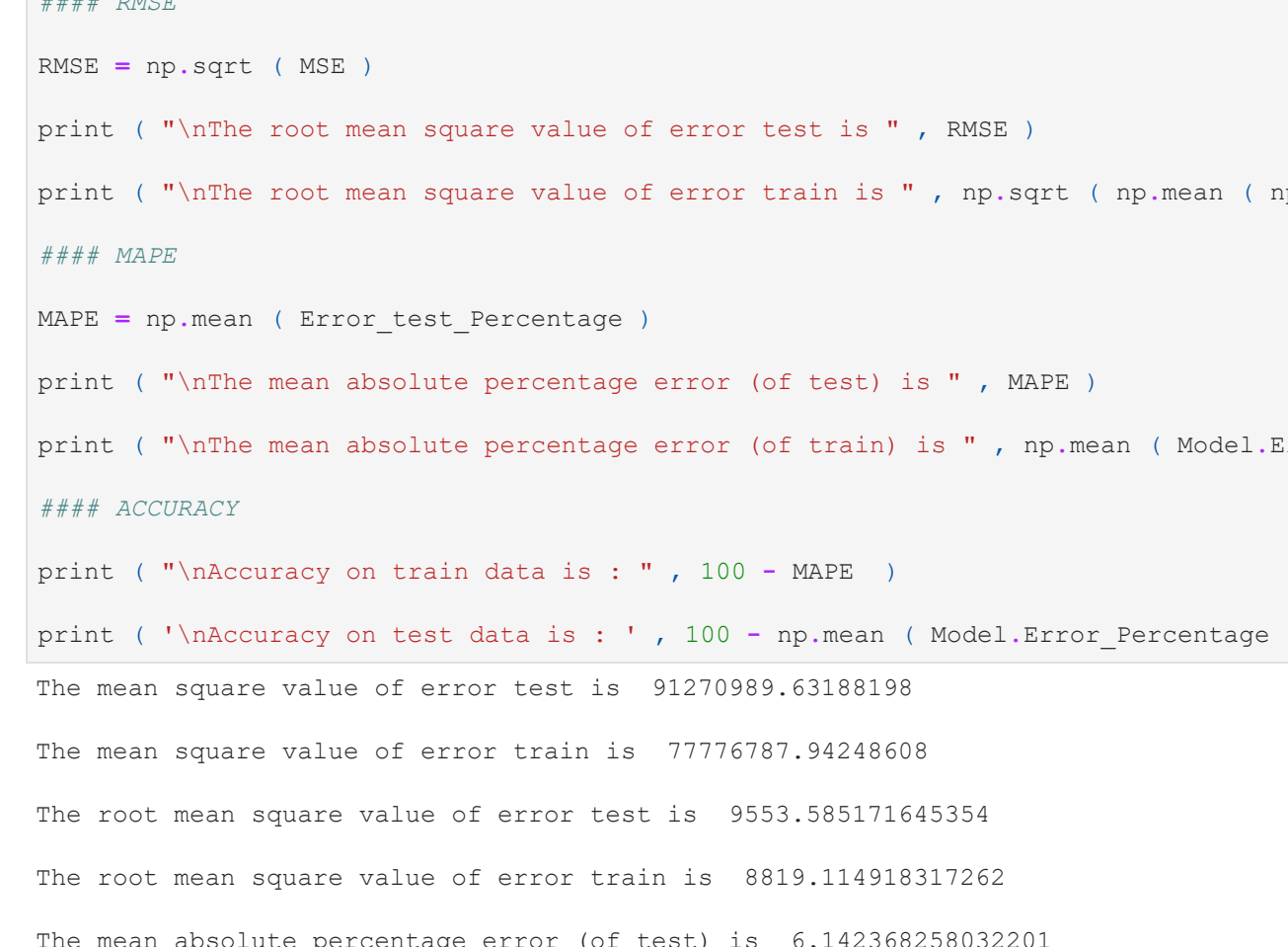
NORMAL DISTRIBUTION

In [27]:
```
plt.figure ( figsize = ( 7 , 7 ) )

plt.xlabel ( 'Error = Actual - Predicted' )

plt.ylabel ( 'Frequency' )

plt.grid ( )

plt.hist ( x = Model.Error );
```



HETEROSCEDASTICITY

In [28]:
```
sns.regplot ( x = Model.Actual_Values , y = Model.Predicted_Values , data = Model );
```



METRICS

In [29]:
```
Predicted_Test = LR.predict ( Test_X )

Error_Test = Test_Y - Predicted_Test

Error_test_Percentage = abs(Error_Test) * 100 / Test_Y

MSE = np.mean ( np.square ( Error_Test ) )

print ( "\nThe mean square value of error test is " , MSE )

print ( "\nThe mean square value of error train is " , np.mean ( np.square (Model.Error) ) )

#### RMSE

RMSE = np.sqrt ( MSE )

print ( "\nThe root mean square value of error test is " , RMSE )

print ( "\nThe root mean square value of error train is " , np.sqrt ( np.mean ( np.square ( Model.Error) ) ) )

#### MAPE

MAPE = np.mean ( Error_test_Percentage )

print ( "\nThe mean absolute percentage error (of test) is " , MAPE )

print ( "\nThe mean absolute percentage error (of train) is " , np.mean ( Model.Error_Percentage ) )

#### ACCURACY

print ( "\nAccuracy on train data is : " , 100 - MAPE  )

print ( '\nAccuracy on test data is : ' , 100 - np.mean ( Model.Error_Percentage ) )
```
```
The mean square value of error test is  91270989.63188198

The mean square value of error train is  77776787.94248608

The root mean square value of error test is  9553.585171645354

The root mean square value of error train is  8819.114918317262

The mean absolute percentage error (of test) is  6.142368258032201

The mean absolute percentage error (of train) is  11.677243912327064

Accuracy on train data is :  93.8576317419678

Accuracy on test data is :  88.32275608767293
```

APPLYING LASSO

A high Lasso score indicates that the model has a large number of nonzero coefficients,

which can be interpreted as a sign of overfitting or high complexity.

In [30]:
```
from sklearn.linear_model import Lasso

L = Lasso ( )

L.fit ( Train_X , Train_Y )

print ( 'The lasso score is : ' , L.score ( Train_X , Train_Y ) * 100 )
```
```
The lasso score is :  95.0503585218569
```

In [37]:
```
Model = pd.DataFrame ( )

Model [ 'Predicted_Values' ] = L.predict ( Train_X )

Model [ 'Actual_Values' ] = Train_Y

Model [ 'Error' ] = Train_Y - Model [ 'Predicted_Values' ]

Model [ 'Absolute_Error' ] = np.abs ( Model.Error )

Model [ 'Error_Percentage' ] = np.abs ( ( Model [ 'Error' ] * 100  ) / Train_Y )

Model = Model.sort_values ( 'Error_Percentage' , ascending = False )

Model
```

Out[37]:

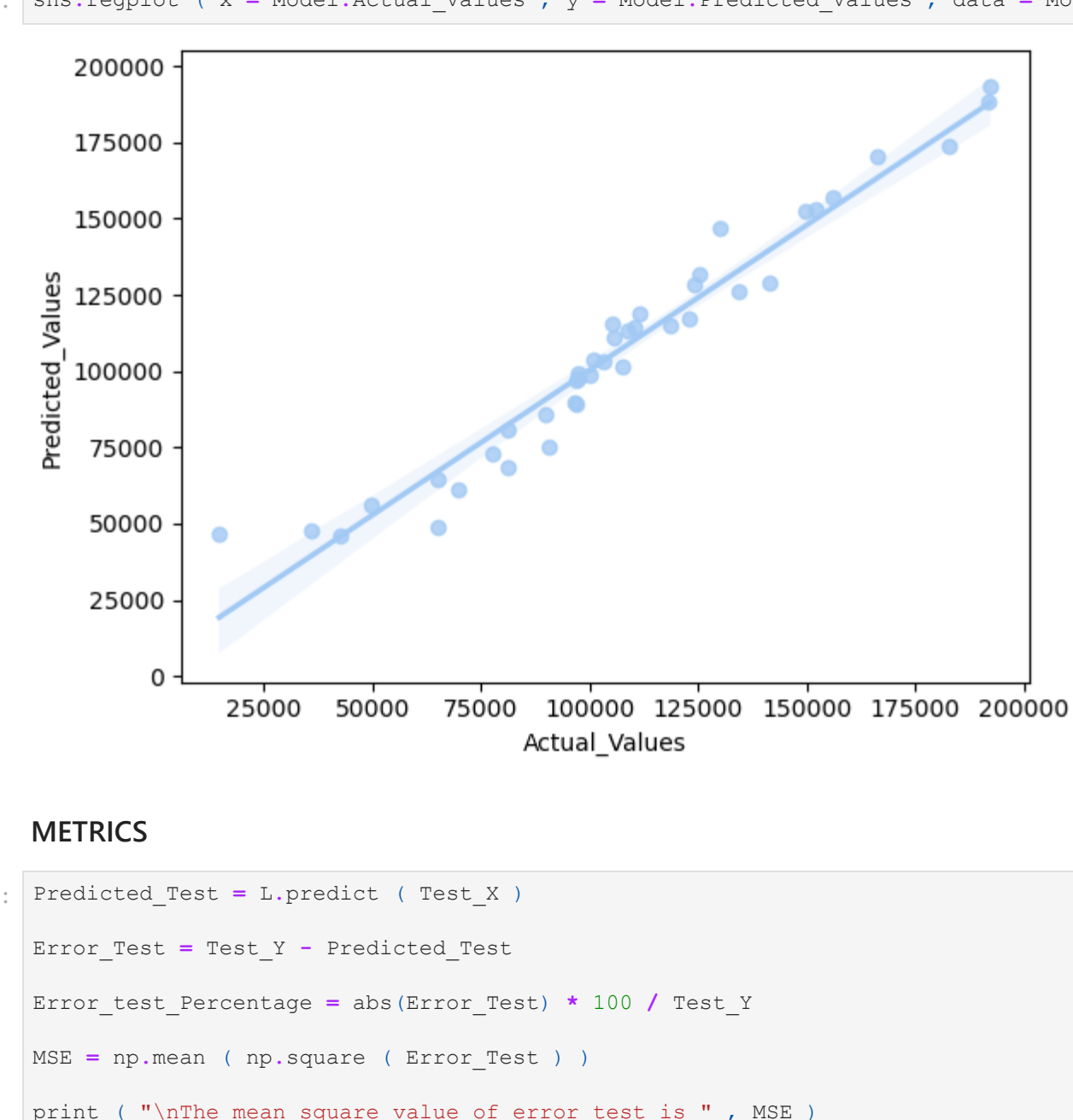
	Predicted_Values	Actual_Values	Error	Absolute_Error	Error_Percentage
36	46829.765869	14681.40	-32148.365869	32148.365869	218.973435
34	47540.719160	35673.41	-11867.309160	11867.309160	33.266540
25	48696.100348	64926.08	16229.979652	16229.979652	24.997628
0	75230.427772	90708.19	15477.762228	15477.762228	17.063247
28	68512.417316	81229.06	12716.642684	12716.642684	15.655287
23	146656.009642	129917.04	-16738.969642	16738.969642	12.884353
1	55852.666507	49490.75	-6361.916507	6361.916507	12.854759
11	61259.442877	69758.98	8499.537123	8499.537123	12.184148
14	115260.328262	105008.31	-10252.018262	10252.018262	9.763054
6	128752.563532	141585.52	12832.956468	12832.956468	9.063749
20	88935.416912	96712.80	7777.383088	7777.383088	8.041731
3	45781.364629	42559.73	-3221.634629	3221.634629	7.569678
27	119089.553413	111313.02	-7776.533413	7776.533413	6.986185
35	89898.903973	96479.51	6580.606027	6580.606027	6.820729
17	126326.811290	134307.35	7980.538710	7980.538710	5.941997
15	73179.733817	77798.83	4619.096183	4619.096183	5.937231
7	101356.362039	107404.34	6047.977961	6047.977961	5.631037
31	173829.433254	182901.99	9072.556746	9072.556746	4.960338
24	131552.282780	125370.37	-6181.912780	6181.912780	4.930920
4	110919.968149	105733.54	-5186.428149	5186.428149	4.905187
19	85590.424888	89949.14	4358.715112	4358.715112	4.845755
38	117313.097236	122776.86	5463.762764	5463.762764	4.450157
29	113103.407859	108552.04	-4551.367859	4551.367859	4.192798
8	114266.238548	110352.25	-3913.988548	3913.988548	3.546814
32	128159.784067	124266.90	-3892.884067	3892.884067	3.132680
12	114768.186170	118474.03	3705.843830	3705.843830	3.127980
39	103579.797889	101004.64	-2575.157889	2575.157889	2.549544
30	170184.595343	166187.94	-3996.655343	3996.655343	2.404901
26	99481.651410	97483.56	-1998.091410	1998.091410	2.049670
2	152572.285825	149759.96	-2812.325825	2812.325825	1.877889
22	188233.874852	191792.06	3558.185148	3558.185148	1.855231
10	98782.840308	99937.59	1154.749692	1154.749692	1.155471
16	64505.013899	65200.33	695.316101	695.316101	1.066430
33	157142.935344	156122.51	-1020.425344	1020.425344	0.653606
18	153102.724597	152211.77	-890.954597	890.954597	0.585339
37	193228.628642	192261.83	-966.798642	966.798642	0.502855
9	97732.148518	97427.84	-304.308518	304.308518	0.312342
5	103450.923495	103282.38	-168.543495	168.543495	0.163187
13	80898.688424	81005.76	107.071576	107.071576	0.132178
21	96831.011148	96778.92	-52.091148	52.091148	0.053825

MEAN

In [38]:
```
print ( "The mean of the error is : " , np.mean ( Model.Error ) )

print ( "\nThe mean of absolute error is : " , np.mean ( Model.Absolute_Error ) )
```
```
The mean of the error is :  -1.4006218407303095e-11

The mean of absolute error is :  6343.93405472469
```

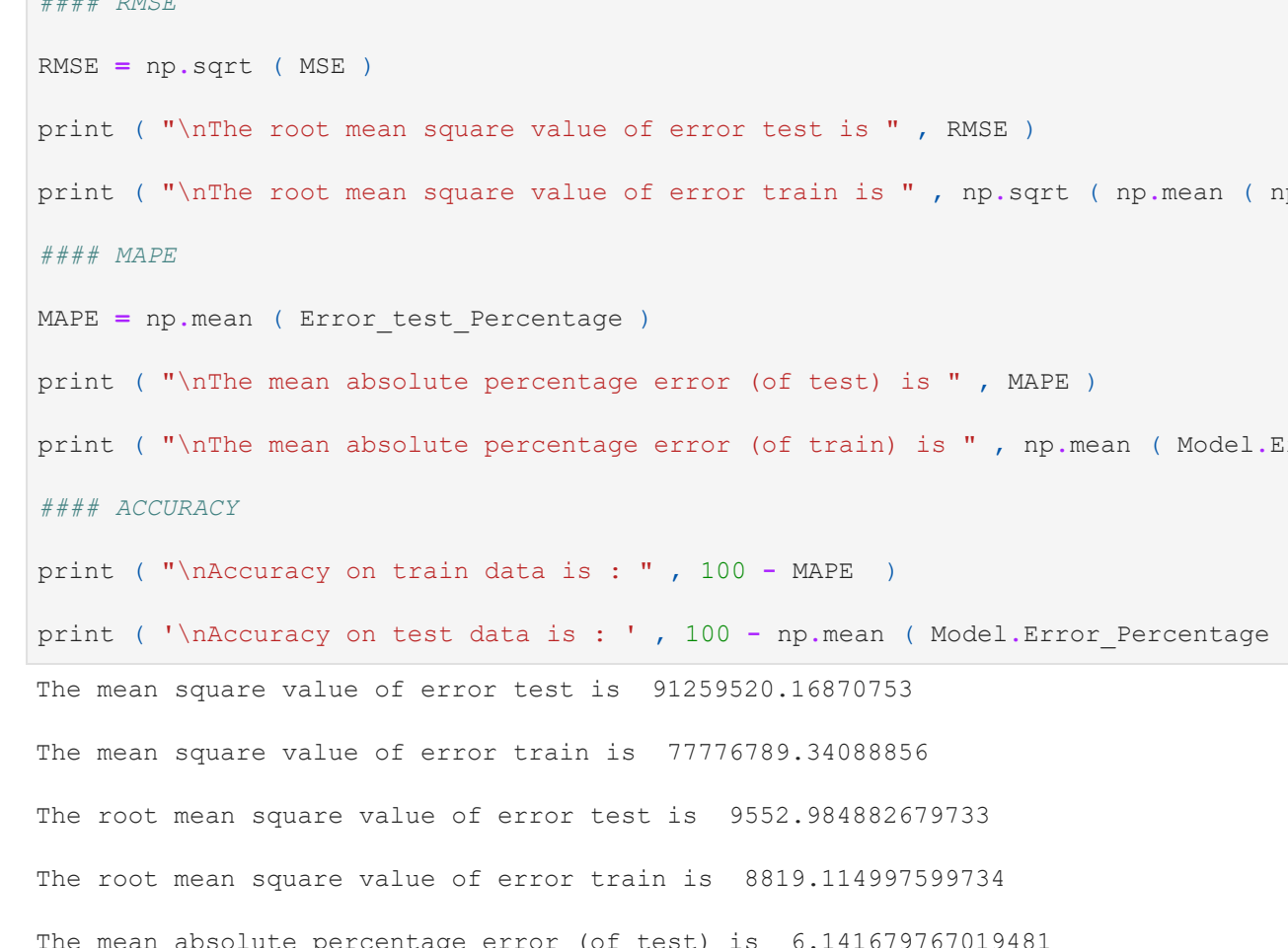
NORMAL DISTRIBUTION

In [39]:
```
plt.figure ( figsize = ( 7 , 7 ) )

plt.xlabel ( 'Error = Actual - Predicted' )

plt.ylabel ( 'Frequency' )

plt.grid ( )

plt.hist ( x = Model.Error );
```



HETEROSCEDASTICITY

In [40]:
```
sns.regplot ( x = Model.Actual_Values , y = Model.Predicted_Values , data = Model );
```



METRICS

In [41]:
```
Predicted_Test = L.predict ( Test_X )

Error_Test = Test_Y - Predicted_Test

Error_test_Percentage = abs(Error_Test) * 100 / Test_Y

MSE = np.mean ( np.square ( Error_Test ) )

print ( "\nThe mean square value of error test is " , MSE )

print ( "\nThe mean square value of error train is " , np.mean ( np.square (Model.Error) ) )

#### RMSE

RMSE = np.sqrt ( MSE )

print ( "\nThe root mean square value of error test is " , RMSE )

print ( "\nThe root mean square value of error train is " , np.sqrt ( np.mean ( np.square ( Model.Error) ) ) )

#### MAPE

MAPE = np.mean ( Error_test_Percentage )

print ( "\nThe mean absolute percentage error (of test) is " , MAPE )

print ( "\nThe mean absolute percentage error (of train) is " , np.mean ( Model.Error_Percentage ) )

#### ACCURACY

print ( "\nAccuracy on train data is : " , 100 - MAPE  )

print ( '\nAccuracy on test data is : ' , 100 - np.mean ( Model.Error_Percentage ) )
```
```
The mean square value of error test is  91259520.16870753

The mean square value of error train is  77776789.34088856

The root mean square value of error test is  9552.984882679733

The root mean square value of error train is  8819.114997599734

The mean absolute percentage error (of test) is  6.141679767019481

The mean absolute percentage error (of train) is  11.677247076702795

Accuracy on train data is :  93.85832023298052

Accuracy on test data is :  88.3227529232972
```

In [ ]: