Linear Regression With Python.

What is Regression:-

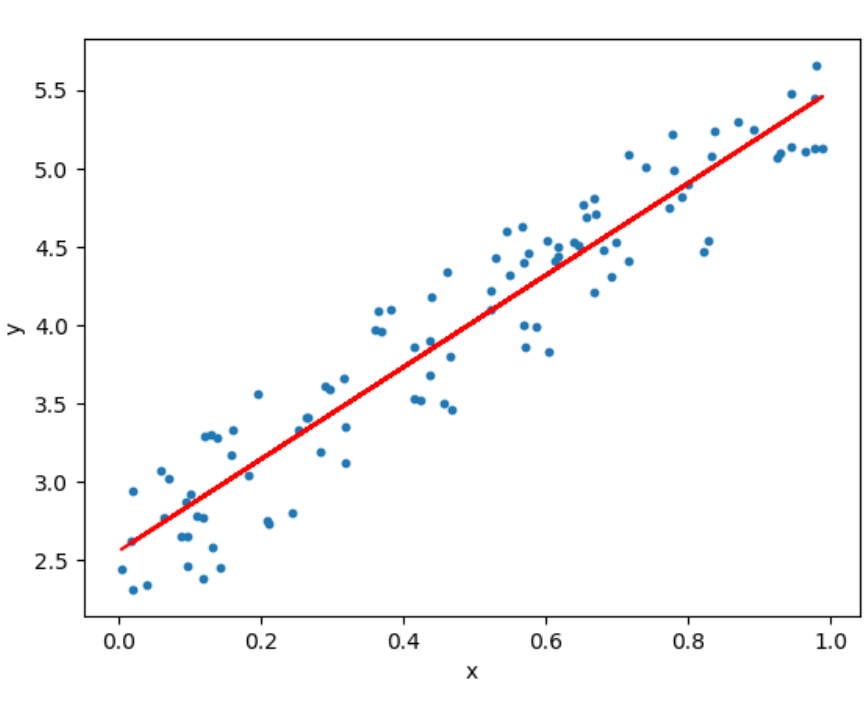
- Regression is a method of modelling a target value based on independent predictors. This method is mostly used for forecasting and finding out cause and effect relationship between variables.
- Regression techniques mostly differ based on the number of independent variables and the type of relationship between the independent and dependent variables.

Linear Regression :-

- Simple linear regression is a type of regression analysis where the number of independent variables is one and there is a linear relationship between the independent(x) and dependent(y) variable. The red line in the below graph is referred to as the best fit straight line. Based on the given data points, we try to plot a line that gives us best fit line with least cost function or else we can say an error between the actual and predicted values. The line can be modelled based on the linear equation shown below.
- The motive of the linear regression algorithm is to find the optimal or best values for β0 and β1.
- Before moving on to the algorithm, let's have a look at two important concepts you must know to better understand linear regression.

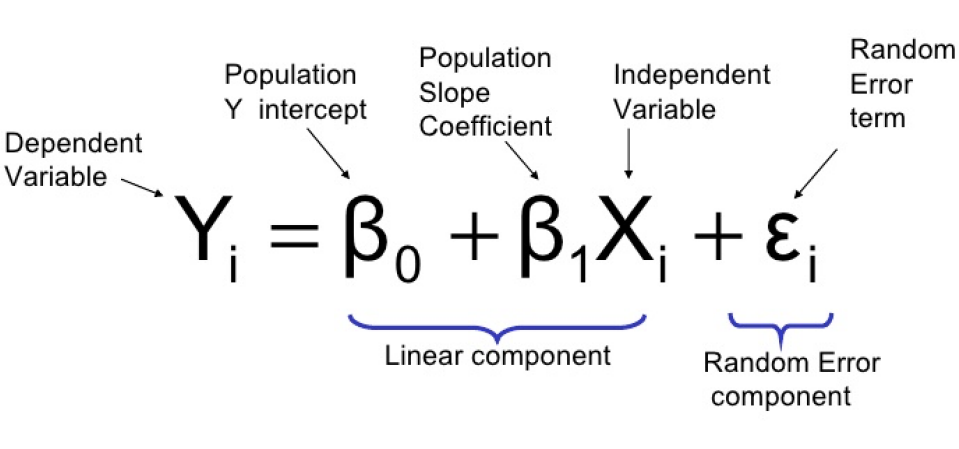
In [6]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/line.png',height=500,width=500)
```

Out[6]:



In [7]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/equation.png',height=500,width=500)
```

Out[7]:



Cost Function:-

- The cost function helps us to figure out the best possible values for β0 and β1 which would provide the best fit line for the data points. Since we want the best values for β0 and β1.
- Algorith initailly needs to convert this randomly allocated values of β0 and β1 into their minima.It means we want the best values of these coeficient at optimal value where our model will gives us at its best.That means we will reduce the error between the predicted and actual data points.
- The difference between the predicted values and ground truth measures the error difference. We square the error difference and sum over all data points and divide that value by the total number of data points.
- This provides the average squared error over all the data points. Therefore, this cost function is also known as the Mean Squared Error(MSE) function. Now, using this MSE function we are going to change the values of β0 and β1 such that the MSE value settles at the its minima.

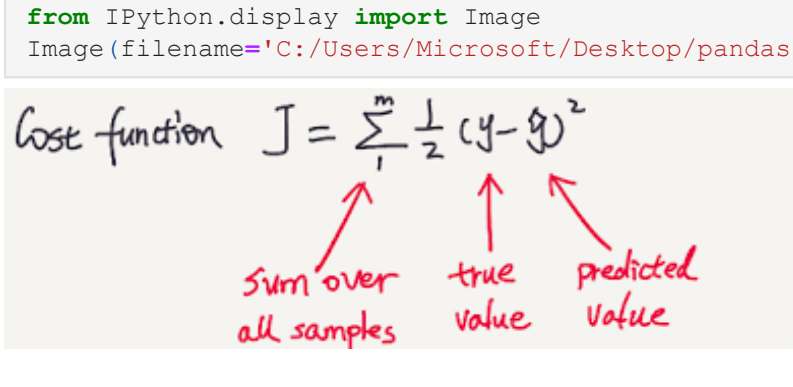
In [10]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/images.png',height=300,width=300)
```

Out[10]:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

In [14]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/images (1).png',height=400,width=400)
```

Out[14]:

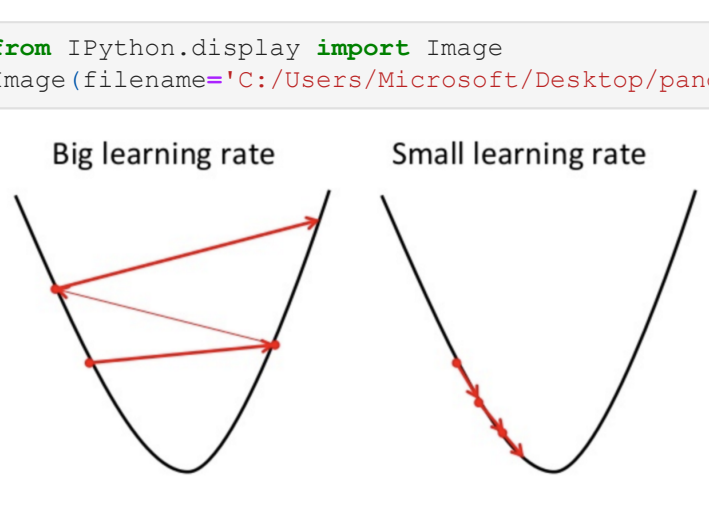


Gradient Descent:-

- The next important concept needed to understand linear regression is gradient descent. Gradient descent is a method where algorithm is getting updating their values of linear regression coeficient β0 and β1 gradually with ceratin learning rate in order to get the relationship between the features and settle down the value of regression coeficient at its best value where our model will perform at its best in other words it is going to reduce the cost function(MSE).
- The idea is that Algorithm will start intially with start with some random values of β0,β1,..βn..and son on and then algorithm change these values iteratively to reduce the cost function.
- Whenever we have high learnig rate that time the model is highly unreliable in terms of prediction because it learned in zig-zag way and not train itself on all patterns avaliable inside the data correctly.
- Whereas,if we have very small learning rate then it tries figure out each pattern avaliable data but it will stop learning soon before reaching to the end of data.
- Thats means we also want the best value of learning rate as well because it is going to help for understanding the pattern inside the data at its best with minina of cost function.
- Sometimes the cost function can be a non-convex function where you could settle at a local minima but but in case of linear regression, it is always a convex function.
- `A smaller learning rate could get you closer to the minima but takes more time to reach the minima, a larger learning rate converges sooner but there is a chance that you could overshoot the minima.`

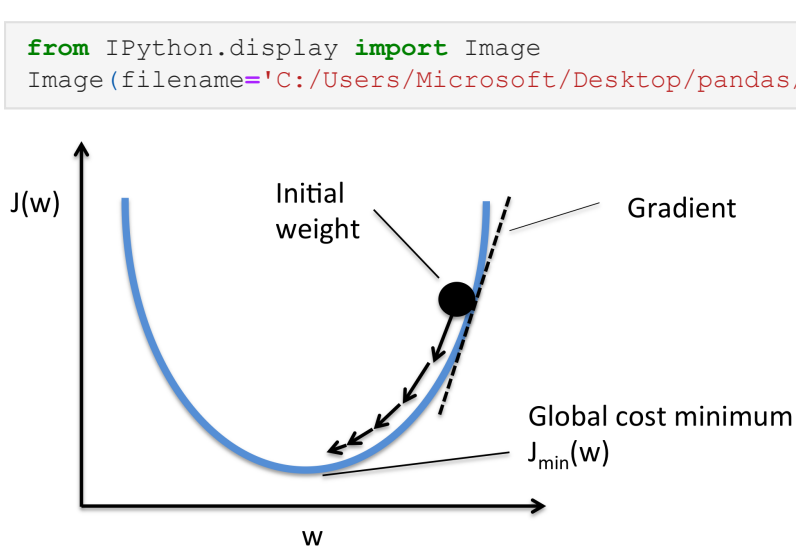
In [15]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/big.png',height=400,width=400)
```

Out[15]:



In [16]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/ball.png',height=400,width=400)
```

Out[16]:



Code :-

- We use pandas library to read the train and test files. We retrieve the independent(x) and dependent(y) variables and since we have only one feature(x) we reshape them so that we could feed them into our linear regression model.
- We use scikit learn to import the LinearRegression model. we fit the model on the training data and predict the values for the testing data. We use R2 score to measure the accuracy of our model.

Visualized our data:-

- By using the pairplot we can we visualized our data to get the perfect behaviour.

Example

1. House Price Prediction on Booston Housing Datasets.

- You are the real estate agent and you want to predict the house price.It would great if you can make some kind of automated system which can predict the price of the house based on various input which is known as features.
- Supervised Machine Learning Algorithm needs some data to train the model before making any prediction.
- We have a Bostan Datasets.

Where can The Linear Regression be used ?

- It is very powerfull techiques and can be used to understand facts that can understand the probablity.It can be used to predict forecast sales in the upcoming months by analyzing the sales data of the previous months.It can used gain various insight about the customer behaviour.
- Here we are having the values of the `x` which are the independent to each other and `y` is the dependedt variable .
- `y` used to calaculate with the help of `x.`
- Here `x (Independent variable)` and `y(dependent variable)` should always in the form of **continuous data** in order to predict.
- So,that we can say y is function of x
- `y = f(x)`
- For an exmaple:- If we having the housing price prediction,In that case the price of the house is on the `y-axis` dependent variable and rooms,blocks,halls and other thing that would be count in `x-axis` that is independet variable.

Types Of Linear Regression

- There are to kinds of regression
  - Simple linear regression.
  - Multiple regression.

Simple linear regression

- Let's suppose if we go with the single input linear regression.
- In that we having the single input on the `x-axis` and according this input we will get the straight line and it will give us the predited value of house.
- Once,the line will fit in the linear regression then we will found what is goodness of the fit.
- That means how well your model will behave on unseen data.
- It behave good or it produce more accuracy or good mse reduction but it might not behave good on the unseen data.
- So,we define goodness of the fit on the unseen data and our goal is mimizes the total error and the total square error is a mean square error or mean absolute error.
- That can mininmize by using the Gradient decent.

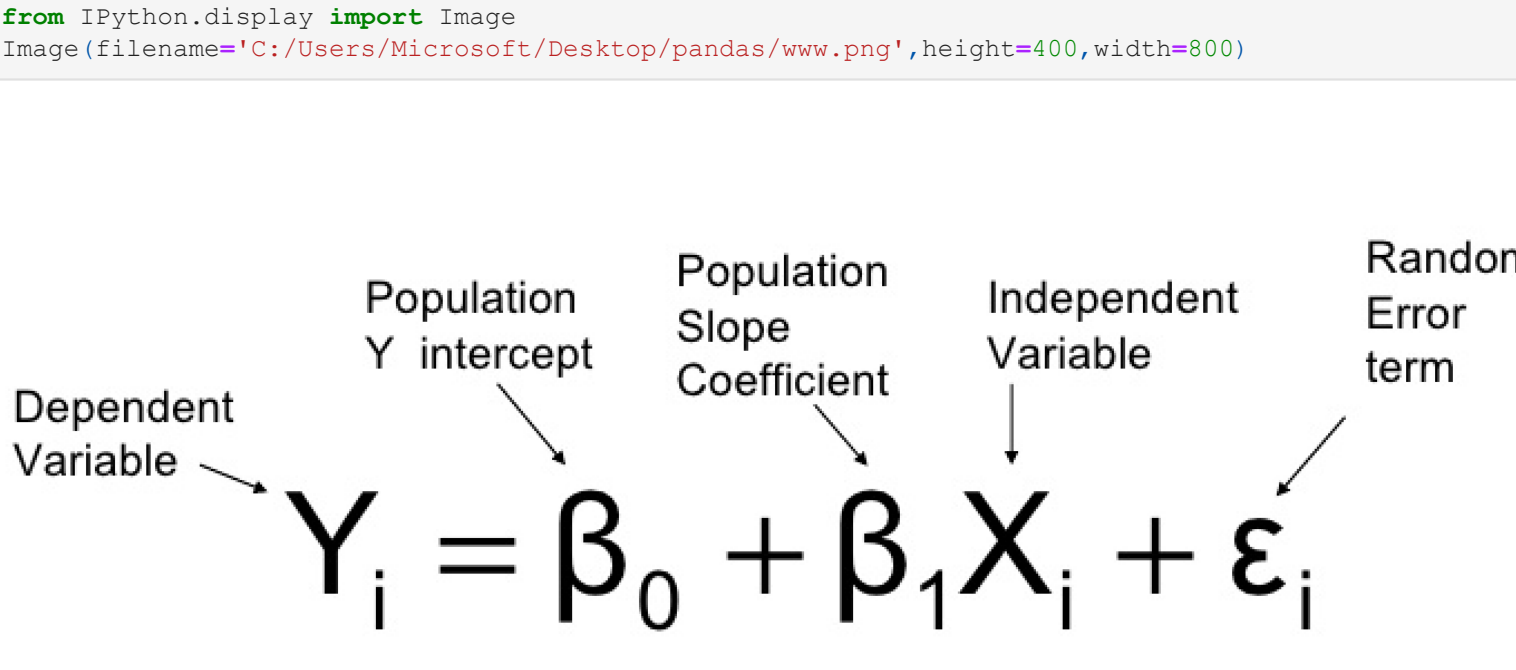
What gradient decent do in that ?

- We start fitting our regression line in simultaneously with our data and we will calculate the mean squared error with this fitted line.
- If error is in high amount then it will look foreward again for getting the best position of itself inside the data where it can get the minimum error between predicted and actual values.
- It is happening until and unless we it will not get the best possible regression line or best fit line where cost function at its minima.

About the Line.

In [19]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/www.png',height=400,width=800)
```

Out[19]:

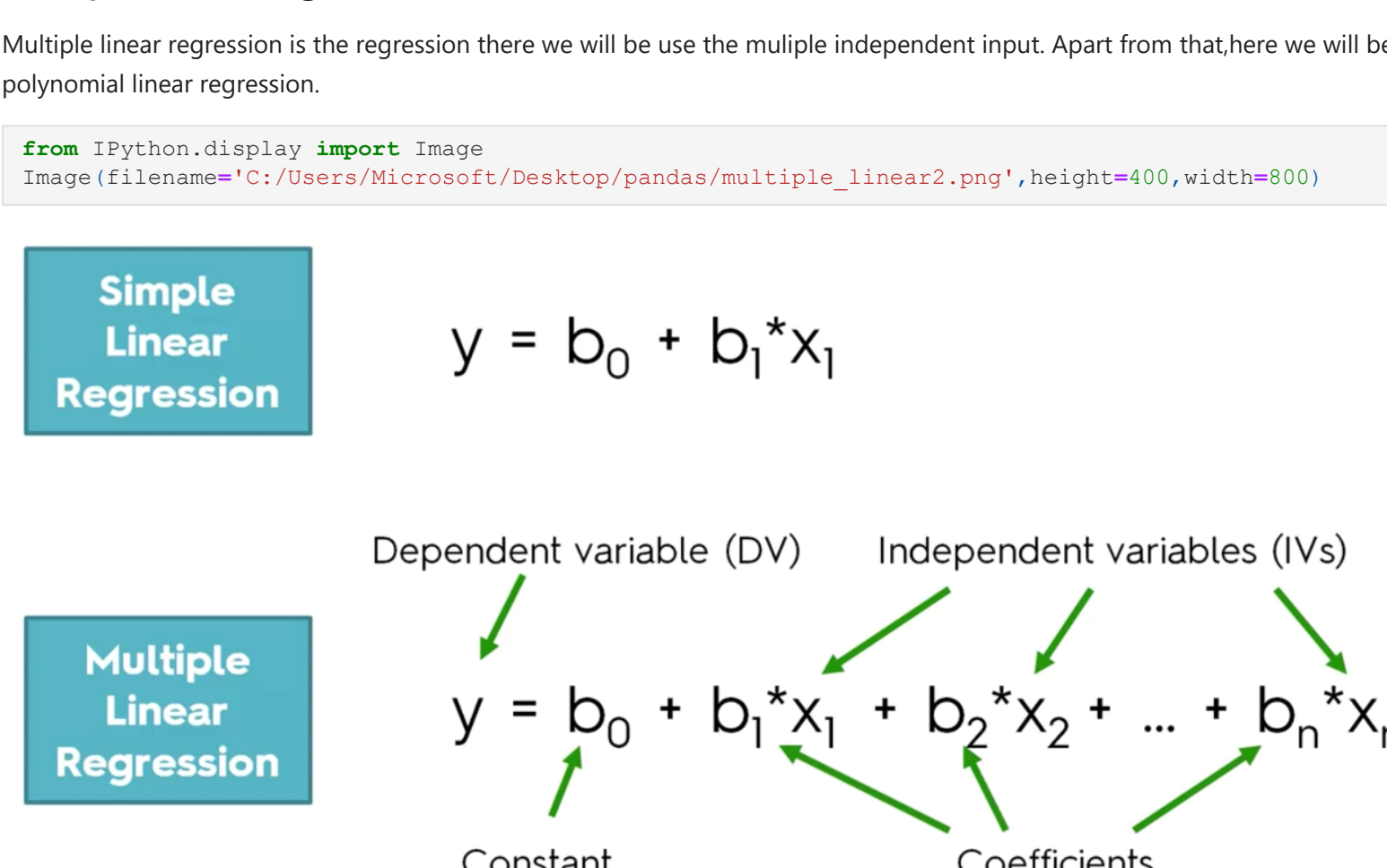


Multiple Linear Regression.

Multiple linear regression is the regression there we will be use the muliple independent input. Apart from that,here we will be use the polynomial linear regression.

In [20]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/multiple_linear2.png',height=400,width=800)
```

Out[20]:



Plane Fitting On the Linear Regression.

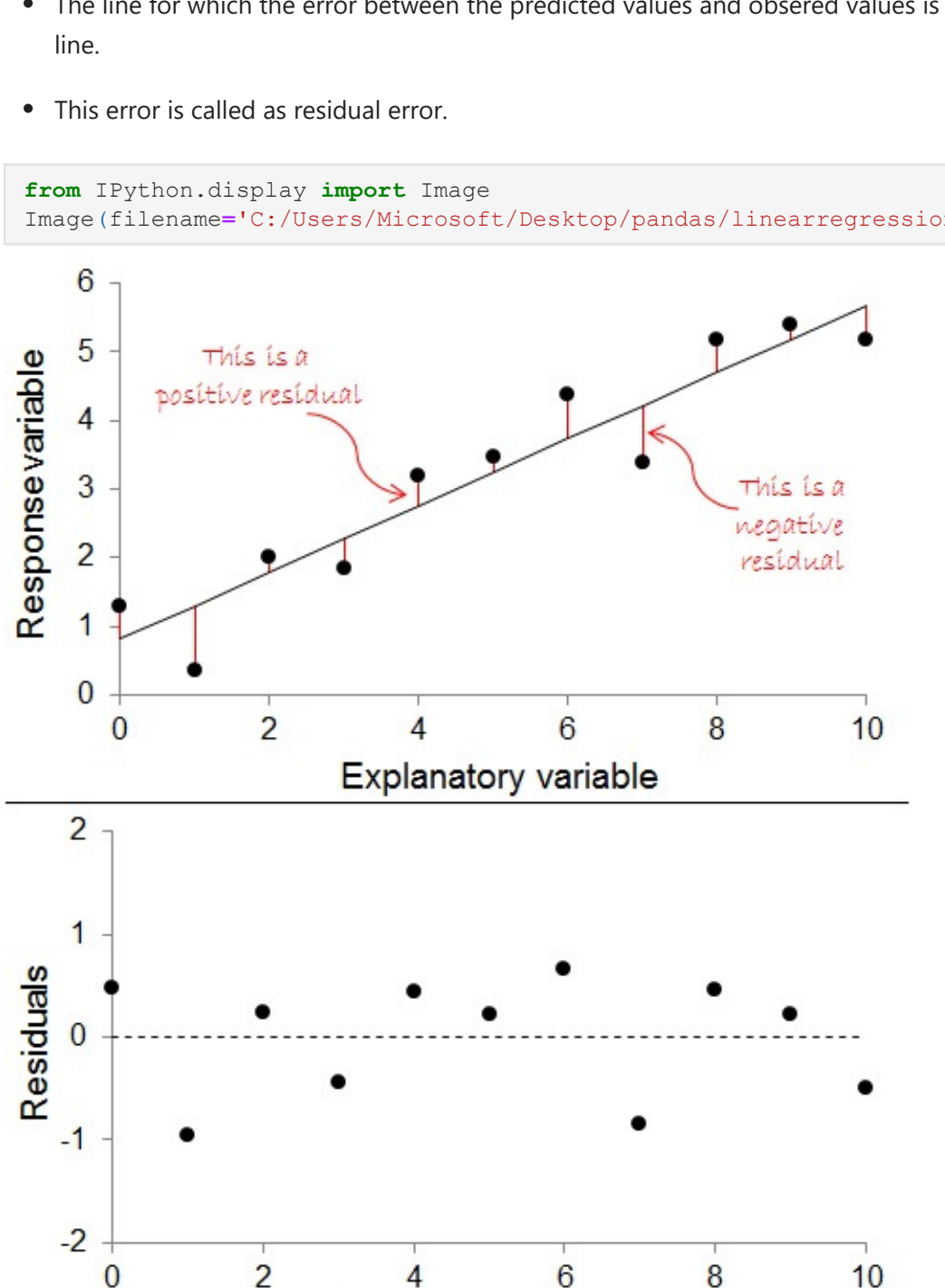
- If we use the multiple regression input In that case it will be use the plane fitting not the line fitting.Apart from the we can use the multi-diamensional data more than **x1 and x2**.like of the number of bedrooms,Lot Size etc. as input to our model.
- That will become multiple linear regression.

Assessing the Performace of the model.

- Algorthm calculate the vertical distance from the regression line to the data points and determine which data points are close or away from the that line.
- The diffrence between them we called it as residual.
- The line for which the error between the predicted values and obsered values is minimum is called the best fit line of the regression line.
- This error is called as residual error.

In [24]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/linearregression_re.jpg',height=400,width=500)
```

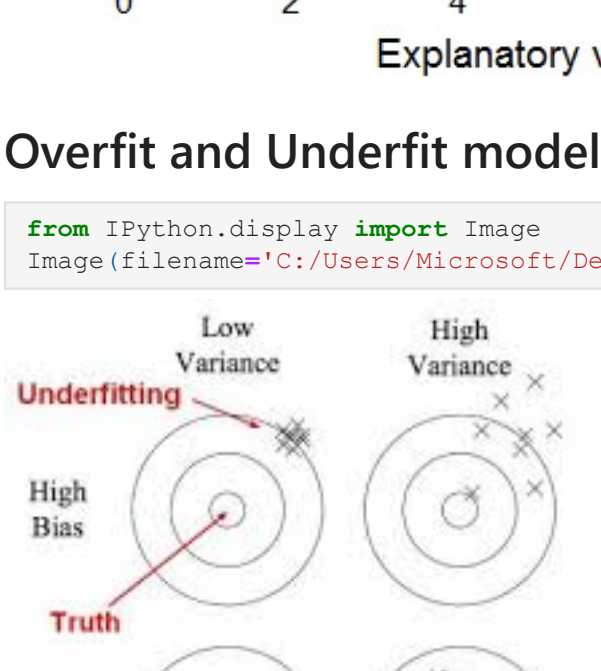
Out[24]:



Overfit and Underfit model

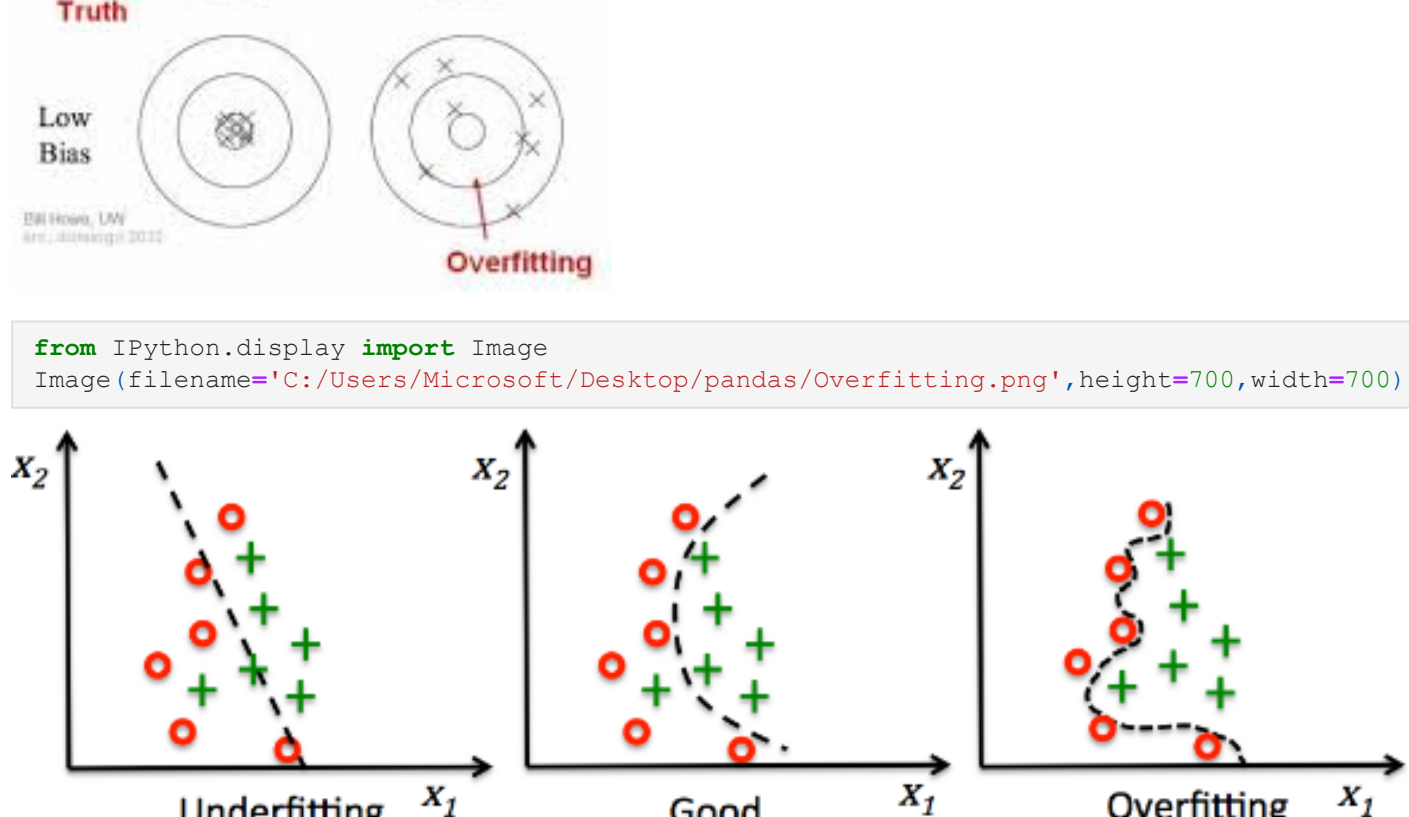
In [31]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/images.jpg',height=400,width=300)
```

Out[31]:



In [36]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/Overfitting.png',height=700,width=700)
```

Out[36]:



Underfitting Model:-

- In this kind of model data will be neither train properly nor preddict properly.It means it having the lower variance and high bias in the data.
- Underfitting model will gives us the bised prediction.
- It will not learn all data properly due to lack of evidance so for getting good prediction we required more data.
- Underfitting is a scenario in data science where a data model is unable to capture the relationship between the input and output variables accurately.
- Generating a high error rate on both the training set and unseen data.

Overfitting Model:-

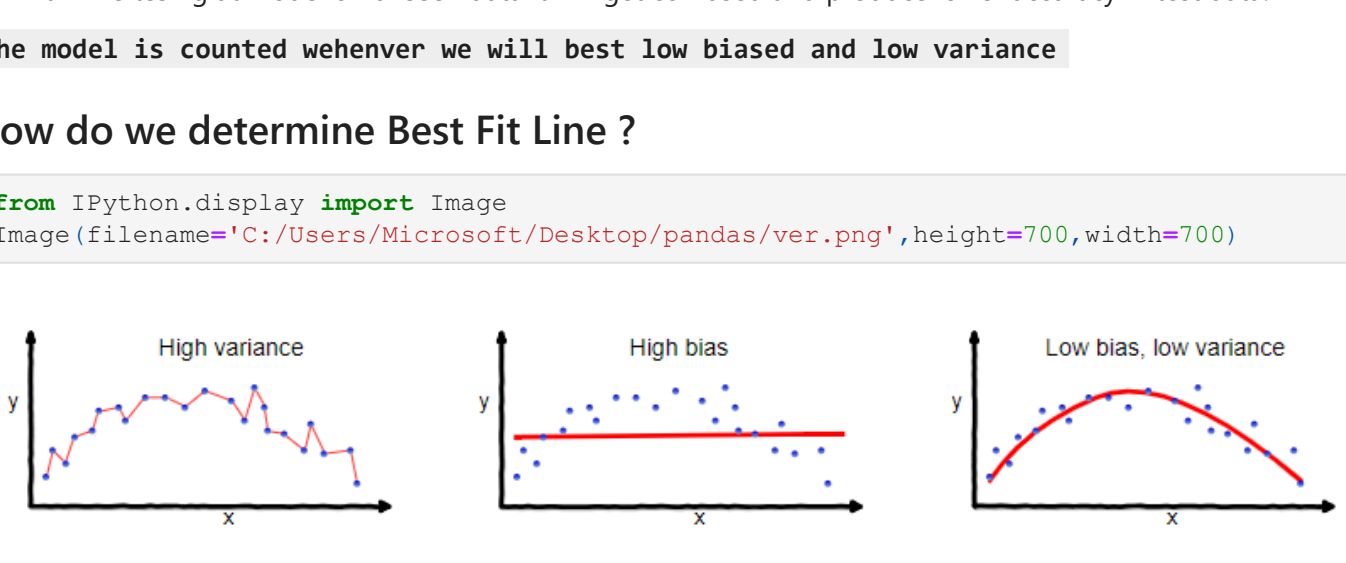
- In this kind of model will be train on data properly but during the tarining,while testing that model on the unseen data it won't be give us better prediction.It means that there could be the high variance and low bias.
- Due to high diversity inside the data or we can say muti-colilearity inside the data model will get train on all features.
- At while tesing at model on unseen data it will get confused and produce lower accuracy in test data.

`The model is counted wehenver we will best low biased and low variance`

How do we determine Best Fit Line ?

In [37]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/ver.png',height=700,width=700)
```

Out[37]:



- If we observed that whenever the data is not properly fit on the best fit line that time our model complexity get deteriorates but for those data which is strongly fitted in model there we will found high model complexity.
- So,from we can say when we high model complexity we will get the overfitting of our data.
- If we have low model complexity we will get the underfitting of our data.

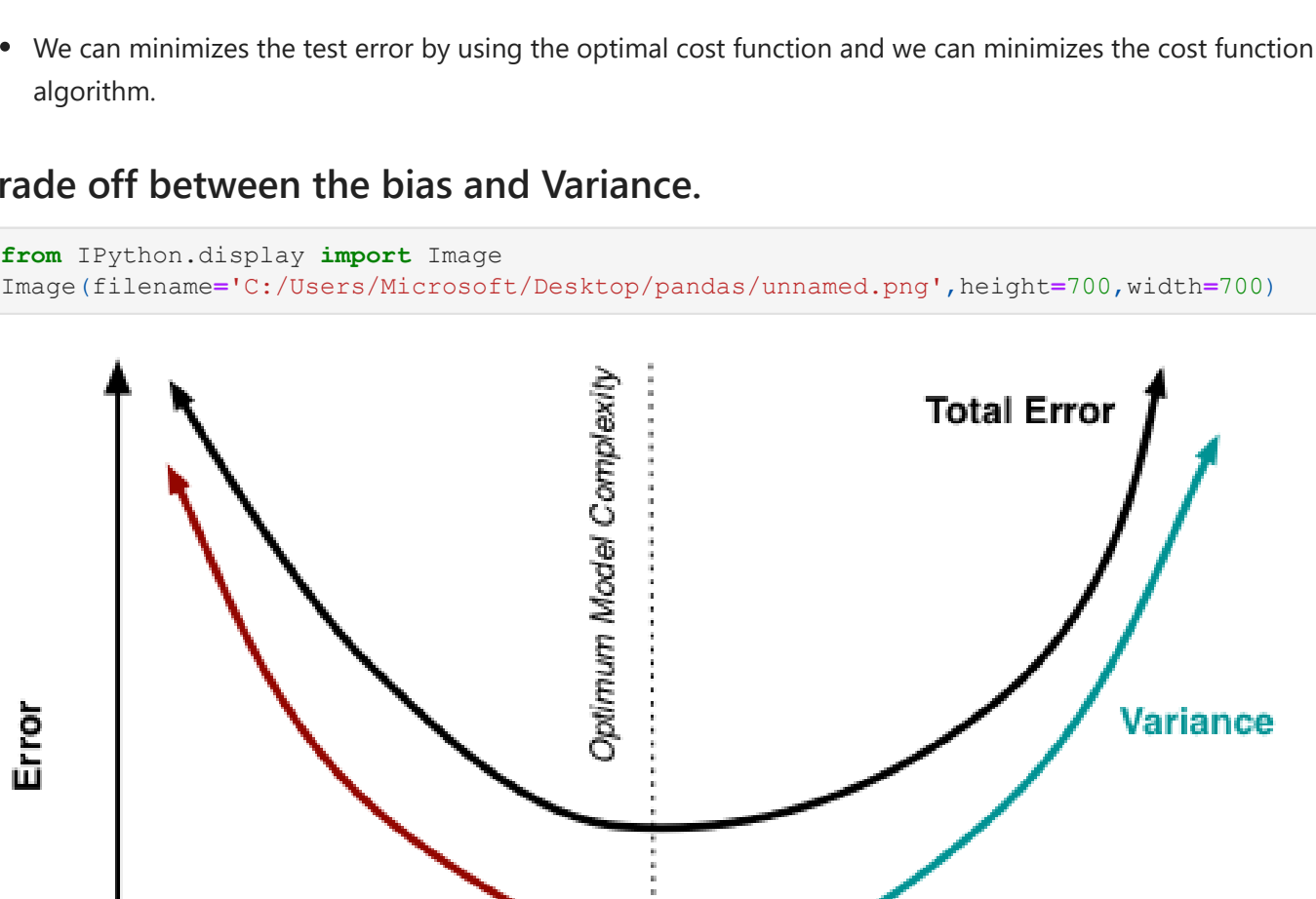
There is some kind of trade off between these.

- When the modle complexity increase that time the training data's error got decreasing but the testing datasets error started getting increses vice versa.
- For that we have to decide the minimun spot or sweet spot there test error would be the low.
- It is alsways recommanded that we should always focus on test error.
- We can minimizes the test error by using the optimal cost function and we can minimizes the cost function with the gradient decent algorithm.

Trade off between the bias and Variance.

In [38]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/unnamed.png',height=700,width=700)
```

Out[38]:



- While modeling, we will get to know about the Bias and Variance trade off.
- If our Model complexity will low then data variance will be low and bias will be high because there is no any high scattered data but there is high biased.
- But in the case of the high complexity there we having the low bias and the high variance that means there is data are in at random or highly scattered fashion so fitting the model became complex.
- When the variance get increase that time the biase gets decrese.There is inverse proportion between the Bias and variance.
- When the variance is low that time our model complexity is also low and vice versa because model does not required so much time to train the model because euclidean distance is not much high between the datapoints.
- So,model will tried have to find the sweet spot on that spot the our bias and variance get the settle down to the mimimun value,miminum error (mean square error) on the test datasets with optimal model complexity.

Implementation With Sklearn.

- It is simple tool to implement on the data mininig and data analysis.
- It build on numpy,SciPy and matplotlib.
- To get the detail documentation :-https://scikit-learn.org/
- To get the all deatails about the regression,classification,clusering :-https://cdn-images-1.medium.com/max/2400/1*2NR51X0FDjLB13u4WdYc4g.png
- To get the installation we have use the
- !pip install -v scikit-learn

Training and Testing Splitting.

- Best training and testing spliting is 60% for the training and the 25 % testing and 15 % for the cross validation if in case if we don't want to do validation on the datasets that time take the training datasets 80% and 20 % testing datasets.

Introduction & Summery of Linear Regression.

- **Linear Regression**
- Regression means prediction dependent on the basis on independent variable.
- In the linear Regression the value should be continuous format.
- In that will use the Eucleadean distance to measure the error (eror is the perpandicular distance from the base fit line or Regression Line)
- It will minimized by using the StandarScaler or gradient indicent algorthm to find global minima there we will get the least eror or root mean squared error.
- **Kinds Of Regression.**
- There are two types of LinearRegression
- 1.Simple Regression
- 2.Mupliple regression i.e.Polynomial Regression.
- Both uses the linear relationship by using the straight line equation $y = mx+c$
- **Bias-Variance Trade Off**
- Bias:- It makes the prediction away from actual with low variance.
- Variance :- It is dispursion of the actual datapoint away from the mean of that datapoint.
- Bias and Variance are mutually invesly proportional to each other.
- If Bias get increase the Variance get decrease voice versa.
- whenver the bias is high and variace is low on that time model complexity got decreases but due to this reason model undefitted that it either proprly fit on model nor predict the actual of our data.
- Whenver the Bias get decrease and Variance get increse that time our model get confused due to inctrease in the model complexity and model become Overfitted .It means that model train on each and every datapoint of the datasets but failed to generlized the model either we say not predict in our favour.
- **Underfit and Overfit**
- Underfit - Whenevr the variance is low and bias is high that means it is Underfitted.
- Overfiited - Whenever the variance is high but the bias is low that time is Overfitted.fi
- **Identification Of Overfit & Underfit**
- Whenver the R2 score of training based fit model is greater than R2 score of testing based fit model that time we can say it is Ovefitted model or vice versa.
- **Errors**
- Mean Squared Error
- Mean Absolute error
- **Measure of Performance Count Of the Linear Regression**
- R2score
- **Assumptions Of Linear Regression**
- 1.Linearity - Graphical method regplot inbetween the residule and prediction.2nd Linear_rainbow method 3rd is the mean the residual is cloest to zero.
- 2.Normality - Joque bera test (pvalue=5.99<Pjoquebera) and Graphical method by using the distribution plot of residual.Here null hypothesis is 5.99.
- 3.Homocidasticity - Graphical method lmplot inbetween the residul and prediction and Goldfread test or Bust wagon test.Here the pvalue of the null hypothesis is 0.05
- 4.No Autocorrelation- durbin-watson-test (0-2(possitive Auto-correlation) and 2 is no autocorrelation and 2 to 4 (Negetive Auto-correlation) and
- 5.No Muticolinearity - Variance Influence factor.
- **Feature Selection Done On The Basis Of Highibe Correlation of the Independent with Dependedent Variable**

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
data= pd.read_csv('50_Startups.csv')
```

In [3]:
```
data.head()
```

Out[3]:

	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

In [4]:
```
data.tail()
```

Out[4]:

	R&D Spend	Administration	Marketing Spend	State	Profit
45	1000.23	124153.04	1903.93	New York	64926.08
46	1315.46	115816.21	297114.46	Florida	49490.75
47	0.00	135426.92	0.00	California	42559.73
48	542.05	51743.15	0.00	New York	35673.41
49	0.00	116983.80	45173.06	California	14681.40

In [5]:
```
data.rename(columns={'R&D Spend':'R&D','Marketing Spend':'Marketing'},inplace=True)
```

In [6]:
```
data.head()
```

Out[6]:

	R&D	Administration	Marketing	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

In [7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
R&D               50 non-null float64
Administration    50 non-null float64
Marketing         50 non-null float64
State             50 non-null object
Profit            50 non-null float64
dtypes: float64(4), object(1)
memory usage: 1.8+ KB
```

In [8]:
```
data.describe()
```

Out[8]:

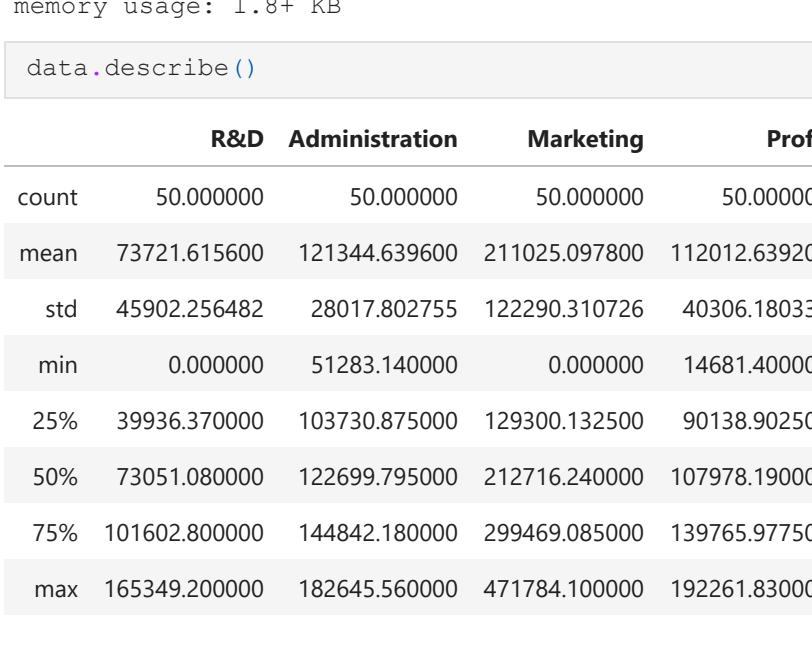
	R&D	Administration	Marketing	Profit
count	50.000000	50.000000	50.000000	50.000000
mean	73721.615600	121344.639600	211025.097800	112012.639200
std	45902.256482	28017.802755	122290.310726	40306.180338
min	0.000000	51283.140000	0.000000	14681.400000
25%	39936.370000	103730.875000	129300.132500	90138.902500
50%	73051.080000	122699.795000	212716.240000	107978.190000
75%	101602.800000	144842.180000	299469.085000	139765.977500
max	165349.200000	182645.560000	471784.100000	192261.830000

In [9]:
```
data.isnull().sum()
```

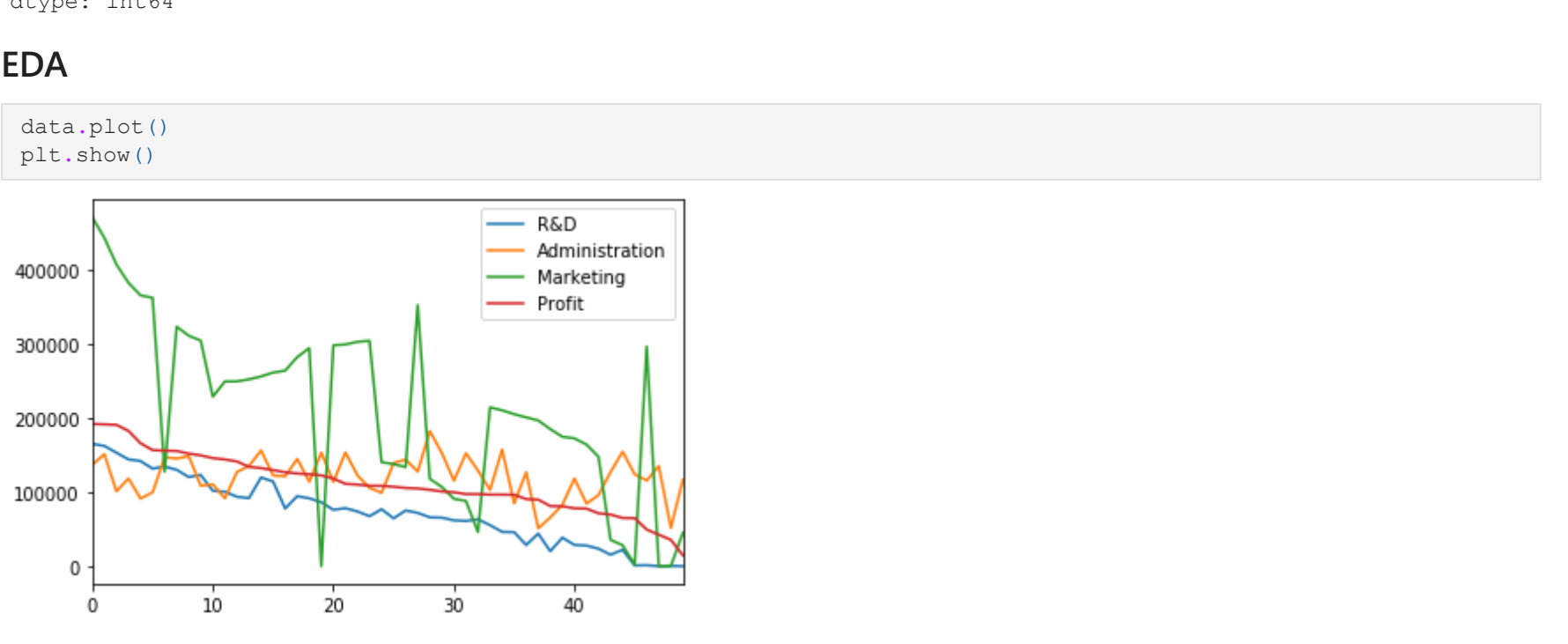
Out[9]:
```
R&D               0
Administration    0
Marketing         0
State             0
Profit            0
dtype: int64
```

EDA

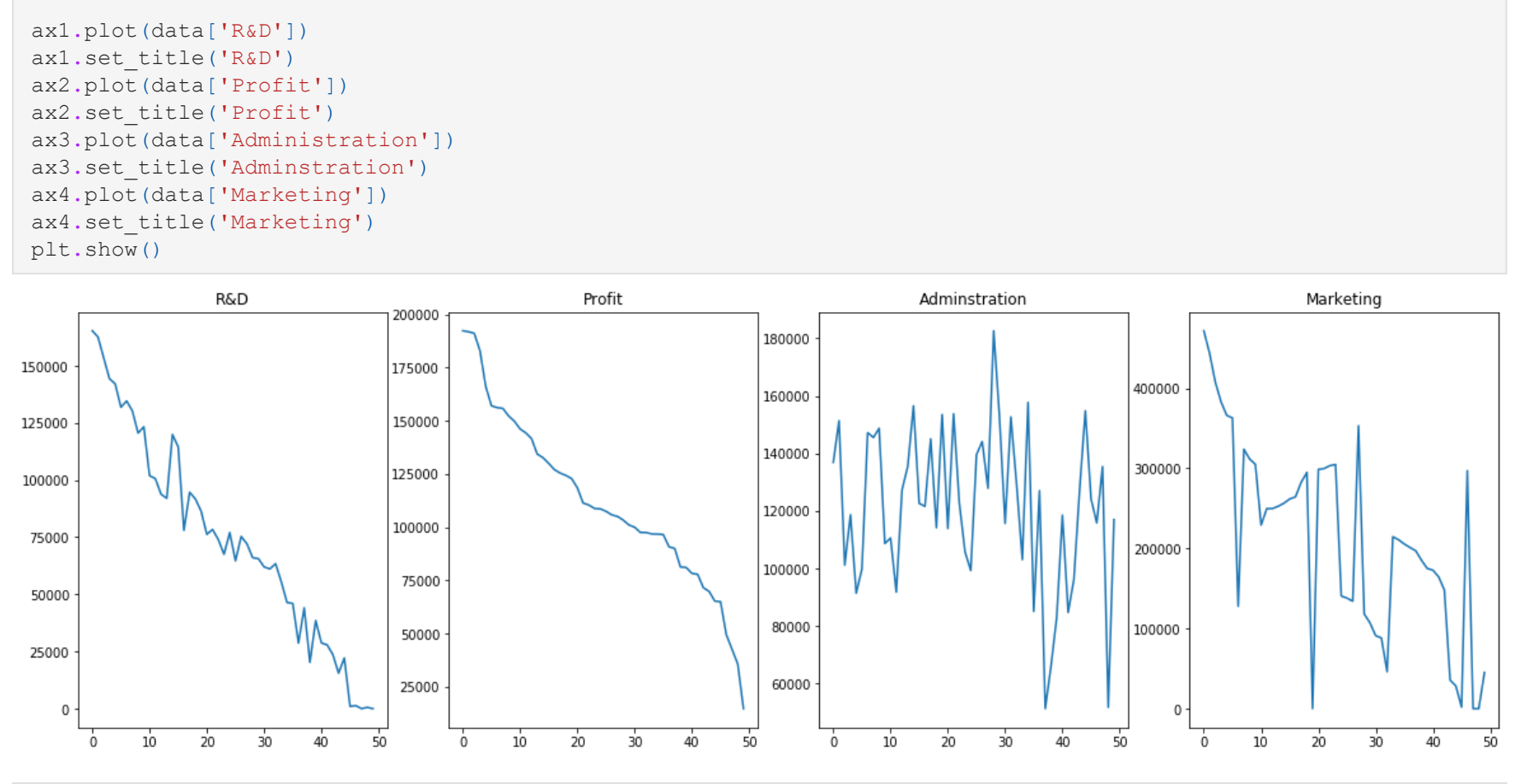
In [10]:
```
data.plot()
plt.show()
```



In [11]:
```
fig,[ax1,ax2,ax3,ax4]= plt.subplots(ncols=4,nrows=1,figsize=(20,6))

ax1.plot(data['R&D'])
ax1.set_title('R&D')
ax2.plot(data['Profit'])
ax2.set_title('Profit')
ax3.plot(data['Administration'])
ax3.set_title('Adminstration')
ax4.plot(data['Marketing'])
ax4.set_title('Marketing')
plt.show()
```



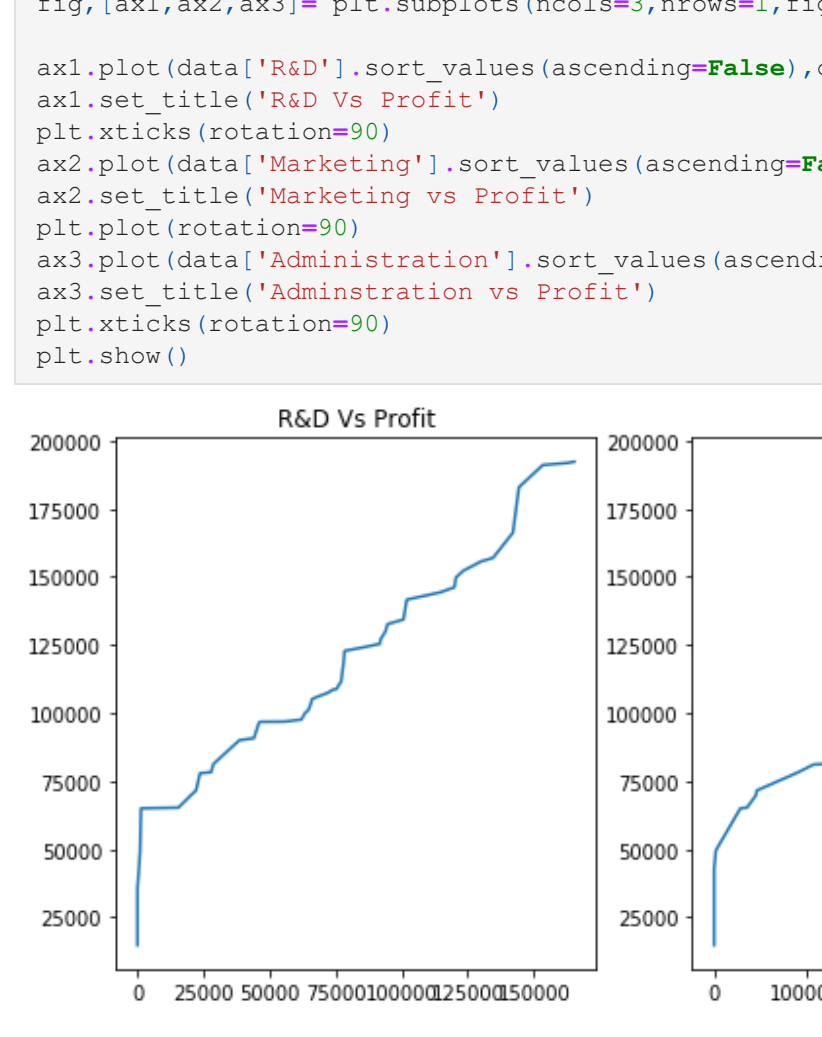
In [12]:
```
fig,[ax1,ax2,ax3]= plt.subplots(ncols=3,nrows=1,figsize=(15,5))

ax1.plot(data['R&D'].sort_values(ascending=False),data['Profit'].sort_values(ascending=False))
ax1.set_title('R&D Vs Profit')
plt.xticks(rotation=90)
ax2.plot(data['Marketing'].sort_values(ascending=False),data['Profit'].sort_values(ascending=False))
ax2.set_title('Marketing vs Profit')
plt.plot(rotation=90)
ax3.plot(data['Administration'].sort_values(ascending=False),data['Profit'].sort_values(ascending=False))
ax3.set_title('Adminstration vs Profit')
plt.xticks(rotation=90)
plt.show()
```



In [13]:
```
data['State'].value_counts()
```

Out[13]:
```
California    17
New York      17
Florida       16
Name: State, dtype: int64
```

In [14]:
```
sns.countplot(data['State'])
plt.show()
```



In [15]:
```
data.groupby(data['State']).count()
```

Out[15]:

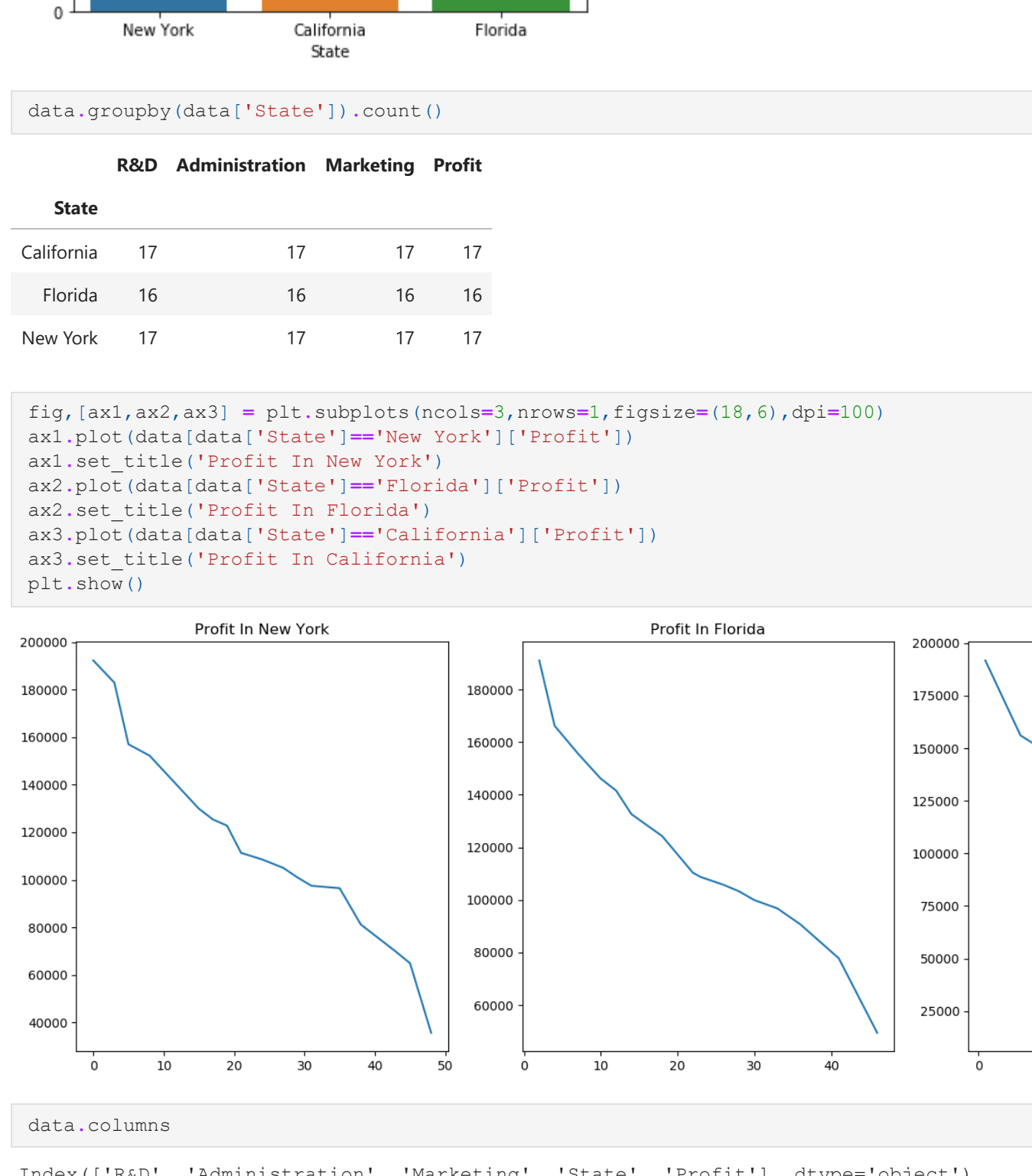
State	R&D	Administration	Marketing	Profit
California	17	17	17	17
Florida	16	16	16	16
New York	17	17	17	17

In [16]:
```
fig,[ax1,ax2,ax3] = plt.subplots(ncols=3,nrows=1,figsize=(18,6),dpi=100)
ax1.plot(data[data['State']=='New York']['Profit'])
ax1.set_title('Profit In New York')
ax2.plot(data[data['State']=='Florida']['Profit'])
ax2.set_title('Profit In Florida')
ax3.plot(data[data['State']=='California']['Profit'])
ax3.set_title('Profit In California')
plt.show()
```

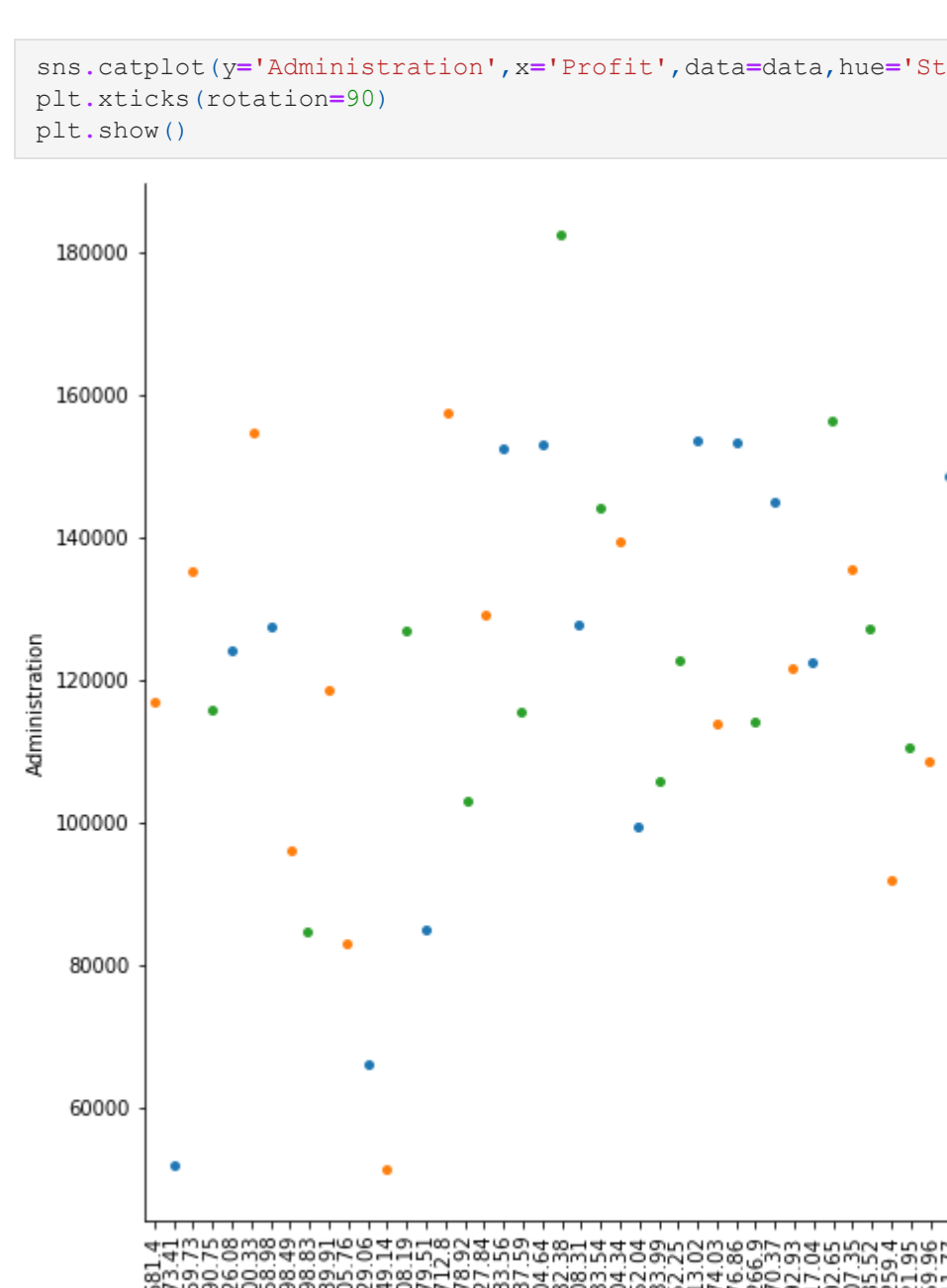
In [17]:
```
data.columns
```

Out[17]:
```
Index(['R&D', 'Administration', 'Marketing', 'State', 'Profit'], dtype='object')
```

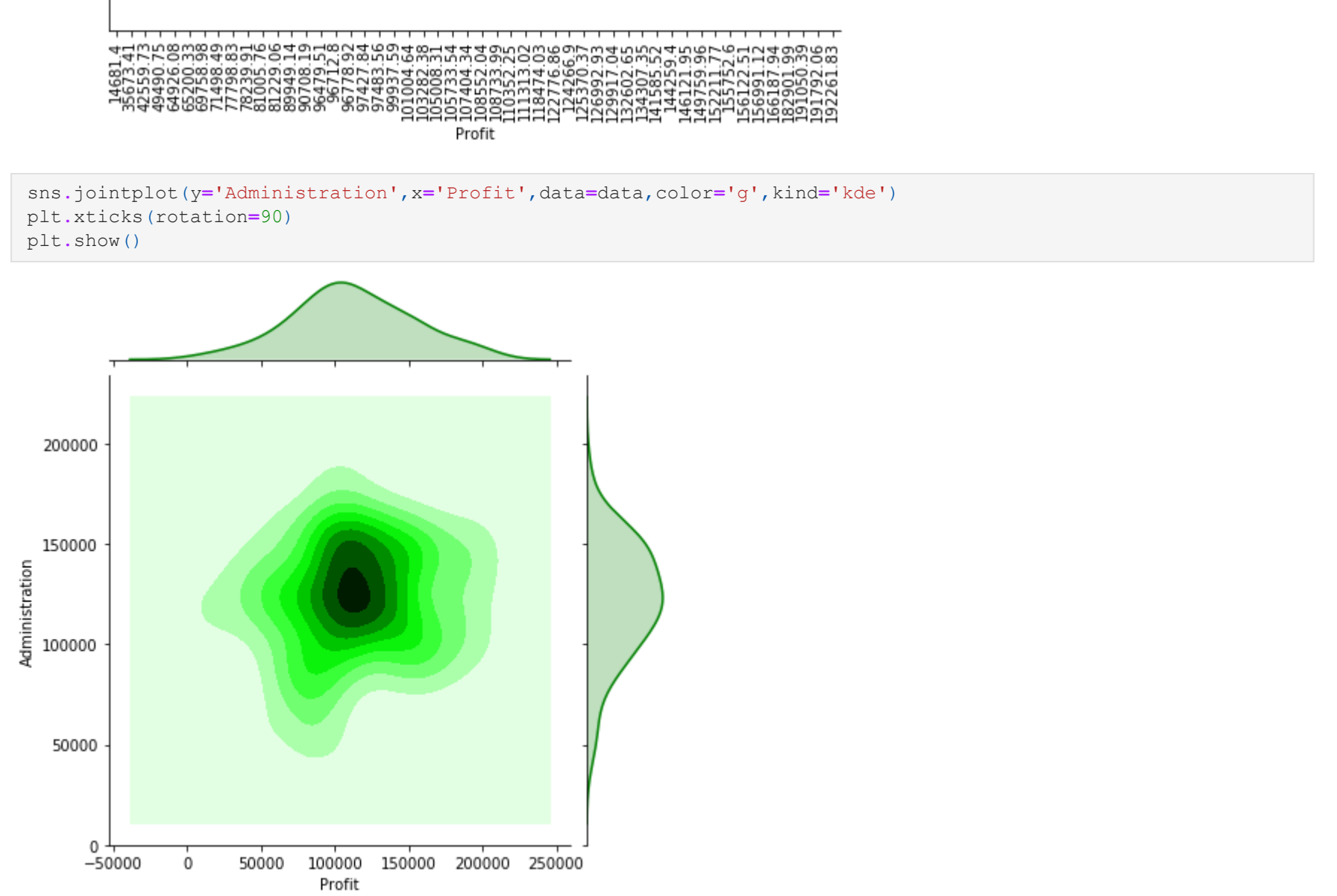
In [18]:
```
sns.catplot(y='Administration',x='Profit',data=data,hue='State',height=8)
plt.xticks(rotation=90)
plt.show()
```

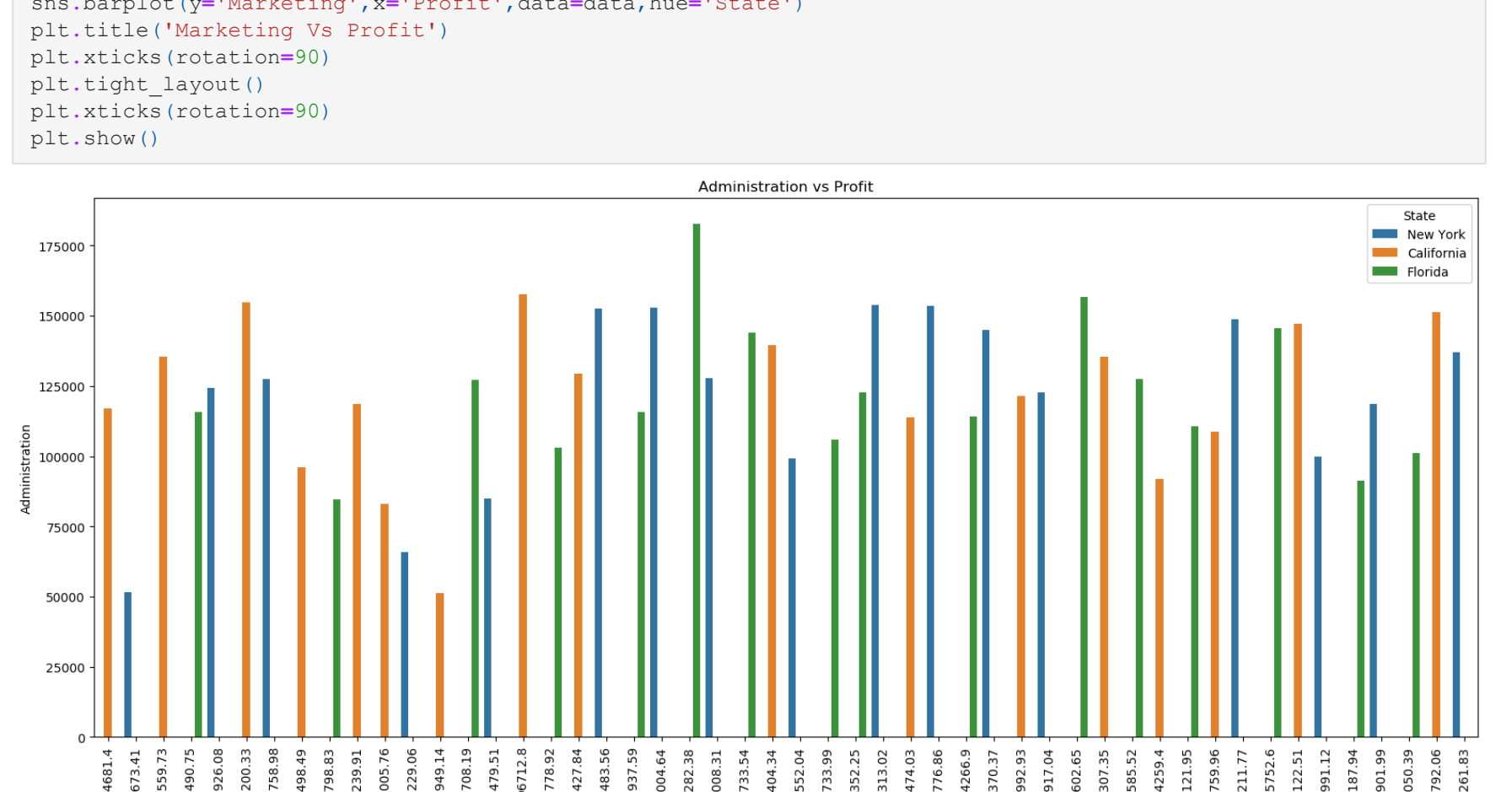


In [19]:
```
sns.jointplot(y='Administration',x='Profit',data=data,color='g',kind='kde')
plt.xticks(rotation=90)
plt.show()
```



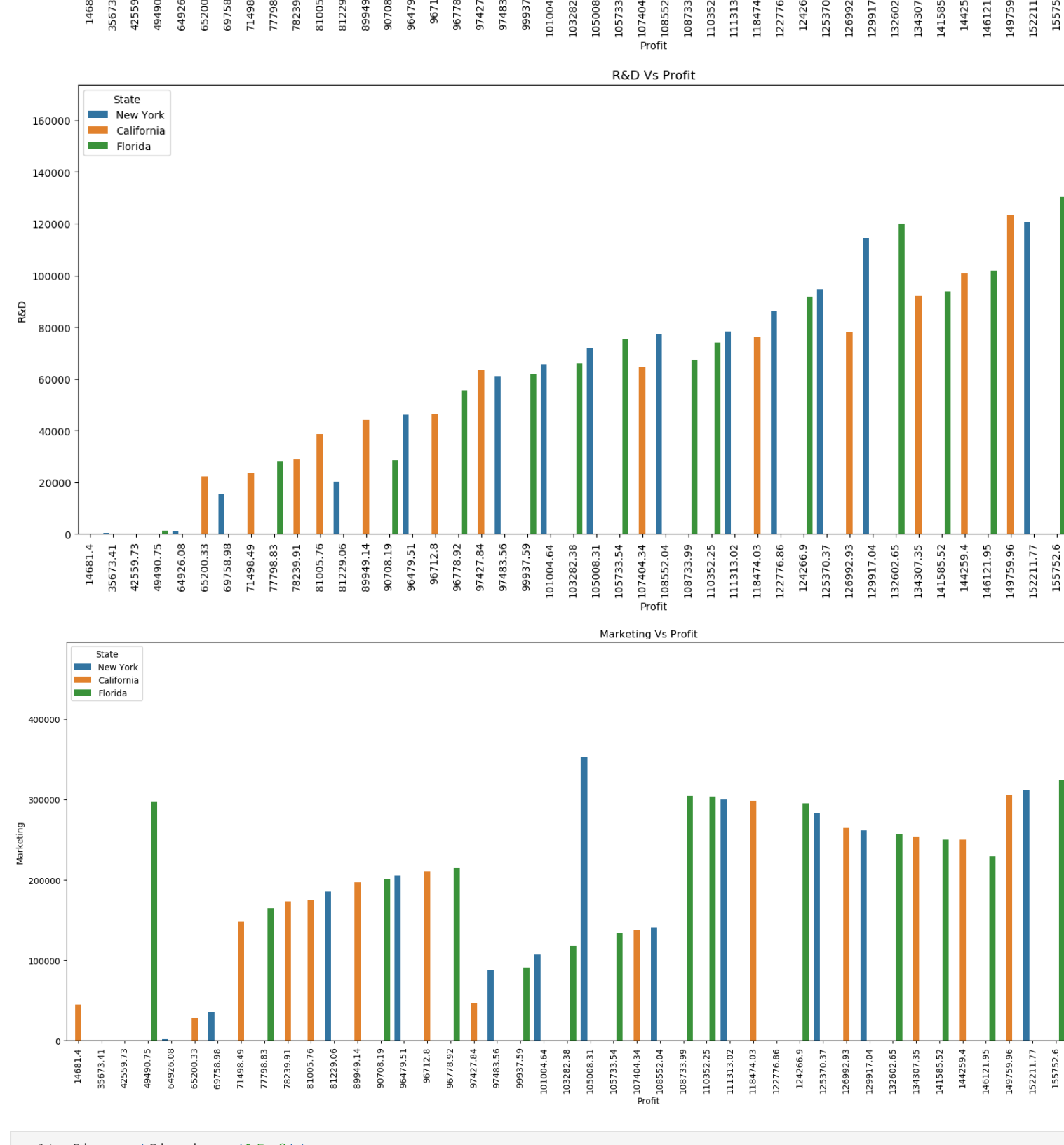
In [20]:
```
plt.figure(figsize=(20,8),dpi=100)
sns.barplot(y='Administration',x='Profit',data=data,hue='State')
plt.title('Administration vs Profit')
plt.xticks(rotation=90)
plt.show()
plt.figure(figsize=(20,8),dpi=100)
sns.barplot(y='R&D',x='Profit',data=data,hue='State')
plt.title('R&D Vs Profit')
plt.xticks(rotation=90)
plt.show()
plt.figure(figsize=(20,8),dpi=100)
sns.barplot(y='Marketing',x='Profit',data=data,hue='State')
plt.title('Marketing Vs Profit')
plt.xticks(rotation=90)
plt.tight_layout()
plt.xticks(rotation=90)
plt.show()
```





In [21]:
```
plt.figure(figsize=(15,8))
sns.pairplot(data)
plt.show()
```

```
<Figure size 1080x576 with 0 Axes>
```



In [22]:
```
df =pd.get_dummies(data['State'],drop_first=True)
dataframe = pd.concat([df,data],axis=1)
dataframe.drop(columns=['State'],inplace=True)
dataframe.head()
```

Out[22]:

	Florida	New York	R&D	Administration	Marketing	Profit
0	0	1	165349.20	136897.80	471784.10	192261.83
1	0	0	162597.70	151377.59	443898.53	191792.06
2	1	0	153441.51	101145.55	407934.54	191050.39
3	0	1	144372.41	118671.85	383199.62	182901.99
4	1	0	142107.34	91391.77	366168.42	166187.94

Start To Prepare Model

In [23]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score,mean_squared_error,median_absolute_error
from sklearn.preprocessing import StandardScaler
```

In [24]:
```
x=dataframe.drop(columns=['Profit'])
y=dataframe['Profit']
```

In [25]:
```
x = np.array(x)
y = np.array(y)
```

Train_test_split

In [26]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.80,random_state=42)
```

In [27]:
```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

In [28]:
```
model= LinearRegression()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
print('R2 score :',r2_score(y_test,y_pred))
```

```
R2 score : 0.8987266414319837
```

In [29]:
```
print('Slop OF the Line :-',model.coef_)
print()
print('Y intersection of line :-',model.intercept_)
```

```
Slop OF the Line :- [ 4.47775725e+02  3.27289103e+00  3.81022693e+04 -1.86475430e+03
  3.38617581e+03]

Y intersection of line :- 115651.72050000001
```

In [30]:
```
print('Accuaracy over the training data :-',model.score(x_train,y_train))
print('Accuaracy over the training data :-',model.score(x_test,y_test))
```

```
Accuaracy over the training data :- 0.9537019995248526
Accuaracy over the training data :- 0.8987266414319837
```

In [31]:
```
from sklearn.model_selection import cross_val_score
```

In [38]:
```
val_score_mean = cross_val_score(model,x_train,y_train,cv=10).mean()
val_score_mean
```

Out[38]:
```
0.9148232109588694
```

In [39]:
```
val_score_min = cross_val_score(model,x_train,y_train,cv=10).min()
val_score_min
```

Out[39]:
```
0.7064528129403105
```

In [40]:
```
val_score_max = cross_val_score(model,x_train,y_train,cv=10).max()
val_score_max
```

Out[40]:
```
0.9750119091746022
```

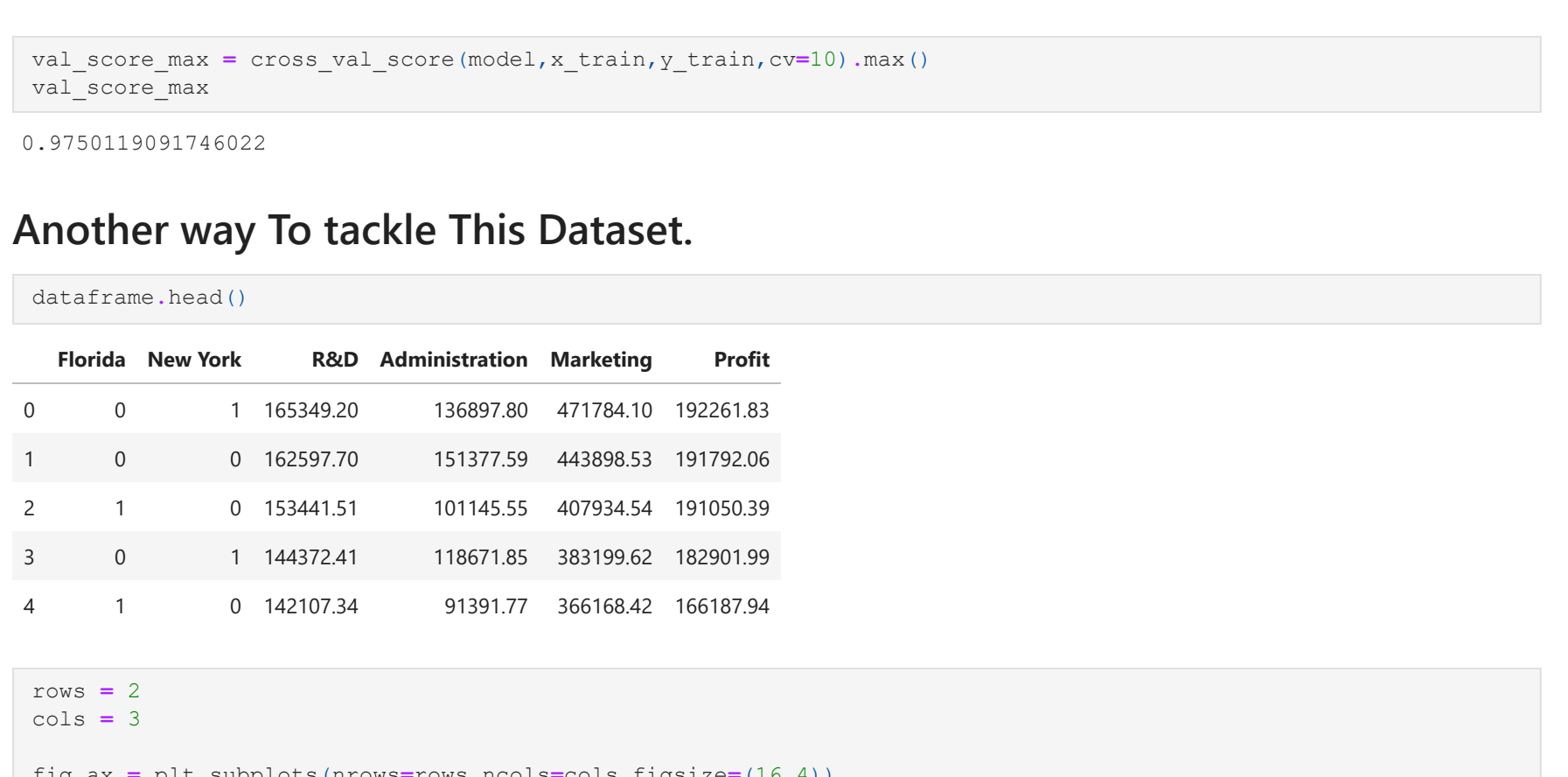
Another way To tackle This Dataset.

In [41]:
```
dataframe.head()
```

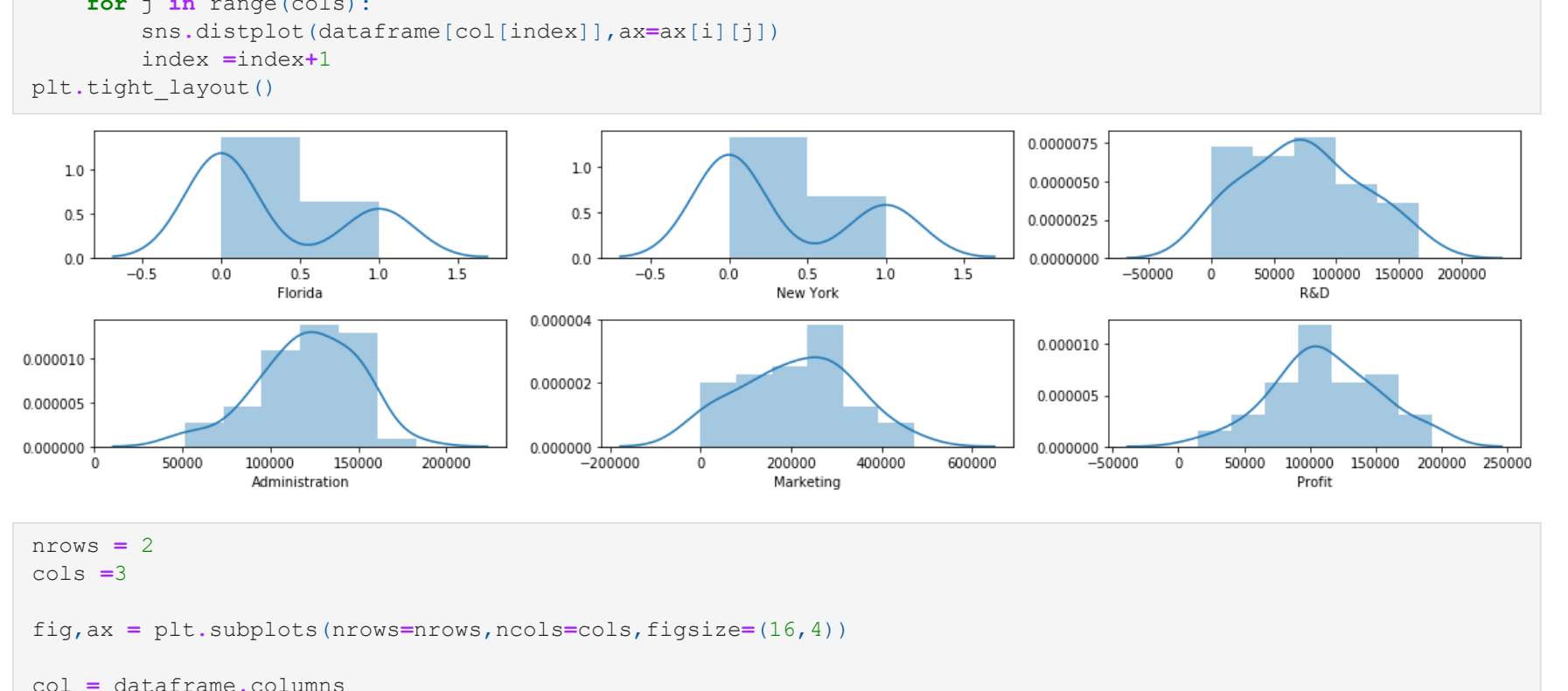
Out[41]:

	Florida	New York	R&D	Administration	Marketing	Profit
0	0	1	165349.20	136897.80	471784.10	192261.83
1	0	0	162597.70	151377.59	443898.53	191792.06
2	1	0	153441.51	101145.55	407934.54	191050.39
3	0	1	144372.41	118671.85	383199.62	182901.99
4	1	0	142107.34	91391.77	366168.42	166187.94

In [42]:
```
rows = 2
cols = 3

fig,ax = plt.subplots(nrows=rows,ncols=cols,figsize=(16,4))
col = dataframe.columns

index = 0

for i in range(rows):
    for j in range(cols):
        sns.distplot(dataframe[col[index]],ax=ax[i][j])
        index =index+1
plt.tight_layout()
```



In [43]:
```
nrows = 2
cols =3

fig,ax = plt.subplots(nrows=nrows,ncols=cols,figsize=(16,4))

col = dataframe.columns
index =0
for i in range(cols):
    for j in range(rows):
        sns.distplot(dataframe[col[index]],ax = ax[j][i])
        index = index +1
plt.tight_layout()
```



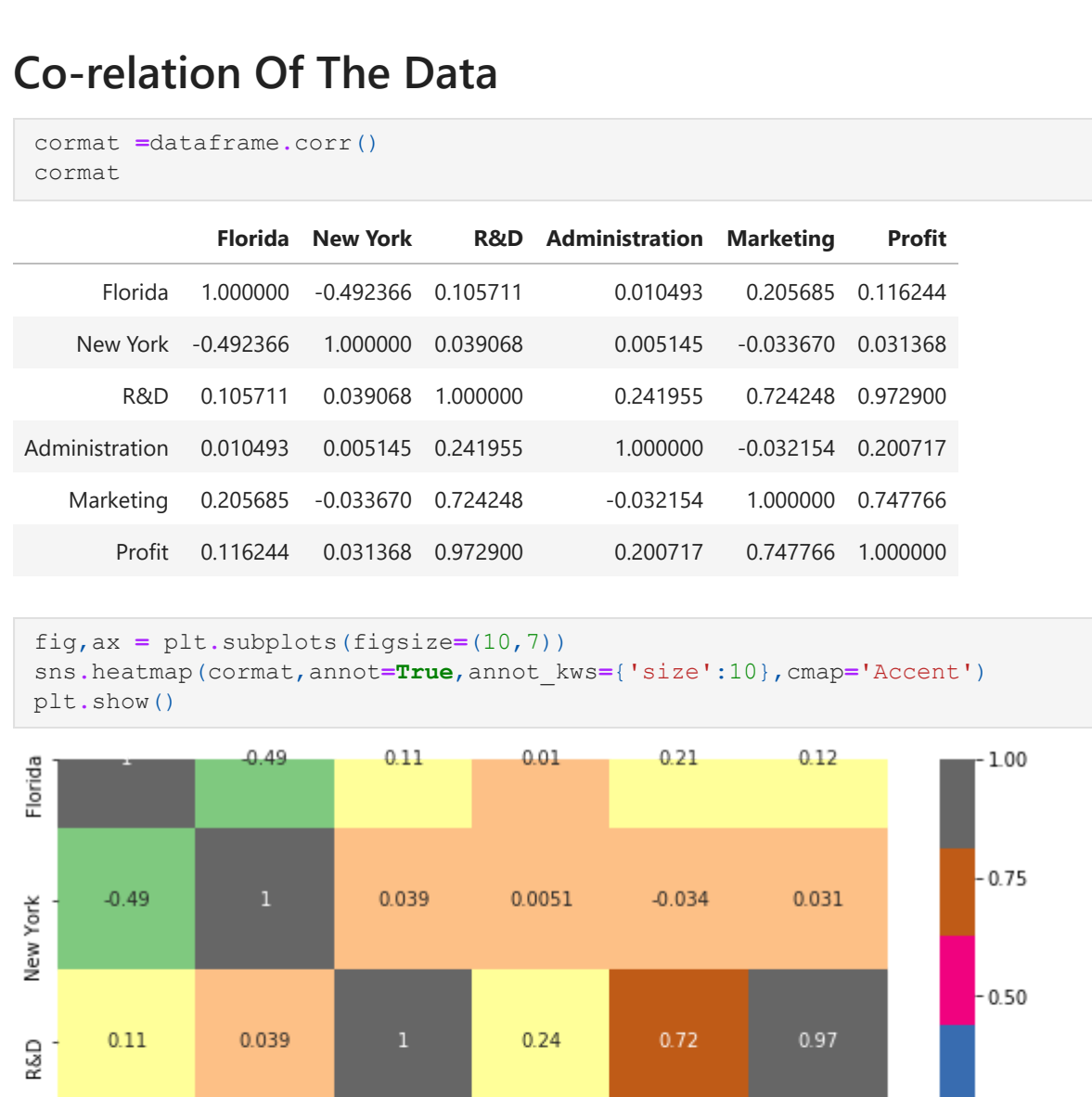
Co-relation Of The Data

In [44]:
```
cormat =dataframe.corr()
cormat
```

Out[44]:

	Florida	New York	R&D	Administration	Marketing	Profit
Florida	1.000000	-0.492366	0.105711	0.010493	0.205685	0.116244
New York	-0.492366	1.000000	0.039068	0.005145	-0.033670	0.031368
R&D	0.105711	0.039068	1.000000	0.241955	0.724248	0.972900
Administration	0.010493	0.005145	0.241955	1.000000	-0.032154	0.200717
Marketing	0.205685	-0.033670	0.724248	-0.032154	1.000000	0.747766
Profit	0.116244	0.031368	0.972900	0.200717	0.747766	1.000000

In [45]:
```
fig,ax = plt.subplots(figsize=(10,7))
sns.heatmap(cormat,annot=True,annot_kws={'size':10},cmap='Accent')
plt.show()
```

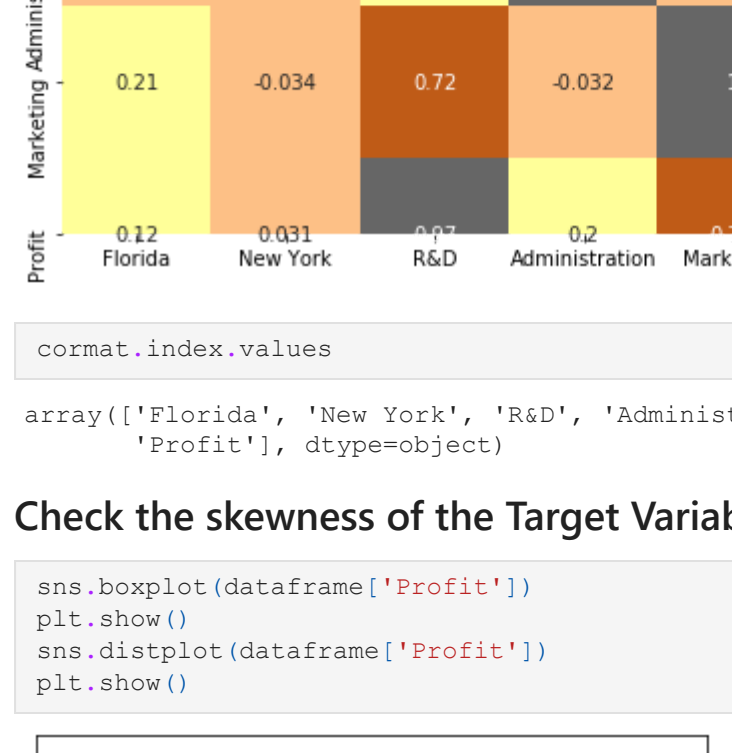


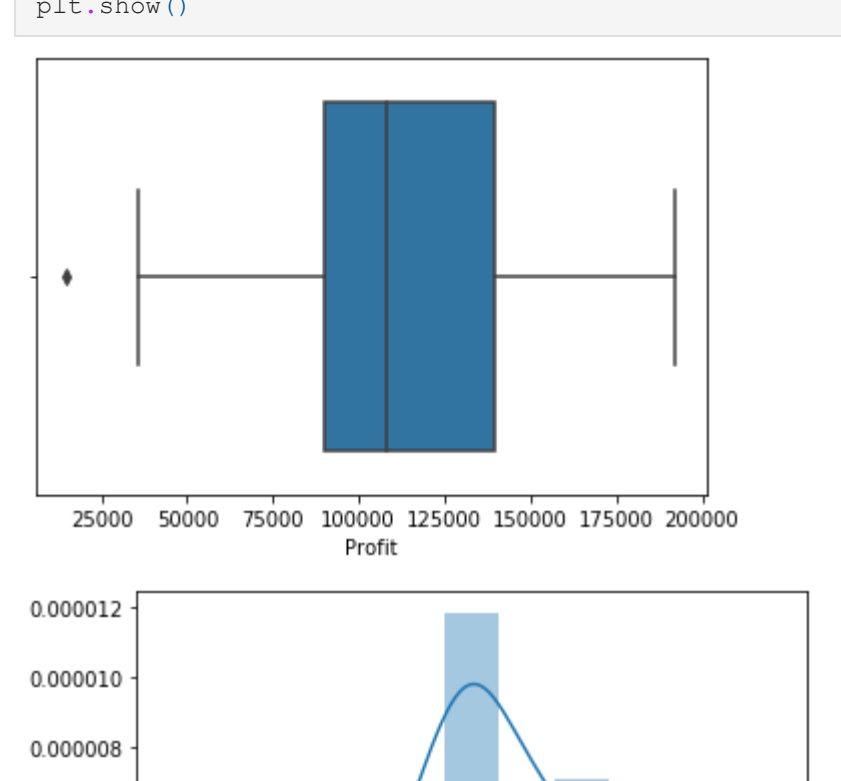
In [46]:
```
cormat.index.values
```

Out[46]:
```
array(['Florida', 'New York', 'R&D', 'Administration', 'Marketing',
       'Profit'], dtype=object)
```

Check the skewness of the Target Variable and Outlier treatment

In [47]:
```
sns.boxplot(dataframe['Profit'])
plt.show()
sns.distplot(dataframe['Profit'])
plt.show()
```





Z-Score

In [48]:
```
dataframe['Profit'].mean()
```

Out[48]:
```
112012.63920000002
```

In [49]:
```
dataframe['Profit'].median()
```

Out[49]:
```
107978.19
```

In [50]:
```
from scipy.stats import zscore
```

In [51]:
```
z = np.abs(zscore(dataframe['Profit']))
```

In [52]:
```
dataframe['z']=z
```

In [53]:
```
dataframe['z'].mean()
```

Out[53]:
```
0.791398820354949
```

In [54]:
```
dataframe['z'].median()
```

Out[54]:
```
0.649952180975987
```

In [55]:
```
dataz = dataframe[dataframe['z']<3]
dataz.head()
```

Out[55]:

	Florida	New York	R&D	Administration	Marketing	Profit	z
0	0	1	165349.20	136897.80	471784.10	192261.83	2.011203
1	0	0	162597.70	151377.59	443898.53	191792.06	1.999430
2	1	0	153441.51	101145.55	407934.54	191050.39	1.980842
3	0	1	144372.41	118671.85	383199.62	182901.99	1.776627
4	1	0	142107.34	91391.77	366168.42	166187.94	1.357740

In [56]:
```
dataz.shape
```

Out[56]:
```
(50, 7)
```

IQR

In [57]:
```
Q1 = dataframe['Profit'].quantile(0.25)
Q3 = dataframe['Profit'].quantile(0.75)

IQR = Q3-Q1
IQR
```

Out[57]:
```
49627.07499999998
```

In [58]:
```
dataiqr = dataframe[~((dataframe['Profit']<Q1-1.5*IQR) | (dataframe['Profit']>Q3+1.5*IQR))]
dataiqr.head()
```

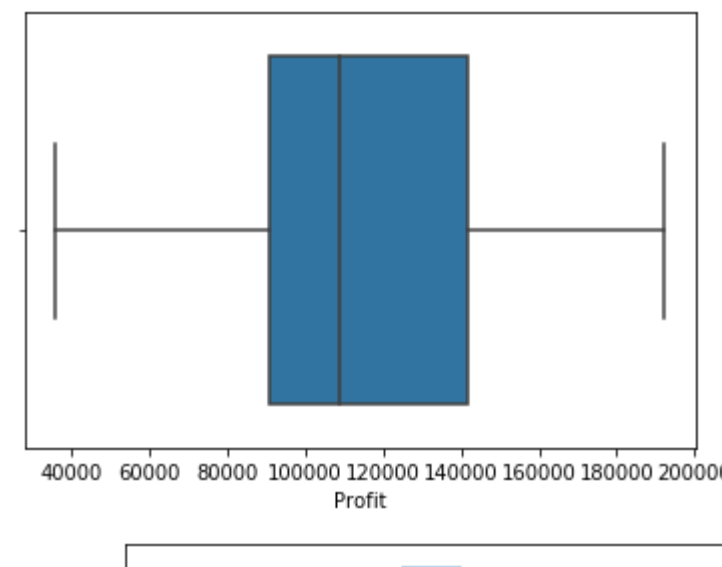
Out[58]:

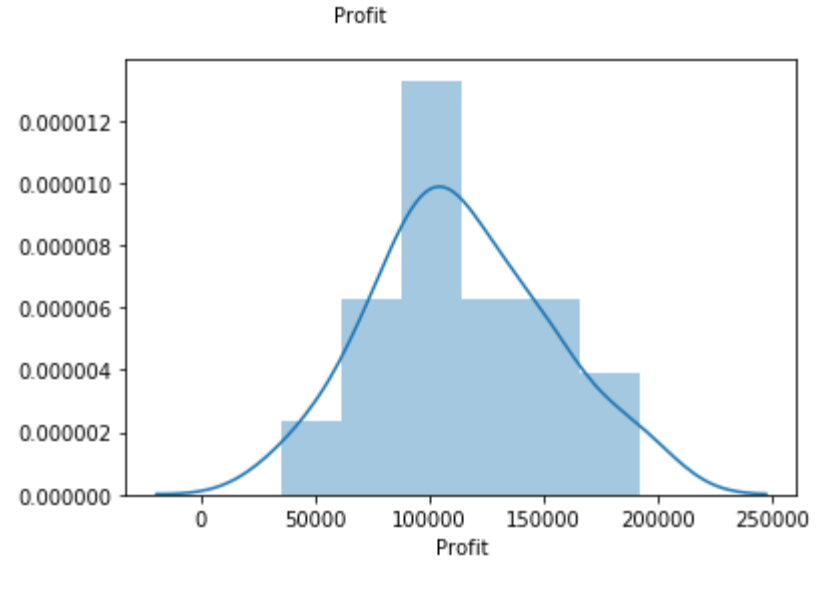
	Florida	New York	R&D	Administration	Marketing	Profit	z
0	0	1	165349.20	136897.80	471784.10	192261.83	2.011203
1	0	0	162597.70	151377.59	443898.53	191792.06	1.999430
2	1	0	153441.51	101145.55	407934.54	191050.39	1.980842
3	0	1	144372.41	118671.85	383199.62	182901.99	1.776627
4	1	0	142107.34	91391.77	366168.42	166187.94	1.357740

In [59]:
```
dataiqr.shape
```

Out[59]:
```
(49, 7)
```

In [60]:
```
sns.boxplot(dataiqr['Profit'])
plt.show()
sns.distplot(dataiqr['Profit'])
plt.show()
```





Feature Selection By Using Correlation Between the Independent and Depenedent Feature.

In [61]:
```
corr=dataframe.corr()['Profit']
corr
```

Out[61]:
```
Florida           0.116244
New York          0.031368
R&D               0.972900
Administration    0.200717
Marketing         0.747766
Profit            1.000000
z                 0.047770
Name: Profit, dtype: float64
```

In [62]:
```
good_corr = corr[corr>0.5]
good_corr
```

Out[62]:
```
R&D          0.972900
Marketing    0.747766
Profit       1.000000
Name: Profit, dtype: float64
```

In [63]:
```
index = good_corr.index
```

In [64]:
```
corrdata = dataframe[index]
```

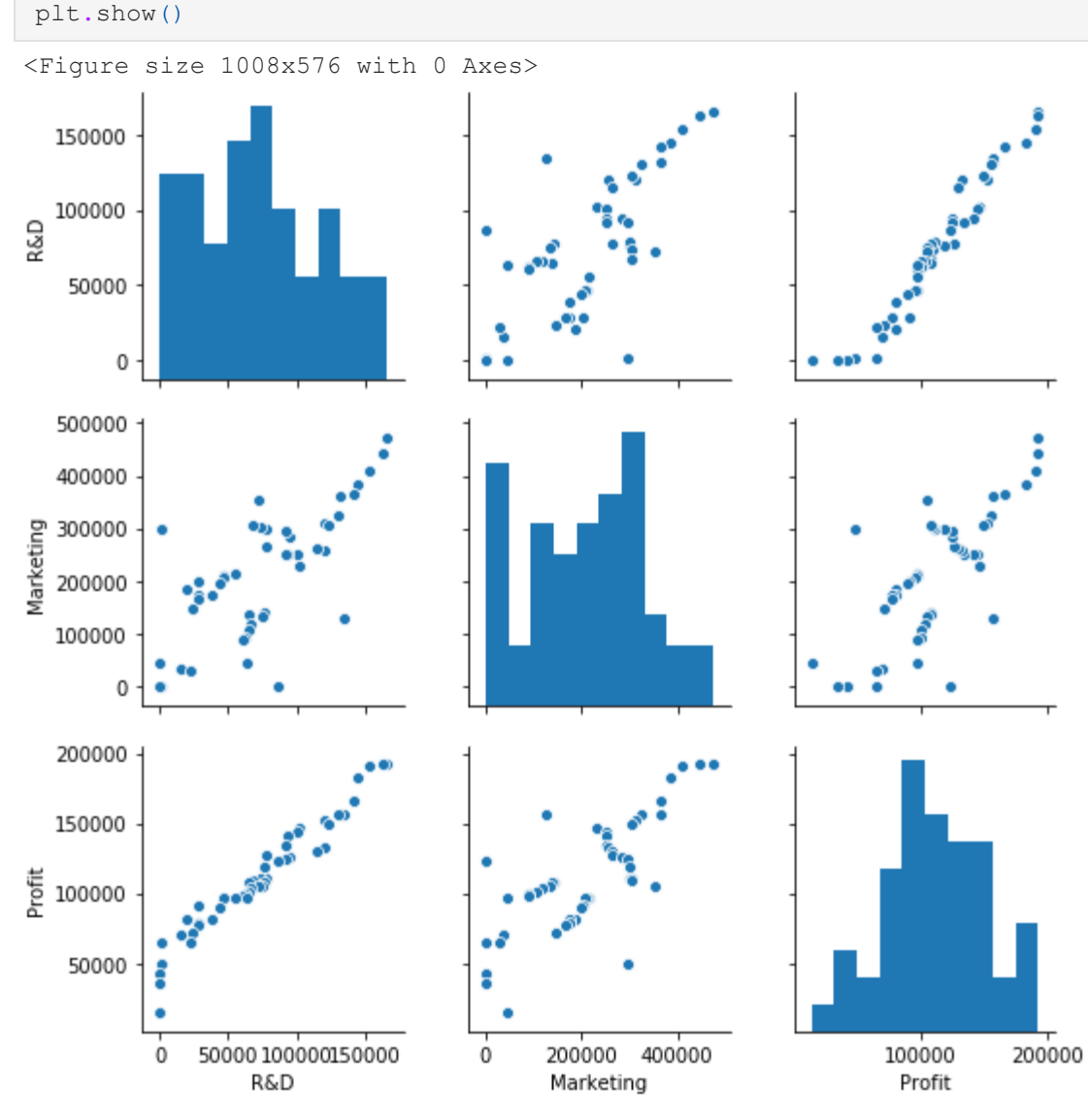
In [65]:
```
corrdata.head()
```

Out[65]:

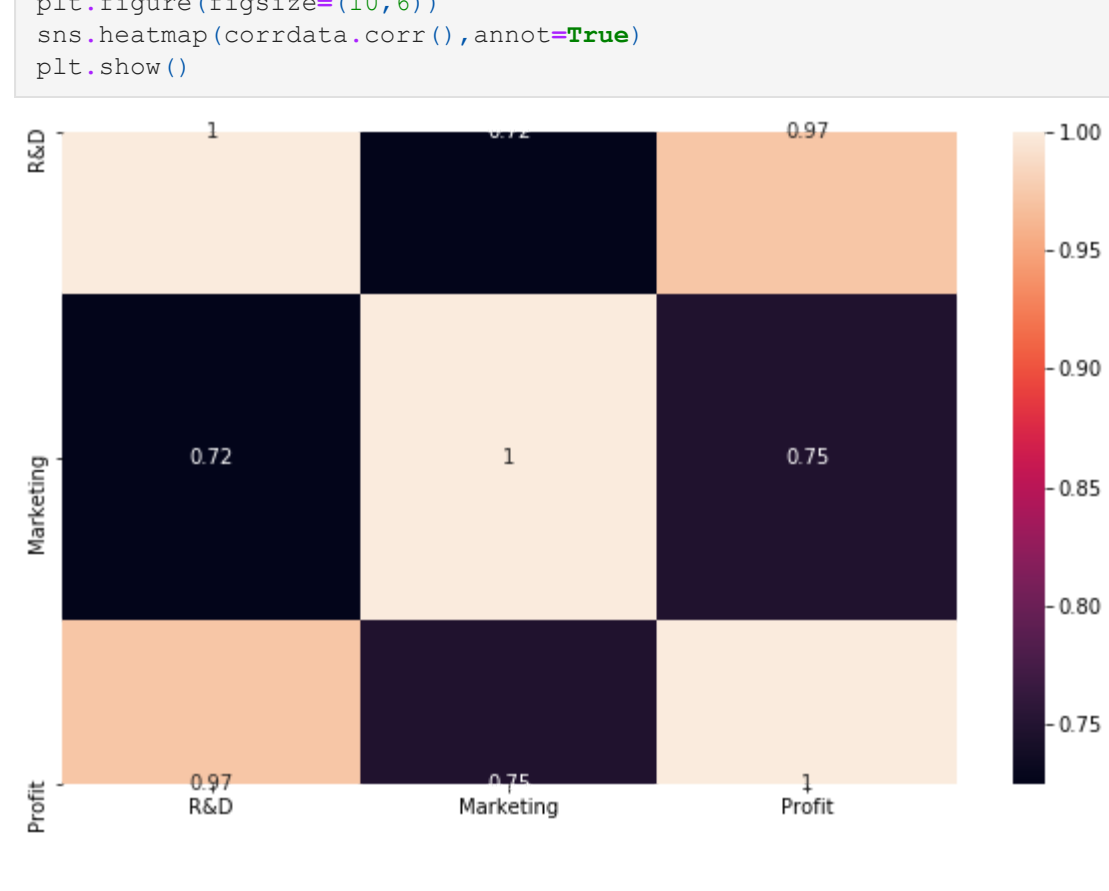
	R&D	Marketing	Profit
0	165349.20	471784.10	192261.83
1	162597.70	443898.53	191792.06
2	153441.51	407934.54	191050.39
3	144372.41	383199.62	182901.99
4	142107.34	366168.42	166187.94

In [66]:
```
plt.figure(figsize=(14,8))
sns.pairplot(corrdata)
plt.show()
```

```
<Figure size 1008x576 with 0 Axes>
```



In [67]:
```
plt.figure(figsize=(10,6))
sns.heatmap(corrdata.corr(),annot=True)
plt.show()
```



Create model

In [68]:
```
X = corrdata.drop(columns=['Profit'])
Y = corrdata['Profit']
```

In [69]:
```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,train_size=0.80,random_state=42)
```

In [70]:
```
X_train.shape,Y_train.shape
```

Out[70]:
```
((40, 2), (40,))
```

In [71]:
```
scaler =  StandardScaler()
X_train=scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

R2 score

R2 score is nothing but the **1-(mean Absolue Square).**

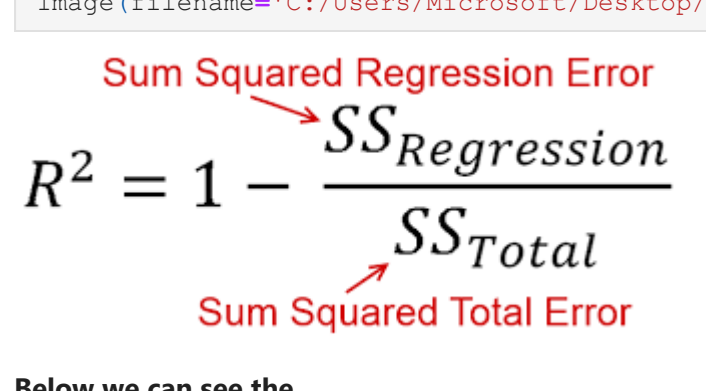
- If the r2 score is nearer to one then we can say the model is good
- If the r2 score is nearest to zero then we can say the model is not so much good.

What Is R-Squared?

R-squared (R2) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model. Whereas correlation explains the strength of the relationship between an independent and dependent variable, R-squared explains to what extent the variance of one variable explains the variance of the second variable. So, if the R2 of a model is 0.50, then approximately half of the observed variation can be explained by the model's inputs.

In [72]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/download (2).png')
```

Out[72]:

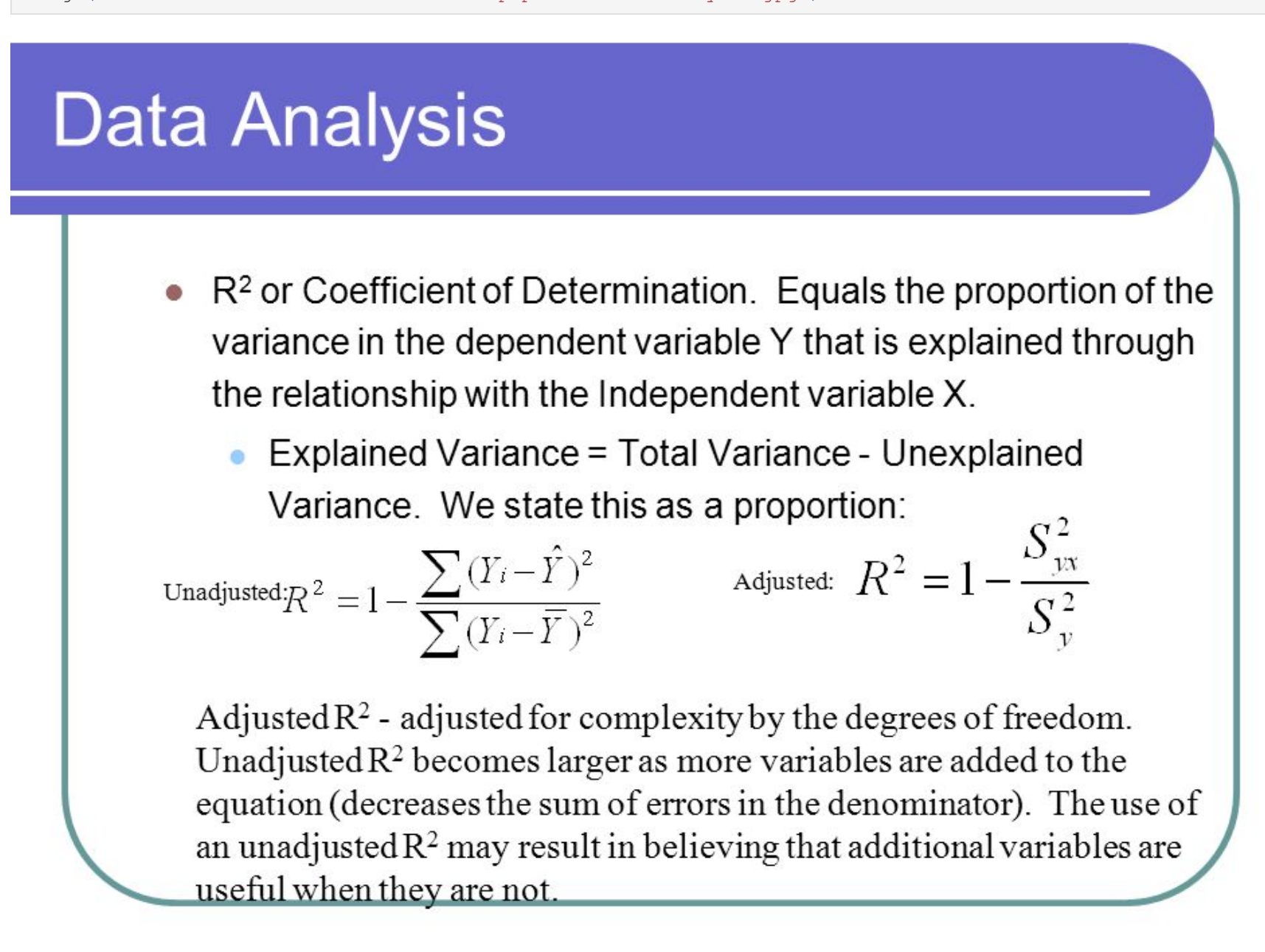


Below we can see the

- yi= Actal Point in data.
- ^y_pred = Predicted Y in the data
- -y_mean = Mean of the actual data.

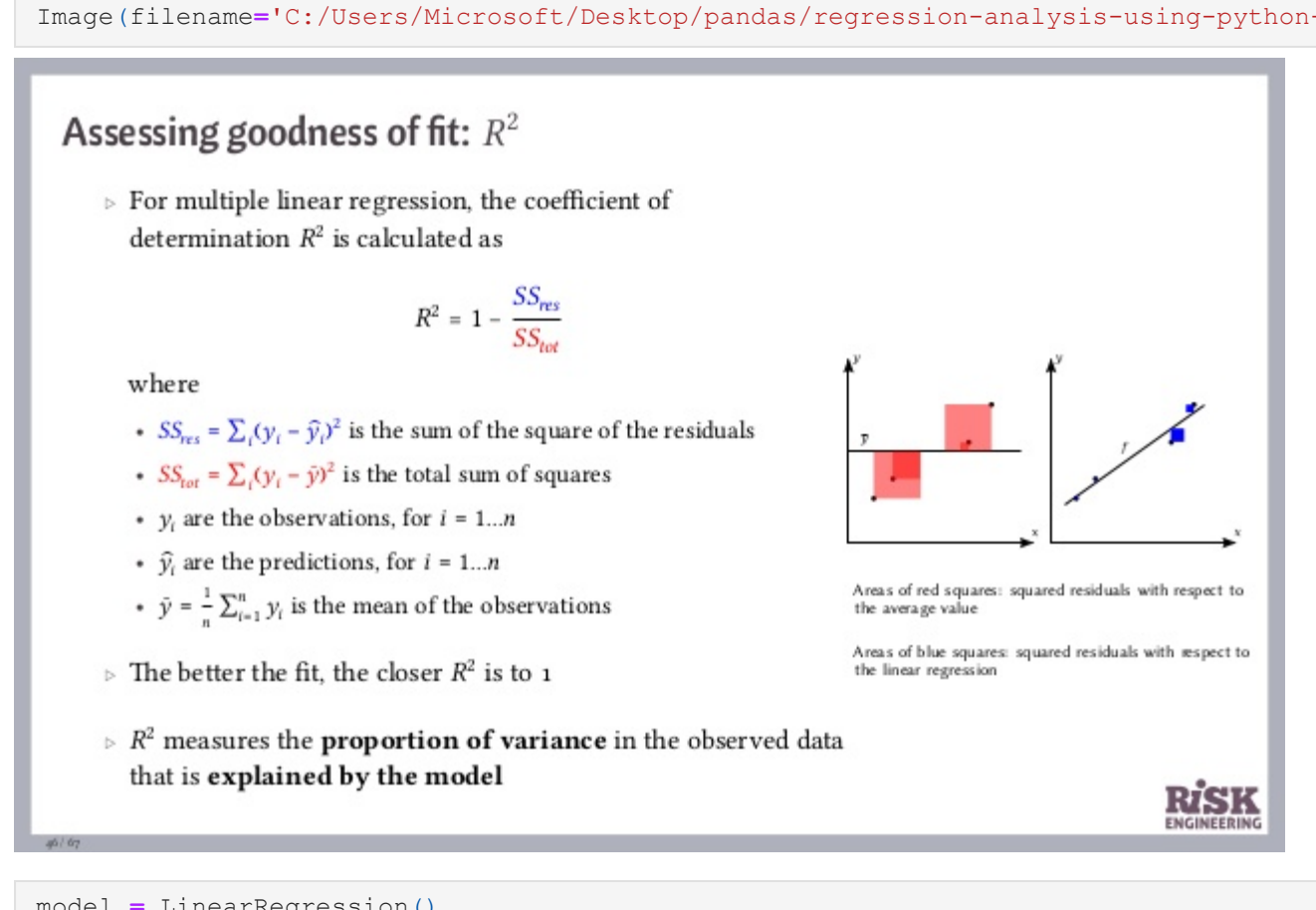
In [73]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/Data+Analysis.jpg')
```

Out[73]:



In [74]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/regression-analysis-using-python-47-638.jpg')
```

Out[74]:



In [75]:
```
model = LinearRegression()
model.fit(X_train,Y_train)
y_pred = model.predict(X_test)
print('Accuarcy :',r2_score(Y_test,y_pred))
```

```
Accuarcy : 0.9168381183550247
```

In [76]:
```
dff = pd.DataFrame({'Actual':Y_test,'Predicted':y_pred})
dff
```

Out[76]:

	Actual	Predicted
13	134307.35	127521.386041
39	81005.76	82615.074115
30	99937.59	97683.246234
45	64926.08	46400.656776
17	125370.37	130782.536119
48	35673.41	45967.020525
26	105733.54	109813.190619
25	107404.34	101612.689214
32	97427.84	97023.640139
19	122776.86	113241.365758

Matrix Measures Of The Linear Regression.

- R2 score.
- Mean Absolute Error.(MAE)
- Mean Squared Error.(MSE)

In [77]:
```
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
```

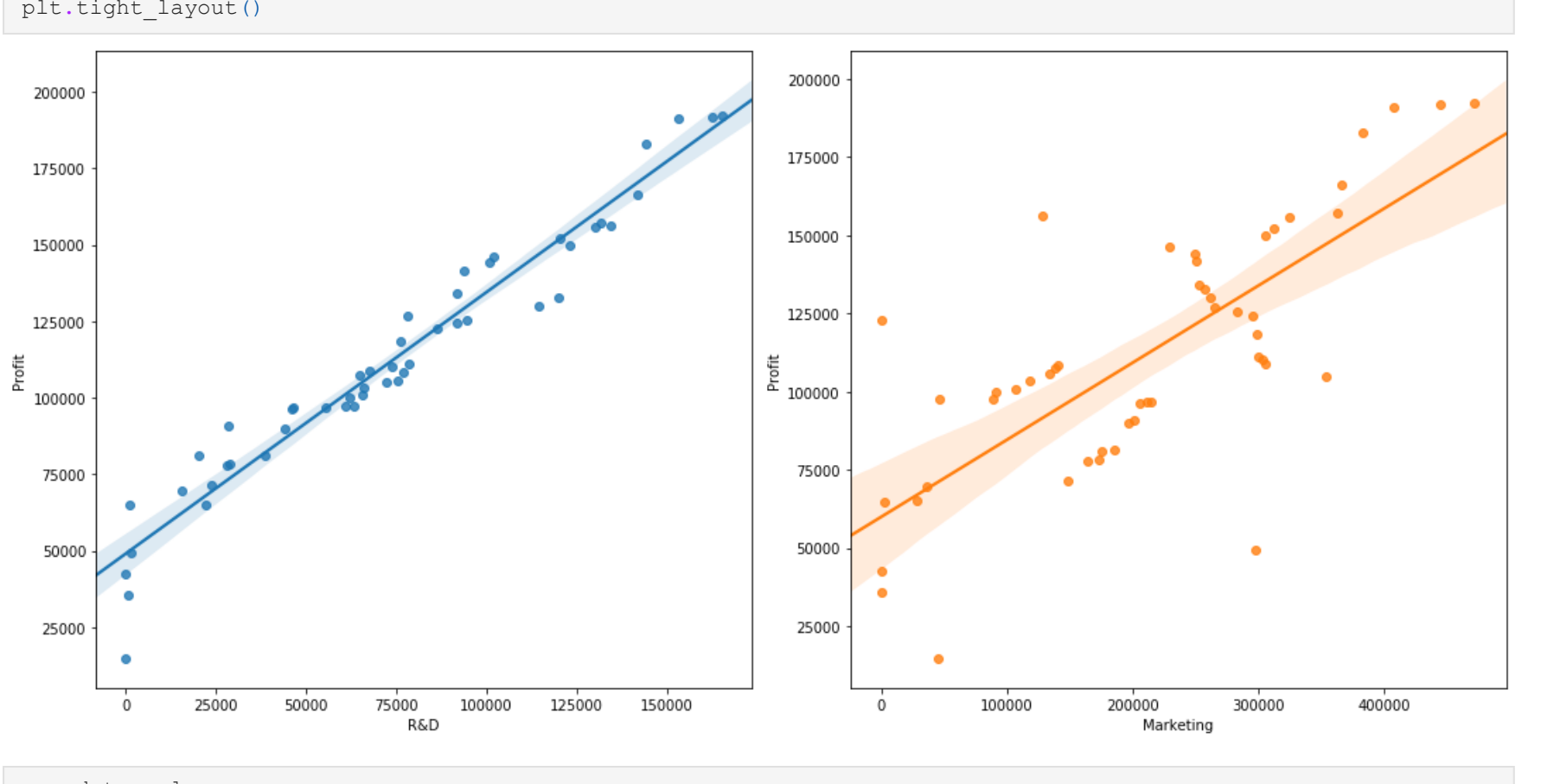
In [78]:
```
score = r2_score(Y_test,y_pred)
mae =  mean_absolute_error(Y_test,y_pred)
mse =  mean_squared_error(Y_test,y_pred)

print('r2_score :',score)
print('Mean absolute error :',mae)
print('Mean squared Error :',mse)
```

```
r2_score : 0.9168381183550247
Mean absolute error : 6469.1817214685925
Mean squared Error : 67343832.5897961
```

In [79]:
```
MSE = np.sqrt(mse)
MSE
```

Out[79]:
```
8206.328813165854
```

In [80]:
```
col = 2
row = 1
fig,ax = plt.subplots(nrows=row,ncols=col,figsize=(15,7))

cols = corrdata.columns
index=0
for i in range(rows):
        sns.regplot(x=corrdata[cols[index]],y=corrdata['Profit'],ax=ax[i])

        index = index+1
plt.tight_layout()
```



In [81]:
```
corrdata.columns
```

Out[81]:
```
Index(['R&D', 'Marketing', 'Profit'], dtype='object')
```

OLS Model

In [96]:
```
from statsmodels.api import add_constant
```

In [111...
```
from statsmodels.api import OLS
```

In [115...
```
x_constant = add_constant(X)
```

In [116...
```
OLs_model = OLS(Y,x_constant).fit()
```

In [117...
```
OLs_model.summary()
```

Out[117...

OLS Regression Results

Dep. Variable:	Profit	**R-squared:**	0.950
Model:	OLS	**Adj. R-squared:**	0.948
Method:	Least Squares	**F-statistic:**	450.8
Date:	Sat, 28 Mar 2020	**Prob (F-statistic):**	2.16e-31
Time:	00:16:10	**Log-Likelihood:**	-525.54
No. Observations:	50	**AIC:**	1057.
Df Residuals:	47	**BIC:**	1063.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.698e+04	2689.933	17.464	0.000	4.16e+04	5.24e+04
R&D	0.7966	0.041	19.266	0.000	0.713	0.880
Marketing	0.0299	0.016	1.927	0.060	-0.001	0.061

Omnibus:	14.677	**Durbin-Watson:**	1.257
Prob(Omnibus):	0.001	**Jarque-Bera (JB):**	21.161
Skew:	-0.939	**Prob(JB):**	2.54e-05
Kurtosis:	5.575	**Cond. No.**	5.32e+05

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.32e+05. This might indicate that there are strong multicollinearity or other numerical problems.

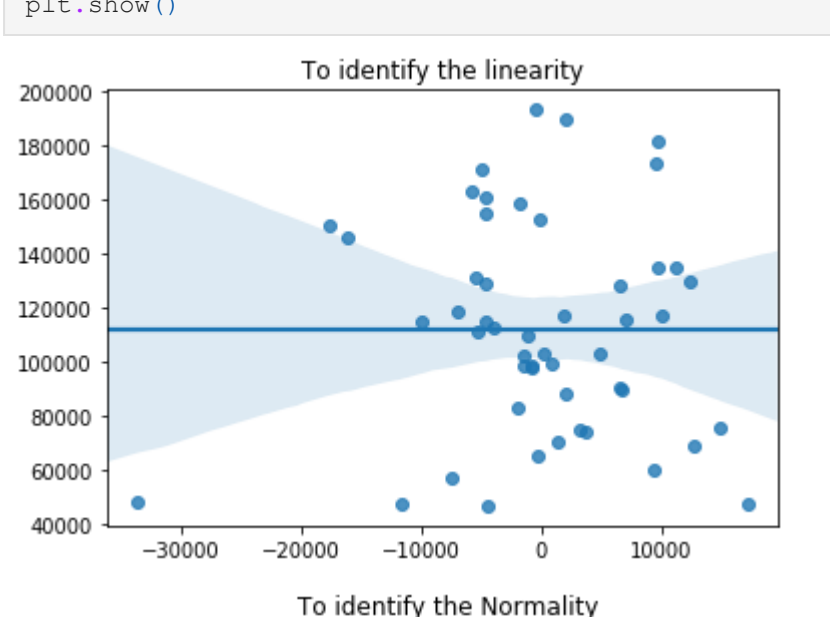
In [119...
```
residual=OLs_model.resid
```

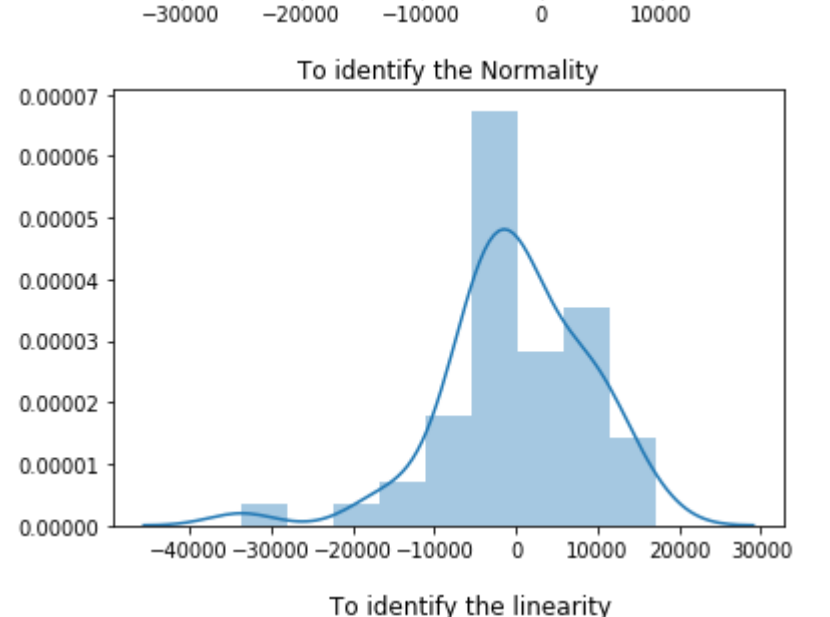
In [126...
```
prediction= OLs_model.predict(x_constant)
```

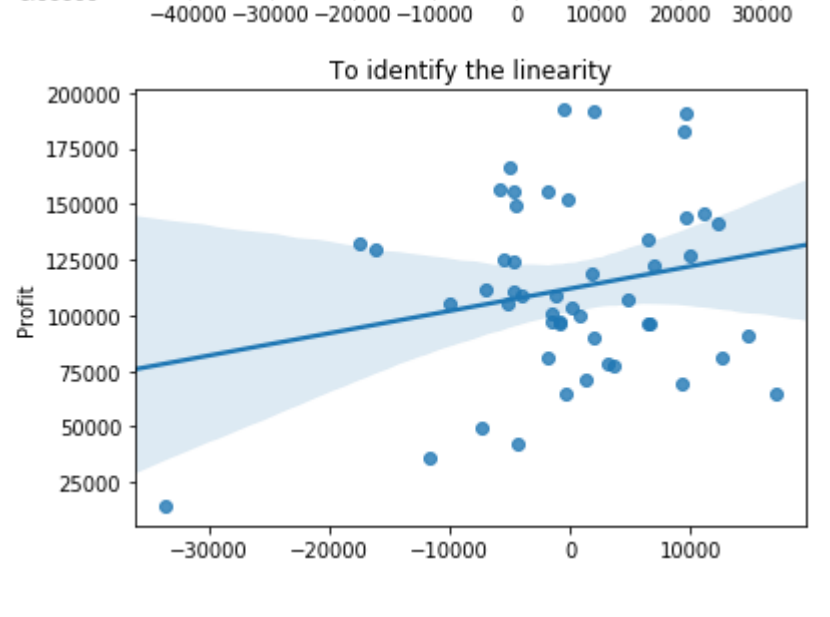
In [124...
```
Y.head()
```

Out[124...
```
0    192261.83
1    191792.06
2    191050.39
3    182901.99
4    166187.94
Name: Profit, dtype: float64
```

In [135...
```
sns.regplot(OLs_model.resid,OLs_model.predict(x_constant))
plt.title('To identify the linearity')
plt.show()
sns.distplot(OLs_model.resid)
plt.title('To identify the Normality')
plt.show()
sns.regplot(OLs_model.resid,Y)
plt.title('To identify the linearity')
plt.show()
```



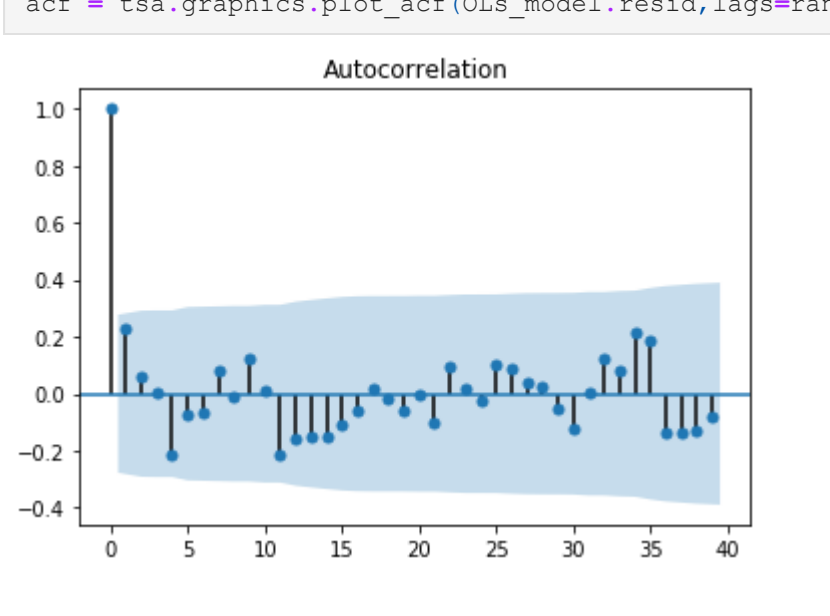




To indentify the autocorrelation

In [136...
```
import statsmodels.tsa.api as tsa
```

In [139...
```
acf = tsa.graphics.plot_acf(OLs_model.resid,lags=range(40))
```



- 0 To 20 is possitive correlation.20 is no correlation and 20 to 40 is Negetive Correlation.

Colinearity

In [143...
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [145...
```
vif = [variance_inflation_factor(x_constant.values,i) for i in range(x_constant.shape[1])]
```

In [146...
```
vif
```

Out[146...
```
[4.31092413950283, 2.1032058162760427, 2.1032058162760423]
```

In [147...
```
dataframe=pd.DataFrame(vif,columns=['vif'],index=x_constant.columns)
dataframe
```

Out[147...

	vif
const	4.310924
R&D	2.103206
Marketing	2.103206

In [27]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import zscore
import warnings
warnings.filterwarnings('ignore')
```

In [28]:
```python
data=pd.read_csv('house_train (1) (1).csv')
```

In [29]:
```python
data.head()
```

Out[29]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeatu
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	Na
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	Na
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	Na
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	Na
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	Na

5 rows × 81 columns

In [30]:
```python
data.tail()
```

Out[30]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	Mis
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	GdPrv	
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	

5 rows × 81 columns

In [31]:
```python
data.columns
```

Out[31]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

In [32]:
```python
len(data.columns)
```

Out[32]: 81

In [33]:
```python
data.shape
```

Out[33]: (1460, 81)

In [34]:
```python
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
Id               1460 non-null int64
MSSubClass       1460 non-null int64
MSZoning         1460 non-null object
LotFrontage      1201 non-null float64
LotArea          1460 non-null int64
Street           1460 non-null object
Alley            91 non-null object
LotShape         1460 non-null object
LandContour      1460 non-null object
Utilities        1460 non-null object
LotConfig        1460 non-null object
LandSlope        1460 non-null object
Neighborhood     1460 non-null object
Condition1       1460 non-null object
Condition2       1460 non-null object
BldgType         1460 non-null object
HouseStyle       1460 non-null object
OverallQual      1460 non-null int64
OverallCond      1460 non-null int64
YearBuilt        1460 non-null int64
YearRemodAdd     1460 non-null int64
RoofStyle        1460 non-null object
RoofMatl         1460 non-null object
Exterior1st      1460 non-null object
Exterior2nd      1460 non-null object
MasVnrType       1452 non-null object
MasVnrArea       1452 non-null float64
ExterQual        1460 non-null object
ExterCond        1460 non-null object
Foundation       1460 non-null object
BsmtQual         1423 non-null object
BsmtCond         1423 non-null object
BsmtExposure     1422 non-null object
BsmtFinType1     1423 non-null object
BsmtFinSF1       1460 non-null int64
BsmtFinType2     1422 non-null object
BsmtFinSF2       1460 non-null int64
BsmtUnfSF        1460 non-null int64
TotalBsmtSF      1460 non-null int64
Heating          1460 non-null object
HeatingQC        1460 non-null object
CentralAir       1460 non-null object
Electrical       1459 non-null object
1stFlrSF         1460 non-null int64
2ndFlrSF         1460 non-null int64
LowQualFinSF     1460 non-null int64
GrLivArea        1460 non-null int64
BsmtFullBath     1460 non-null int64
BsmtHalfBath     1460 non-null int64
FullBath         1460 non-null int64
HalfBath         1460 non-null int64
BedroomAbvGr     1460 non-null int64
KitchenAbvGr     1460 non-null int64
KitchenQual      1460 non-null object
TotRmsAbvGrd     1460 non-null int64
Functional       1460 non-null object
Fireplaces       1460 non-null int64
FireplaceQu      770 non-null object
GarageType       1379 non-null object
GarageYrBlt      1379 non-null float64
GarageFinish     1379 non-null object
GarageCars       1460 non-null int64
GarageArea       1460 non-null int64
GarageQual       1379 non-null object
GarageCond       1379 non-null object
PavedDrive       1460 non-null object
WoodDeckSF       1460 non-null int64
OpenPorchSF      1460 non-null int64
EnclosedPorch    1460 non-null int64
3SsnPorch        1460 non-null int64
ScreenPorch      1460 non-null int64
PoolArea         1460 non-null int64
PoolQC           7 non-null object
Fence            281 non-null object
MiscFeature      54 non-null object
MiscVal          1460 non-null int64
MoSold           1460 non-null int64
YrSold           1460 non-null int64
SaleType         1460 non-null object
SaleCondition    1460 non-null object
SalePrice        1460 non-null int64
dtypes: float64(3), int64(35), object(43)
memory usage: 678.7+ KB
None
```

In [35]:
```python
data.describe()
```

Out[35]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.000000
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.639726
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207	456.098091
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.000000
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.500000
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000	712.250000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000

8 rows × 38 columns

In [36]:
```python
df = pd.DataFrame({'Data':data.dtypes,'missing values':data.isnull().sum(),'unique values':data.nunique(),'cour
```

In [37]:
```python
df
```

Out[37]:

	Data	missing values	unique values	count
Id	int64	0	1460	1460
MSSubClass	int64	0	15	1460
MSZoning	object	0	5	1460
LotFrontage	float64	259	110	1201
LotArea	int64	0	1073	1460
...	...	...	...	...
MoSold	int64	0	12	1460
YrSold	int64	0	5	1460
SaleType	object	0	9	1460
SaleCondition	object	0	6	1460
SalePrice	int64	0	663	1460

81 rows × 4 columns

In [38]:
```python
df.T
```

Out[38]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence
Data	int64	int64	object	float64	int64	object	object	object	object	object	...	int64	object	object
missing values	0	0	0	259	0	0	1369	0	0	0	...	0	1453	1179
unique values	1460	15	5	110	1073	2	2	4	4	2	...	8	3	4
count	1460	1460	1460	1201	1460	1460	91	1460	1460	1460	...	1460	7	281

4 rows × 81 columns

In [39]:
```python
data.isnull().sum().sort_values(ascending=False).head(15) #find the top null values.
```

Out[39]:
```
PoolQC          1453
MiscFeature     1406
Alley           1369
Fence           1179
FireplaceQu      690
LotFrontage      259
GarageCond        81
GarageType        81
GarageYrBlt       81
GarageFinish      81
GarageQual        81
BsmtExposure      38
BsmtFinType2      38
BsmtFinType1      37
BsmtCond          37
dtype: int64
```

In [40]:
```python
data.drop(['Alley','PoolQC','Fence','MiscFeature'],axis=1,inplace=True)
```

In [41]:
```python
data.isnull().sum().sort_values(ascending=False).head(10)
```

Out[41]:
```
FireplaceQu     690
LotFrontage     259
GarageYrBlt      81
GarageType       81
GarageFinish     81
GarageQual       81
GarageCond       81
BsmtExposure     38
BsmtFinType2     38
BsmtFinType1     37
dtype: int64
```

In [42]:
```python
data.dropna(subset=['GarageType'],inplace=True) #sunbset use for drop ythe null of same columns.
#it remove the duplicates
```

In [43]:
```python
data.isnull().sum().sort_values(ascending=False).head(10)
```

Out[43]:
```
FireplaceQu     618
LotFrontage     252
BsmtFinType2     31
BsmtExposure     31
BsmtFinType1     30
BsmtCond         30
BsmtQual         30
MasVnrArea        8
MasVnrType        8
Electrical        1
dtype: int64
```

Fill Null Vlues.

In [44]:
```python
data['LotFrontage'].mean()  #mean and medium is close each other that's we use the medium.
```

Out[44]: 70.6832298136646

In [45]:
```python
data['LotFrontage'].median()
```

Out[45]: 70.0

In [46]:
```python
#HERE we can see the mean = medium that means here oulier is low.
```

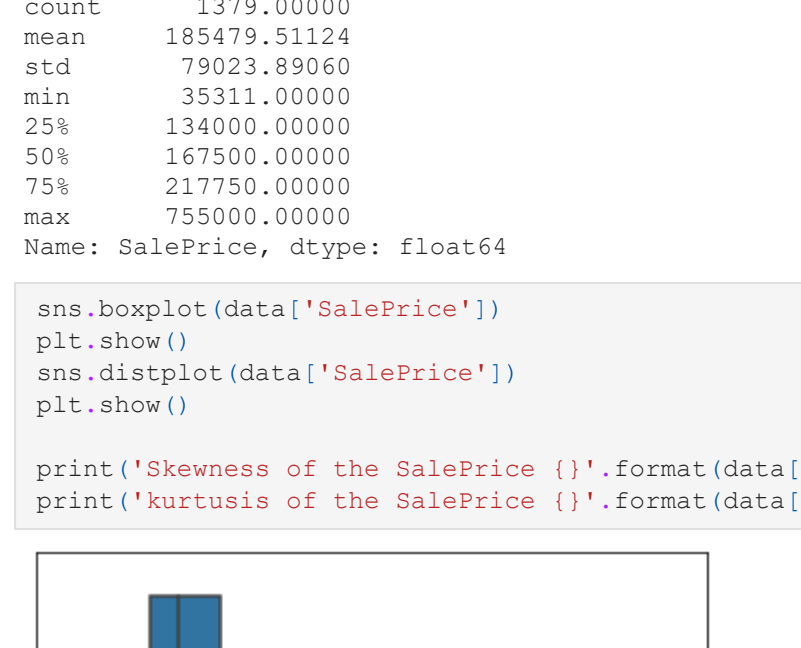
In [47]:
```python
data['LotFrontage']= data['LotFrontage'].fillna(data['LotFrontage'].median())
```

In [48]:
```python
data['LotFrontage'].isnull().sum()
```

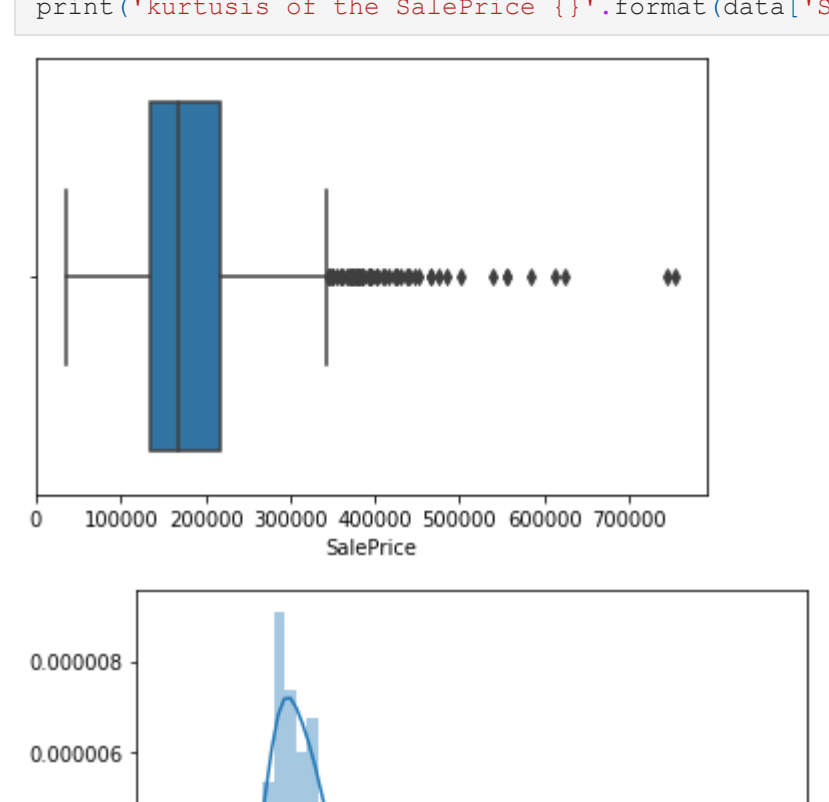
Out[48]: 0

Analysis the data.

In [49]:
```python
sns.countplot(x='OverallQual',data=data)
plt.show()
```



In [50]:
```python
sns.stripplot(x='OverallQual',y='SalePrice',data=data)
plt.show()
```

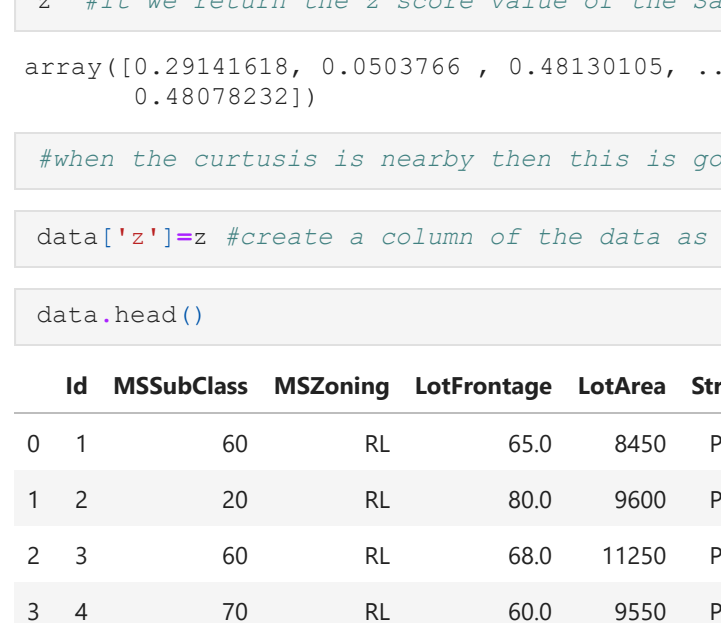


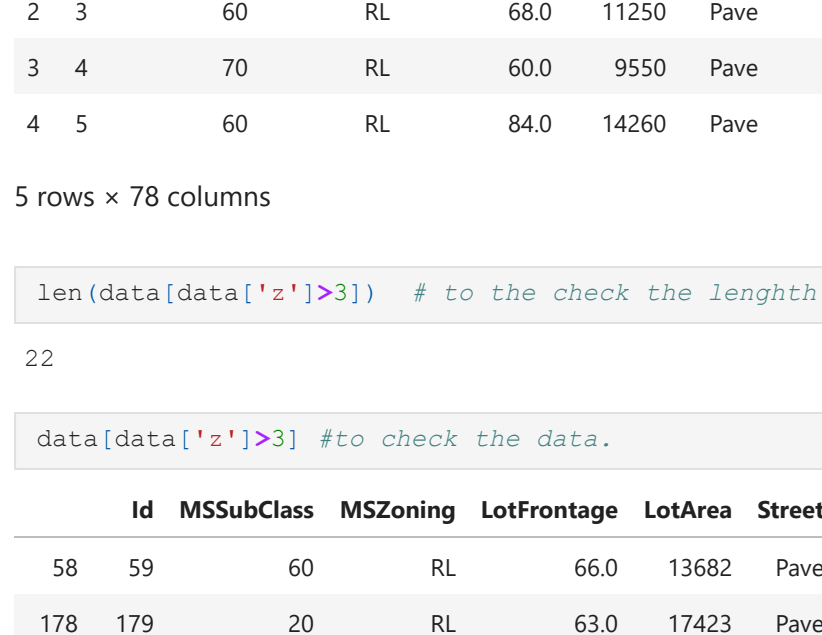
Z-Score

In [51]:
```python
data['SalePrice'].describe()
```

Out[51]:
```
count      1379.00000
mean     185479.51124
std       79023.89060
min       35311.00000
25%      134000.00000
50%      167500.00000
75%      217750.00000
max      755000.00000
Name: SalePrice, dtype: float64
```

In [52]:
```python
sns.boxplot(data['SalePrice'])
plt.show()
sns.distplot(data['SalePrice'])
plt.show()

print('Skewness of the SalePrice {}'.format(data['SalePrice'].skew()))
print('kurtusis of the SalePrice {}'.format(data['SalePrice'].kurt()))
```






```
Skewness of the SalePrice 1.935362098363132
kurtusis of the SalePrice 6.735649337267559
```

In [53]:
```python
from scipy.stats import zscore
```

In [55]:
```python
z = np.abs(zscore(data['SalePrice']))

#The abs() function is used to return the absolute value of a number.
z  #It we return the z score value of the SalesPrice
```

Out[55]:
```
array([0.29141618, 0.0503766 , 0.48130105, ..., 1.0256377 , 0.5488244 ,
       0.48078232])
```

In [ ]:
```python
#when the curtusis is nearby then this is good for us.
```

In [56]:
```python
data['z']=z #create a column of the data as name z
```

In [57]:
```python
data.head()
```

Out[57]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	...	3SsnPorch	ScreenPorch	PoolAr
0	1	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub	Inside	...	0	0	
1	2	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub	FR2	...	0	0	
2	3	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub	Inside	...	0	0	
3	4	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub	Corner	...	0	0	
4	5	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub	FR2	...	0	0	

5 rows × 78 columns

In [60]:
```python
len(data[data['z']>3])  # to the check the lenghth of the data to detect the outliers.
```

Out[60]: 22

In [61]:
```python
data[data['z']>3] #to check the data.
```

Out[61]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	...	3SsnPorch	ScreenPorch	P
58	59	60	RL	66.0	13682	Pave	IR2	HLS	AllPub	CulDSac	...	0	0	
178	179	20	RL	63.0	17423	Pave	IR1	Lvl	AllPub	CulDSac	...	0	0	
185	186	75	RM	90.0	22950	Pave	IR2	Lvl	AllPub	Inside	...	0	410	
349	350	60	RL	56.0	20431	Pave	IR2	Lvl	AllPub	Inside	...	0	0	
389	390	60	RL	96.0	12474	Pave	Reg	Lvl	AllPub	Inside	...	0	0	
440	441	20	RL	105.0	15431	Pave	Reg	Lvl	AllPub	Inside	...	0	170	
473	474	20	RL	110.0	14977	Pave	IR1	Lvl	AllPub	Inside	...	0	0	
496	497	20	RL	70.0	12692	Pave	IR1	Lvl	AllPub	Inside	...	0	0	
527	528	60	RL	67.0	14948	Pave	IR1	Lvl	AllPub	Inside	...	0	0	
591	592	60	RL	97.0	13478	Pave	IR1	Lvl	AllPub	Corner	...	0	0	
664	665	20	RL	49.0	20896	Pave	IR2	Lvl	AllPub	CulDSac	...	0	0	
691	692	60	RL	104.0	21535	Pave	IR1	Lvl	AllPub	Corner	...	0	0	
769	770	60	RL	47.0	53504	Pave	IR2	HLS	AllPub	CulDSac	...	0	210	
798	799	60	RL	104.0	13518	Pave	Reg	Lvl	AllPub	Inside	...	0	0	
803	804	60	RL	107.0	13891	Pave	Reg	Lvl	AllPub	Inside	...	0	192	
898	899	20	RL	100.0	12919	Pave	IR1	Lvl	AllPub	Inside	...	0	0	
1046	1047	60	RL	85.0	16056	Pave	IR1	Lvl	AllPub	Inside	...	0	0	
1142	1143	60	RL	77.0	9965	Pave	Reg	Lvl	AllPub	Inside	...	0	0	
1169	1170	60	RL	118.0	35760	Pave	IR1	Lvl	AllPub	CulDSac	...	0	0	
1182	1183	60	RL	160.0	15623	Pave	IR1	Lvl	AllPub	Corner	...	0	0	
1243	1244	20	RL	107.0	13891	Pave	Reg	Lvl	AllPub	Inside	...	0	0	
1373	1374	20	RL	70.0	11400	Pave	Reg	Lvl	AllPub	Inside	...	0	0	

22 rows × 78 columns

In [62]:
```python
dataz = data[data['z']<3]
dataz.shape
#new variable is create to drop the outliers store the values of z score.data
```

Out[62]: (1357, 78)

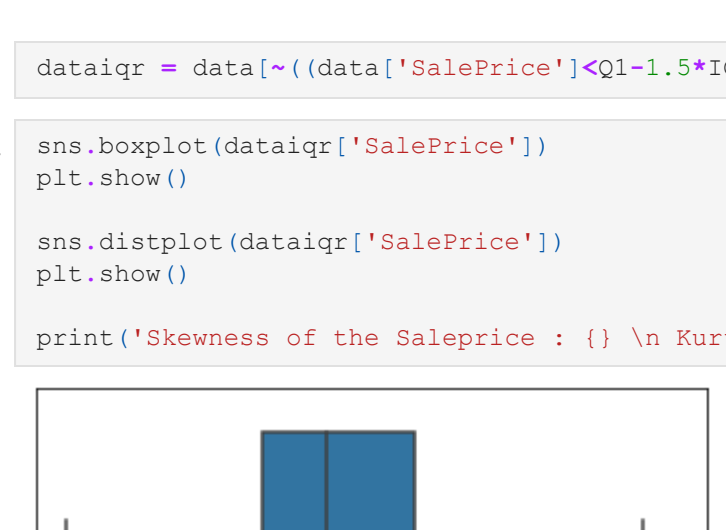
In [63]:
```python
dataz.describe()
```

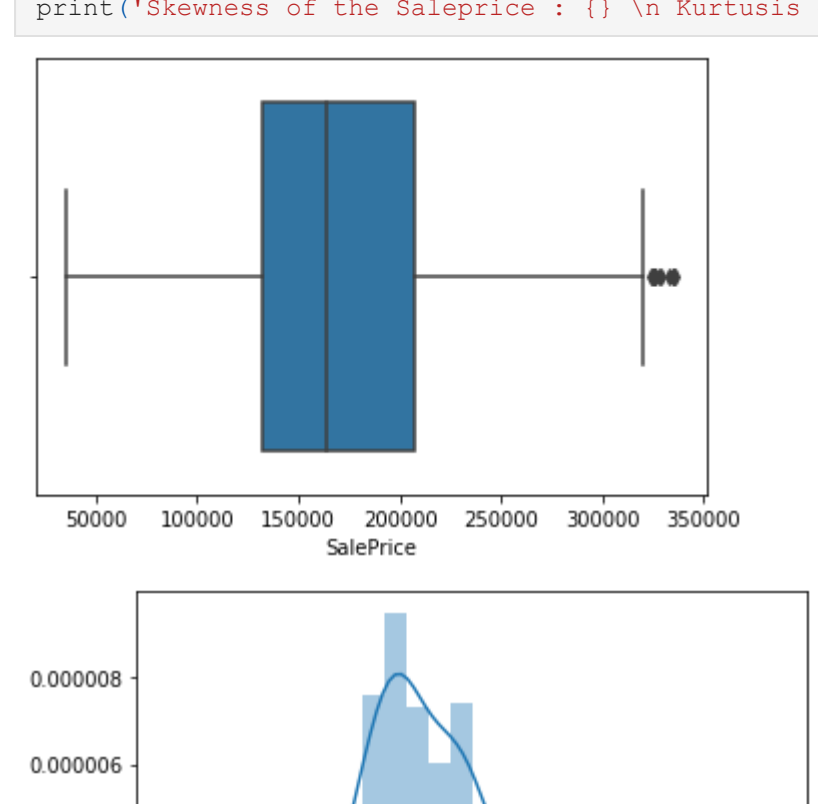
Out[63]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1
count	1357.000000	1357.000000	1357.000000	1357.000000	1357.000000	1357.000000	1357.000000	1357.000000	1350.000000	1357.000000
mean	732.333088	56.182756	70.266765	10576.330877	6.137804	5.583640	1972.537951	1985.131909	102.869630	443.855564
std	421.872735	41.260349	21.737293	10183.379768	1.295337	1.083449	29.271908	20.457622	172.286634	445.945664
min	1.000000	20.000000	21.000000	1300.000000	2.000000	2.000000	1880.000000	1950.000000	0.000000	0.000000
25%	365.000000	20.000000	60.000000	7681.000000	5.000000	5.000000	1955.000000	1967.000000	0.000000	0.000000
50%	735.000000	50.000000	70.000000	9549.000000	6.000000	5.000000	1974.000000	1994.000000	0.000000	394.000000
75%	1095.000000	70.000000	79.000000	11606.000000	7.000000	6.000000	2001.000000	2004.000000	168.000000	714.000000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000

8 rows × 39 columns

In [84]:
```python
sns.boxplot(dataz['SalePrice'])
plt.show()

sns.distplot(data['SalePrice'])
plt.show()

print('Skewness of the SalePrice {}'.format(dataz['SalePrice'].skew()))
print('kurtusis of the SalePrice {}'.format(dataz['SalePrice'].kurt()))
print('Mean of the SalesPrice {}'.format(dataz['SalePrice'].mean()))
print('Median of SalesPrice {}'.format(dataz['SalePrice'].median()))
```





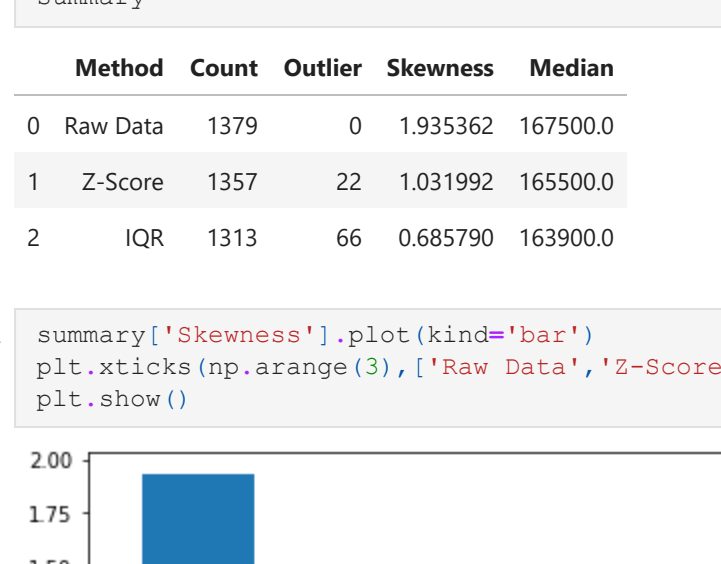

```
Skewness of the SalePrice 1.0319917234183782
kurtusis of the SalePrice 0.945553045084834
Mean of the SalesPrice 180173.70891672806
Median of SalesPrice 165500.0
```

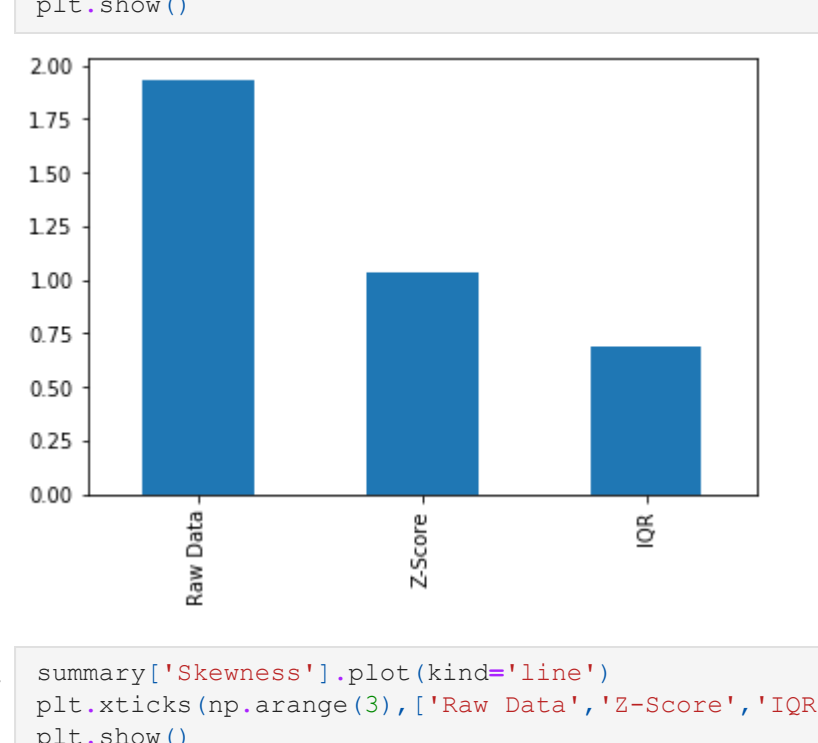
In [86]:
```python
Q1 = dataz['SalePrice'].quantile(0.25)
Q3 = dataz['SalePrice'].quantile(0.75)
IQR = Q3-Q1
IQR
```

Out[86]: 81000.0

In [89]:
```python
dataiqr = data[~((data['SalePrice']<Q1-1.5*IQR)|(data['SalePrice']>Q3+1.5*IQR))] #~ with help of this we remove
```

In [102…
```python
sns.boxplot(dataiqr['SalePrice'])
plt.show()

sns.distplot(dataiqr['SalePrice'])
plt.show()

print('Skewness of the Saleprice : {} \n Kurtosis of the SalePrice : {} \n Median of the SalePrice : {}'.format
```






```
Skewness of the Saleprice : 0.6857904124634652
 Kurtosis of the SalePrice : 0.022244822602960124
 Median of the SalePrice : 163900.0
```

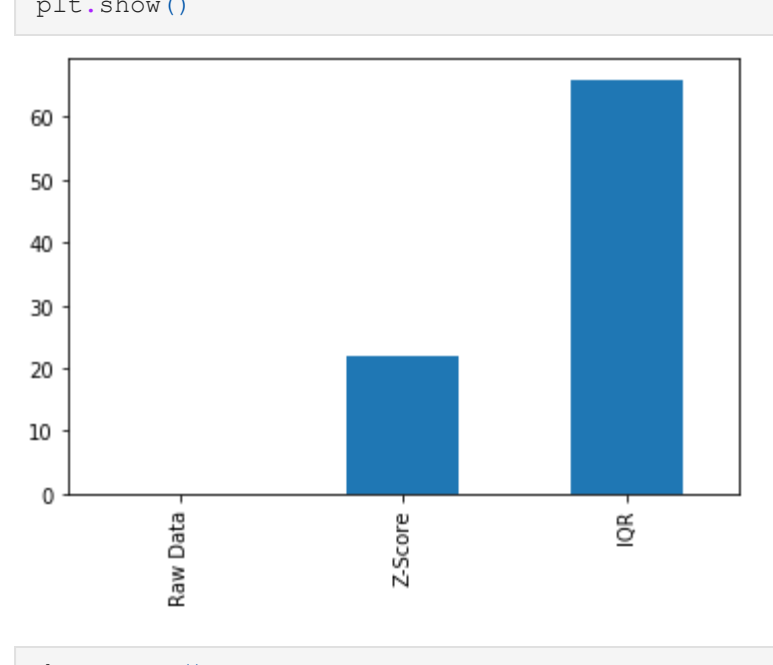
In [97]:
```python
summary = pd.DataFrame({'Method':['Raw Data','Z-Score','IQR'],'Count':[data.shape[0],dataz.shape[0],dataiqr.sha
```

In [98]:
```python
summary
```

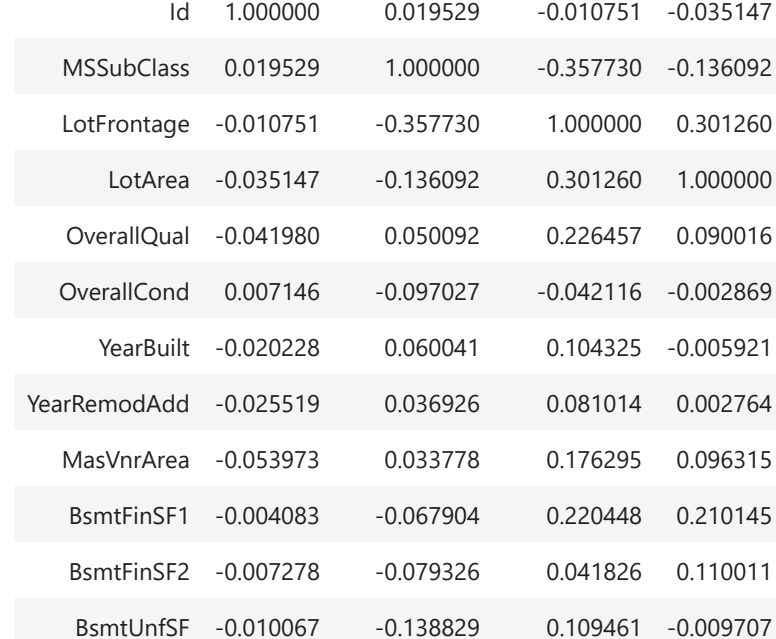
Out[98]:

	Method	Count	Outlier	Skewness	Median
0	Raw Data	1379	0	1.935362	167500.0
1	Z-Score	1357	22	1.031992	165500.0
2	IQR	1313	66	0.685790	163900.0

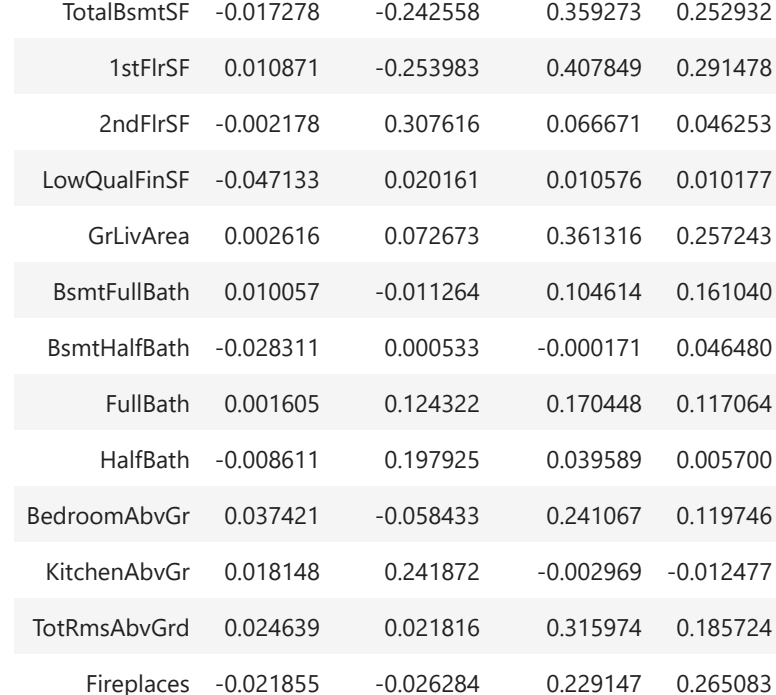
In [103…
```python
summary['Skewness'].plot(kind='bar')
plt.xticks(np.arange(3),['Raw Data','Z-Score','IQR'])
plt.show()
```



In [104…
```python
summary['Skewness'].plot(kind='line')
plt.xticks(np.arange(3),['Raw Data','Z-Score','IQR'])
plt.show()
```



In [106…
```python
summary['Outlier'].plot(kind='bar')
plt.xticks(np.arange(3),['Raw Data','Z-Score','IQR'])
plt.show()
```



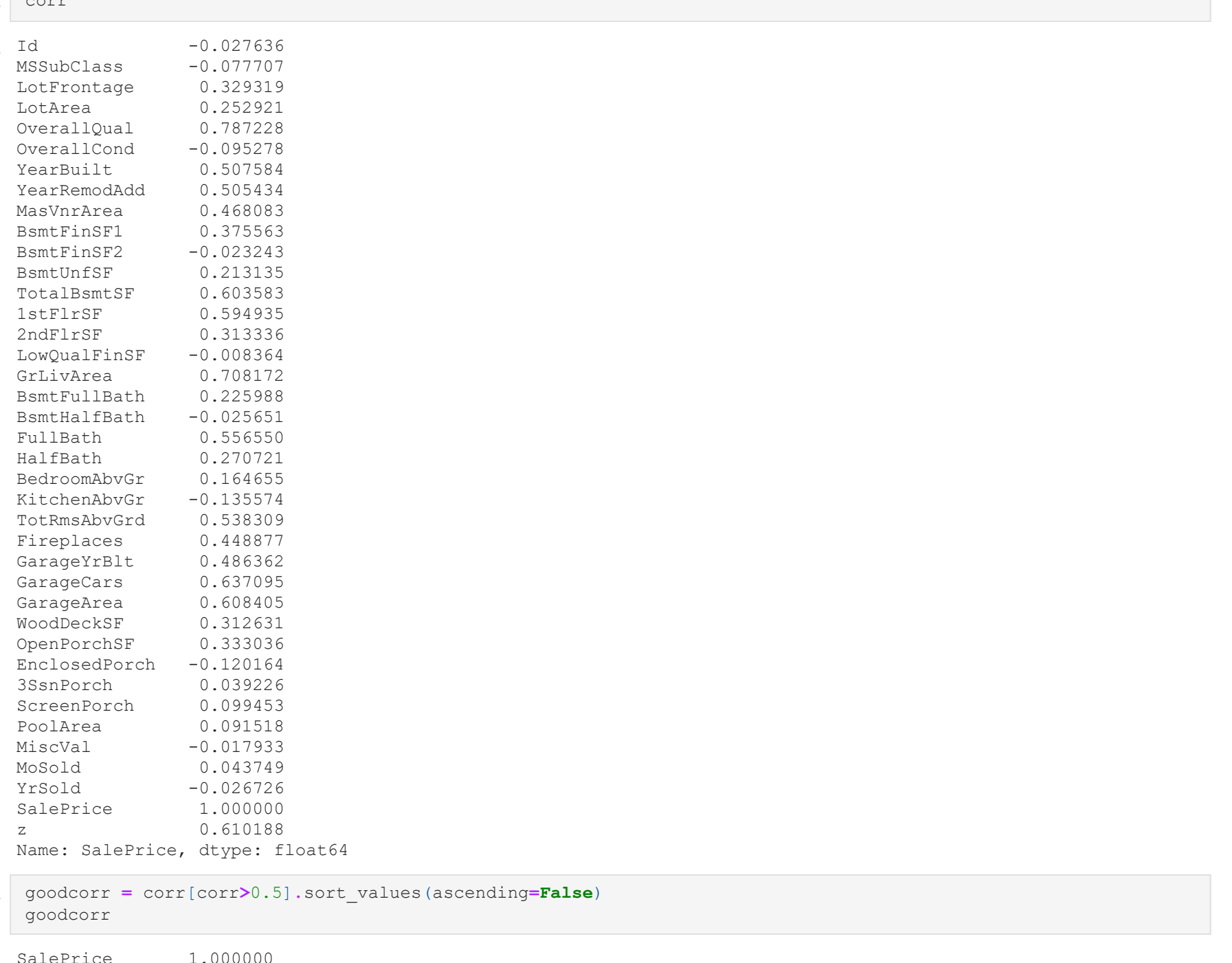
In [107…
```python
data.corr()
```

Out[107…

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1
Id	1.000000	0.019529	-0.010751	-0.035147	-0.041980	0.007146	-0.020228	-0.025519	-0.053973	-0.004083
MSSubClass	0.019529	1.000000	-0.357730	-0.136092	0.050092	-0.097027	0.060041	0.036926	0.033778	-0.067904
LotFrontage	-0.010751	-0.357730	1.000000	0.301260	0.226457	-0.042116	0.104325	0.081014	0.176295	0.220448
LotArea	-0.035147	-0.136092	0.301260	1.000000	0.090016	-0.002869	-0.005921	0.002764	0.096315	0.210145
OverallQual	-0.041980	0.050092	0.226457	0.090016	1.000000	-0.136232	0.572082	0.557772	0.403120	0.223690
OverallCond	0.007146	-0.097027	-0.042116	-0.002869	-0.136232	1.000000	-0.403602	0.048340	-0.135258	-0.050818
YearBuilt	-0.020228	0.060041	0.104325	-0.005921	0.572082	-0.403602	1.000000	0.618058	0.302720	0.229643
YearRemodAdd	-0.025519	0.036926	0.081014	0.002764	0.557772	0.048340	0.618058	1.000000	0.174329	0.114668
MasVnrArea	-0.053973	0.033778	0.176295	0.096315	0.403120	-0.135258	0.302720	0.174329	1.000000	0.257343
BsmtFinSF1	-0.004083	-0.067904	0.220448	0.210145	0.223690	-0.050818	0.229643	0.114668	0.257343	1.000000
BsmtFinSF2	-0.007278	-0.079326	0.041826	0.110011	-0.073166	0.038770	-0.066657	-0.077820	-0.078169	-0.054966
BsmtUnfSF	-0.010067	-0.138829	0.109461	-0.009707	0.318552	-0.140730	0.157525	0.188918	0.113865	-0.502954
TotalBsmtSF	-0.017278	-0.242558	0.359273	0.252932	0.531977	-0.182020	0.376977	0.283572	0.357466	0.521366
1stFlrSF	0.010871	-0.253983	0.407849	0.291478	0.467425	-0.147171	0.258161	0.235044	0.336894	0.446596
2ndFlrSF	-0.002178	0.307616	0.066671	0.046253	0.286109	0.016720	0.017152	0.128197	0.173770	-0.142969
LowQualFinSF	-0.047133	0.020161	0.010576	0.010177	-0.014550	0.043216	-0.159333	-0.063419	-0.061079	-0.050021
GrLivArea	0.002616	0.072673	0.361316	0.257243	0.589584	-0.092217	0.194663	0.278864	0.392325	0.208016
BsmtFullBath	0.010057	-0.011264	0.104614	0.161040	0.108955	-0.061775	0.181559	0.112381	0.085275	0.647400
BsmtHalfBath	-0.028311	0.000533	-0.000171	0.046480	-0.056924	0.121662	-0.054074	-0.020593	0.017518	0.060852
FullBath	0.001605	0.124322	0.170448	0.117064	0.556231	-0.218474	0.482739	0.444677	0.272766	0.044227
HalfBath	-0.008611	0.197925	0.039589	0.005700	0.264944	-0.062217	0.235722	0.182239	0.195650	-0.007579
BedroomAbvGr	0.037421	-0.058433	0.241067	0.119746	0.081714	0.014985	-0.071794	-0.067375	0.108825	-0.118036
KitchenAbvGr	0.018148	0.241872	-0.002969	-0.012477	-0.173051	-0.101463	-0.164898	-0.173640	-0.018467	-0.065825
TotRmsAbvGrd	0.024639	0.021816	0.315974	0.185724	0.426834	-0.068573	0.101539	0.173375	0.286585	0.044389
Fireplaces	-0.021855	-0.026284	0.229147	0.265083	0.380163	-0.026489	0.113546	0.100770	0.236022	0.253887
GarageYrBlt	0.000072	0.085072	0.066466	-0.024947	0.547766	-0.324297	0.825667	0.642277	0.252691	0.153484
GarageCars	0.011099	0.011426	0.270049	0.137978	0.581381	-0.247317	0.523349	0.450659	0.363550	0.201996
GarageArea	0.012683	-0.066485	0.332576	0.167622	0.528107	-0.196678	0.444951	0.382323	0.368884	0.289128
WoodDeckSF	-0.031905	-0.009195	0.071603	0.166600	0.227272	-0.005638	0.213102	0.204308	0.150878	0.194729
OpenPorchSF	-0.000082	-0.001663	0.146908	0.085056	0.319045	-0.053090	0.220775	0.235763	0.130223	0.120338
EnclosedPorch	0.001548	-0.022147	0.013035	-0.013427	-0.106609	0.063196	-0.378630	-0.210831	-0.105034	-0.096743
3SsnPorch	-0.048460	-0.043958	0.061589	0.018443	0.024314	0.026735	0.026285	0.043737	0.015528	0.023903
ScreenPorch	0.000460	-0.021732	0.032446	0.038605	0.050580	0.057507	-0.069431	-0.048102	0.054151	0.056216
PoolArea	0.058550	0.010296	0.185238	0.076868	0.064305	-0.002274	0.001167	0.004089	0.009797	0.141615
MiscVal	-0.004923	-0.021951	-0.000588	0.040288	-0.029078	0.079088	-0.033892	-0.003292	-0.029563	0.006846
MoSold	0.015639	-0.016801	0.017675	-0.000557	0.065338	-0.010564	0.008620	0.017090	-0.010239	-0.017515
YrSold	0.008721	-0.023073	0.010736	-0.013797	-0.019273	0.050309	-0.006809	0.039937	-0.004724	0.017754
SalePrice	-0.027636	-0.077707	0.329319	0.252921	0.787228	-0.095278	0.507584	0.505434	0.468083	0.375563
z	-0.025039	-0.076561	0.153219	0.144413	0.319140	-0.086742	0.038882	0.059486	0.339871	0.241549

39 rows × 39 columns

In [110…
```python
plt.figure(figsize=(15,10))
sns.heatmap(data.corr())
plt.show()
```



In [111…
```python
corr = data.corr()['SalePrice']
```

In [112…
```python
corr
```

Out[112…
```
Id              -0.027636
MSSubClass      -0.077707
LotFrontage      0.329319
LotArea          0.252921
OverallQual      0.787228
OverallCond     -0.095278
YearBuilt        0.507584
YearRemodAdd     0.505434
MasVnrArea       0.468083
BsmtFinSF1       0.375563
BsmtFinSF2      -0.023243
BsmtUnfSF        0.213135
TotalBsmtSF      0.603583
1stFlrSF         0.594935
2ndFlrSF         0.313336
LowQualFinSF    -0.008364
GrLivArea        0.708172
BsmtFullBath     0.225988
BsmtHalfBath    -0.025651
FullBath         0.556550
HalfBath         0.270721
BedroomAbvGr     0.164655
KitchenAbvGr    -0.135574
TotRmsAbvGrd     0.538309
Fireplaces       0.448877
GarageYrBlt      0.486362
GarageCars       0.637095
GarageArea       0.608405
WoodDeckSF       0.312631
OpenPorchSF      0.333036
EnclosedPorch   -0.120164
3SsnPorch        0.039226
ScreenPorch      0.099453
PoolArea         0.091518
MiscVal         -0.017933
MoSold           0.043749
YrSold          -0.026726
SalePrice        1.000000
z                0.610188
Name: SalePrice, dtype: float64
```

In [116…
```python
goodcorr = corr[corr>0.5].sort_values(ascending=False)
goodcorr
```

Out[116…
```
SalePrice       1.000000
OverallQual     0.787228
GrLivArea       0.708172
GarageCars      0.637095
z               0.610188
GarageArea      0.608405
TotalBsmtSF     0.603583
1stFlrSF        0.594935
FullBath        0.556550
TotRmsAbvGrd    0.538309
YearBuilt       0.507584
YearRemodAdd    0.505434
Name: SalePrice, dtype: float64
```

In [117…
```python
goodcorr.drop(['z'],inplace=True)
```

In [118…
```python
Hc = goodcorr.index
Hc
```

Out[118…
```
Index(['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'GarageArea',
       'TotalBsmtSF', '1stFlrSF', 'FullBath', 'TotRmsAbvGrd', 'YearBuilt',
       'YearRemodAdd'],
      dtype='object')
```

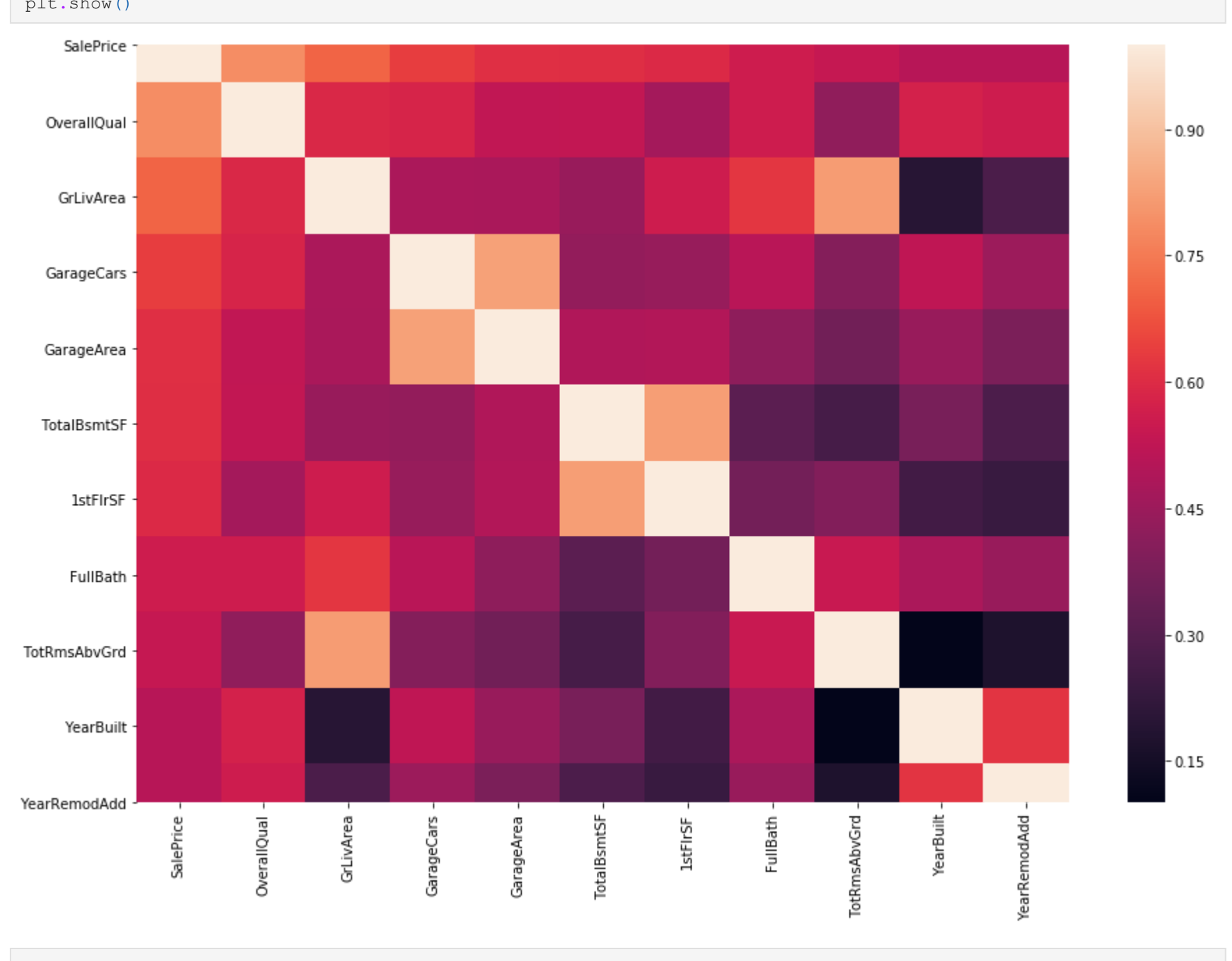
In [124…
```python
GD = data[Hc]
GD.head()  #In that we got the relavnt feature columns.by feeding the index.
```

Out[124…

	SalePrice	OverallQual	GrLivArea	GarageCars	GarageArea	TotalBsmtSF	1stFlrSF	FullBath	TotRmsAbvGrd	YearBuilt	YearRemodAdd
0	208500	7	1710	2	548	856	856	2	8	2003	2003
1	181500	6	1262	2	460	1262	1262	2	6	1976	1976
2	223500	7	1786	2	608	920	920	2	6	2001	2002
3	140000	7	1717	3	642	756	961	1	7	1915	1970
4	250000	8	2198	3	836	1145	1145	2	9	2000	2000

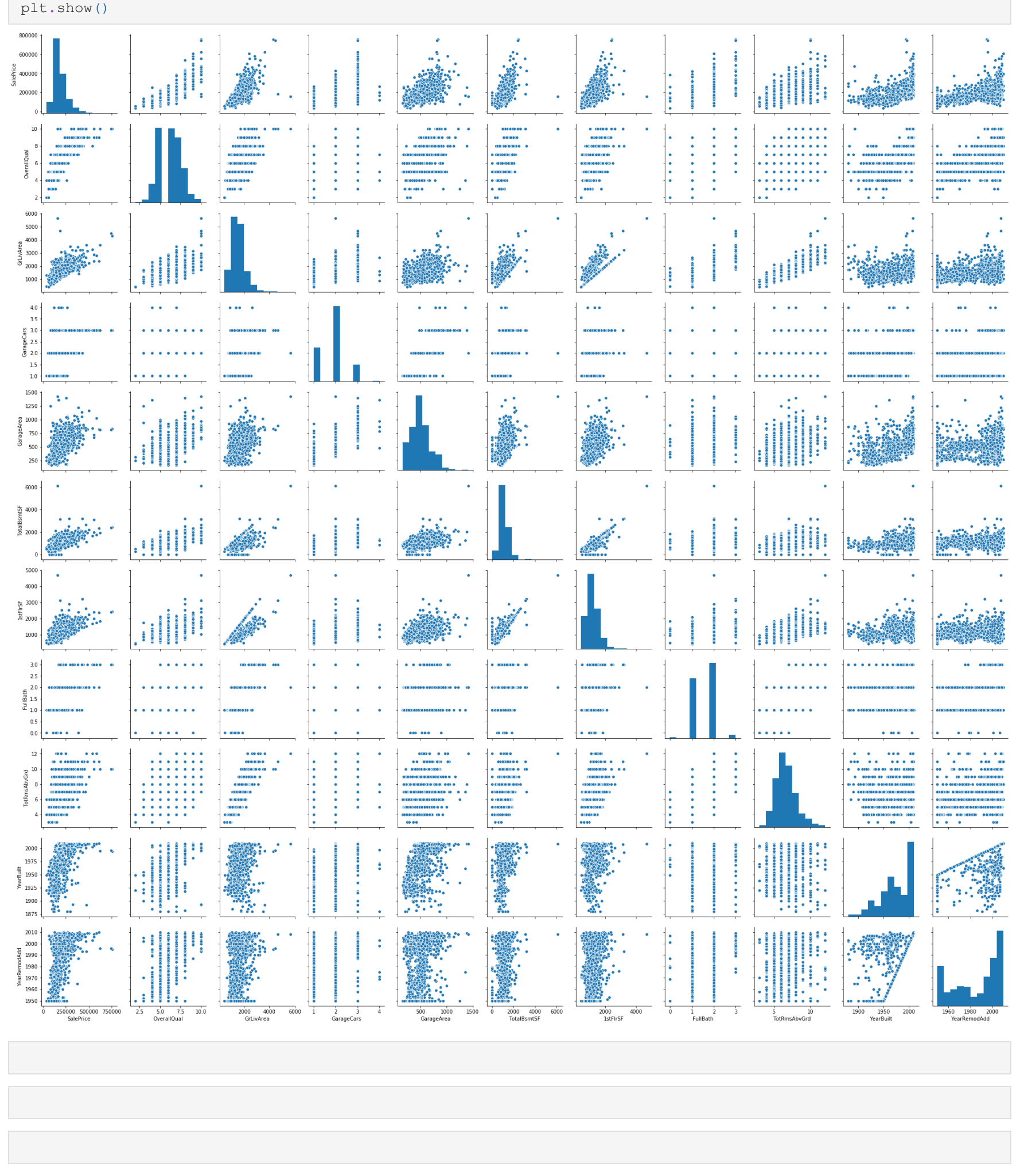
In [122…

```
plt.figure(figsize=(15,10))
sns.heatmap(GD.corr())
plt.show()
```



In [123…

```
sns.pairplot(GD)
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

```
In [2]:
import numpy as np
import pandas as pd
from pandas import DataFrame,Series
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import zscore
import warnings
warnings.filterwarnings('ignore')
from sklearn import preprocessing
%matplotlib inline
matplotlib.style.use('ggplot')
```

```
In [3]: from sklearn.datasets import load_boston
```

```
In [4]: boston=load_boston()
```

```
In [5]: print(boston.keys())
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

```
In [6]: print(boston.data.shape)
```

```
(506, 13)
```

```
In [7]: print(boston.feature_names)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

```
In [8]: print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the targe
t.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity',
Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth Internat
ional Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
```

```
In [9]: bos=DataFrame(boston.data)
```

```
In [10]: bos.head()
```

Out[10]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

```
In [11]: bos.columns=boston.feature_names
```

```
In [12]: bos
```

Out[12]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

506 rows × 13 columns

```
In [13]: bos['Price']=boston.target
```

```
In [14]: bos
```

Out[14]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88	11.9

506 rows × 14 columns

```
In [15]: bos.head(5)
```

Out[15]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

```
In [16]: bos.describe()
```

Out[16]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.

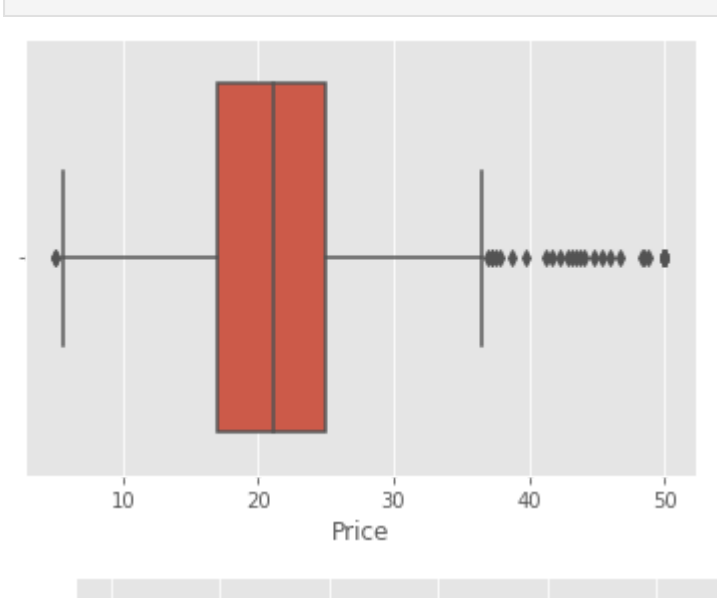
```
In [17]: bos.info()
```

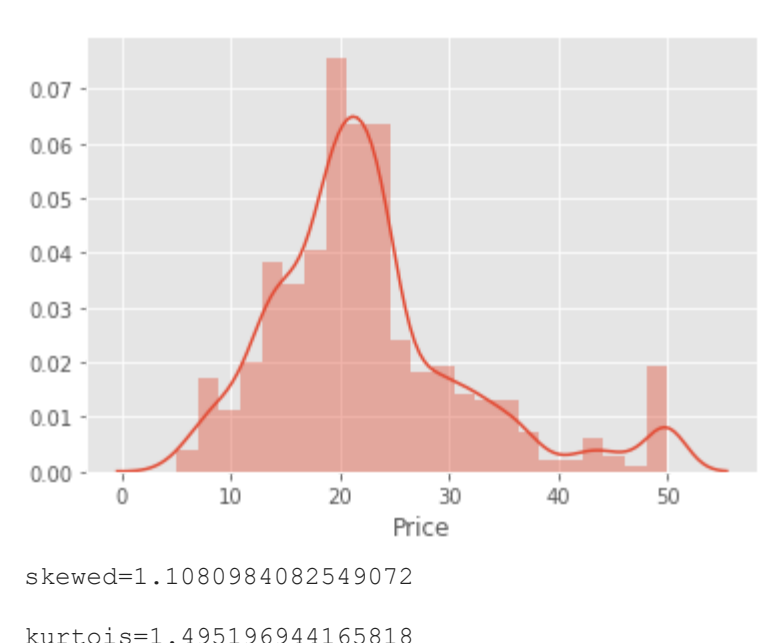
```
&lt;class 'pandas.core.frame.DataFrame'&gt;
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
CRIM       506 non-null float64
ZN         506 non-null float64
INDUS      506 non-null float64
CHAS       506 non-null float64
NOX        506 non-null float64
RM         506 non-null float64
AGE        506 non-null float64
DIS        506 non-null float64
RAD        506 non-null float64
TAX        506 non-null float64
PTRATIO    506 non-null float64
B          506 non-null float64
LSTAT      506 non-null float64
Price      506 non-null float64
dtypes: float64(14)
memory usage: 55.4 KB
```

```
In [18]: bos['Price'].describe()
```

```
Out[18]:
count    506.000000
mean      22.532806
std        9.197104
min        5.000000
25%       17.025000
50%       21.200000
75%       25.000000
max       50.000000
Name: Price, dtype: float64
```

```
In [19]:
sns.boxplot(bos['Price'])
plt.show()
sns.distplot(bos['Price'])
plt.show()
print('skewed={}\n'.format(bos['Price'].skew()))
print('kurtois={}\n'.format(bos['Price'].kurt()))
```






```
skewed=1.1080984082549072

kurtois=1.495196944165818
```

```
In [20]: bos.head(2)
```

Out[20]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.9	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.9	9.14	21.6

```
In [21]:
x=bos.iloc[:,:13].values
x
```

```
Out[21]: array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
         4.9800e+00],
        [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
         9.1400e+00],
        [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
         4.0300e+00],
        ...,
        [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         5.6400e+00],
        [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
         6.4800e+00],
        [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         7.8800e+00]])
```

```
In [22]:
y=bos.iloc[:,13].values
y
```

```
Out[22]: array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
       18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
       15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
       13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
       21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
       35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
       19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
       20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
       23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
       33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
       21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
       20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
       23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
       15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
       17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
       25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
       23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
       32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
       34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
       20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
       26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
       31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
       22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
       42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
       36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
       32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
       20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
       20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
       22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
       21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
       19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
       32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
       18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
       16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
       13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
        7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
       12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
       27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
        8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
        9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
       10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
       15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
       19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
       29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
       20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
       23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9])
```

```
In [23]: from sklearn.model_selection import train_test_split
```

```
In [24]: x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=0)
```

```
In [25]: x_test.shape
```

```
Out[25]: (127, 13)
```

```
In [26]: x_train.shape
```

```
Out[26]: (379, 13)
```

```
In [27]: y_train.shape
```

```
Out[27]: (379,)
```

```
In [28]: y_test.shape
```

```
Out[28]: (127,)
```

```
In [29]: from sklearn.linear_model import LinearRegression
```

```
In [30]:
regg=LinearRegression()
regg
```

```
Out[30]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
In [31]: regg.fit(x,y)
```

```
Out[31]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
In [32]: y_predict_x=regg.predict(x)
```

```
In [33]: from sklearn import metrics
```

```
In [34]:
r2_score=metrics.r2_score(y,y_predict_x)
r2_score
```

```
Out[34]: 0.7406426641094095
```

```
In [35]: regg.fit(x_train,y_train)
```

```
Out[35]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
In [36]: y_predict_train=regg.predict(x_train)
```

```
In [37]:
r2_score=metrics.r2_score(y_train,y_predict_train)
r2_score
```

```
Out[37]: 0.7697699488741149
```

```
In [38]: regg.fit(x_test,y_test)
```

```
Out[38]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
In [39]: y_predict_test=regg.predict(x_test)
```

```
In [40]:
r2_score=metrics.r2_score(y_test,y_predict_test)
r2_score
```

```
Out[40]: 0.690382480076236
```

Conclusion:--

```
In [41]:
print('x & y:--',metrics.r2_score(y,y_predict_x))
print('x_train & y_train:--',metrics.r2_score(y_train,y_predict_train))####data is overfitt because of train >
print('x_test & y_test:--',metrics.r2_score(y_test,y_predict_test))
```

```
x & y:-- 0.7406426641094095
x_train & y_train:-- 0.7697699488741149
x_test & y_test:-- 0.690382480076236
```

```
In [42]: bos.head(1)
```

Out[42]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.09	1.0	296.0	15.3	396.9	4.98	24.0

OLS:- Ordinary Least Square Method

```
In [43]: import statsmodels.api as sm
```

```
In [44]: #! pip install statsmodels  #To install
```

```
In [45]:
x=bos['LSTAT']
Y=bos['Price']
```

```
In [46]: x_constant=sm.add_constant(x)
```

```
In [ ]:
```

```
In [47]: x_constant
```

Out[47]:

	const	LSTAT
0	1.0	4.98
1	1.0	9.14
2	1.0	4.03
3	1.0	2.94
4	1.0	5.33
...	...	...
501	1.0	9.67
502	1.0	9.08
503	1.0	5.64
504	1.0	6.48
505	1.0	7.88

506 rows × 2 columns

```
In [48]: linear_reg=sm.OLS(y,x_constant).fit()
```

```
In [49]: linear_reg
```

```
Out[49]: &lt;statsmodels.regression.linear_model.RegressionResultsWrapper at 0xf8e5cb0&gt;
```

```
In [50]: linear_reg.summary()
```

Out[50]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.544
Model:	OLS	Adj. R-squared:	0.543
Method:	Least Squares	F-statistic:	601.6
Date:	Sat, 01 Feb 2020	Prob (F-statistic):	5.08e-88
Time:	23:32:03	Log-Likelihood:	-1641.5
No. Observations:	506	AIC:	3287.
Df Residuals:	504	BIC:	3295.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	34.5538	0.563	61.415	0.000	33.448	35.659
LSTAT	-0.9500	0.039	-24.528	0.000	-1.026	-0.874

Omnibus:	137.043	Durbin-Watson:	0.892
Prob(Omnibus):	0.000	Jarque-Bera (JB):	291.373
Skew:	1.453	Prob(JB):	5.36e-64
Kurtosis:	5.319	Cond. No.	29.7

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

ASSUMPTIONS

```
In [51]:
x=bos.iloc[:,:13]
y=bos['Price']
```

```
In [52]: y
```

```
Out[52]:
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: Price, Length: 506, dtype: float64
```

```
In [53]: x_constant=sm.add_constant(x)
```

```
In [54]: linear_reg=sm.OLS(y,x_constant).fit()
```

```
In [55]: linear_reg.summary()
```

Out[55]:

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.741
Model:	OLS	Adj. R-squared:	0.734
Method:	Least Squares	F-statistic:	108.1
Date:	Sat, 01 Feb 2020	Prob (F-statistic):	6.72e-135
Time:	23:32:03	Log-Likelihood:	-1498.8
No. Observations:	506	AIC:	3026.
Df Residuals:	492	BIC:	3085.
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	36.4595	5.103	7.144	0.000	26.432	46.487
CRIM	-0.1080	0.033	-3.287	0.001	-0.173	-0.043
ZN	0.0464	0.014	3.382	0.001	0.019	0.073
INDUS	0.0206	0.061	0.334	0.738	-0.100	0.141
CHAS	2.6867	0.862	3.118	0.002	0.994	4.380
NOX	-17.7666	3.820	-4.651	0.000	-25.272	-10.262
RM	3.8099	0.418	9.116	0.000	2.989	4.631
AGE	0.0007	0.013	0.052	0.958	-0.025	0.027
DIS	-1.4756	0.199	-7.398	0.000	-1.867	-1.084
RAD	0.3060	0.066	4.613	0.000	0.176	0.436
TAX	-0.0123	0.004	-3.280	0.001	-0.020	-0.005
PTRATIO	-0.9527	0.131	-7.283	0.000	-1.210	-0.696
B	0.0093	0.003	3.467	0.001	0.004	0.015
LSTAT	-0.5248	0.051	-10.347	0.000	-0.624	-0.425

Omnibus:	178.041	Durbin-Watson:	1.078
Prob(Omnibus):	0.000	Jarque-Bera (JB):	783.126
Skew:	1.521	Prob(JB):	8.84e-171
Kurtosis:	8.281	Cond. No.	1.51e+04

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.51e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

RESIDUALS

```
In [56]: linear_reg.resid.head() ####actual-predict
```

```
Out[56]:
0   -6.003843
1   -3.425562
2    4.132403
3    4.792964
4    8.256476
dtype: float64
```

```
In [57]: linear_reg.predict(x_constant).head() #### it is an y_predict value
```

```
Out[57]:
0    30.003843
1    25.025562
2    30.567597
3    28.607036
4    27.943524
dtype: float64
```

```
In [58]: y.head()
```

```
Out[58]:
0    24.0
1    21.6
2    34.7
3    33.4
4    36.2
Name: Price, dtype: float64
```

1.AutoCorrelation

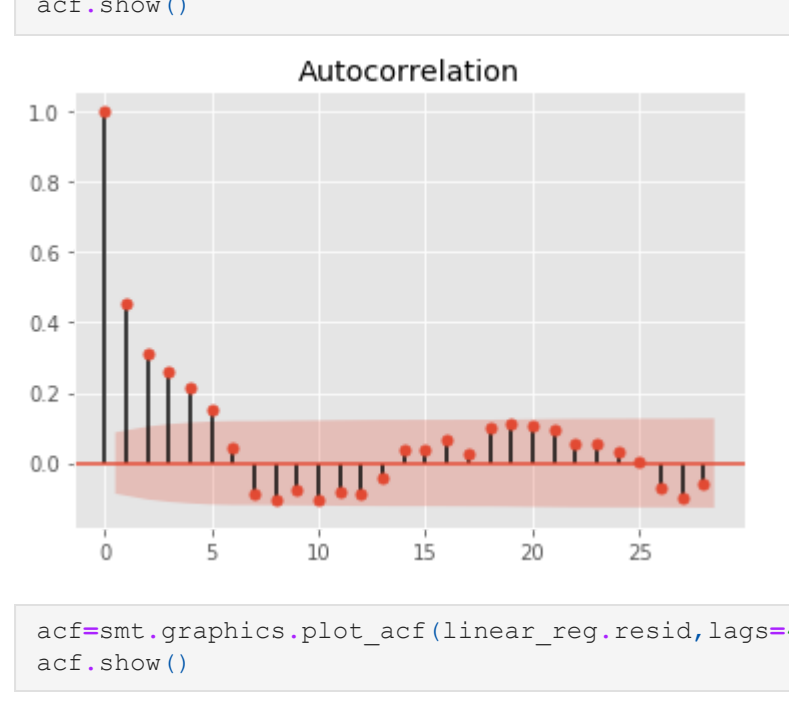
Durbin-Watson test

is used to check autocorrelation between resid.range 0 to 4.0 to 2 is positive correlation,2 is no autocorrelation,2 to 4 is negative correlation.

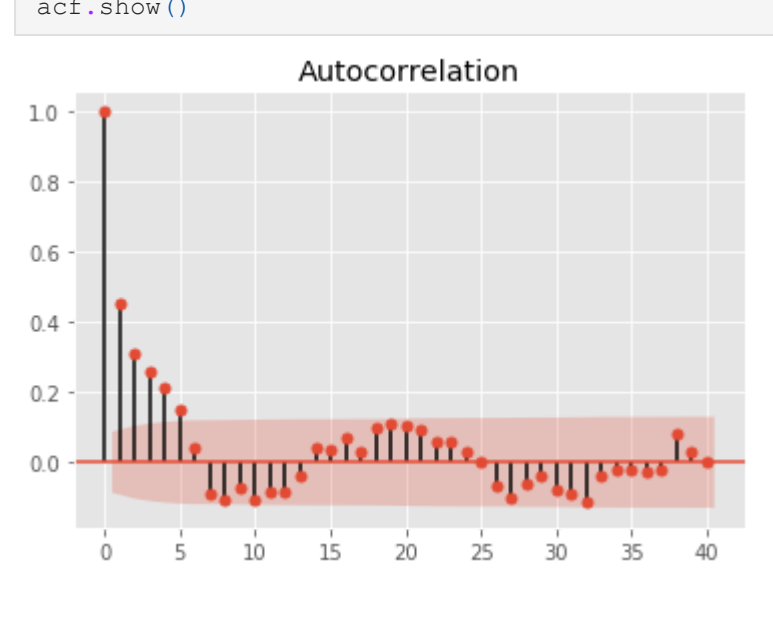
Graphical method to analyse autocorrelation

```
In [59]: import statsmodels.tsa.api as smt
```

```
In [60]:
acf=smt.graphics.plot_acf(linear_reg.resid)
acf.show()
```

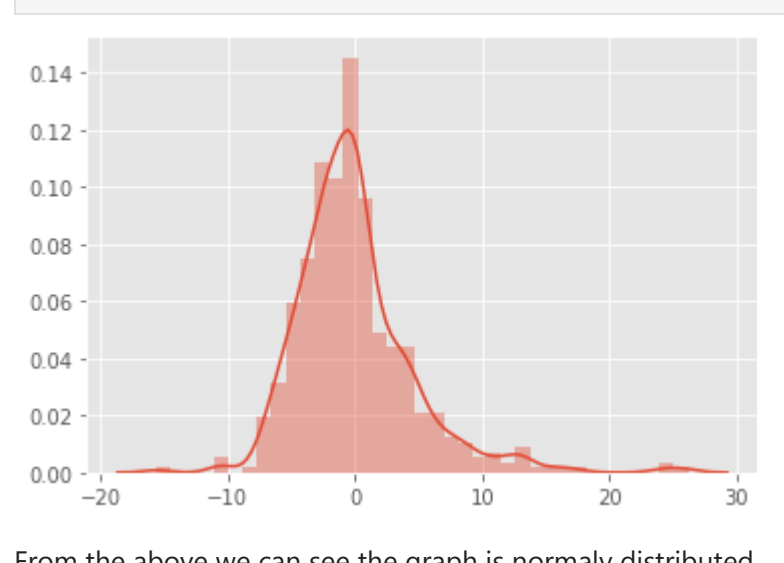



```
In [61]:
acf=smt.graphics.plot_acf(linear_reg.resid,lags=40)
acf.show()
```



2nd Assumptions:Multivariate Normalality.

```
In [62]:
sns.distplot(linear_reg.resid)
plt.show()
```



From the above we can see the graph is normaly distributed.

test2 :-Jarque Bera Test.

This test is for goodness of fit test of wheather the sample data have skweness and kulrtoisis matching to the normly distributed or not.

Null Hypothesis:- Error should be normally distrubted

Alternative hypothesis :- Error terms are not normally distributed.

When the significance is greater than 5.99 that time we can say error terms are normally distributed.

When the significance is lesser than 5.99 that time we can say the error terms are normally distributed.

```
In [63]: from scipy import stats
```

```
In [64]: stats.jarque_bera(linear_reg.resid)
```

```
Out[64]: (783.1262780507636, 0.0)
```

Left one is the significant value and it is geater than 5.99(significant value) that means the it is normally distributed.

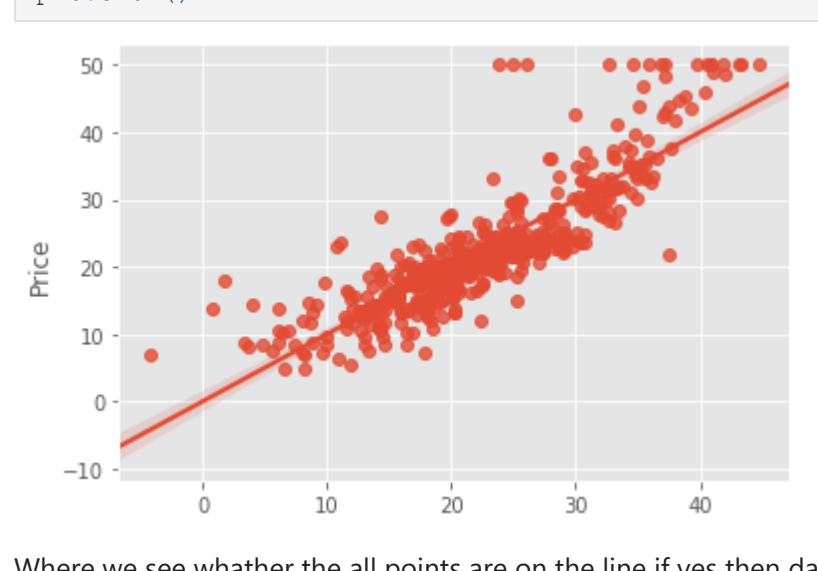
significant value for jaque Bera is 5.99.

To get the first value of the significant:-

```
stats.jarque_bera(linear_reg.resid)[0]
```

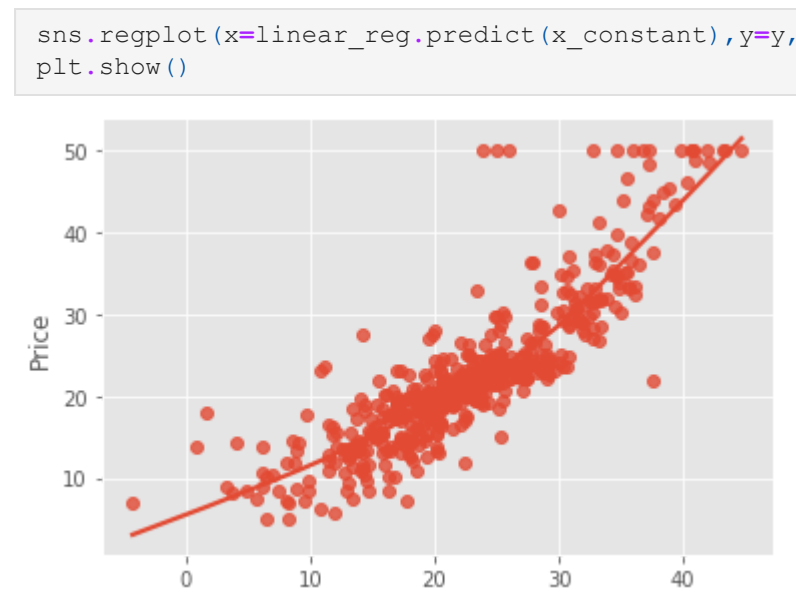
3.Linearity.

```
In [65]:
sns.regplot(x=linear_reg.predict(x_constant),y=y)
plt.show()
```



Where we see whather the all points are on the line if yes then data is linear.

```
In [66]:
sns.regplot(x=linear_reg.predict(x_constant),y=y,lowess=True)
plt.show()
```

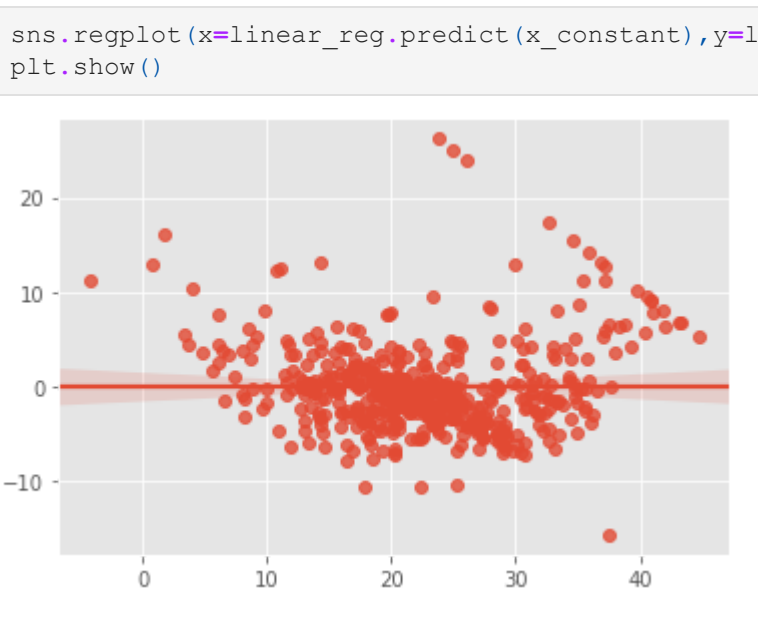


To check the more accurately linearity by using lowness.

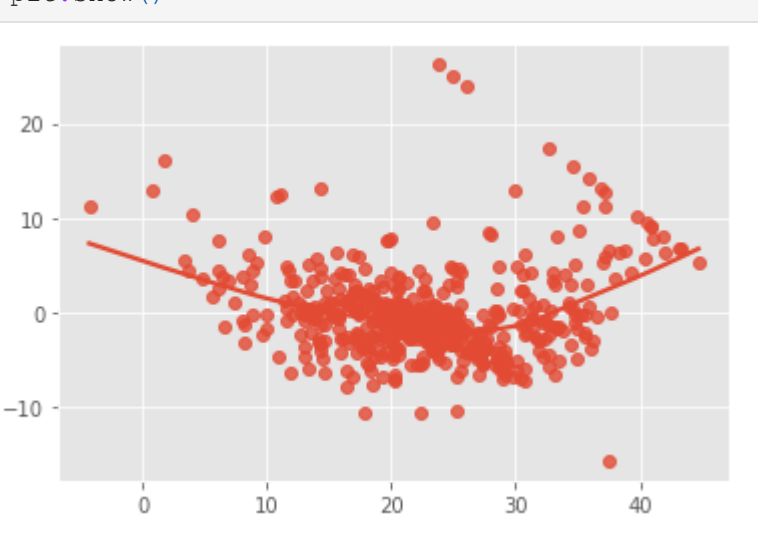
To cross check the whather our data is linear or not.

method 2:-

```
In [67]:
sns.regplot(x=linear_reg.predict(x_constant),y=linear_reg.resid)
plt.show()
```




```
In [68]:
sns.regplot(x=linear_reg.predict(x_constant),y=linear_reg.resid,lowess=True)
plt.show()
```



method 3:- RainBow Test

Null Hypothesis is the belongs to linear hypothesis. Alternate hypothesis is belongs to non-linear Hypothesis.

```
In [69]: sm.stats.diagnostic.linear_rainbow(res=linear_reg)
```

```
Out[69]: (0.580850763961426, 0.999988852262534)
```

in the return we having the first value is f-test and second is the p-test. here we can see that the pvalue is greter than the 0.05 that means the model is linear.

here we can see that model is linear model.

Method 4:-

if residual mean is close to the zero that means the the model is linear.

```
In [70]: np.mean(linear_reg.resid)
```

```
Out[70]: 1.227301089040173e-13
```

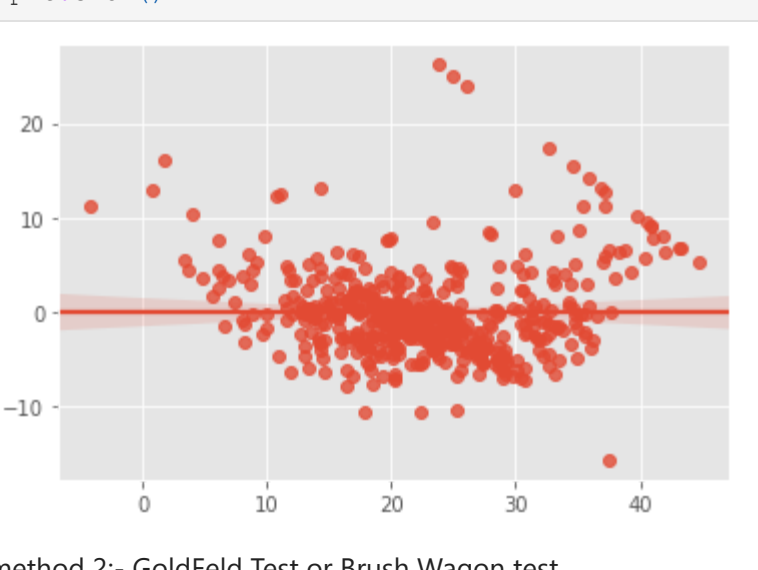
Since the residual mean is close to zero then it is linear.

Homosidacity.

If the variance of the residual are symmerically distributed accorss the residual line then data is said to homesidasctic.

If the variance is uneqaul residual acorss the residual line then data is said to hetrosidacity.

```
In [71]:
sns.regplot(x=linear_reg.predict(x_constant),y=linear_reg.resid)
plt.show()
```



method 2:- GoldFeld Test or Brush Wagon test.

When the varince is constant across the homosidaesity belongs to null hypothesis voice versa.

```
In [72]: import statsmodels.stats.api as sms
```

```
In [73]: sms.het_goldfeldquandt(linear_reg.resid,linear_reg.model.exog)
```

```
Out[73]: (2.4311473253497438, 7.302071915608868e-12, 'increasing')
```

It return the t-value,p_value,order

Here we can see that pvalue is lesser than the 0.05 that means the it rejeted the null hypothesis and accepted the alternate hypothesis.It relate to the hetrosidasticity.

```
In [74]: linear_reg.model.exog.shape
```

```
Out[74]: (506, 14)
```

```
In [7]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import warnings
        warnings.filterwarnings('ignore')
```

```
In [8]: data = pd.read_csv('Ecommerce Customers.csv')
```

```
In [9]: data.head()
```

Out[9]:

	Email	Address	Avatar	Avg. Session Length	Time on App	Time on Website	Length of Membership	Yearly Amount Spent
0	mstephenson@fernandez.com	835 Frank Tunnel\nWrightmouth, MI 82180-9605	Violet	34.497268	12.655651	39.577668	4.082621	587.951054
1	hduke@hotmail.com	4547 Archer Common\nDiazchester, CA 06566-8576	DarkGreen	31.926272	11.109461	37.268959	2.664034	392.204933
2	pallen@yahoo.com	24645 Valerie Unions Suite 582\nCobbborough, D...	Bisque	33.000915	11.330278	37.110597	4.104543	487.547505
3	riverarebecca@gmail.com	1414 David Throughway\nPort Jason, OH 22070-1220	SaddleBrown	34.305557	13.717514	36.721283	3.120179	581.852344
4	mstephens@davidson-herman.com	14023 Rodriguez Passage\nPort Jacobville, PR 3...	MediumAquaMarine	33.330673	12.795189	37.536653	4.446308	599.406092

```
In [10]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
Email                   500 non-null object
Address                 500 non-null object
Avatar                  500 non-null object
Avg. Session Length     500 non-null float64
Time on App             500 non-null float64
Time on Website         500 non-null float64
Length of Membership    500 non-null float64
Yearly Amount Spent     500 non-null float64
dtypes: float64(5), object(3)
memory usage: 25.5+ KB
```

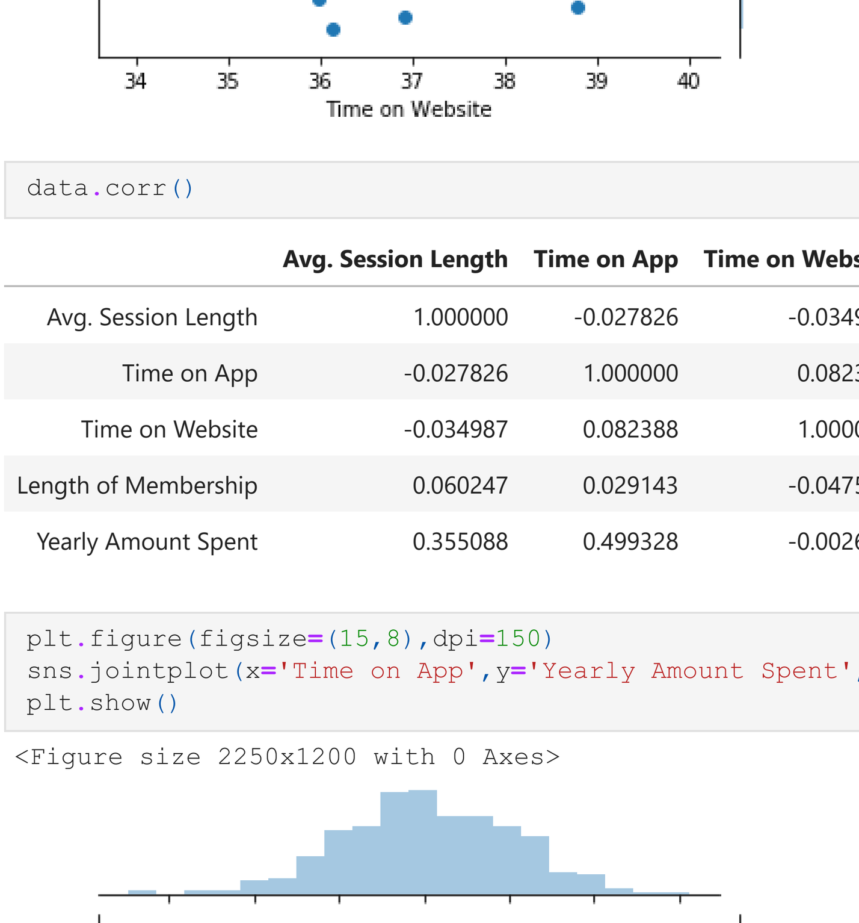
```
In [11]: data.describe()
```

Out[11]:

	Avg. Session Length	Time on App	Time on Website	Length of Membership	Yearly Amount Spent
count	500.000000	500.000000	500.000000	500.000000	500.000000
mean	33.053194	12.052488	37.060445	3.533462	499.314038
std	0.992563	0.994216	1.010489	0.999278	79.314782
min	29.532429	8.508152	33.913847	0.269901	256.670582
25%	32.341822	11.388153	36.349257	2.930450	445.038277
50%	33.082008	11.983231	37.069367	3.533975	498.887875
75%	33.711985	12.753850	37.716432	4.126502	549.313828
max	36.139662	15.126994	40.005182	6.922689	765.518462

Use seaborn to create a jointplot to compare the Time on Website and Yearly Amount Spent columns. Does the correlation make sense?

```
In [14]: plt.figure(figsize=(15,8),dpi=150)
         sns.jointplot(x='Time on Website',y='Yearly Amount Spent',data=data)
         plt.show()

<Figure size 2250x1200 with 0 Axes>
```

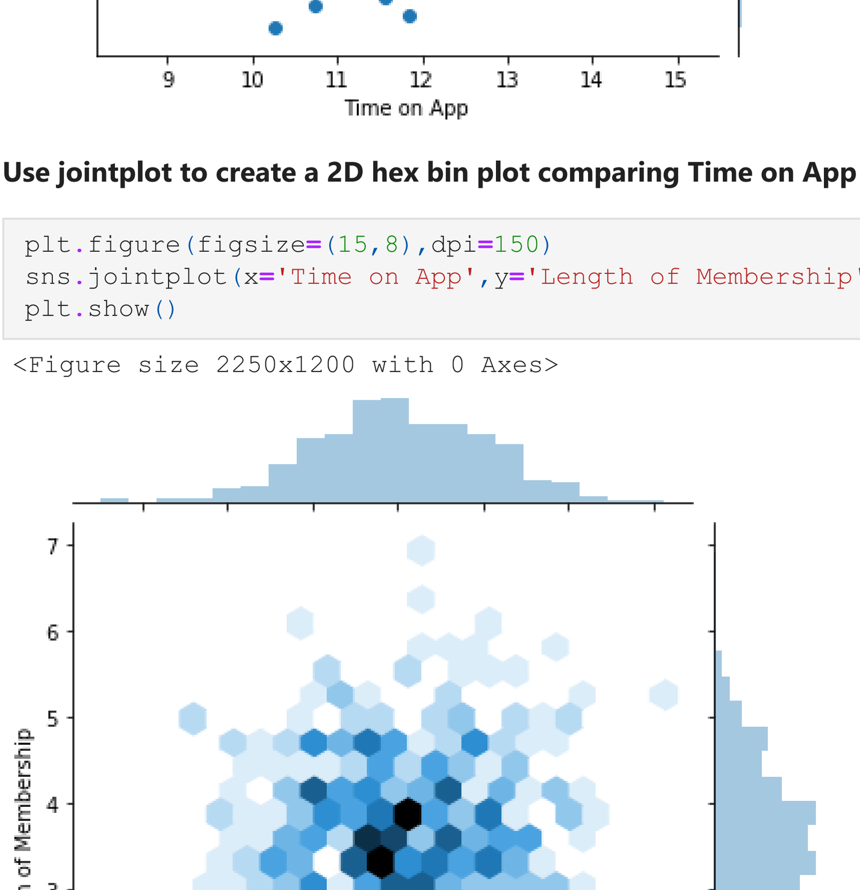



```
In [15]: data.corr()
```

Out[15]:

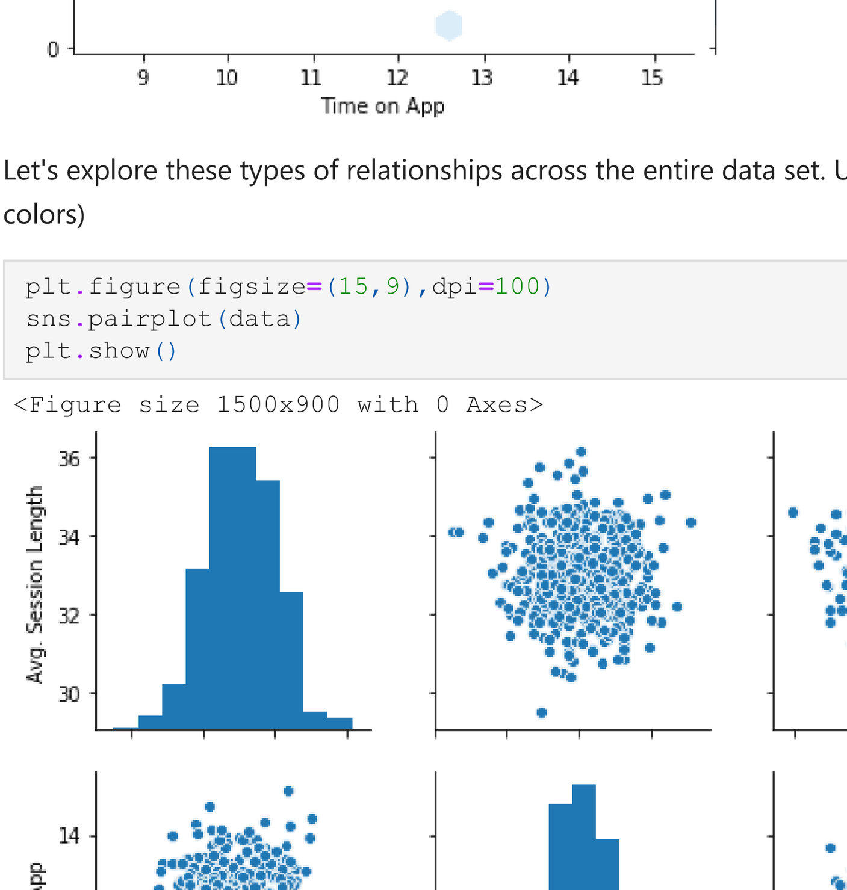
	Avg. Session Length	Time on App	Time on Website	Length of Membership	Yearly Amount Spent
Avg. Session Length	1.000000	-0.027826	-0.034987	0.060247	0.355088
Time on App	-0.027826	1.000000	0.082388	0.029143	0.499328
Time on Website	-0.034987	0.082388	1.000000	-0.047582	-0.002641
Length of Membership	0.060247	0.029143	-0.047582	1.000000	0.809084
Yearly Amount Spent	0.355088	0.499328	-0.002641	0.809084	1.000000

```
In [16]: plt.figure(figsize=(15,8),dpi=150)
         sns.jointplot(x='Time on App',y='Yearly Amount Spent',data=data)
         plt.show()

<Figure size 2250x1200 with 0 Axes>
```



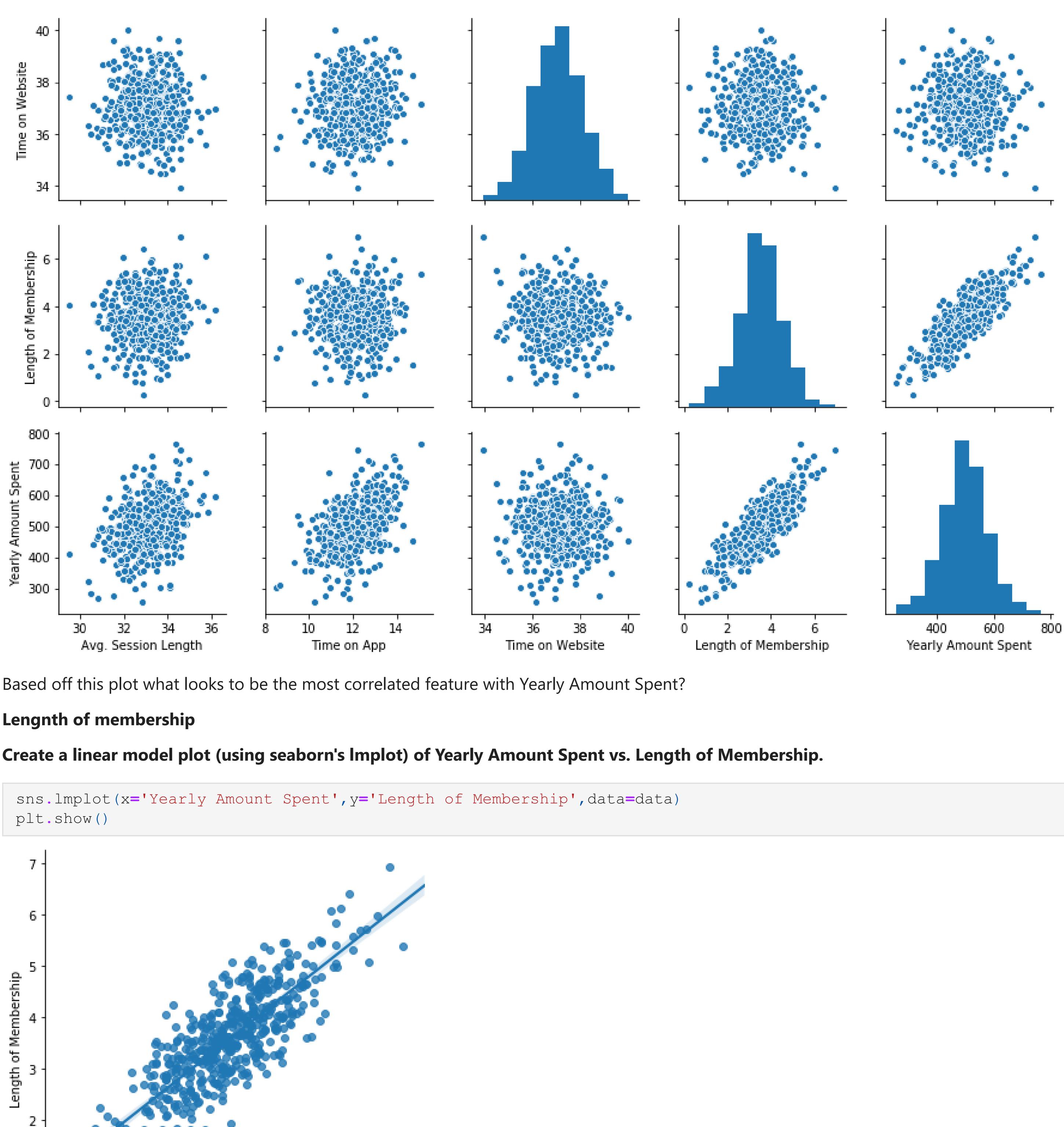
Use jointplot to create a 2D hex bin plot comparing Time on App and Length of Membership.

```
In [18]: plt.figure(figsize=(15,8),dpi=150)
         sns.jointplot(x='Time on App',y='Length of Membership',data=data,kind='hex')
         plt.show()

<Figure size 2250x1200 with 0 Axes>
```



Let's explore these types of relationships across the entire data set. Use pairplot to recreate the plot below.(Don't worry about the the colors)

```
In [19]: plt.figure(figsize=(15,9),dpi=100)
         sns.pairplot(data)
         plt.show()

<Figure size 1500x900 with 0 Axes>
```

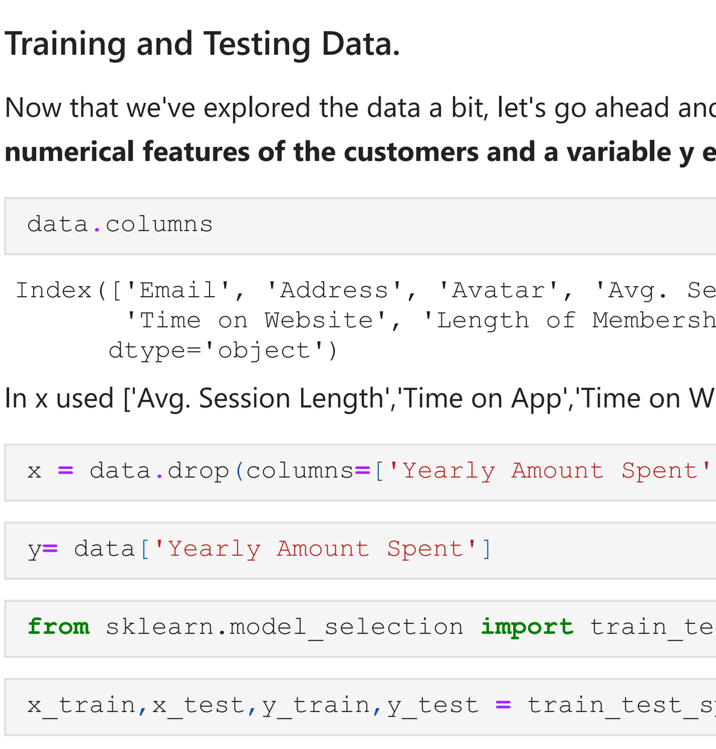


Based off this plot what looks to be the most correlated feature with Yearly Amount Spent?

Lenngth of membership

Create a linear model plot (using seaborn's lmplot) of Yearly Amount Spent vs. Length of Membership.

```
In [20]: sns.lmplot(x='Yearly Amount Spent',y='Length of Membership',data=data)
         plt.show()
```



Training and Testing Data.

Now that we've explored the data a bit, let's go ahead and split the data into training and testing sets. **Set a variable X equal to the numerical features of the customers and a variable y equal to the "Yearly Amount Spent" column.**

```
In [21]: data.columns

Out[21]: Index(['Email', 'Address', 'Avatar', 'Avg. Session Length', 'Time on App',
                'Time on Website', 'Length of Membership', 'Yearly Amount Spent'],
               dtype='object')
```

In x used ['Avg. Session Length','Time on App','Time on Website','Length of Membership'] this columns of values

```
In [35]: x = data.drop(columns=['Yearly Amount Spent','Email','Address','Avatar'])
```

```
In [36]: y= data['Yearly Amount Spent']
```

```
In [37]: from sklearn.model_selection import train_test_split
```

```
In [38]: x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=.80,random_state=0)
```

```
In [39]: x_train.shape,y_train.shape

Out[39]: ((400, 4), (400,))
```

```
In [40]: y_train.shape,y_test.shape

Out[40]: ((400,), (100,))
```

```
In [41]: from sklearn.linear_model import LinearRegression
```

```
In [42]: model = LinearRegression()
         model.fit(x_train,y_train)

Out[42]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
In [44]: y_pred=model.predict(x_test)
         y_pred

Out[44]: array([438.46488066, 489.6618454 , 370.06954186, 513.8590556 ,
                495.69799868, 450.88726525, 458.01303756, 336.6655623 ,
                528.51544919, 663.27832042, 559.42244861, 526.02431274,
                552.20571747, 457.67586182, 450.15607779, 431.65097987,
                563.58511617, 408.35440214, 599.61853988, 486.84826737,
                716.65493547, 496.79327186, 411.49845257, 556.55139426,
                551.84576857, 398.18338551, 421.30757507, 395.15395642,
                548.52797678, 463.48718386, 558.43909636, 505.52726976,
                505.87531544, 463.94575658, 425.98372449, 401.07293164,
                503.60337906, 430.34122821, 569.63274307, 421.89472114,
                487.26750654, 602.37423872, 616.44671945, 589.39491734,
                449.73449152, 379.92615446, 558.17328317, 462.72008747,
                502.16865444, 401.43462317, 523.84624174, 416.80395478,
                576.8712195 , 478.50455211, 457.24801931, 397.67512992,
                564.8697802 , 305.13130981, 562.63197525, 476.33492249,
                499.04418098, 433.98746406, 463.72067764, 576.91811781,
                504.00395793, 356.62137493, 508.85162889, 416.95315655,
                634.98001693, 455.88862211, 451.89537374, 767.06469393,
                518.9651027 , 457.34831524, 379.85564649, 375.87959328,
                598.72889754, 515.75175435, 513.36496692, 429.46162916,
                491.12451373, 491.58530432, 532.48659843, 494.43012939,
                441.61268032, 507.23551171, 558.93278557, 753.43951807,
                395.91913096, 516.9802983 , 518.31262767, 523.33224278,
                384.28654972, 486.27601323, 505.10493179, 383.74450882,
                524.16819228, 500.68268787, 606.78782704, 408.80519096])
```

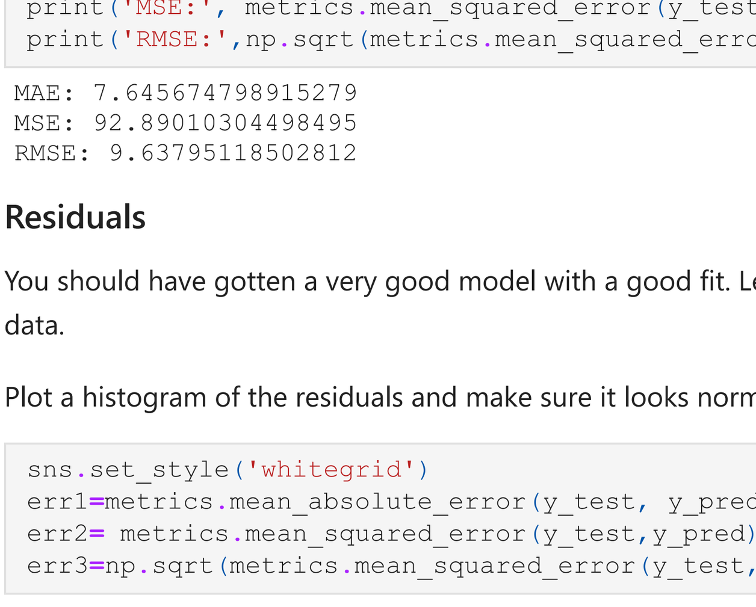
```
In [45]: from sklearn.metrics import r2_score
```

```
In [46]: print('R2 Score :',r2_score(y_test,y_pred))

R2 Score : 0.9861924261981548
```

```
In [47]: model.coef_

Out[47]: array([25.88815047, 38.87046474,  0.47066154, 61.78369022])
```

Create a scatterplot of the real test values versus the predicted values.

```
In [48]: sns.scatterplot(x=y_test,y=y_pred)
         plt.show()
```




```
In [51]: # calculate these metrics by hand!
         from sklearn import metrics

         print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
         print('MSE:', metrics.mean_squared_error(y_test,y_pred))
         print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test,y_pred)))

MAE: 7.645674798915279
MSE: 92.89010304498495
RMSE: 9.63795118502812
```

Residuals

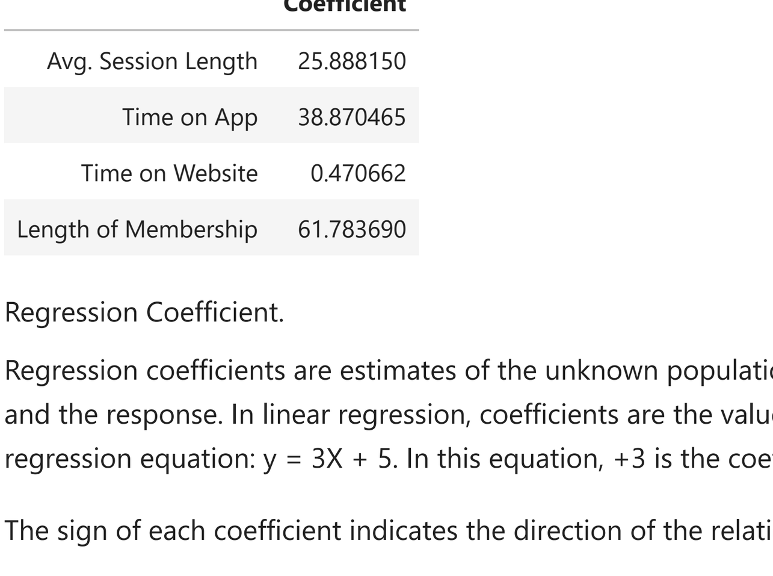
You should have gotten a very good model with a good fit. Let's quickly explore the residuals to make sure everything was okay with our data.

Plot a histogram of the residuals and make sure it looks normally distributed. Use either seaborn distplot, or just plt.hist().

```
In [66]: sns.set_style('whitegrid')
         err1=metrics.mean_absolute_error(y_test, y_pred)
         err2= metrics.mean_squared_error(y_test,y_pred)
         err3=np.sqrt(metrics.mean_squared_error(y_test,y_pred))
```

```
In [74]: error =np.array([err1,err2,err3])
```

```
In [77]: sns.distplot(data['Yearly Amount Spent'])

Out[77]: <matplotlib.axes._subplots.AxesSubplot at 0x146fb330>
```




```
In [92]: dataframe = pd.DataFrame({'Coefficient':model.coef_},index=['Avg. Session Length','Time on App','Time on Websit
```

```
In [93]: dataframe
```

Out[93]:

	Coefficient
Avg. Session Length	25.888150
Time on App	38.870465
Time on Website	0.470662
Length of Membership	61.783690

Regression Coefficient.

Regression coefficients are estimates of the unknown population parameters and describe the relationship between a predictor variable and the response. In linear regression, coefficients are the values that multiply the predictor values. Suppose you have the following regression equation: $y = 3X + 5$. In this equation, +3 is the coefficient, X is the predictor, and +5 is the constant.

The sign of each coefficient indicates the direction of the relationship between a predictor variable and the response variable.

A positive sign indicates that as the predictor variable increases, the response variable also increases. A negative sign indicates that as the predictor variable increases, the response variable decreases. The coefficient value represents the mean change in the response given a one unit change in the predictor. For example, if a coefficient is +3, the mean response value increases by 3 for every one unit change in the predictor.

Do you think the company should focus more on their mobile app or on their website.

On their app

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
sns.set_style('darkgrid')
```

```python
from sklearn.datasets import load_boston
```

```python
boston = load_boston()
```

```python
print(boston.keys())
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

```python
print(boston.data.shape)
```

```
(506, 13)
```

```python
print(boston.feature_names)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

```python
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the targe
t.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity',
Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth Internat
ional Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
```

```python
data = pd.DataFrame(boston.data,columns=boston.feature_names)
```

```python
data.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

```python
data.tail()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

```python
data['Price']=boston.target
```

```python
data.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

Inferanetial stastics.

EDA

Outiler treatement apply on target.

Apply Machine Algorithm i.e. Linear regression here.

```python
data.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
CRIM       506 non-null float64
ZN         506 non-null float64
INDUS      506 non-null float64
CHAS       506 non-null float64
NOX        506 non-null float64
RM         506 non-null float64
AGE        506 non-null float64
DIS        506 non-null float64
RAD        506 non-null float64
TAX        506 non-null float64
PTRATIO    506 non-null float64
B          506 non-null float64
LSTAT      506 non-null float64
Price      506 non-null float64
dtypes: float64(14)
memory usage: 55.4 KB
```

```python
data.isnull().sum()
```

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
Price      0
dtype: int64
```

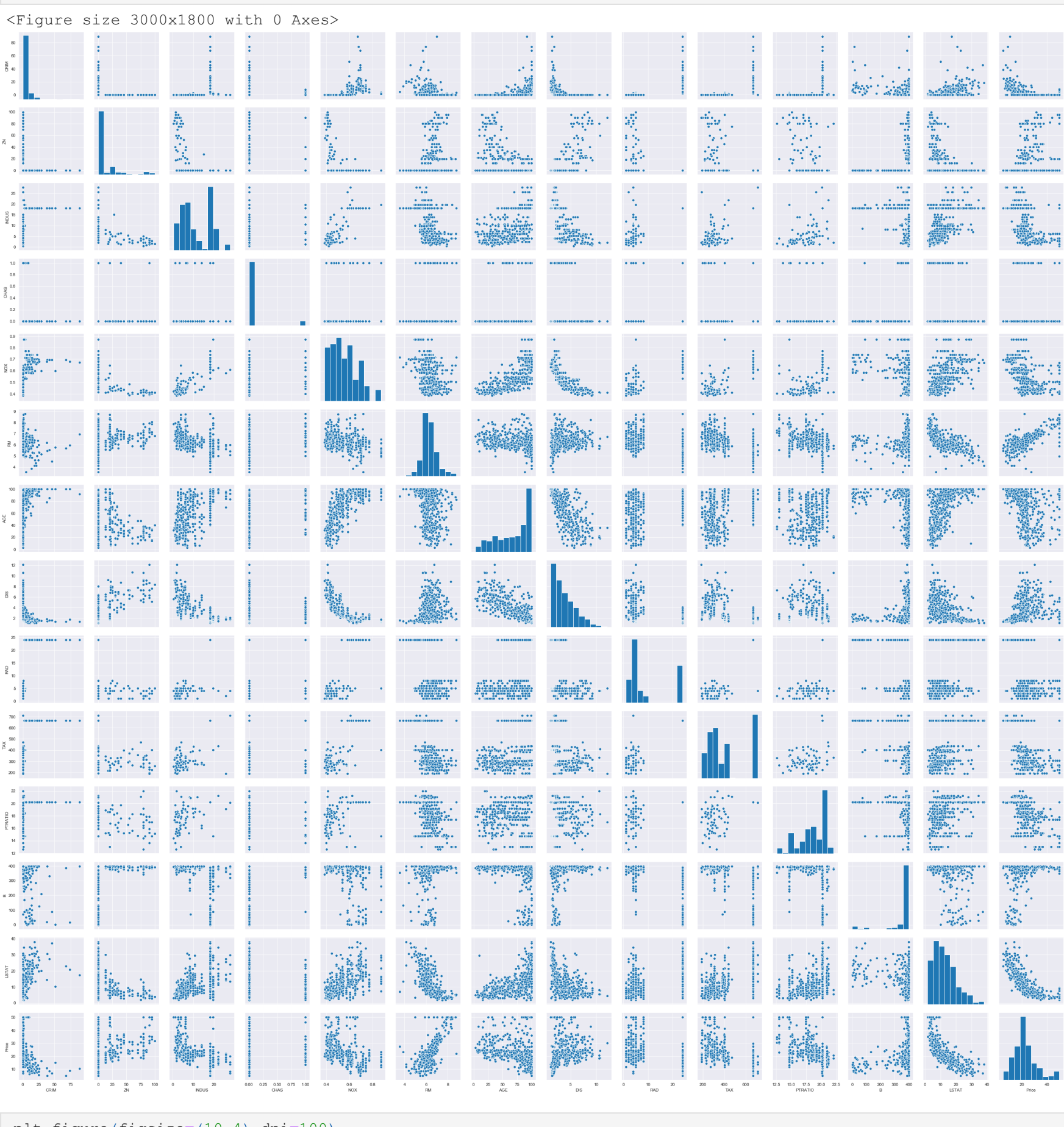
```python
data.duplicated().sum()
```

```
0
```

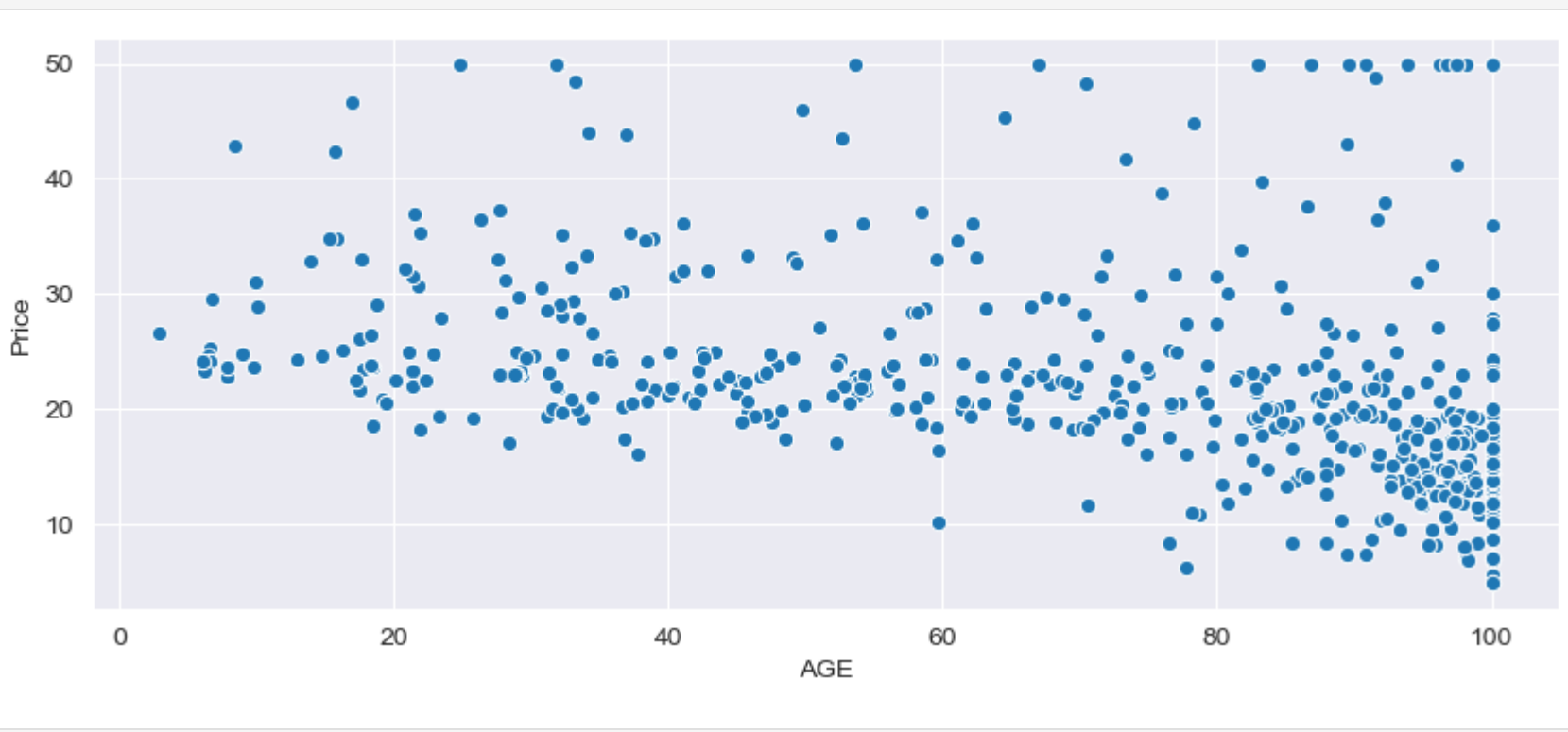
EDA

```python
plt.figure(figsize =(15,9),dpi=200)
sns.pairplot(data)
plt.show()
```

```
<Figure size 3000x1800 with 0 Axes>
```



```python
plt.figure(figsize=(10,4),dpi=100)
sns.scatterplot(x='AGE',y='Price',data=data)
plt.show()
```




```python
data.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

```python
data['Price'].describe()
```

```
count    506.000000
mean      22.532806
std        9.197104
min        5.000000
25%       17.025000
50%       21.200000
75%       25.000000
max       50.000000
Name: Price, dtype: float64
```

```python
from scipy.stats import zscore
```

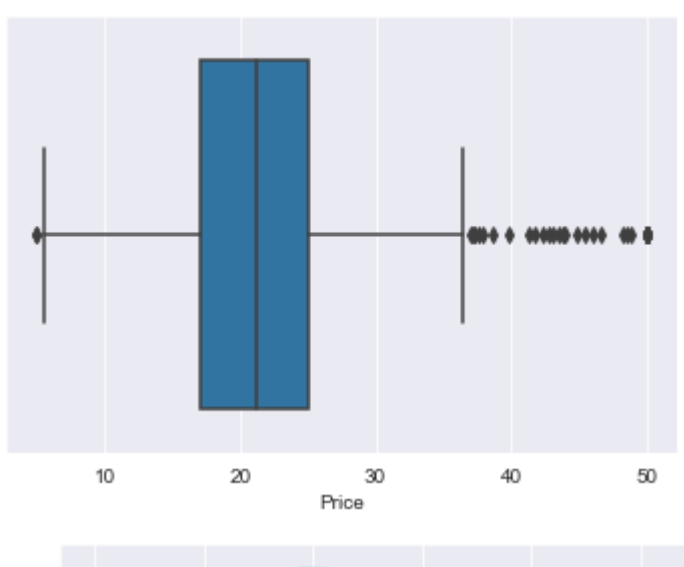
```python
z= np.abs(zscore(data['Price']))
```

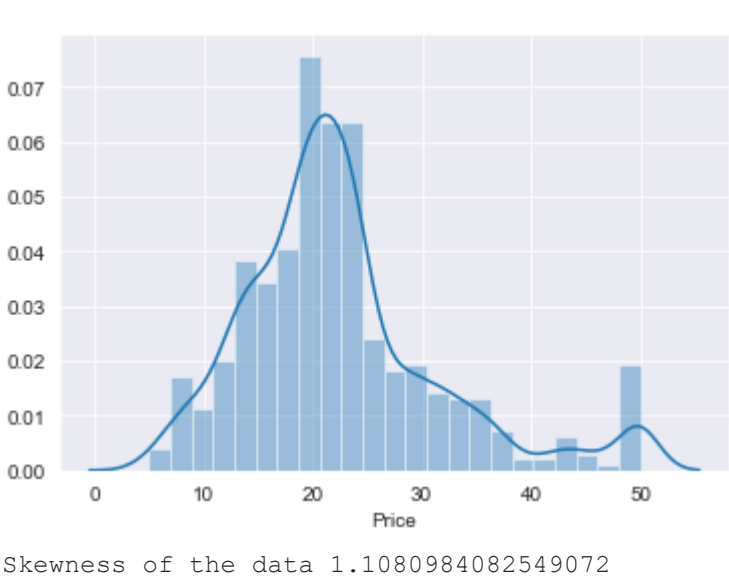
```python
data['z']=z
```

```python
data.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price	z
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0	0.159686
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6	0.101524
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7	1.324247
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4	1.182758
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2	1.487503

```python
sns.boxplot(data['Price'])
plt.show()
sns.distplot(data['Price'])
plt.show()
print('Skewness of the data',data['Price'].skew())
print('Kurtoisis of the data',data['Price'].kurt())
```






```
Skewness of the data 1.1080984082549072
Kurtoisis of the data 1.495196944165818
```

```python
dataz = data[data['z']<3]
dataz.shape
```

```
(506, 15)
```

```python
Q1 = data['Price'].quantile(0.25)
Q3 = data['Price'].quantile(0.75)

IQR = Q3-Q1
IQR
```

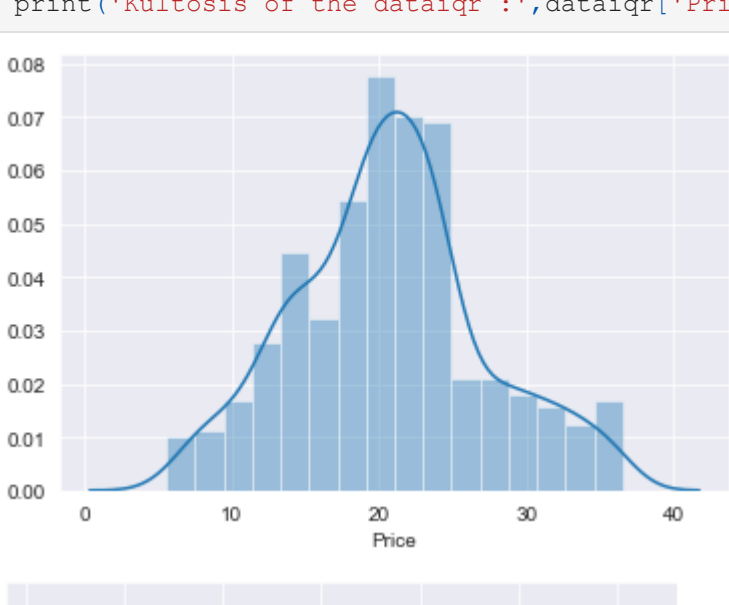
```
7.975000000000001
```

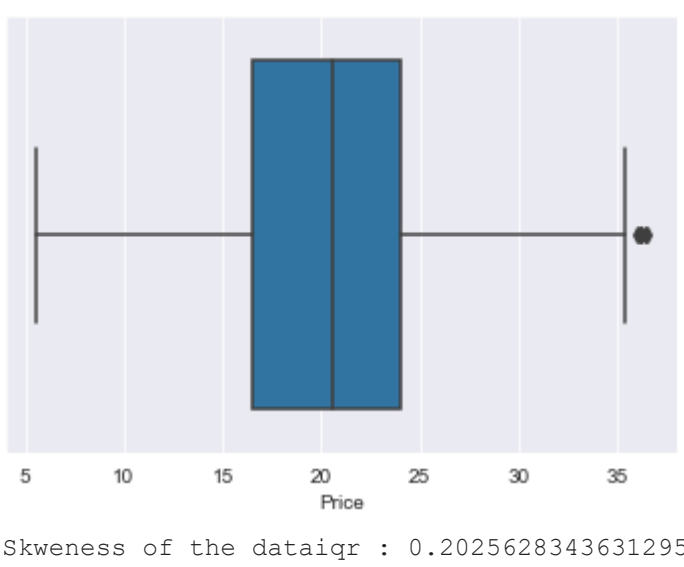
```python
dataiqr = data[~((data['Price']<Q1-1.5*IQR) |(data['Price']>Q3+1.5*IQR))]
```

```python
dataiqr.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price	z
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0	0.159686
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6	0.101524
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7	1.324247
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4	1.182758
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2	1.487503

```python
sns.distplot(dataiqr['Price'])
plt.show()

sns.boxplot(dataiqr['Price'])
plt.show()

print('Skweness of the dataiqr :',dataiqr['Price'].skew())
print('Kultosis of the dataiqr :',dataiqr['Price'].kurt())
```






```
Skweness of the dataiqr : 0.20256283436312952
Kultosis of the dataiqr : -0.11829814359992152
```

```python
dataiqr.shape
```

```
(466, 15)
```

```python
data.corr()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670	0.625505	0.582764	0.289946	-0.385064	0.455
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0.311948	-0.314563	-0.391679	0.175520	-0.412
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0.595129	0.720760	0.383248	-0.356977	0.603
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0.007368	-0.035587	-0.121515	0.048788	-0.053
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0.611441	0.668023	0.188933	-0.380051	0.590
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246	-0.209847	-0.292048	-0.355501	0.128069	-0.613
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881	0.456022	0.506456	0.261515	-0.273534	0.602
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000	-0.494588	-0.534432	-0.232471	0.291512	-0.496
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588	1.000000	0.910228	0.464741	-0.444413	0.488
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0.910228	1.000000	0.460853	-0.441808	0.543
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0.464741	0.460853	1.000000	-0.177383	0.374
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512	-0.444413	-0.441808	-0.177383	1.000000	-0.366
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0.488676	0.543993	0.374044	-0.366087	1.00
Price	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929	-0.381626	-0.468536	-0.507787	0.333461	-0.737
z	0.244482	0.098276	0.043936	0.093913	0.166539	0.416958	0.180597	-0.227608	0.193582	0.151138	-0.162696	-0.148942	0.005

```python
data['Price'].describe()
```

```
count    506.000000
mean      22.532806
std        9.197104
min        5.000000
25%       17.025000
50%       21.200000
75%       25.000000
max       50.000000
Name: Price, dtype: float64
```

```python
corr=np.abs(data.corr()['Price'])
```

```python
corr
```

```
CRIM       0.388305
ZN         0.360445
INDUS      0.483725
CHAS       0.175260
NOX        0.427321
RM         0.695360
AGE        0.376955
DIS        0.249929
RAD        0.381626
TAX        0.468536
PTRATIO    0.507787
B          0.333461
LSTAT      0.737663
Price      1.000000
z          0.442946
Name: Price, dtype: float64
```

```python
good_corr = corr[corr>0.4]
good_corr
```

```
INDUS      0.483725
NOX        0.427321
RM         0.695360
TAX        0.468536
PTRATIO    0.507787
LSTAT      0.737663
Price      1.000000
z          0.442946
Name: Price, dtype: float64
```

```python
good_corr.index
```

```
Index(['INDUS', 'NOX', 'RM', 'TAX', 'PTRATIO', 'LSTAT', 'Price', 'z'], dtype='object')
```

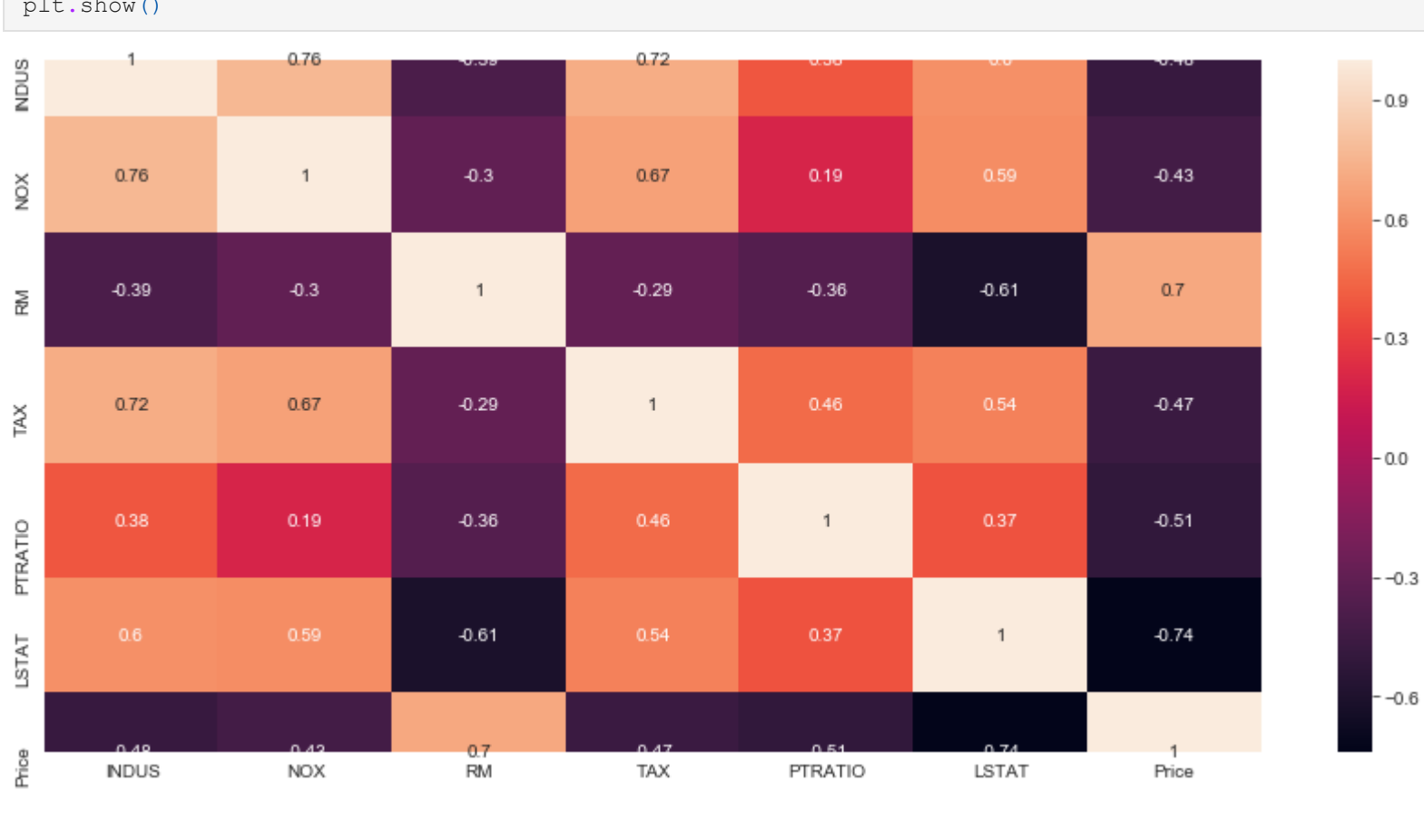
```python
data1 = data[good_corr.index]
```

```python
data1.drop(columns='z',inplace=True)
```

```python
data1.head()
```

	INDUS	NOX	RM	TAX	PTRATIO	LSTAT	Price
0	2.31	0.538	6.575	296.0	15.3	4.98	24.0
1	7.07	0.469	6.421	242.0	17.8	9.14	21.6
2	7.07	0.469	7.185	242.0	17.8	4.03	34.7
3	2.18	0.458	6.998	222.0	18.7	2.94	33.4
4	2.18	0.458	7.147	222.0	18.7	5.33	36.2

```python
plt.figure(figsize=(15,7))
sns.heatmap(data1.corr(),annot=True)
plt.show()
```



Model Selection & Identification of the overfit or Underfit.

```python
x = data.drop(columns='Price')
y = data['Price']
```

```python
x.shape,y.shape
```

```
((506, 14), (506,))
```

```python
from sklearn.model_selection import train_test_split
```

```python
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.8,random_state=0)
```

```python
x_train.shape,y_train.shape
```

```
((404, 14), (404,))
```

```python
x_test.shape,y_test.shape
```

```
((102, 14), (102,))
```

```python
from sklearn.linear_model import LinearRegression
```

```python
model = LinearRegression()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
```

```python
from sklearn.metrics import r2_score
```

```python
print('R2 score :',r2_score(y_test,y_pred))
```

```
R2 score : 0.7288492772784789
```

```python
model.fit(x_train,y_train)
y_train_pred = model.predict(x_train)
print('R2 Score :',r2_score(y_train,y_train_pred))
```

```
R2 Score : 0.8662636965197937
```

```python
model.fit(x,y)
y_pred_x = model.predict(x)
print('R2 Score :',r2_score(y,y_pred_x))
```

```
R2 Score : 0.8420524260377601
```

```python
print('R2 score for x_test and y_test   :',r2_score(y_test,y_pred))
print('R2 Score for x_train and y_train :',r2_score(y_train,y_train_pred))
print('R2 Score for x and y             :',r2_score(y,y_pred_x))
```

```
R2 score for x_test and y_test   : 0.7288492772784789
R2 Score for x_train and y_train : 0.8662636965197937
R2 Score for x and y             : 0.8420524260377601
```

Data is overfit because the train> test.

```python
data.drop(columns='z',inplace=True)
```

```python
data.head(2)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.9	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.9	9.14	21.6

OLS:- Ordinary Least Square Method.

```python
import statsmodels.api as sm
```

```python
x=data['LSTAT']
Y=data['Price']
```

```python
x_constant=sm.add_constant(x)
```

```python
x_constant
```

	const	LSTAT
0	1.0	4.98
1	1.0	9.14
2	1.0	4.03
3	1.0	2.94
4	1.0	5.33
...	...	...
501	1.0	9.67
502	1.0	9.08
503	1.0	5.64
504	1.0	6.48
505	1.0	7.88

506 rows × 2 columns

```python
linear_reg=sm.OLS(y,x_constant).fit()
```

```python
linear_reg
```

```
<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1111deb0>
```

```python
linear_reg.summary()
```

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.544
Model:	OLS	**Adj. R-squared:**	0.543
Method:	Least Squares	**F-statistic:**	601.6
Date:	Mon, 03 Feb 2020	**Prob (F-statistic):**	5.08e-88
Time:	11:35:02	**Log-Likelihood:**	-1641.5
No. Observations:	506	**AIC:**	3287.
Df Residuals:	504	**BIC:**	3295.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	34.5538	0.563	61.415	0.000	33.448	35.659
LSTAT	-0.9500	0.039	-24.528	0.000	-1.026	-0.874

Omnibus:	137.043	**Durbin-Watson:**	0.892
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	291.373
Skew:	1.453	**Prob(JB):**	5.36e-64
Kurtosis:	5.319	**Cond. No.**	29.7

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```python
x=data.iloc[:,:13]
y=data['Price']
```

```python
x_constant=sm.add_constant(x)
```

While selecting the attribute for OLS fit choose first depended variable and second is the independed variable those are the concered with the x_contant.

```python
linear_reg=sm.OLS(y,x_constant).fit()
```

```python
linear_reg.summary()
```

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.741
Model:	OLS	**Adj. R-squared:**	0.734
Method:	Least Squares	**F-statistic:**	108.1
Date:	Mon, 03 Feb 2020	**Prob (F-statistic):**	6.72e-135
Time:	11:35:02	**Log-Likelihood:**	-1498.8
No. Observations:	506	**AIC:**	3026.
Df Residuals:	492	**BIC:**	3085.
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	36.4595	5.103	7.144	0.000	26.432	46.487
CRIM	-0.1080	0.033	-3.287	0.001	-0.173	-0.043
ZN	0.0464	0.014	3.382	0.001	0.019	0.073
INDUS	0.0206	0.061	0.334	0.738	-0.100	0.141
CHAS	2.6867	0.862	3.118	0.002	0.994	4.380
NOX	-17.7666	3.820	-4.651	0.000	-25.272	-10.262
RM	3.8099	0.418	9.116	0.000	2.989	4.631
AGE	0.0007	0.013	0.052	0.958	-0.025	0.027
DIS	-1.4756	0.199	-7.398	0.000	-1.867	-1.084
RAD	0.3060	0.066	4.613	0.000	0.176	0.436
TAX	-0.0123	0.004	-3.280	0.001	-0.020	-0.005
PTRATIO	-0.9527	0.131	-7.283	0.000	-1.210	-0.696
B	0.0093	0.003	3.467	0.001	0.004	0.015
LSTAT	-0.5248	0.051	-10.347	0.000	-0.624	-0.425

Omnibus:	178.041	**Durbin-Watson:**	1.078
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	783.126
Skew:	1.521	**Prob(JB):**	8.84e-171
Kurtosis:	8.281	**Cond. No.**	1.51e+04

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.51e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

RESIDUALS

```python
linear_reg.resid.head() ####actual-predict
```

```
0   -6.003843
1   -3.425562
2    4.132403
3    4.792964
4    8.256476
dtype: float64
```

```python
linear_reg.predict(x_constant).head() #### it is an y_predict value
```

```
0    30.003843
1    25.025562
2    30.567597
3    28.607036
4    27.943524
dtype: float64
```

```python
y.head()
```

```
0    24.0
1    21.6
2    34.7
3    33.4
4    36.2
Name: Price, dtype: float64
```

1.Autocorrelation.

Durbin-Watson test

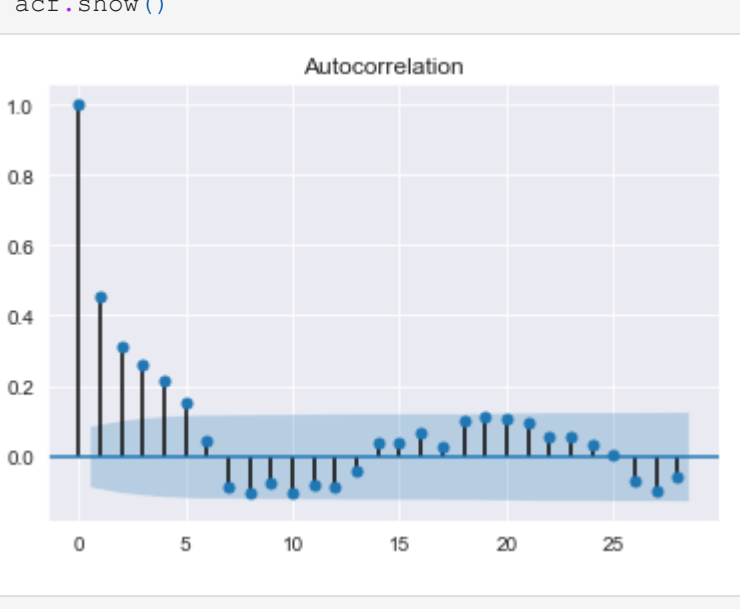
It is used to check autocorrelation.

when the Durine-Watson value inbetween 0 to 2 that means it is positive correlation and if we have 2 then it is no autcorrelation and if we have more than 2 this means it belongs to the negetive autocorrelation.

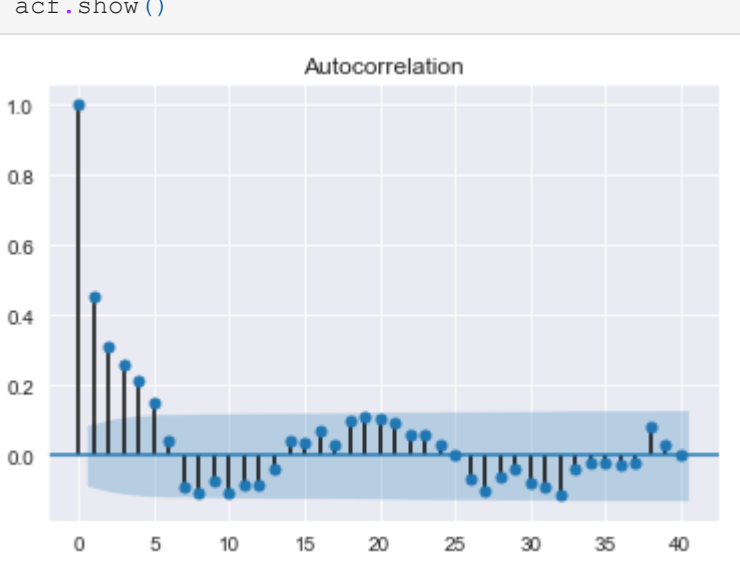
It is always present in the range(0 to 4). please check the summary.

```python
import statsmodels.tsa.api as smt  #tsa = Time Series Analysis.
```

```python
acf=smt.graphics.plot_acf(linear_reg.resid)
acf.show()
```

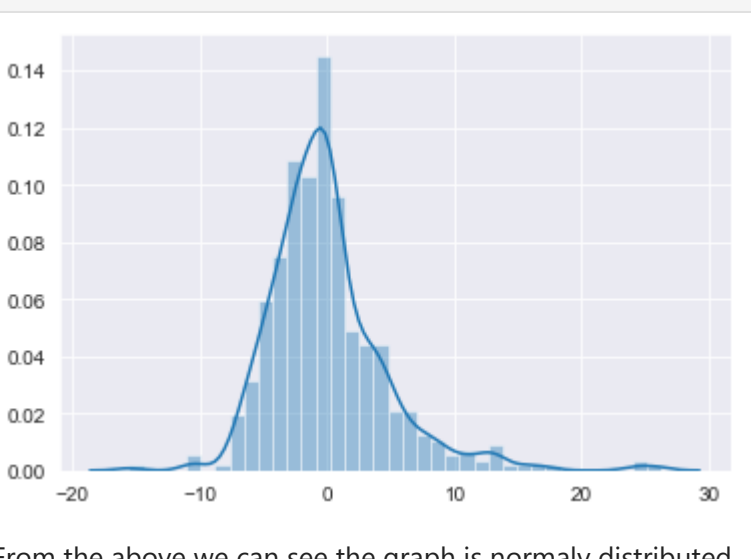



```python
acf=smt.graphics.plot_acf(linear_reg.resid,lags=40)
acf.show()
```



2nd Assumptions:Multivariate Normalality.

In [248...
```
sns.distplot(linear_reg.resid)
plt.show()
```



From the above we can see the graph is normaly distributed.

test2 :-Jarque Bera Test.

This test is for goodness of fit test of wheather the sample data have skweness and kulrtoisis matching to the normly distributed or not.

Null Hypothesis:- Error should be normally distrubted

Atternative hypothesis :- Error terms are not normally distributed.

When the significance is greater than 5.99 that time we can say error terms are normally distributed.

When the significance is lesser than 5.99 that time we can say the error terms are normally distributed.

Left one is the significant value and it is geater than 5.99(significant value) that means the it is normally distributed. significant value for jaque Bera is 5.99.

To get the first value of the significant:-

```
stats.jarque_bera(linear_reg.resid)[0]
```

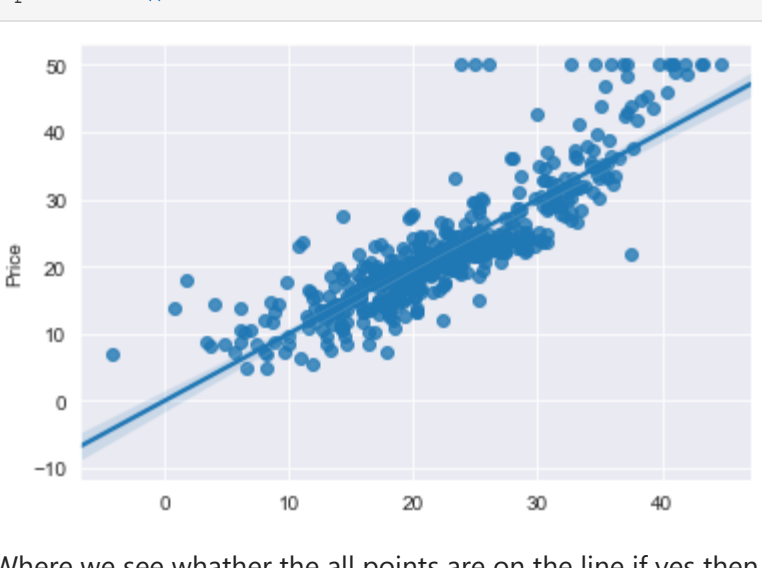
In [249...
```
from scipy import stats
stats.jarque_bera(linear_reg.resid)
```

Out[249... (783.1262780507636, 0.0)

Here we can see the not more than 5.99 that means it is not normally distributed.

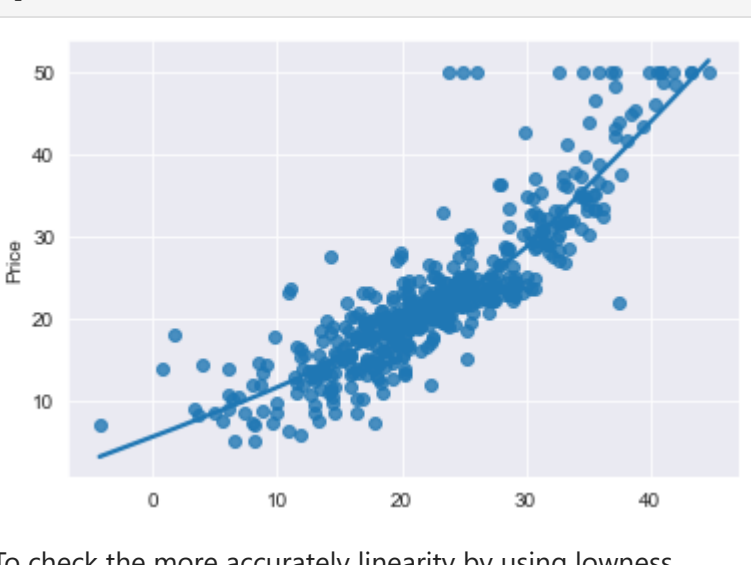
3.Linearity.

In [250...
```
sns.regplot(x=linear_reg.predict(x_constant),y=y)
plt.show()
```



Where we see whather the all points are on the line if yes then data is linear.

In [251...
```
sns.regplot(x=linear_reg.predict(x_constant),y=y,lowess=True)
plt.show()
```

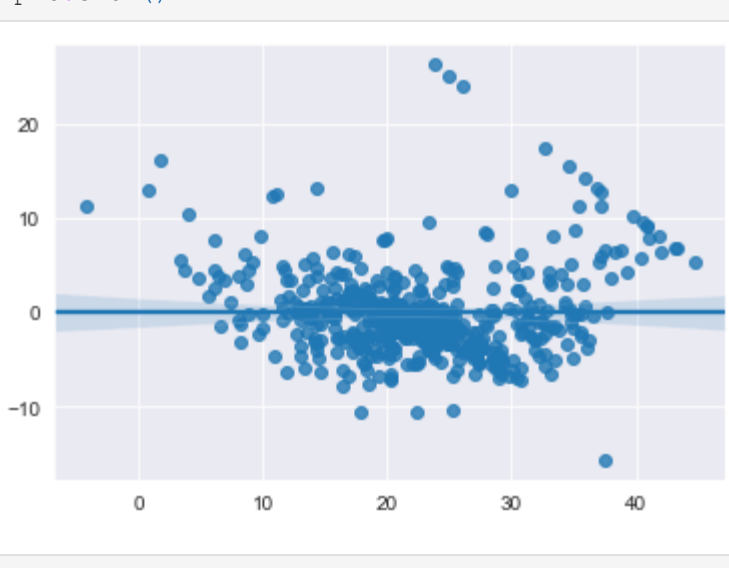


To check the more accurately linearity by using lowness.

To cross check the whather our data is linear or not.

method 2:-

In [252...
```
sns.regplot(x=linear_reg.predict(x_constant),y=linear_reg.resid)
plt.show()
```



In [253...
```
sns.regplot(x=linear_reg.predict(x_constant),y=linear_reg.resid,lowess=True)
plt.show()
```

method 3:- RainBow Test

Null Hypothesis is the belongs to linear hypothesis. Alternate hypothesis is belongs to non-linear Hypothesis.

In [254...
```
sm.stats.diagnostic.linear_rainbow(res=linear_reg)
```

Out[254... (0.580850763961426, 0.999988852262534)

in the return we having the first value is f-test and second is the p-test. here we can see that the pvalue is greter than the 0.05 that means the model is linear.

here we can see that model is linear model.

Method 4:-

if residual mean is close to the zero that means the the model is linear.

In [255...
```
np.mean(linear_reg.resid)
```

Out[255... 1.227301089040173e-13

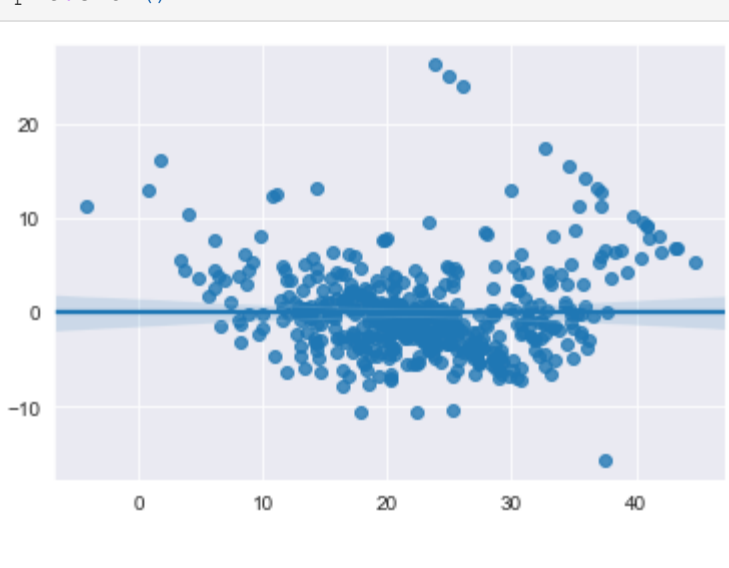
Since the residual mean is close to zero then it is linear.

Homocidasticity

If the variance of the residual are symmerically distributed accorss the residual line then data is said to homesidasctic.

If the variance is uneqaul residual acorss the residual line then data is said to hetrosidacity.

In [256...
```
sns.regplot(x=linear_reg.predict(x_constant),y=linear_reg.resid)
plt.show()
```



method 2:- GoldFeld Test or Brush Wagon test.

Homocidasticity:-When the variance is constant across the line belongs to null Hypothesis.

Hetrocidasticity:-When the variance is not constant accross line belongs to Alternate Hypothesis.

It return the t-value,p_value,order

Here we can see that pvalue is lesser than the 0.005 that means the it rejeted the null hypothesis and accepted the alternate hypothesis.It relate to the hetrosidasticity.

In [257...
```
import statsmodels.stats.api as sms
```

In [258...
```
sms.het_goldfeldquandt(linear_reg.resid,linear_reg.model.exog)
```

Out[258... (2.4311473253497438, 7.302071915608868e-12, 'increasing')

In [259...
```
linear_reg.model.exog.shape
```

Out[259... (506, 14)

It return the t-value,p_value,order

Here we can see that pvalue is lesser than the 0.005 that means the it rejeted the null hypothesis and accepted the alternate hypothesis.It relate to the hetrosidasticity.

Multi-Colinearity

It found in Multivariant data.

It present can adversly in our analysis.There should be No and low Multi-colinearity.

Correlation Causation:-Formulative relation between the two entity.

Causation:-Formulated relationship between the data like temp in 0c and F(Farenite).

VIF - varianace Influential Factor. (to check the multi-colinearity)

VIF is the quotient of variance in model of multiple term by the variance with one term.It tells about the multi-Colinearity that is ratio.

In [264...
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [265...
```
vif = [variance_inflation_factor(x_constant.values,i) for i in range(x_constant.shape[1])]
```

In [266...
```
vif
```

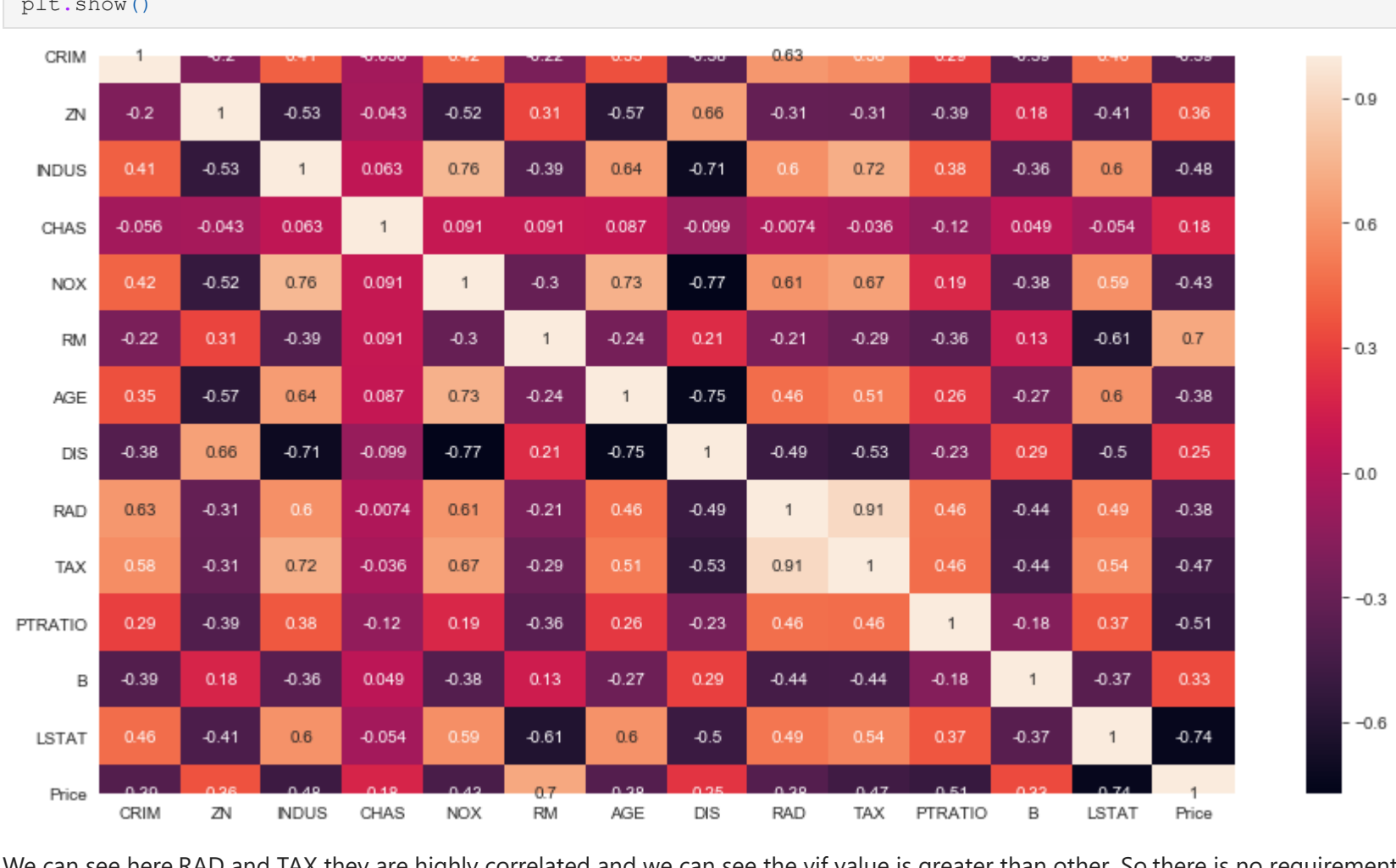
Out[266...
```
[585.2652379423121,
 1.7921915474332406,
 2.2987581787494418,
 3.9915964183460297,
 1.0739953275537886,
 4.393719847577493,
 1.9337444357832574,
 3.1008255128153372,
 3.9559449063727263,
 7.484496335274472,
 9.00855394759707,
 1.7990840492488978,
 1.3485210764063758,
 2.9414910780919366]
```

In [275...
```
dataframe=pd.DataFrame(vif,columns=['vif'],index=x_constant.columns)
dataframe
```

Out[275...

	vif
const	585.265238
CRIM	1.792192
ZN	2.298758
INDUS	3.991596
CHAS	1.073995
NOX	4.393720
RM	1.933744
AGE	3.100826
DIS	3.955945
RAD	7.484496
TAX	9.008554
PTRATIO	1.799084
B	1.348521
LSTAT	2.941491

In [280...
```
plt.figure(figsize=(15,8))
sns.heatmap(data.corr(),annot=True)
plt.show()
```



We can see here,RAD and TAX they are highly correlated and we can see the vif value is greater than other. So,there is no requirement for prediction because they are mutually multi-correlation.

In [281...
```
data.drop(columns=['RAD','TAX'],inplace=True)
```

In [282...
```
data.head()
```

Out[282...

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	PTRATIO	B	LSTAT	Price
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	18.7	396.90	5.33	36.2

.....Continue till will get the best model.