In [10]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
```

EDA

EDA stands for exploratotary data analysis

in stastics exploratory data analysis is approach of analysing datasets to summarise their main characterstics,often using stastitical graphics and other data visualisation methods

EDA can be done in 3 steps

1. Data Sorcing

2. Data cleaning

3. Data visualisation

DATA SOURCING

It is three types

1.Private Data

EX : Health care data,telephone data

2.Public Data

EX : Data generated by Government and third parties

3. Web Scraping

EX: Creating and Gathering our Own data

reading data from csv file

In [11]:
```
data1=pd.read_csv("AAPL.csv")
```

In [12]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10285 entries, 0 to 10284
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       10285 non-null  object 
 1   Open       10285 non-null  float64
 2   High       10285 non-null  float64
 3   Low        10285 non-null  float64
 4   Close      10285 non-null  float64
 5   Adj Close  10285 non-null  float64
 6   Volume     10285 non-null  int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 562.6+ KB
```

In [13]:
```
data1.head(20)
```

Out[13]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1980-12-12	0.128348	0.128906	0.128348	0.128348	0.100600	469033600
1	1980-12-15	0.122210	0.122210	0.121652	0.121652	0.095352	175884800
2	1980-12-16	0.113281	0.113281	0.112723	0.112723	0.088353	105728000
3	1980-12-17	0.115513	0.116071	0.115513	0.115513	0.090540	86441600
4	1980-12-18	0.118862	0.119420	0.118862	0.118862	0.093165	73449600
5	1980-12-19	0.126116	0.126674	0.126116	0.126116	0.098851	48630400
6	1980-12-22	0.132254	0.132813	0.132254	0.132254	0.103662	37363200
7	1980-12-23	0.137835	0.138393	0.137835	0.137835	0.108036	46950400
8	1980-12-24	0.145089	0.145647	0.145089	0.145089	0.113722	48003200
9	1980-12-26	0.158482	0.159040	0.158482	0.158482	0.124219	55574400
10	1980-12-29	0.160714	0.161272	0.160714	0.160714	0.125969	93161600
11	1980-12-30	0.157366	0.157366	0.156808	0.156808	0.122907	68880000
12	1980-12-31	0.152902	0.152902	0.152344	0.152344	0.119408	35750400
13	1981-01-02	0.154018	0.155134	0.154018	0.154018	0.120720	21660800
14	1981-01-05	0.151228	0.151228	0.150670	0.150670	0.118096	35728000
15	1981-01-06	0.144531	0.144531	0.143973	0.143973	0.112847	45158400
16	1981-01-07	0.138393	0.138393	0.137835	0.137835	0.108036	55686400
17	1981-01-08	0.135603	0.135603	0.135045	0.135045	0.105849	39827200
18	1981-01-09	0.142299	0.142857	0.142299	0.142299	0.111535	21504000
19	1981-01-12	0.142299	0.142299	0.141183	0.141183	0.110660	23699200

In [14]:
```
data1.describe()
```

Out[14]:

	Open	High	Low	Close	Adj Close	Volume
count	10285.000000	10285.000000	10285.000000	10285.000000	10285.000000	1.028500e+04
mean	12.179027	12.307921	12.047393	12.182293	11.592432	3.350720e+08
std	25.558444	25.834993	25.272890	25.564842	25.267256	3.403284e+08
min	0.049665	0.049665	0.049107	0.049107	0.038490	0.000000e+00
25%	0.276964	0.284821	0.270647	0.277344	0.232861	1.270752e+08
50%	0.457589	0.464286	0.448661	0.457589	0.379018	2.231640e+08
75%	12.859643	12.867500	12.711429	12.796429	10.987222	4.171552e+08
max	156.979996	157.259995	154.389999	156.690002	156.690002	7.421641e+09

In [16]:
```
data1.nunique()
```

Out[16]:
```
Date         10285
Open          5643
High          5582
Low           5562
Close         5752
Adj Close     7279
Volume        9723
dtype: int64
```

DATA CLEANING

Checking Data Types

Fixing Rows and Columns

Handling missing Values

Standardizing values

checking null values

In [17]:
```
data1.isnull().sum()
```

Out[17]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

Checking Datatypes of Columns

In [18]:
```
data1.dtypes
```

Out[18]:
```
Date          object
Open         float64
High         float64
Low          float64
Close        float64
Adj Close    float64
Volume         int64
dtype: object
```

replacing nan values with 0

In [19]:
```
#handling missing values
data1=data1.fillna(method='bfill',axis=0).fillna(0)
```

In [20]:
```
data1
```

Out[20]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1980-12-12	0.128348	0.128906	0.128348	0.128348	0.100600	469033600
1	1980-12-15	0.122210	0.122210	0.121652	0.121652	0.095352	175884800
2	1980-12-16	0.113281	0.113281	0.112723	0.112723	0.088353	105728000
3	1980-12-17	0.115513	0.116071	0.115513	0.115513	0.090540	86441600
4	1980-12-18	0.118862	0.119420	0.118862	0.118862	0.093165	73449600
...	...	...	...	...	...	...	...
10280	2021-09-21	143.929993	144.600006	142.779999	143.429993	143.429993	75834000
10281	2021-09-22	144.449997	146.429993	143.699997	145.850006	145.850006	76404300
10282	2021-09-23	146.649994	147.080002	145.639999	146.830002	146.830002	64838200
10283	2021-09-24	145.660004	147.470001	145.559998	146.919998	146.919998	53434200
10284	2021-09-27	145.470001	145.960007	143.820007	145.369995	145.369995	74031500

10285 rows × 7 columns

In [21]:
```
date=data1['Date'].head(50)
open=data1['Open'].head(50)
high=data1['High'].head(50)
low=data1['Low'].head(50)
```

In [22]:
```
date
```

Out[22]:
```
0     1980-12-12
1     1980-12-15
2     1980-12-16
3     1980-12-17
4     1980-12-18
5     1980-12-19
6     1980-12-22
7     1980-12-23
8     1980-12-24
9     1980-12-26
10    1980-12-29
11    1980-12-30
12    1980-12-31
13    1981-01-02
14    1981-01-05
15    1981-01-06
16    1981-01-07
17    1981-01-08
18    1981-01-09
19    1981-01-12
20    1981-01-13
21    1981-01-14
22    1981-01-15
23    1981-01-16
24    1981-01-19
25    1981-01-20
26    1981-01-21
27    1981-01-22
28    1981-01-23
29    1981-01-26
30    1981-01-27
31    1981-01-28
32    1981-01-29
33    1981-01-30
34    1981-02-02
35    1981-02-03
36    1981-02-04
37    1981-02-05
38    1981-02-06
39    1981-02-09
40    1981-02-10
41    1981-02-11
42    1981-02-12
43    1981-02-13
44    1981-02-17
45    1981-02-18
46    1981-02-19
47    1981-02-20
48    1981-02-23
49    1981-02-24
Name: Date, dtype: object
```

In [23]:
```
open
```

Out[23]:
```
0     0.128348
1     0.122210
2     0.113281
3     0.115513
4     0.118862
5     0.126116
6     0.132254
7     0.137835
8     0.145089
9     0.158482
10    0.160714
11    0.157366
12    0.152902
13    0.154018
14    0.151228
15    0.144531
16    0.138393
17    0.135603
18    0.142299
19    0.142299
20    0.136719
21    0.136719
22    0.139509
23    0.138951
24    0.146763
25    0.142857
26    0.145089
27    0.146763
28    0.146763
29    0.144531
30    0.143973
31    0.138951
32    0.133929
33    0.127232
34    0.119420
35    0.123326
36    0.127790
37    0.127790
38    0.128348
39    0.122768
40    0.121652
41    0.118304
42    0.117188
43    0.114955
44    0.116629
45    0.121652
46    0.114955
47    0.108817
48    0.109933
49    0.107143
Name: Open, dtype: float64
```

In [24]:
```
high
```

Out[24]:
```
0     0.128906
1     0.122210
2     0.113281
3     0.116071
4     0.119420
5     0.126674
6     0.132813
7     0.138393
8     0.145647
9     0.159040
10    0.161272
11    0.157366
12    0.152902
13    0.155134
14    0.151228
15    0.144531
16    0.138393
17    0.135603
18    0.142857
19    0.142299
20    0.136719
21    0.137277
22    0.140625
23    0.138951
24    0.147321
25    0.142857
26    0.146205
27    0.147879
28    0.147321
29    0.144531
30    0.143973
31    0.138951
32    0.133929
33    0.127232
34    0.119420
35    0.123884
36    0.128348
37    0.128906
38    0.128906
39    0.122768
40    0.122210
41    0.118304
42    0.117188
43    0.114955
44    0.117188
45    0.122768
46    0.114955
47    0.108817
48    0.110491
49    0.107143
Name: High, dtype: float64
```

In [25]:
```
low
```

Out[25]:
```
0     0.128348
1     0.121652
2     0.112723
3     0.115513
4     0.118862
5     0.126116
6     0.132254
7     0.137835
8     0.145089
9     0.158482
10    0.160714
11    0.156808
12    0.152344
13    0.154018
14    0.150670
15    0.143973
16    0.137835
17    0.135045
18    0.142299
19    0.141183
20    0.136161
21    0.136719
22    0.139509
23    0.138393
24    0.146763
25    0.142299
26    0.145089
27    0.146763
28    0.146205
29    0.143973
30    0.142857
31    0.138393
32    0.133371
33    0.126116
34    0.118862
35    0.123326
36    0.127790
37    0.127790
38    0.128348
39    0.121652
40    0.121652
41    0.117746
42    0.116629
43    0.113839
44    0.116629
45    0.121652
46    0.114397
47    0.108259
48    0.109933
49    0.106027
Name: Low, dtype: float64
```

In [26]:
```
data1.columns
```

Out[26]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='o
bject')
```

DATA VISUALISATION

Univariate

Bivariate

Multivariate

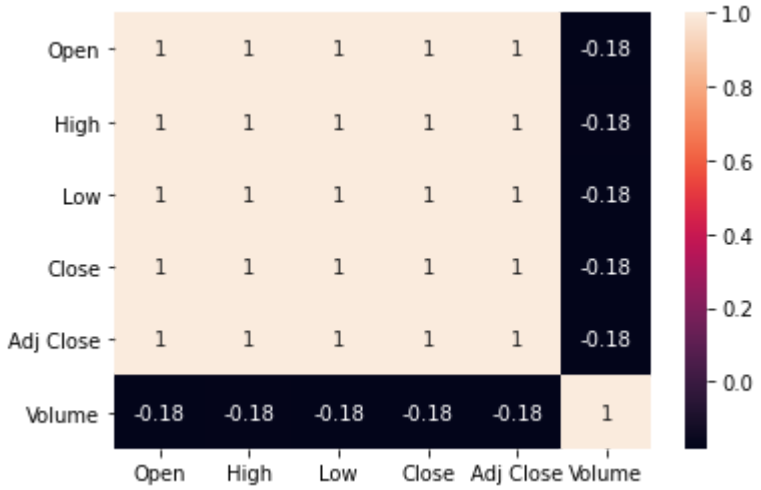
EX: bf.null

sns.heatmap

In [32]:
```
corelation = data1.corr()
```

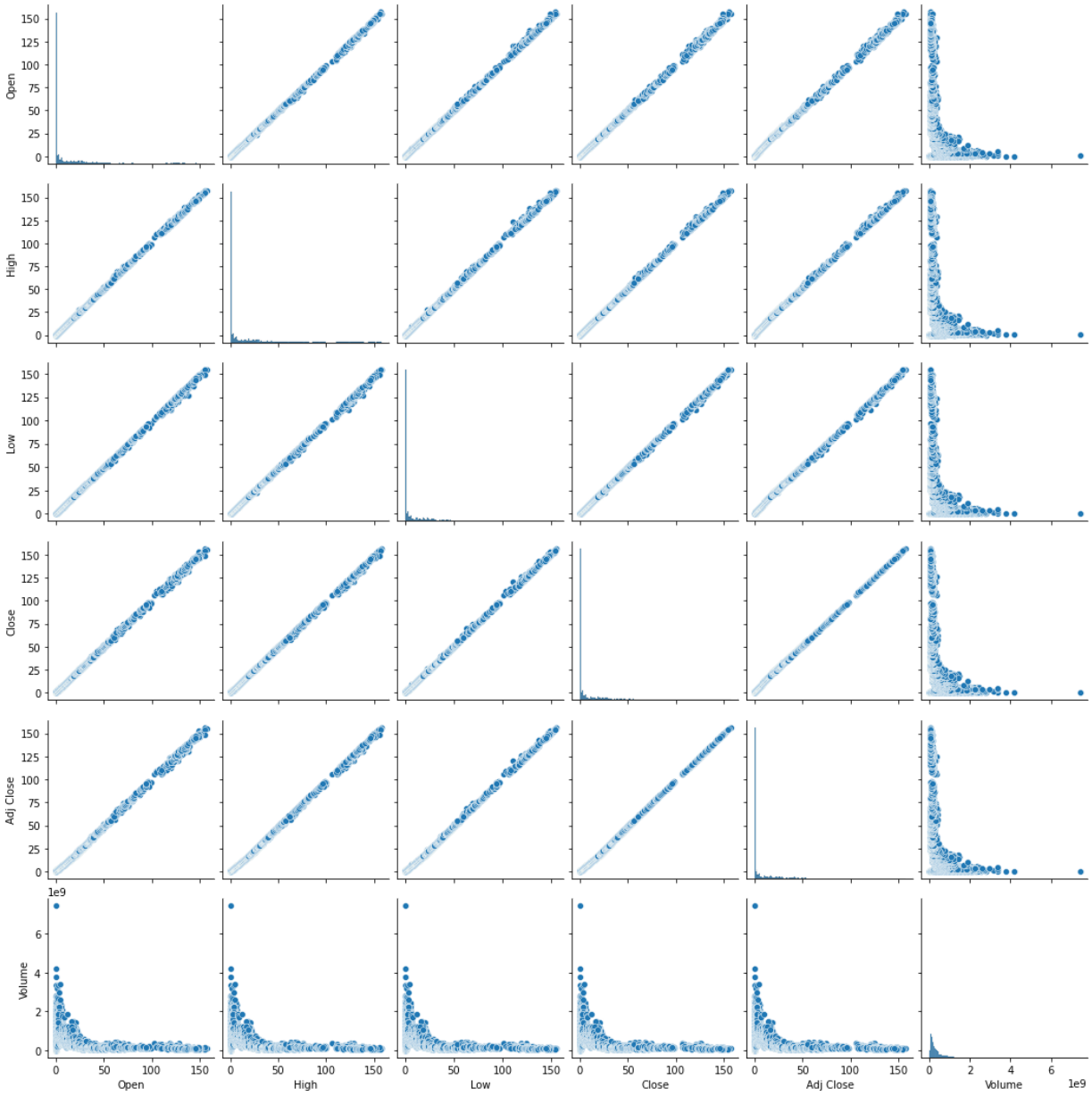
In [33]:
```
sns.heatmap(corelation, xticklabels=corelation.columns,yticklabels=corelation.col
```

Out[33]:
```
<AxesSubplot:>
```

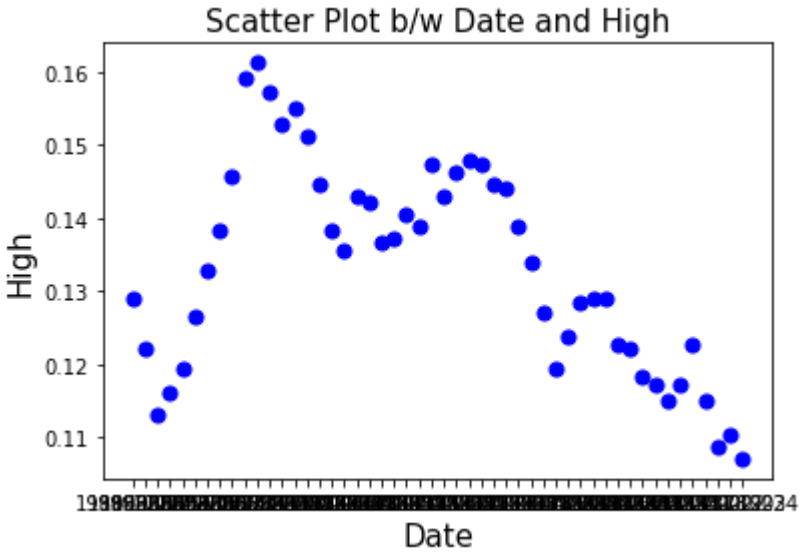


In [34]:
```
sns.pairplot(data1)
```

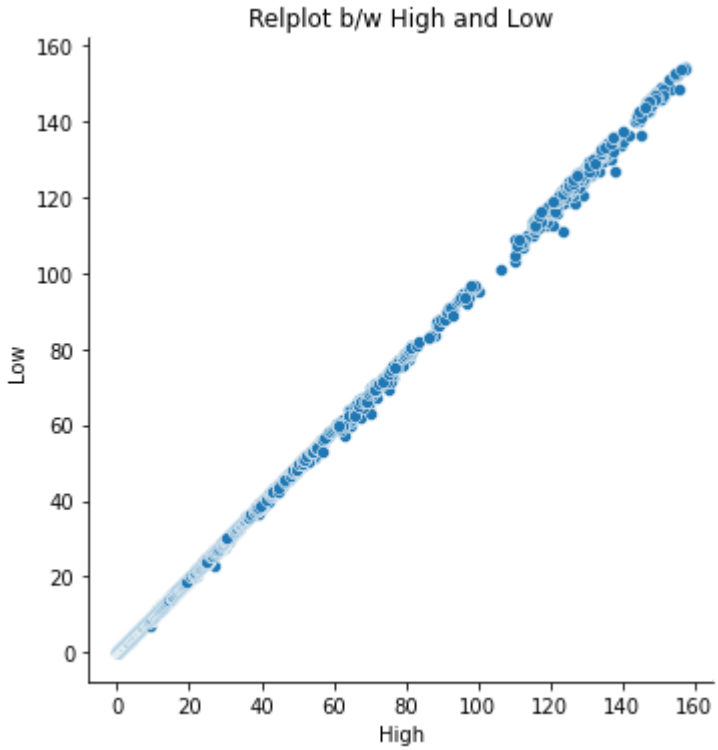
Out[34]:
```
<seaborn.axisgrid.PairGrid at 0x1cc71dbe190>
```



In [35]:
```
#Data Visualization using Matplotlib
plt.title("Scatter Plot b/w Date and High",fontsize=15)
plt.xlabel("Date",fontsize=15)
plt.ylabel("High",fontsize=15)
plt.scatter(date,high,c='blue',s=50)
plt.show()
```

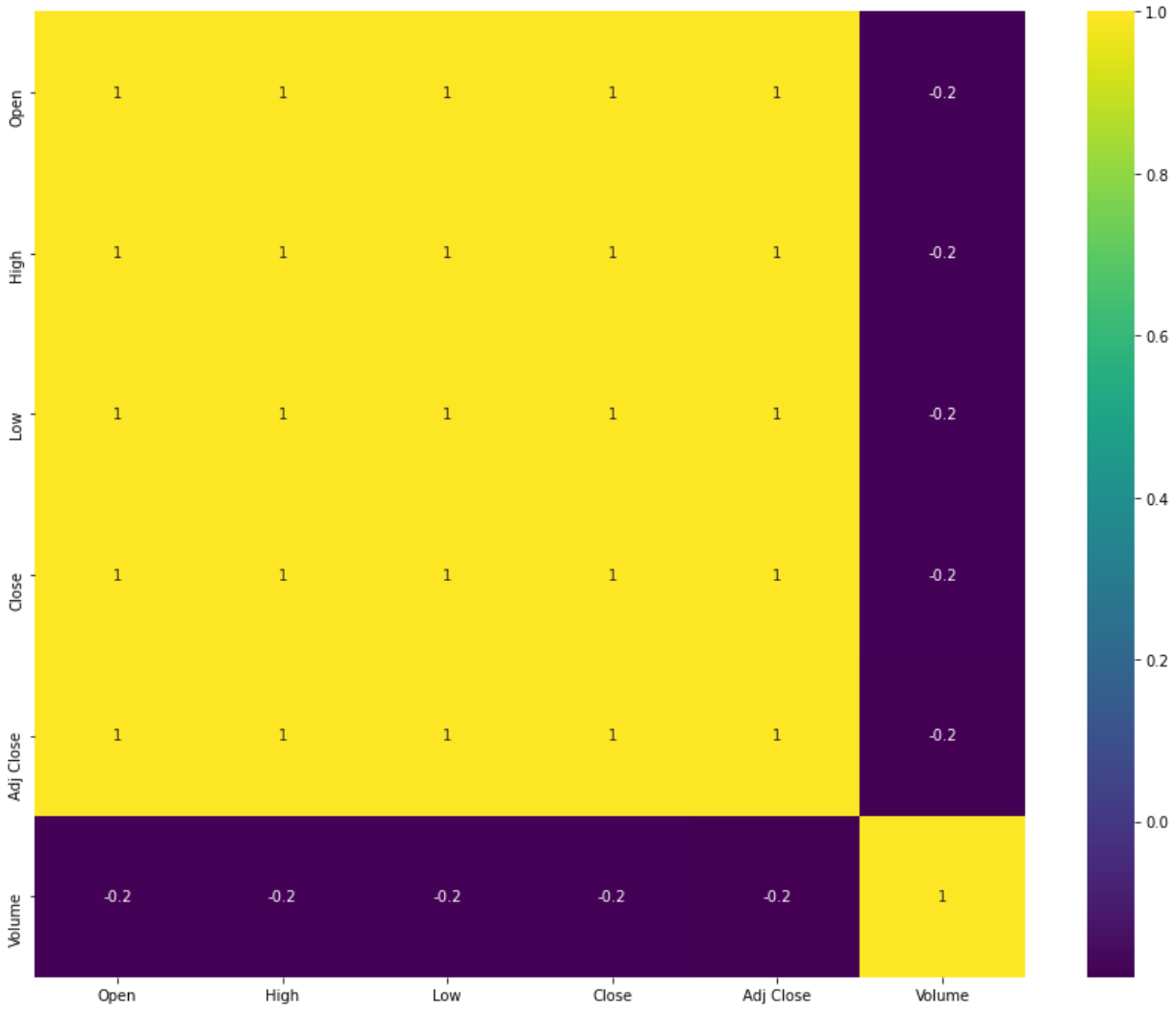


In [36]:
```
#Data Visualisation Using seaborn
sns.relplot(data=data1,x="High",y="Low")
plt.title("Relplot b/w High and Low")
plt.show()
```



In [40]:
```
fig, ax=plt.subplots(figsize=(15,12))
sns.heatmap(data.corr(),annot=True, fmt='.1g',cmap='viridis')
```

Out[40]: <AxesSubplot:>



MACHINE LEARNING MODELS

In [20]:
```
X = data1.drop("Volume", axis=1)
y = data1["Volume"]
```

In [21]:
```
X.head()
```

Out[21]:

	Open	High	Low	Close	Adj Close
0	0.128348	0.128906	0.128348	0.128348	0.100600
1	0.122210	0.122210	0.121652	0.121652	0.095352
2	0.113281	0.113281	0.112723	0.112723	0.088353
3	0.115513	0.116071	0.115513	0.115513	0.090540
4	0.118862	0.119420	0.118862	0.118862	0.093165

In [22]:
```
y.head()
```

Out[22]:
```
0    469033600
1    175884800
2    105728000
3     86441600
4     73449600
Name: Volume, dtype: int64
```

Splitting dataset into test and train

It is important part of evaluating datamining methods

In [23]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_sta
```

In [24]:
```
len(X_train),len(X_test)
```

Out[24]:
```
(7190, 3082)
```

In [25]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train, y_train)
X_test = scaler.transform(X_test)
```

LINEAR REGRESSION MODEL

It is used in statistical software to understnd the relation between the dependent variable and one or more independent variables by estimating probabilitiesof using a logistic regression equation

Logistic Regression

Logisticregressin is a supervised learning classification algorithm used to predict the probability of a target variable

It is one of the simplest ML algothms that can be used for various classification problems such as spam detection,diabetes prediction ,cancer detection etc..

In [ ]:
```
from sklearn.linear_model import LogisticRegression
mod = LogisticRegression()
mod.fit(X_train, y_train)
```

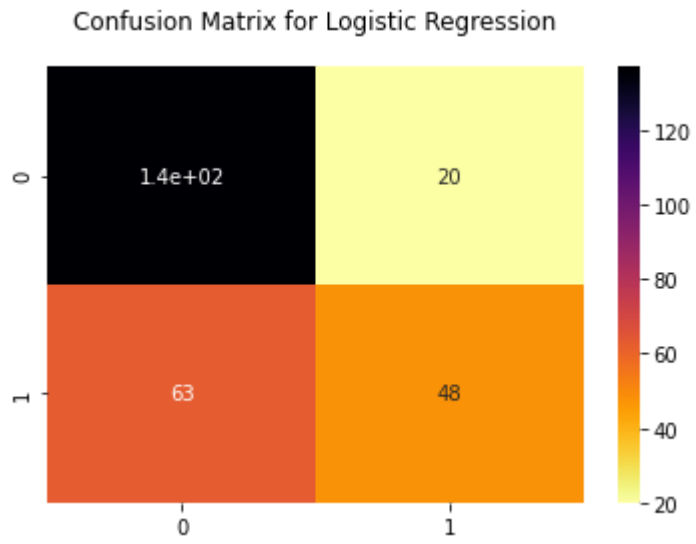
In [ ]:
```
LogisticRegressionScore = mod.score(X_test, y_test)
```

In [29]:
```
from sklearn.metrics import confusion_matrix, classification_report
y_pred_mod = mod.predict(X_test)
cf_matrix = confusion_matrix(y_test,y_pred_mod)
cf_matrix
```

Out[29]:
```
array([[137,  20],
       [ 63,  48]], dtype=int64)
```

In [30]:
```
sns.heatmap(cf_matrix, annot=True, cmap='inferno_r')
plt.title('Confusion Matrix for Logistic Regression',fontsize=12, y=1.06)
```

Out[30]: Text(0.5, 1.06, 'Confusion Matrix for Logistic Regression')



In [31]:
```
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred_mod))
```

```
              precision    recall  f1-score   support

           0       0.69      0.87      0.77       157
           1       0.71      0.43      0.54       111

    accuracy                           0.69       268
   macro avg       0.70      0.65      0.65       268
weighted avg       0.69      0.69      0.67       268
```

In [32]:
```
LogisticRegressionScore*100
```

Out[32]: 69.02985074626866

Random Forest Classifier

A Random Forest is a classifier that contains a number of decision

trees on various subsets of the given data sets and takes the average to improve the predictive accuracy of that dataset

It is a supervised learning algorithm that uses ensembled learning method from regression

In [33]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X_train,y_train)
```

Out[33]:
```
RandomForestClassifier()
```

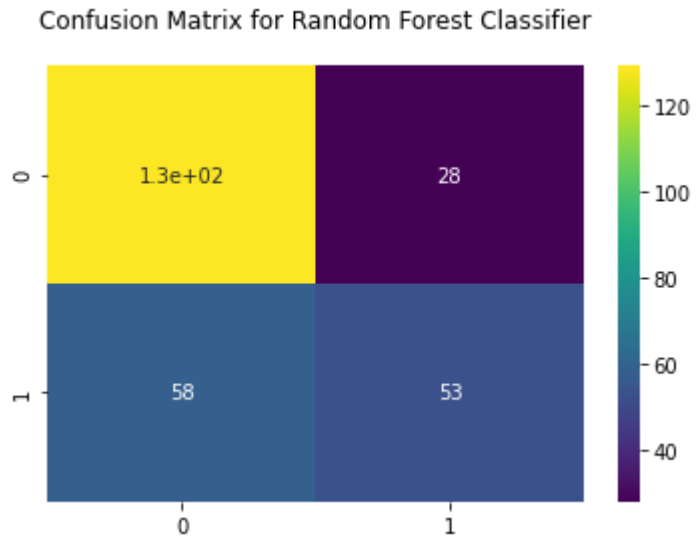
In [34]:
```
RandomForestClassifierScore = rfc.score(X_test,y_test)
```

In [35]:
```
y_pred_rfc = rfc.predict(X_test)
cf_matrix = confusion_matrix(y_test, y_pred_rfc)
cf_matrix
```

Out[35]:
```
array([[129,  28],
       [ 58,  53]], dtype=int64)
```

In [36]:
```
sns.heatmap(cf_matrix, annot=True, cmap='viridis')
plt.title('Confusion Matrix for Random Forest Classifier',fontsize=12,y=1.06)
```

Out[36]:
```
Text(0.5, 1.06, 'Confusion Matrix for Random Forest Classifier')
```



In [37]:
```
print(metrics.classification_report(y_test,y_pred_rfc))
```

```
              precision    recall  f1-score   support

           0       0.69      0.82      0.75       157
           1       0.65      0.48      0.55       111

    accuracy                           0.68       268
   macro avg       0.67      0.65      0.65       268
weighted avg       0.68      0.68      0.67       268
```

In [38]:
```
RandomForestClassifierScore*100
```

Out[38]:
```
67.91044776119402
```

Decision Tree Classifier

Decision tree is a graphical representation for getting all the possible solutions to a problem or decision based on given conditions

The Decision tree classifier creates the classification model by building a decision model tree

In [39]:
```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(X_train,y_train)
```

Out[39]:
```
DecisionTreeClassifier()
```

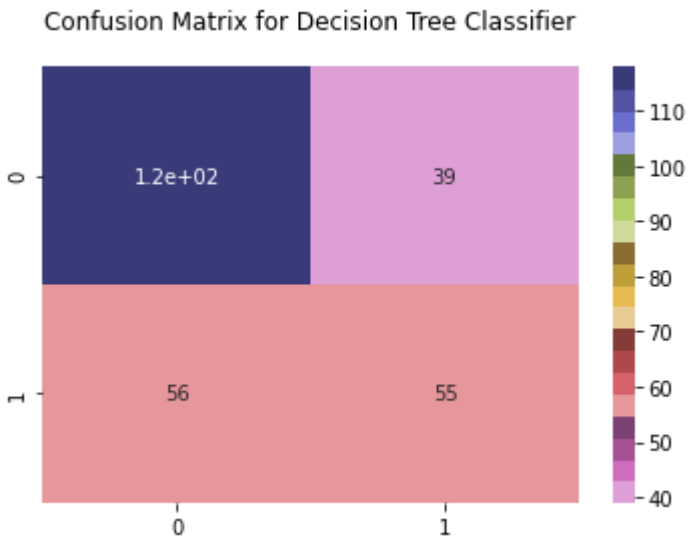
In [40]:
```
DecisionTreeClassifierScore = dtc.score(X_test,y_test)
```

In [41]:
```
y_pred_dtc = dtc.predict(X_test)
cf_matrix = confusion_matrix(y_test,y_pred_dtc)
cf_matrix
```

Out[41]:
```
array([[118,  39],
       [ 56,  55]], dtype=int64)
```

In [42]:
```
sns.heatmap(cf_matrix, annot=True, cmap='tab20b_r')
plt.title("Confusion Matrix for Decision Tree Classifier",fontsize=12, y=1.06)
```

Out[42]: Text(0.5, 1.06, 'Confusion Matrix for Decision Tree Classifier')



In [43]:
```
print(metrics.classification_report(y_test,y_pred_dtc))
```

```
              precision    recall  f1-score   support

           0       0.68      0.75      0.71       157
           1       0.59      0.50      0.54       111

    accuracy                           0.65       268
   macro avg       0.63      0.62      0.62       268
weighted avg       0.64      0.65      0.64       268
```

In [44]:
```
DecisionTreeClassifierScore*100
```

Out[44]: 64.55223880597015

In [46]:
```
from sklearn.model_selection import cross_val_score
rf = RandomForestClassifier()
scores = cross_val_score(rf,X_train,y_train,cv=2,scoring='accuracy')
print("scores :",scores)
print("mean :",scores.mean())
print("standard deviation :",scores.std())
```

```
scores : [0.6474359  0.66559486]
mean : 0.6565153763706819
standard deviation : 0.00907947893478439
```

Accuracy of Three Machine learning Models

In [47]:
```
print("Accuracy obtained by LogisticRegressionModel : ",LogisticRegressionScore*1
print("Accuracy obtained by RandomForestClassifierModel :",RandomForestClassifier
print("Accuracy obtained by DecisionTreeClassifierModel :",DecisionTreeClassifier
```

```
Accuracy obtained by LogisticRegressionModel :  69.02985074626866
Accuracy obtained by RandomForestClassifierModel : 67.91044776119402
Accuracy obtained by DecisionTreeClassifierModel : 64.55223880597015
```

In [ ]:
```

```