Import libraries and modules

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from scipy import stats
from xgboost import XGBRegressor
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score,
confusion_matrix,classification_report

import warnings
warnings.filterwarnings('ignore')

data=pd.read_csv(r"C:\Users\Admin\Downloads\AMZN (1).csv")
data.head()

         Date      Open      High       Low     Close  Adj Close
Volume
0  1997-05-16  0.098438  0.098958  0.085417  0.086458   0.086458
294000000
1  1997-05-19  0.088021  0.088542  0.081250  0.085417   0.085417
122136000
2  1997-05-20  0.086458  0.087500  0.081771  0.081771   0.081771
109344000
3  1997-05-21  0.081771  0.082292  0.068750  0.071354   0.071354
377064000
4  1997-05-22  0.071875  0.072396  0.065625  0.069792   0.069792
235536000

data.shape

(6378, 7)

data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6378 entries, 0 to 6377
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
```

```
---  ------     --------------  -----
 0   Date       6378 non-null   object
 1   Open       6378 non-null   float64
 2   High       6378 non-null   float64
 3   Low        6378 non-null   float64
 4   Close      6378 non-null   float64
 5   Adj Close  6378 non-null   float64
 6   Volume     6378 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 348.9+ KB
```

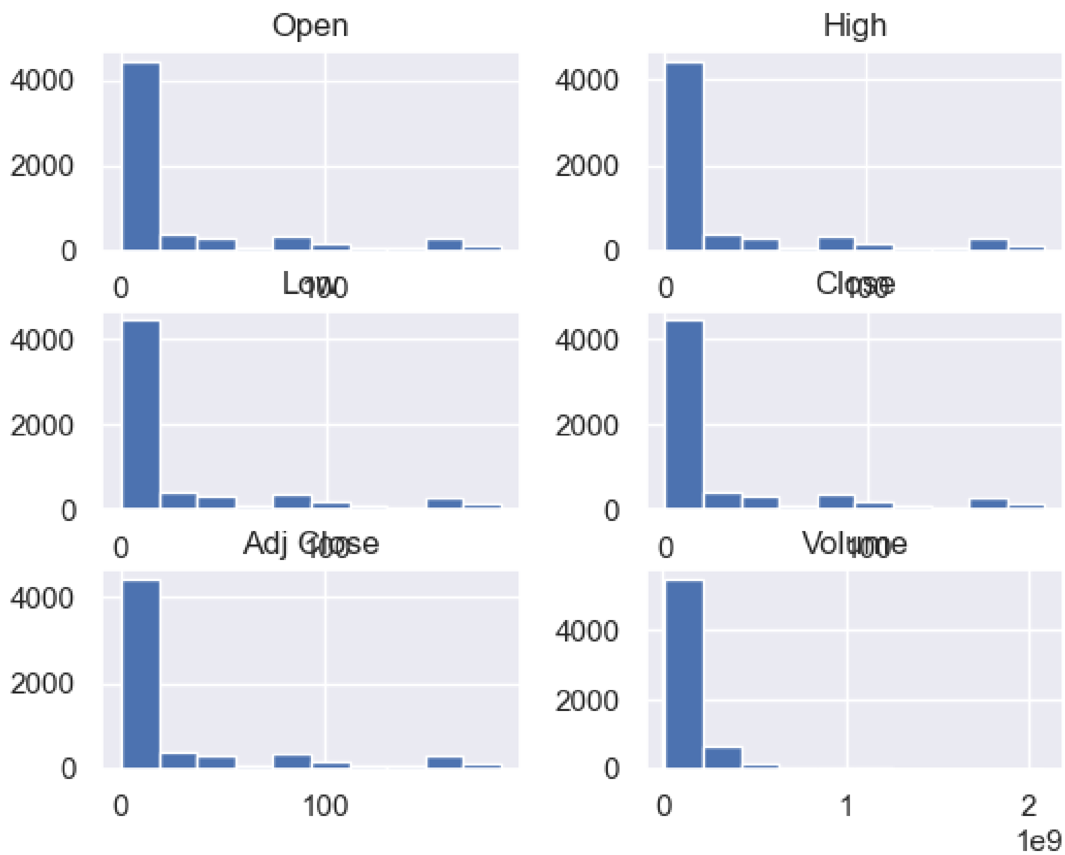
```
data.isnull().sum()
```

```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

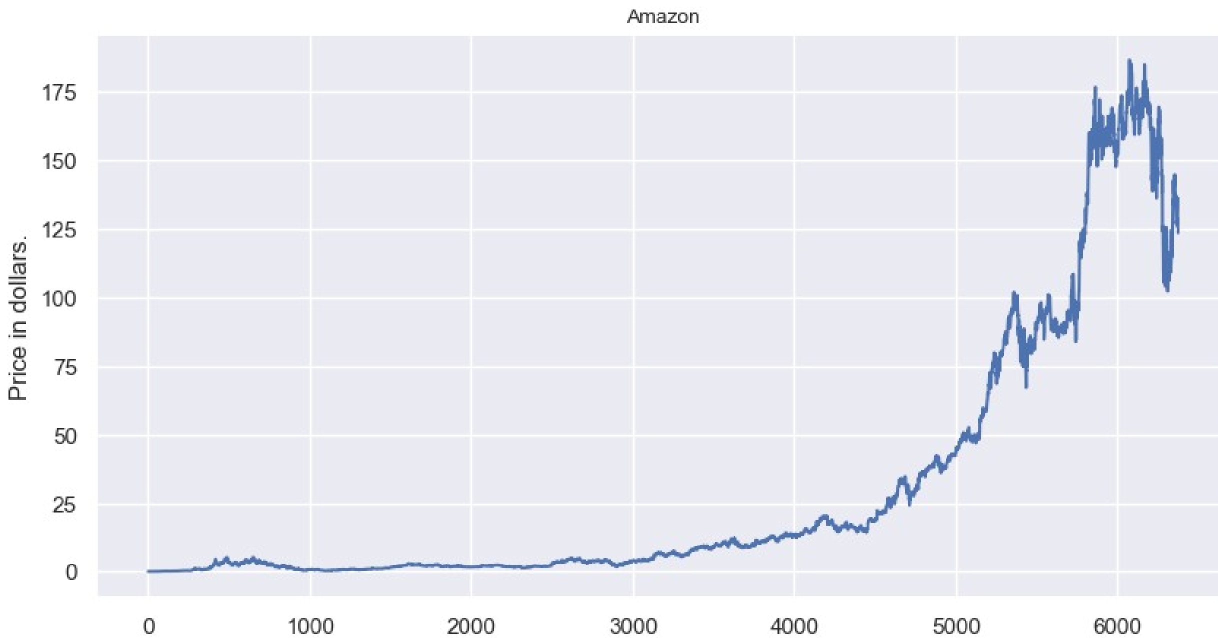
```
data.describe()
```

```
              Open         High          Low        Close    Adj Close  
\
count  6378.000000  6378.000000  6378.000000  6378.000000  6378.000000   

mean     30.166931    30.520510    29.778388    30.156127    30.156127   

std      47.549504    48.092332    46.941417    47.514592    47.514592   

min       0.070313     0.072396     0.065625     0.069792     0.069792   

25%       1.972500     2.011250     1.940719     1.974750     1.974750   

50%       6.074000     6.200250     5.956500     6.103250     6.103250   

75%      35.039376    35.393752    34.880251    35.150125    35.150125   

max     187.199997   188.654007   184.839493   186.570496   186.570496   


             Volume  
count  6.378000e+03  
mean   1.438332e+08  
std    1.402886e+08  
min    9.744000e+06  
25%    6.978400e+07  
50%    1.070510e+08  
```

```
75%    1.626890e+08
max    2.086584e+09

data.hist()

array([[<AxesSubplot:title={'center':'Open'}>,
        <AxesSubplot:title={'center':'High'}>],
       [<AxesSubplot:title={'center':'Low'}>,
        <AxesSubplot:title={'center':'Close'}>],
       [<AxesSubplot:title={'center':'Adj Close'}>,
        <AxesSubplot:title={'center':'Volume'}>]], dtype=object)
```




```
plt.figure(figsize=(10,5))
plt.plot(data['Close'])
plt.title('Amazon', fontsize=10)
plt.ylabel('Price in dollars.')
plt.show()
```

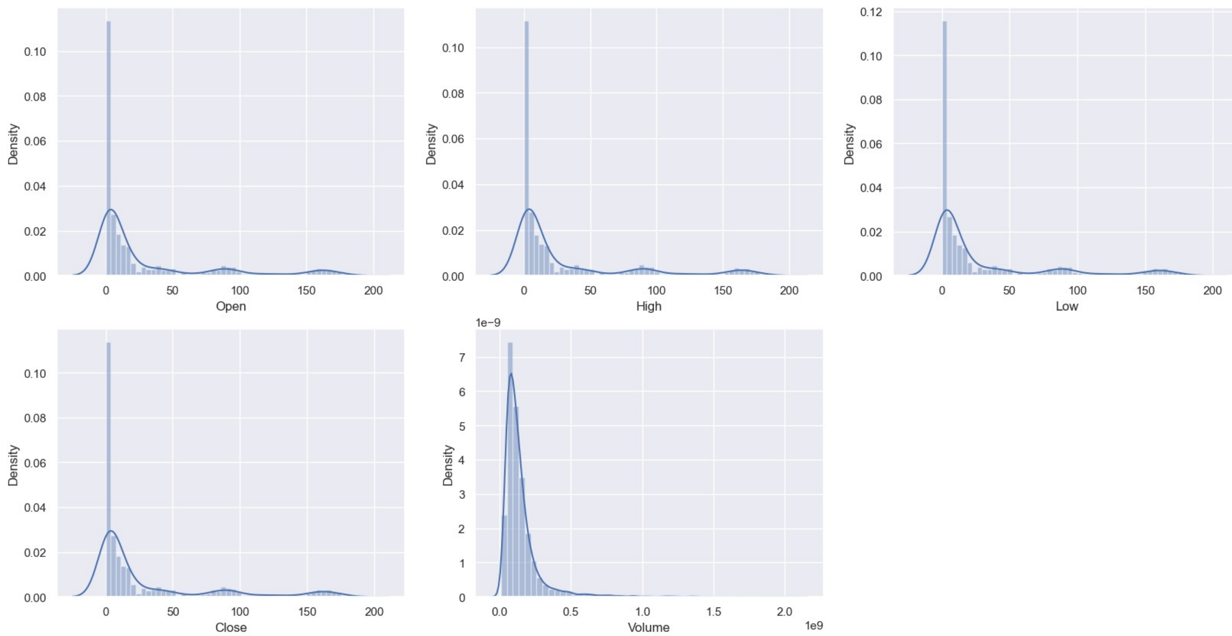



```
data[data['Close'] == data['Adj Close']].shape
```

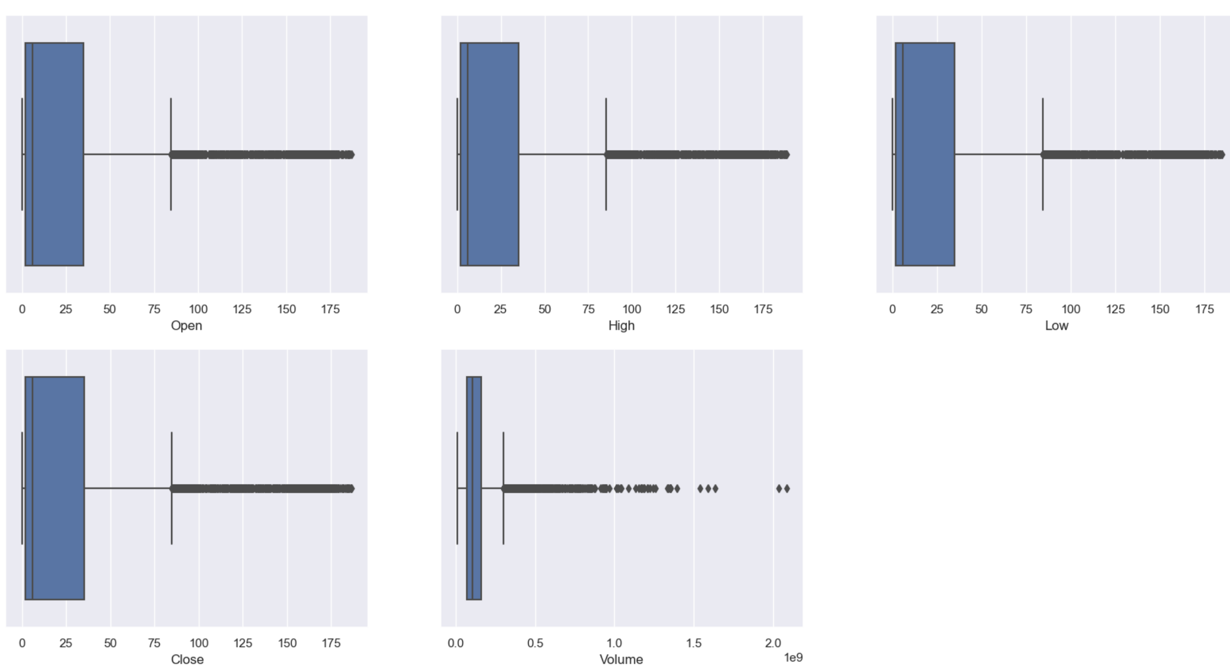
```
(6378, 7)
```

```
data = data.drop(['Adj Close'], axis=1)
```

```
features = ['Open', 'High', 'Low', 'Close', 'Volume']

plt.subplots(figsize=(20,10))

for i, col in enumerate(features):
  plt.subplot(2,3,i+1)
  sns.distplot(data[col])
plt.show()
```

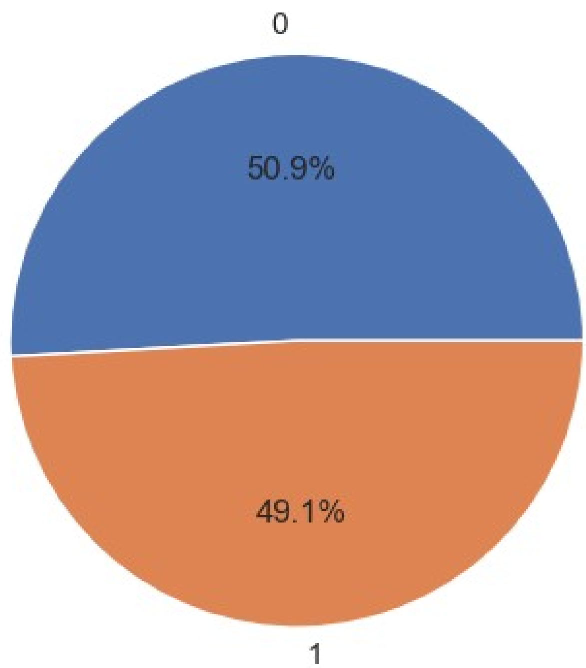



```
plt.subplots(figsize=(20,10))
for i, col in enumerate(features):
  plt.subplot(2,3,i+1)
  sns.boxplot(data[col])
plt.show()
```

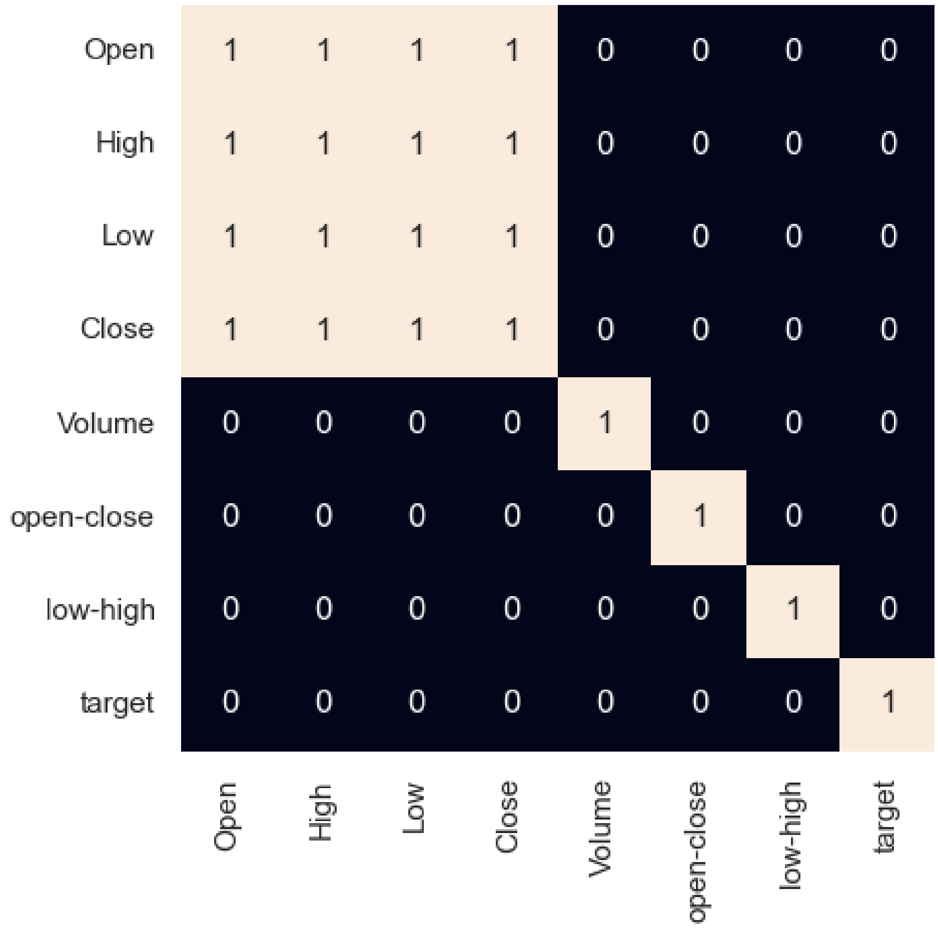



```
data['open-close']  = data['Open'] - data['Close']
data['low-high']  = data['Low'] - data['High']
data['target'] = np.where(data['Close'].shift(-1) > data['Close'], 1,
```

```
0)


data['open-close']  = data['Open'] - data['Close']
data['low-high']  = data['Low'] - data['High']
data['target'] = np.where(data['Close'].shift(-1) > data['Close'], 1,
0)


plt.pie(data['target'].value_counts().values,labels=[0, 1],
autopct='%1.1f%%')
plt.show()
```




```
plt.figure(figsize=(5, 5))
sns.heatmap(data.corr() > 0.9, annot=True, cbar=False)
plt.show()
# As our concern is with the highly
# correlated features only so, we will visualize
# our heatmap as per that criteria only.
```




```
X = data[['open-close', 'low-high']]
y = data['target']


X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.02,rand
om_state=1)
```

---------------------------Random Forest---------------------

```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
X_train=ss.fit_transform(X_train) # xtrain = training input samples
X_test=ss.transform(X_test) # xtest - testing input samples

clfr=RandomForestClassifier(n_estimators=10,criterion='entropy',random
_state=0)
```

```
clfr.fit(X_train,y_train)

RandomForestClassifier(criterion='entropy', n_estimators=10,
random_state=0)

from sklearn.ensemble import RandomForestClassifier

clfr1=RandomForestClassifier(n_estimators=10,criterion='gini',random_s
tate=0)

clfr1.fit(X_train,y_train)


RandomForestClassifier(n_estimators=10, random_state=0)

from sklearn.metrics import
confusion_matrix,classification_report,accuracy_score

ypre=clfr.predict(X_test)# entropy ypre calculation

ypre1=clfr1.predict(X_test)# gini ypre calculation

print('entropy Accuracy Score:')
accuracy_score(y_test,ypre)*100

entropy Accuracy Score:

52.34375

print('gini Accuracy Score:')
accuracy_score(y_test,ypre1)*100

gini Accuracy Score:

50.78125

print('entropy - confusion matrix\n--------------------\n')
print(confusion_matrix(y_test,ypre))
print('gini - confusion matrix\n--------------------\n')
print(confusion_matrix(y_test,ypre1))

entropy - confusion matrix
--------------------

[[42 25]
 [36 25]]
gini - confusion matrix
--------------------

[[38 29]
 [34 27]]
```

```
print('entropy result\n--------------')
print(classification_report(y_test,ypre))
print('gini index result\n---------------------')
print(classification_report(y_test,ypre1))
```

```
entropy result
--------------
              precision    recall  f1-score   support

           0       0.54      0.63      0.58        67
           1       0.50      0.41      0.45        61

    accuracy                           0.52       128
   macro avg       0.52      0.52      0.51       128
weighted avg       0.52      0.52      0.52       128

gini index result
---------------------
              precision    recall  f1-score   support

           0       0.53      0.57      0.55        67
           1       0.48      0.44      0.46        61

    accuracy                           0.51       128
   macro avg       0.50      0.50      0.50       128
weighted avg       0.51      0.51      0.51       128
```

-----------------------LOGISTIC REGRESSION--------------------

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.15,random_state=1)

log_reg = LogisticRegression()
log_reg.fit(X_train,y_train)
```

```
LogisticRegression()
```

```
ypre2 = log_reg.predict(X_test)
```

```
print(confusion_matrix(y_test,ypre2))
```

```
[[  5 487]
 [  7 458]]
```

```
print(classification_report(y_test,ypre2))
```

```
              precision    recall  f1-score   support

           0       0.42      0.01      0.02       492
           1       0.48      0.98      0.65       465

    accuracy                           0.48       957
   macro avg       0.45      0.50      0.33       957
weighted avg       0.45      0.48      0.33       957
```

```
print(accuracy_score(y_test,ypre2))
```

```
0.48380355276907
```

____________________________________K-NEAREST NEIGHBOUR____________________________________

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
print(X_train.shape)
print(y_train.shape)
print(X_train)
```

```
(5102, 2)
(5102,)
      open-close  low-high
2275    0.021500 -0.058500
4603    0.130001 -0.828501
2202    0.011000 -0.058500
471    -0.023438 -0.321875
4060   -0.208500 -0.276500
...          ...       ...
3772   -0.086500 -0.247500
5191   -0.850502 -0.974499
5226    0.501000 -1.338997
5390   -1.822998 -3.116501
860     0.162500 -0.209375

[5102 rows x 2 columns]
```

```
scaler = StandardScaler()
scaled_X_train = scaler.fit_transform(X_train)
scaled_X_test = scaler.transform(X_test)
print(scaled_X_train)
print(scaled_X_test)
```

```
[[ 0.00664754  0.49733098]
 [ 0.12129169 -0.06298151]
```

```
 [-0.00444695  0.49733098]
 ...
 [ 0.51329603 -0.43445802]
 [-1.94228284 -1.72790789]
 [ 0.15563073  0.38754262]]
[[-0.11592018  0.43584222]
 [ 0.24174512  0.24955675]
 [ 0.06528986  0.30886321]
 ...
 [ 1.90855947 -1.68424941]
 [ 2.12304454 -3.94623891]
 [ 0.17042339  0.40964583]]
```

```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7) # here n_neighbors is k
knn.fit(scaled_X_train,y_train)
```

```
KNeighborsClassifier(n_neighbors=7)
```

```python
ypre3 = knn.predict(scaled_X_test)
```

```python
confusion_matrix(y_test,ypre3)
```

```
array([[299, 313],
       [305, 359]], dtype=int64)
```

```python
print(classification_report(y_test,ypre3))
```

```
              precision    recall  f1-score   support

           0       0.50      0.49      0.49       612
           1       0.53      0.54      0.54       664

    accuracy                           0.52      1276
   macro avg       0.51      0.51      0.51      1276
weighted avg       0.52      0.52      0.52      1276
```

```python
accuracy_score(y_test,ypre3)
```

```
0.5156739811912225
```

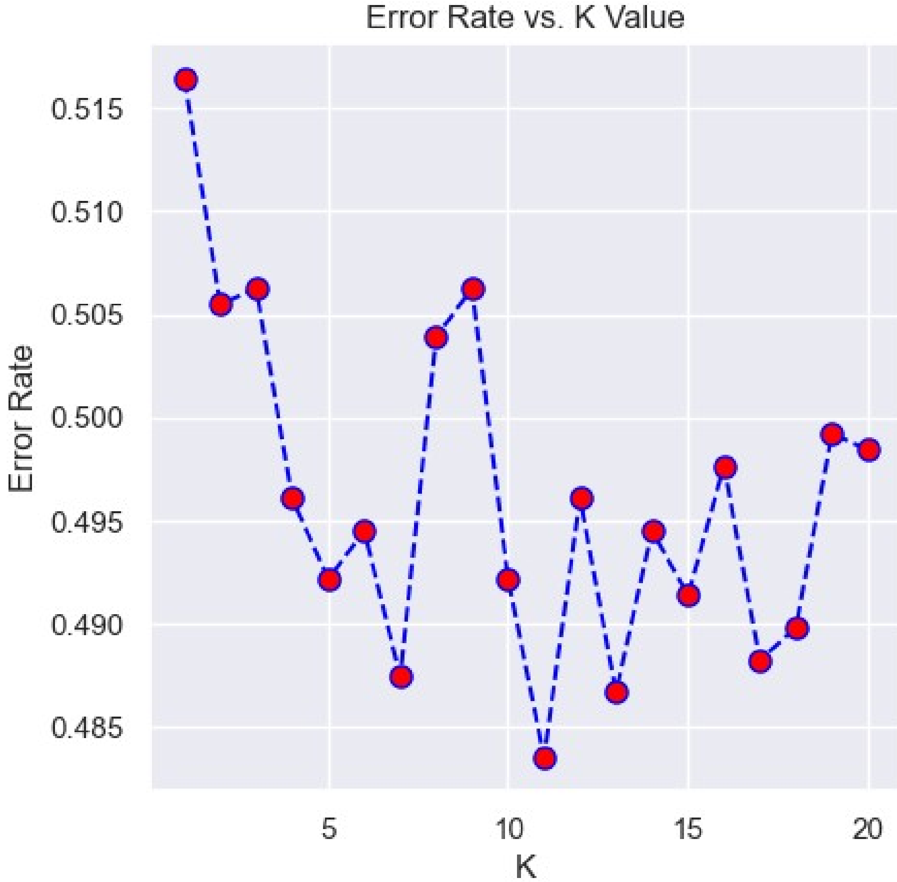
```python
t=1-accuracy_score(y_test,ypre3)
t
```

```
0.48432601880877746
```

```python
error_rate = []
for i in range(1,21):
```

```
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    t=1-accuracy_score(y_test,pred_i)
    error_rate.append(t)

plt.figure(figsize=(5,5))
plt.plot(range(1,21),error_rate,color='blue', linestyle='dashed', 
marker='o',markerfacecolor='red', markersize=8)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

```
Text(0, 0.5, 'Error Rate')
```



----------------------PCA----------------------

```
pca=PCA(n_components=2)
pct=pca.fit_transform(X)# pct - model
```

```
principal_data=pd.DataFrame(pct,columns=['PC1','PC2'])

final_data=pd.concat([principal_data,data[['target']]],axis=1)


principal_data.head()
```

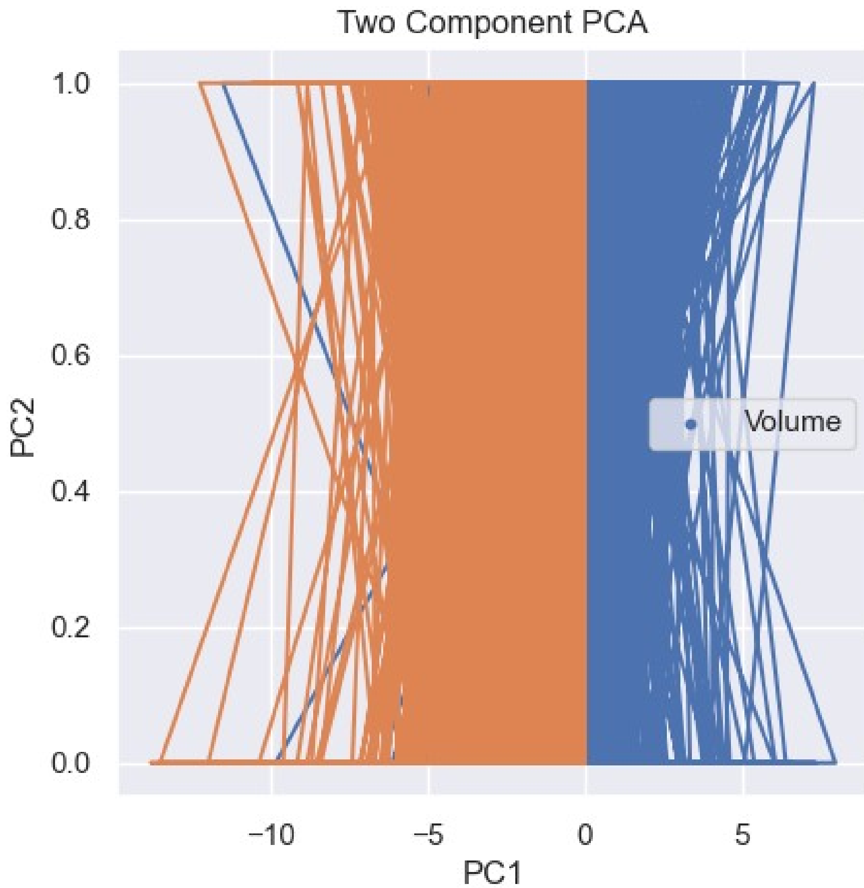
	PC1	PC2
0	-0.725550	-0.066402
1	-0.732614	-0.057623
2	-0.733984	-0.059837
3	-0.725689	-0.064845
4	-0.733179	-0.057151

```
final_data.head()
```

	PC1	PC2	target
0	-0.725550	-0.066402	0
1	-0.732614	-0.057623	0
2	-0.733984	-0.059837	0
3	-0.725689	-0.064845	0
4	-0.733179	-0.057151	1

```
fig=plt.figure(figsize=(5,5))
ax=fig.add_subplot(1,1,1)
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_title('Two Component PCA')
targets=['Volume']
colors=['b']

for target,color in zip(targets,colors):
    index=final_data['target']==target

ax.scatter(final_data.loc[index,'PC1'],final_data.loc[index,'PC2'],c=c
olor,s=10)
    ax.legend(targets)
    #ax.grid()
plt.plot(X,y)

pca.explained_variance_ratio_
```

```
array([0.68298854, 0.31701146])
```



----------------XGBOOST---------------

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.15)

xgbr = XGBRegressor(verbosity=0)
print(xgbr)

XGBRegressor(verbosity=0)


xgbr.fit(X_train,y_train)
score=xgbr.score(X_train,y_train)
print('training score is:',score)




from sklearn.model_selection import cross_val_score
cv_score = cross_val_score(xgbr,X_train,y_train,cv=10)
print('cv mean score is:',cv_score.mean())
```

```
training score is: 0.05559676913145761
cv mean score is: -0.02060240453856226
```

```
from sklearn.metrics import mean_squared_error
ypred=xgbr.predict(X_test)
mse=mean_squared_error(y_test,ypred)
print('MSE is:',mse)
print('rmse is:',mse*(1/2.0))
```

```
MSE is: 0.25168441689997245
rmse is: 0.12584220844998623
```

```
x_ax=range(len(y_test))
plt.plot(x_ax,y_test,color='yellow',label='original')
plt.plot(x_ax,ypred,label='predicted')
plt.title('actual data vs predicted')
plt.legend()
plt.show()
```