In [386…

```
import pandas as pd
import numpy as np
import sklearn
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler , LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings('ignore')
```

In [387…

```
df_1=pd.read_csv("NVDA_weekly_return_volatility.csv")
year=df_1['Year'].unique()
Q1_label=[]
yearly_mean=df_1.groupby('Year')['mean_return'].mean().values
for i in range(len(year)):
    for j in range(len(df_1)):
        if df_1['Year'][j]==year[i] and df_1["mean_return"][j]>yearly_mean[i]:
            Q1_label.append('green')
        elif df_1['Year'][j]==year[i]:
            Q1_label.append('red')
df_1['label']=Q1_label
Q1_X=df_1[df_1["Year"]==2017][["mean_return","volatility"]]
Q1_y=df_1[df_1["Year"]==2017]["label"]
Q2_X=df_1.loc[df_1["Year"]==2018][["mean_return","volatility","label"]].reset_i
Q2_y=df_1.loc[df_1["Year"]==2018]["label"]
#NB_classifier = GaussianNB().fit(Q1_X, Q1_y)
# prediction = NB_classifier.predict(Q2_X)
# error_rate = np.mean(prediction != Q2_y)
# print(error_rate)
```

1. implement a Student-t Naive Bayesian classifier (df = 0.5, 1, 5) and compute its accuracy for year 2

In [388…

```
from scipy import stats
from scipy.stats import t
green_prob=len(Q2_X.loc[Q2_X["label"]=="green",:]["label"])/len(Q2_X)
red_prob=len(Q2_X.loc[Q2_X["label"]=="red",:]["label"])/len(Q2_X)
print(green_prob,red_prob)
df_1, location , scale = stats.t.fit(Q1_X["mean_return"])
# print(df, location , scale)
pdfarr=[]
for j in [0.5,1,5]:
    df = j
    a=t.pdf(Q2_X["mean_return"],df, location,scale)
    #print(Q2_X)
    abc=[]

    for i in range(len(Q2_X)):
        #print(Q2_X["label"])
        if Q2_X["label"][i]=="green":
```

```
            abc.append(a[i]*green_prob)
        else:
            abc.append(a[i]*red_prob)
    Q2_X["prob"+str(j)]=abc
    Q2_X["Predict"+str(j)]=Q2_X["prob"+str(j)].apply(lambda x: 'green' if x > Q

print(Q2_X)
```

```
0.5471698113207547 0.4528301886792453
    index  mean_return  volatility  label   prob0.5 Predict0.5     prob1  \
0      52      0.76600    0.369974  green  0.081721        red  0.095927
1      53      0.32060    0.832617  green  0.294929      green  0.392258
2      54      0.11900    1.415193  green  0.519643      green  0.613887
3      55      0.88960    0.890441  green  0.064334        red  0.071247
4      56     -0.47540    1.871303    red  0.079917        red  0.097285
5      57     -0.72440    4.256168    red  0.048847        red  0.052897
6      58      0.95060    1.044823  green  0.057855        red  0.062320
7      59      0.56450    1.593698  green  0.132374        red  0.170487
8      60     -0.21120    1.086412    red  0.167509        red  0.224811
9      61      0.74240    0.929728  green  0.085905        red  0.101983
10     62     -0.39920    1.212748    red  0.096162        red  0.121264
11     63     -1.61280    1.390139    red  0.016959        red  0.013435
12     64      1.24725    5.054742  green  0.037495        red  0.035747
13     65     -0.20420    2.481235    red  0.171534        red  0.230405
14     66      0.63260    1.479650  green  0.110747        red  0.138462
15     67      0.41540    1.242980  green  0.209543      green  0.281551
16     68      0.18520    1.829708  green  0.468629      green  0.571597
17     69     -0.12540    1.756984    red  0.227248      green  0.303937
18     70      0.53080    0.782874  green  0.145583        red  0.189979
19     71     -0.18780    0.624422    red  0.181502        red  0.244105
20     72      0.41420    0.767087  green  0.210406      green  0.282742
21     73      0.61625    1.069643  green  0.115409        red  0.145363
22     74      0.17000    0.998158  green  0.485604      green  0.586133
23     75     -0.29500    0.724619    red  0.128312        red  0.168709
24     76      0.05820    0.798730    red  0.412774      green  0.494138
25     77     -0.35340    1.376620    red  0.108564        red  0.139633
26     78      0.64500    1.118987  green  0.107405        red  0.133521
27     79      0.83340    0.893494  green  0.071416        red  0.081189
28     80      0.16420    1.159775  green  0.491429      green  0.591013
29     81      0.27900    1.952285  green  0.344499      green  0.449540
30     82      0.07380    1.296973    red  0.422390      green  0.501947
31     83      0.17800    0.510863  green  0.476951      green  0.578782
32     84     -0.18160    0.969596    red  0.185477        red  0.249505
33     85      0.15480    0.847951  green  0.499957      green  0.598058
34     86      0.71640    0.649555  green  0.090926        red  0.109294
35     87     -0.92275    1.353369    red  0.035955        red  0.035779
36     88      0.93700    0.540086  green  0.059205        red  0.064165
37     89      0.16340    1.313901  green  0.492200      green  0.591654
38     90      0.01960    0.345000    red  0.378118      green  0.464494
39     91     -0.38980    1.138941    red  0.098521        red  0.124759
40     92     -0.41660    3.300829    red  0.092021        red  0.115134
41     93     -0.14960    2.072453    red  0.207936      green  0.279309
42     94     -0.24860    4.083815    red  0.148122        red  0.197401
43     95     -0.13220    2.117954    red  0.221609      green  0.296844
44     96      0.65280    2.171354  green  0.105383        red  0.130535
45     97     -0.13500    1.852363    red  0.219336      green  0.293961
46     98     -1.20850    2.256683    red  0.025170        red  0.022539
47     99      1.48820    1.943603  green  0.028326        red  0.024821
48    100     -1.37200    2.593717    red  0.021189        red  0.017998
49    101      0.24740    2.099191  green  0.386299      green  0.493919
50    102     -1.50260    1.837873    red  0.018703        red  0.015280
51    103      0.62325    4.601512  green  0.113376        red  0.142353
52    104      1.17500    0.000000  green  0.041234        red  0.040420

   Predict1     prob5 Predict5
0       red  0.081246      red
1     green  0.531131    green
2     green  0.732610    green
```

```
3       red  0.045487     red
4       red  0.096440     red
5       red  0.030020     red
6       red  0.034374     red
7       red  0.208252     red
8       red  0.308699   green
9       red  0.090853     red
10      red  0.137755     red
11      red  0.001033     red
12      red  0.009617     red
13      red  0.316970   green
14      red  0.152407     red
15    green  0.387608   green
16    green  0.700985   green
17    green  0.415220   green
18      red  0.241862     red
19    green  0.336741   green
20    green  0.389343   green
21      red  0.164423     red
22    green  0.712157   green
23      red  0.219620     red
24    green  0.596161   green
25      red  0.169810     red
26      red  0.143841     red
27      red  0.059121     red
28    green  0.715835   green
29    green  0.592663   green
30    green  0.601895   green
31    green  0.706548   green
32    green  0.344345   green
33    green  0.721083   green
34      red  0.102758     red
35      red  0.012534     red
36      red  0.036572     red
37    green  0.716316   green
38    green  0.573387   green
39      red  0.143858     red
40      red  0.127076     red
41    green  0.384425   green
42      red  0.266527     red
43    green  0.406559   green
44      red  0.138685     red
45    green  0.402992   green
46      red  0.004040     red
47      red  0.003847     red
48      red  0.002256     red
49    green  0.635312   green
50      red  0.001461     red
51      red  0.159176     red
52      red  0.012923     red
```

1. compute the confusion matrices for year 2

In [389…

```
for i in [0.5,1,5]:
    print("the year2 accuracy for",i,"is",accuracy_score(Q2_y, Q2_X["Predict"+s
```

```
the year2 accuracy for 0.5 is 0.5471698113207547
the year2 accuracy for 1 is 0.5094339622641509
the year2 accuracy for 5 is 0.4716981132075472
```

1. what is true positive rate and true negative rate for year 2

In [390…
```
for i in [0.5,1,5]:
    print("the confusion matrix:\n",confusion_matrix(Q2_y, Q2_X["Predict"+str(i
```

```
the confusion matrix:
 [[12 17]
 [ 7 17]]
the confusion matrix:
 [[12 17]
 [ 9 15]]
the confusion matrix:
 [[12 17]
 [11 13]]
```

1. what is the best value of df? Is it better than normal Naive bayesian

In [391…
```
print("The best Naive Bayesian is 0.5")
```

```
The best Naive Bayesian is 0.5
```