In [1]:
```
from google.colab import drive
drive.mount('/content/MyDrive/')
```

```
Mounted at /content/MyDrive/
```

In [65]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import numpy as np
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)
%matplotlib inline
```

In [9]:
```
data = pd.read_csv('/content/MyDrive/MyDrive/Datasets/phishing.txt',sep =
",", names = [ 'UsingIP', 'LongURL', 'ShortURL', 'Symbol@', 'Redirecting
//',
           'PrefixSuffix-', 'SubDomains', 'HTTPS', 'DomainRegLen', 'Favic
on',
           'NonStdPort', 'HTTPSDomainURL', 'RequestURL', 'AnchorURL',
           'LinksInScriptTags', 'ServerFormHandler', 'InfoEmail', 'Abnorm
alURL',
           'WebsiteForwarding', 'StatusBarCust', 'DisableRightClick',
           'UsingPopupWindow', 'IframeRedirection', 'AgeofDomain',
           'DNSRecording', 'WebsiteTraffic', 'PageRank', 'GoogleIndex',
           'LinksPointingToPage', 'StatsReport', 'class' ])
```

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

Out[10]:

	UsingIP	LongURL	ShortURL	Symbol@	Redirecting//	PrefixSuffix-	SubDomains	HTTP
0	-1	1	1	1	-1	-1	-1	
1	1	1	1	1	1	-1	0	
2	1	0	1	1	1	-1	-1	
3	1	0	1	1	1	-1	-1	
4	1	0	-1	1	1	-1	1	

In [11]:
```
data.shape
```

Out[11]:
```
(11055, 31)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11055 entries, 0 to 11054
Data columns (total 31 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   UsingIP              11055 non-null  int64
 1   LongURL              11055 non-null  int64
 2   ShortURL             11055 non-null  int64
 3   Symbol@              11055 non-null  int64
 4   Redirecting//        11055 non-null  int64
 5   PrefixSuffix-        11055 non-null  int64
 6   SubDomains           11055 non-null  int64
 7   HTTPS                11055 non-null  int64
 8   DomainRegLen         11055 non-null  int64
 9   Favicon              11055 non-null  int64
 10  NonStdPort           11055 non-null  int64
 11  HTTPSDomainURL       11055 non-null  int64
 12  RequestURL           11055 non-null  int64
 13  AnchorURL            11055 non-null  int64
 14  LinksInScriptTags    11055 non-null  int64
 15  ServerFormHandler    11055 non-null  int64
 16  InfoEmail            11055 non-null  int64
 17  AbnormalURL          11055 non-null  int64
 18  WebsiteForwarding    11055 non-null  int64
 19  StatusBarCust        11055 non-null  int64
 20  DisableRightClick    11055 non-null  int64
 21  UsingPopupWindow     11055 non-null  int64
 22  IframeRedirection    11055 non-null  int64
 23  AgeofDomain          11055 non-null  int64
 24  DNSRecording         11055 non-null  int64
 25  WebsiteTraffic       11055 non-null  int64
 26  PageRank             11055 non-null  int64
 27  GoogleIndex          11055 non-null  int64
 28  LinksPointingToPage  11055 non-null  int64
 29  StatsReport          11055 non-null  int64
 30  class                11055 non-null  int64
dtypes: int64(31)
memory usage: 2.6 MB
```

In [19]:
```
data.describe()
```

Out[19]:

	UsingIP	LongURL	ShortURL	Symbol@	Redirecting//	PrefixSuffix-
count	11055.000000	11055.000000	11055.000000	11055.000000	11055.000000	11055.000000
mean	0.313795	-0.633198	0.738761	0.700588	0.741474	-0.734962
std	0.949534	0.766095	0.673998	0.713598	0.671011	0.678139
min	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
25%	-1.000000	-1.000000	1.000000	1.000000	1.000000	-1.000000
50%	1.000000	-1.000000	1.000000	1.000000	1.000000	-1.000000
75%	1.000000	-1.000000	1.000000	1.000000	1.000000	-1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [13]:
```
data.isnull().sum()
```

Out[13]:
```
UsingIP                0
LongURL                0
ShortURL               0
Symbol@                0
Redirecting//          0
PrefixSuffix-          0
SubDomains             0
HTTPS                  0
DomainRegLen           0
Favicon                0
NonStdPort             0
HTTPSDomainURL         0
RequestURL             0
AnchorURL              0
LinksInScriptTags      0
ServerFormHandler      0
InfoEmail              0
AbnormalURL            0
WebsiteForwarding      0
StatusBarCust          0
DisableRightClick      0
UsingPopupWindow       0
IframeRedirection      0
AgeofDomain            0
DNSRecording           0
WebsiteTraffic         0
PageRank               0
GoogleIndex            0
LinksPointingToPage    0
StatsReport            0
class                  0
dtype: int64
```

In [14]:
```
for column in data:
  print(f'{column}: {data[column].unique()}')
```

```
UsingIP: [-1  1]
LongURL: [ 1  0 -1]
ShortURL: [ 1 -1]
Symbol@: [ 1 -1]
Redirecting//: [-1  1]
PrefixSuffix-: [-1  1]
SubDomains: [-1  0  1]
HTTPS: [-1  1  0]
DomainRegLen: [-1  1]
Favicon: [ 1 -1]
NonStdPort: [ 1 -1]
HTTPSDomainURL: [-1  1]
RequestURL: [ 1 -1]
AnchorURL: [-1  0  1]
LinksInScriptTags: [ 1 -1  0]
ServerFormHandler: [-1  1  0]
InfoEmail: [-1  1]
AbnormalURL: [-1  1]
WebsiteForwarding: [0 1]
StatusBarCust: [ 1 -1]
DisableRightClick: [ 1 -1]
UsingPopupWindow: [ 1 -1]
IframeRedirection: [ 1 -1]
AgeofDomain: [-1  1]
DNSRecording: [-1  1]
WebsiteTraffic: [-1  0  1]
PageRank: [-1  1]
GoogleIndex: [ 1 -1]
LinksPointingToPage: [ 1  0 -1]
StatsReport: [-1  1]
class: [-1  1]
```

Build a phishing website classifier using Logistic Regression with “C” parameter = 100. Use 70% of data as training data and the remaining 30% as test data. [Hint: Use Scikit-Learn library LogisticRegression] [Hint: Refer to the logistic regression tutorial taught earlier in the course] Print count of misclassified samples in the test data prediction as well as the accuracy score of the model

In [15]:
```
from sklearn.linear_model import LogisticRegression
```

In [16]:
```
X = data.drop(['class'],axis=1)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11055 entries, 0 to 11054
Data columns (total 30 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   UsingIP              11055 non-null  int64
 1   LongURL              11055 non-null  int64
 2   ShortURL             11055 non-null  int64
 3   Symbol@              11055 non-null  int64
 4   Redirecting//        11055 non-null  int64
 5   PrefixSuffix-        11055 non-null  int64
 6   SubDomains           11055 non-null  int64
 7   HTTPS                11055 non-null  int64
 8   DomainRegLen         11055 non-null  int64
 9   Favicon              11055 non-null  int64
 10  NonStdPort           11055 non-null  int64
 11  HTTPSDomainURL       11055 non-null  int64
 12  RequestURL           11055 non-null  int64
 13  AnchorURL            11055 non-null  int64
 14  LinksInScriptTags    11055 non-null  int64
 15  ServerFormHandler    11055 non-null  int64
 16  InfoEmail            11055 non-null  int64
 17  AbnormalURL          11055 non-null  int64
 18  WebsiteForwarding    11055 non-null  int64
 19  StatusBarCust        11055 non-null  int64
 20  DisableRightClick    11055 non-null  int64
 21  UsingPopupWindow     11055 non-null  int64
 22  IframeRedirection    11055 non-null  int64
 23  AgeofDomain          11055 non-null  int64
 24  DNSRecording         11055 non-null  int64
 25  WebsiteTraffic       11055 non-null  int64
 26  PageRank             11055 non-null  int64
 27  GoogleIndex          11055 non-null  int64
 28  LinksPointingToPage  11055 non-null  int64
 29  StatsReport          11055 non-null  int64
dtypes: int64(30)
memory usage: 2.5 MB
```

In [18]:
```
y = data['class']
```

In [20]:
```
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)
```

In [21]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(7738, 30)
(3317, 30)
(7738,)
(3317,)
```

In [22]:
```
LR = LogisticRegression(C=100)
```

In [23]:
```
LR.fit(X_train,y_train)
```

Out[23]:
```
LogisticRegression(C=100)
```

In [25]:
```
score = LR.score(X_train,y_train)
print(score)
```

```
0.9272421814422331
```

In [24]:
```
predicts = LR.predict(X_test)
predicts
```

Out[24]:
```
array([ 1, -1, -1, ...,  1, -1, -1])
```

In [27]:
```
from sklearn.metrics import accuracy_score,classification_report,confusio
n_matrix
```

In [28]:
```
print('Accuracy Score',accuracy_score(y_test,predicts))
```

```
Accuracy Score 0.9258365993367501
```

In [29]:
```
print(classification_report(y_test,predicts))
```

```
              precision    recall  f1-score   support

          -1       0.93      0.90      0.92      1481
           1       0.92      0.94      0.93      1836

    accuracy                           0.93      3317
   macro avg       0.93      0.92      0.92      3317
weighted avg       0.93      0.93      0.93      3317
```

In [30]:
```
print(confusion_matrix(y_test,predicts))
```

```
[[1337  144]
 [ 102 1734]]
```

In [32]:
```
cnf_1 = pd.DataFrame(confusion_matrix(y_test,predicts),index=[-1,1],colum
ns=[-1,1])
```

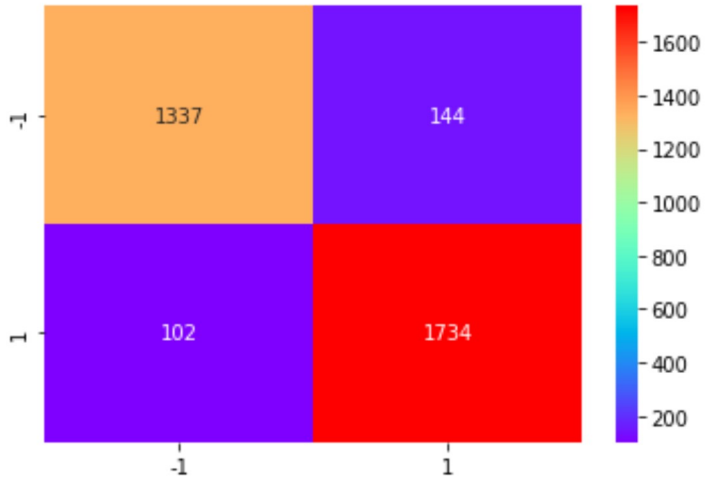
In [33]:
```
cnf_1
```

Out[33]:

	-1	1
-1	1337	144
1	102	1734

In [34]:
```
sns.heatmap(cnf_1,annot=True,fmt='d',cmap=plt.cm.rainbow)
```

Out[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5ae6e4d210>
```



Exercise 2 :

1.Train with only two input parameters - parameter Prefix_Suffix and 13 URL_of _Anchor.
2.Check accuracy using the test data and compare the accuracy with the previou s value.
3.Plot the test samples along with the decision boundary when trained with ind ex 5 and index 13 parameters.

In [35]:
```
x = data[['PrefixSuffix-','AnchorURL']]
```

In [42]:
```
x.head()
```

Out[42]:

	PrefixSuffix-	AnchorURL
0	-1	-1
1	-1	0
2	-1	0
3	-1	0
4	-1	0

In [46]:
```
from sklearn.preprocessing import MinMaxScaler
MM = MinMaxScaler()
```

In [47]:
```
x1=MM.fit_transform(x)
```

In [50]:
```
x1[:5]
```

Out[50]:
```
array([[0. , 0. ],
       [0. , 0.5],
       [0. , 0.5],
       [0. , 0.5],
       [0. , 0.5]])
```

In [53]:
```
x_train_n1,x_test_n1,y_train1,y_test1 = train_test_split(x1,y,test_size=
0.3)
```

In [56]:
```
LR.fit(x_train_n1,y_train1)
```

Out[56]:
```
LogisticRegression(C=100)
```

In [58]:
```
score1 = LR.score(x_train_n1,y_train1)
print('Training score :', score1)
```

```
Training score : 0.8451796329800982
```

In [63]:
```
predicts_n1 = LR.predict(x_test_n1)
predicts_n1
```

Out[63]:
```
array([1, 1, 1, ..., 1, 1, 1])
```

In [64]:
```
print(accuracy_score(y_test1,predicts_n1))
print(classification_report(y_test1,predicts_n1))
print(confusion_matrix(y_test1,predicts_n1))
```

```
0.85770274344287
              precision    recall  f1-score   support

          -1       1.00      0.69      0.81      1494
           1       0.80      1.00      0.89      1823

    accuracy                           0.86      3317
   macro avg       0.90      0.84      0.85      3317
weighted avg       0.89      0.86      0.85      3317

[[1026  468]
 [   4 1819]]
```

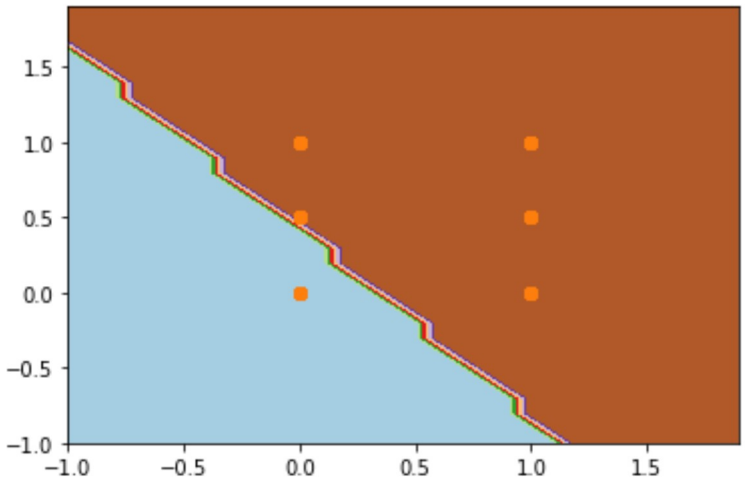
In [51]:
```
# define bounds of the domain
min1, max1 = x1[:, 0].min()-1, x1[:, 0].max()+1
min2, max2 = x1[:, 1].min()-1, x1[:, 1].max()+1
```

In [66]:
```
# define the x and y scale
x1grid = np.arange(min1, max1, 0.1)
x2grid = np.arange(min2, max2, 0.1)
```

In [67]:
```
# create all of the lines and rows of the grid
xx, yy = np.meshgrid(x1grid, x2grid)
# flatten each grid to a vector
r1, r2 = xx.flatten(), yy.flatten()
r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
# horizontal stack vectors to create x1,x2 input for the model
grid = np.hstack((r1,r2))
```

In [68]:
```
yhat = LR.predict(grid)
```

In [69]:
```
# reshape the predictions back into a grid
zz = yhat.reshape(xx.shape)
# plot the grid of x, y and z values as a surface
plt.contourf(xx, yy, zz, cmap='Paired')
# create scatter plot for samples from each class
for class_value in range(2):
        # get row indexes for samples with this class
        row_ix = np.where(y == class_value)
        # create scatter of these samples
        plt.scatter(x1[row_ix, 0], x1[row_ix, 1], cmap='Paired')
# show the plot
plt.show()
```



In [ ]:
```

```