In [1]:

```
#Required imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt #for histogram
import seaborn as sns #for heatmap
```

In [2]:

```
#STEP 1 - Use pandas to read data as a dataframe.
urlsDF = pd.read_csv('dataset.csv')
urlsDF.head()
```

Out[2]:

	index	having_IPhaving_IP_Address	URLURL_Length	Shortining_Service	having_At_Symbol	double_slash_redirect
0	1	-1	1	1	1	
1	2	1	1	1	1	
2	3	1	0	1	1	
3	4	1	0	1	1	
4	5	1	0	-1	1	

5 rows × 32 columns

In [3]:

```
# Determine the number of samples present in the data through shape method
print(f"Number of samples are {urlsDF.shape}")
```

```
Number of samples are (11055, 32)
```

In [4]:

```
# Determine unique elements in all the features by looping through the columns and using
for col in urlsDF.columns[1:]:
    print(f'Unique values of column {col}: {urlsDF[col].unique()}')
```

```
Unique values of column having_IPhaving_IP_Address: [-1  1]
Unique values of column URLURL_Length: [ 1  0 -1]
Unique values of column Shortining_Service: [ 1 -1]
Unique values of column having_At_Symbol: [ 1 -1]
Unique values of column double_slash_redirecting: [-1  1]
Unique values of column Prefix_Suffix: [-1  1]
Unique values of column having_Sub_Domain: [-1  0  1]
Unique values of column SSLfinal_State: [-1  1  0]
Unique values of column Domain_registeration_length: [-1  1]
Unique values of column Favicon: [ 1 -1]
Unique values of column port: [ 1 -1]
Unique values of column HTTPS_token: [-1  1]
Unique values of column Request_URL: [ 1 -1]
Unique values of column URL_of_Anchor: [-1  0  1]
Unique values of column Links_in_tags: [ 1 -1  0]
Unique values of column SFH: [-1  1  0]
Unique values of column Submitting_to_email: [-1  1]
Unique values of column Abnormal_URL: [-1  1]
Unique values of column Redirect: [0 1]
Unique values of column on_mouseover: [ 1 -1]
Unique values of column RightClick: [ 1 -1]
Unique values of column popUpWidnow: [ 1 -1]
Unique values of column Iframe: [ 1 -1]
Unique values of column age_of_domain: [-1  1]
Unique values of column DNSRecord: [-1  1]
Unique values of column web_traffic: [-1  0  1]
Unique values of column Page_Rank: [-1  1]
Unique values of column Google_Index: [ 1 -1]
Unique values of column Links_pointing_to_page: [ 1  0 -1]
Unique values of column Statistical_report: [-1  1]
Unique values of column Result: [-1  1]
```

In [5]:

```
#Check if there is any null value in any features through the info method
urlsDF.info()

#Shows that all columns have 11055 non-null values.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11055 entries, 0 to 11054
Data columns (total 32 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   index                        11055 non-null  int64
 1   having_IPhaving_IP_Address   11055 non-null  int64
 2   URLURL_Length                11055 non-null  int64
 3   Shortining_Service           11055 non-null  int64
 4   having_At_Symbol             11055 non-null  int64
 5   double_slash_redirecting     11055 non-null  int64
 6   Prefix_Suffix                11055 non-null  int64
 7   having_Sub_Domain            11055 non-null  int64
 8   SSLfinal_State               11055 non-null  int64
 9   Domain_registeration_length  11055 non-null  int64
 10  Favicon                      11055 non-null  int64
 11  port                         11055 non-null  int64
 12  HTTPS_token                  11055 non-null  int64
 13  Request_URL                  11055 non-null  int64
 14  URL_of_Anchor                11055 non-null  int64
 15  Links_in_tags                11055 non-null  int64
 16  SFH                          11055 non-null  int64
 17  Submitting_to_email          11055 non-null  int64
 18  Abnormal_URL                 11055 non-null  int64
 19  Redirect                     11055 non-null  int64
 20  on_mouseover                 11055 non-null  int64
 21  RightClick                   11055 non-null  int64
 22  popUpWidnow                  11055 non-null  int64
 23  Iframe                       11055 non-null  int64
 24  age_of_domain                11055 non-null  int64
 25  DNSRecord                    11055 non-null  int64
 26  web_traffic                  11055 non-null  int64
 27  Page_Rank                    11055 non-null  int64
 28  Google_Index                 11055 non-null  int64
 29  Links_pointing_to_page       11055 non-null  int64
 30  Statistical_report           11055 non-null  int64
 31  Result                       11055 non-null  int64
dtypes: int64(32)
memory usage: 2.7 MB
```

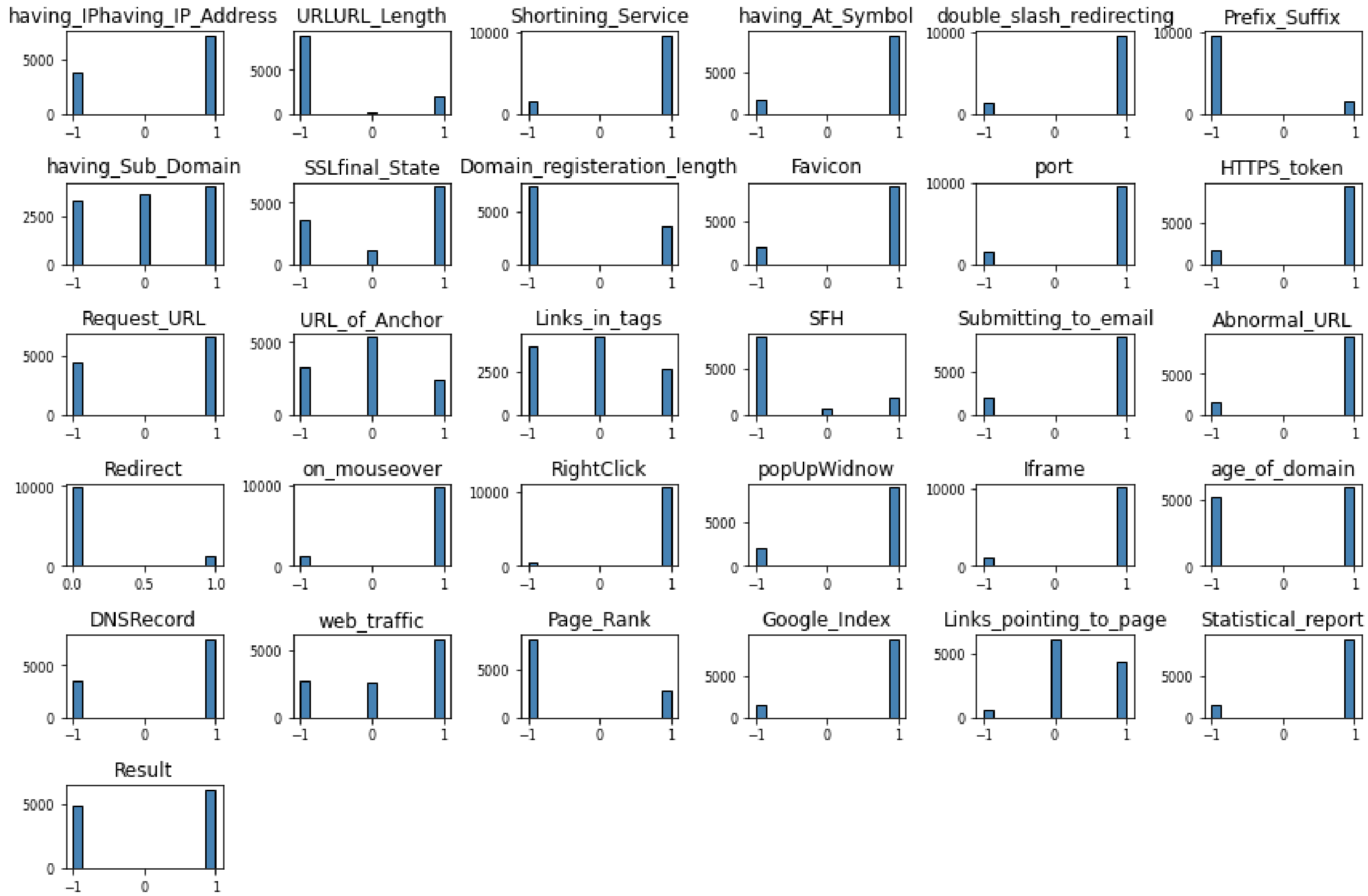
In [6]:

```
#Check if there is any null value in any features - Another way to check this is througl
urlsDF.isnull().sum()

#All have 0 null values indicating that we don't have any null values in any column.
```

Out[6]:
```
index                          0
having_IPhaving_IP_Address     0
URLURL_Length                  0
Shortining_Service             0
having_At_Symbol               0
double_slash_redirecting       0
Prefix_Suffix                  0
having_Sub_Domain              0
SSLfinal_State                 0
Domain_registeration_length    0
Favicon                        0
port                           0
HTTPS_token                    0
Request_URL                    0
URL_of_Anchor                  0
Links_in_tags                  0
SFH                            0
Submitting_to_email            0
Abnormal_URL                   0
Redirect                       0
on_mouseover                   0
RightClick                     0
popUpWidnow                    0
Iframe                         0
age_of_domain                  0
DNSRecord                      0
web_traffic                    0
Page_Rank                      0
Google_Index                   0
Links_pointing_to_page         0
Statistical_report             0
Result                         0
dtype: int64
```

In [7]:
```
#Explore the data using histogram
dfToPlotHistogram = urlsDF.iloc[:,1:] #Excluding the first index column
dfToPlotHistogram.hist(bins=15, color='steelblue', edgecolor='black', linewidth=1.0, xla
plt.tight_layout(rect=(0,0,2.0,2.0))
```



In [8]:

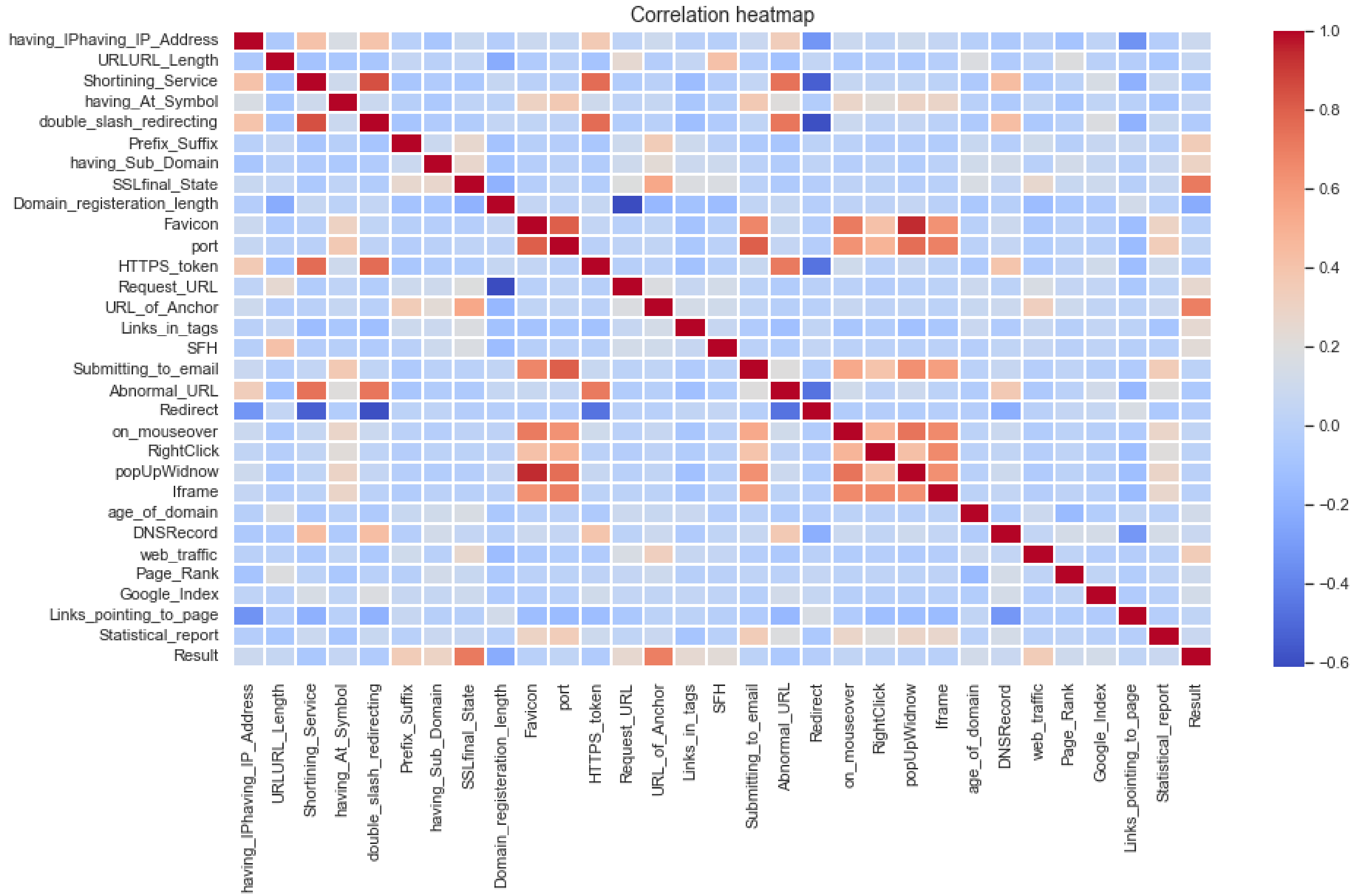
```
#Explore the data using heatmap

#Extract the correlation between the columns. Default is Pearson method
correlationUrlsDF = urlsDF.iloc[:,1:].corr() #Dropping the first column as it is the inc

#Use seaborn and display the heatmap
sns.set(rc = {'figure.figsize':(15,8)})
correlationHeatMap = sns.heatmap(correlationUrlsDF, cmap="coolwarm", fmt='.2f', linewidt
correlationHeatMap.set_title("Correlation heatmap", fontsize=14)
```

Out[8]:

```
Text(0.5, 1.0, 'Correlation heatmap')
```



Interpreting the heatmap:

The color code corresponding to 1.0 on the right side scale indicates higher degree of positive correlation between X and Y columns and similarly -0.6 indicates the negative correlation between X and Y columns.

Example:

There is a higher degree of positive correlation between column "Shortining_Service" and column "double_slash_redirecting" as compared to correlation between column "Shortining_Service" and "having_At_Symbol"

Logic for filtering out the features.

Assuming the threshold as 0.6

Step 1. Find the abs of the correlation matrix of the DF

Step 2. For all the Features(looping), find the Feature that correlate with another Feature with value > than the threshold and less than 1. Note: 1 is the correlation value of the feature with itself. Hence the check of less than 1.

Step 3. For each pair of Feature found in step 2, find the feature that has least correlation with the Target column. (Result column in the given DF)

Step 4. Elimate the least correlated columns from DF

In [9]:
```
threshold = 0.6
```

```
#Step 1: Find the abs of the correlation matrix of the DF
correlationUrlsDF = urlsDF.iloc[:,1:].corr().abs()
correlationUrlsDF
```

Out[9]:

	having_IPhaving_IP_Address	URLURL_Length	Shortining_Service	having_At_Symbol	c
having_IPhaving_IP_Address	1.000000	0.052411	0.403461	0.158699	
URLURL_Length	0.052411	1.000000	0.097881	0.075108	
Shortining_Service	0.403461	0.097881	1.000000	0.104447	
having_At_Symbol	0.158699	0.075108	0.104447	1.000000	
double_slash_redirecting	0.397389	0.081247	0.842796	0.086960	
Prefix_Suffix	0.005257	0.055247	0.080471	0.011726	
having_Sub_Domain	0.080745	0.003997	0.041916	0.058976	
SSLfinal_State	0.071414	0.048754	0.061426	0.031220	
Domain_registeration_length	0.022739	0.221892	0.060923	0.015522	
Favicon	0.087025	0.042497	0.006101	0.304899	
port	0.060979	0.000323	0.002201	0.364891	
HTTPS_token	0.363534	0.089383	0.757838	0.104561	
Request_URL	0.029773	0.246348	0.037235	0.027909	
URL_of_Anchor	0.099847	0.023396	0.000561	0.057914	
Links_in_tags	0.006212	0.052869	0.133379	0.070861	
SFH	0.010962	0.414196	0.022723	0.008672	
Submitting_to_email	0.077989	0.014457	0.049328	0.370123	
Abnormal_URL	0.336549	0.106761	0.739290	0.203945	
Redirect	0.321181	0.046832	0.534530	0.028160	
on_mouseover	0.084059	0.045103	0.062383	0.279697	
RightClick	0.042881	0.013613	0.038118	0.219503	
popUpWidnow	0.096882	0.049381	0.036616	0.290893	
Iframe	0.054694	0.013838	0.016581	0.284410	
age_of_domain	0.010446	0.179426	0.052596	0.005499	
DNSRecord	0.050733	0.040823	0.436064	0.047872	
web_traffic	0.002922	0.008993	0.047074	0.032918	
Page_Rank	0.091774	0.183518	0.014591	0.064735	
Google_Index	0.029153	0.002902	0.155844	0.037061	
Links_pointing_to_page	0.339065	0.022987	0.198410	0.006080	
Statistical_report	0.019103	0.067153	0.085461	0.080357	
Result	0.094160	0.057430	0.067966	0.052948	

31 rows × 31 columns

In [10]:

```
# Step 2. For all the Features(looping), find the Feature that correlate with another Fe
```

```
# and less than 1.

numOfCols = range(len(correlationUrlsDF.columns) - 2) #-2 as we need to exclude the las
numOfRows = range(len(correlationUrlsDF))
dropColumns = []

for colIndx in numOfCols :
    for rowIndx in numOfRows :
        item = correlationUrlsDF.iloc[rowIndx:(rowIndx+1), (colIndx):(colIndx+1)]
        itemCol = item.columns
        itemRow = item.index
        val = item.values
        if val > threshold and val < 1:
                #print(itemCol.values[0], "|", itemRow.values[0], "|", round(val[0][0],

                #Implies we need to remove either the itemCol.values[0] or itemRow.value
#Step 3 - For each pair of Feature found in step 2, elimate that feature that has least

                #Find the correlation value of itemCol.values[0] on the Target
                colCorrelationVal = correlationUrlsDF.loc[itemCol.values[0],'Result']

                #Find the correlation value of itemRow.values[0] on the Target
                rowCorrelationVal = correlationUrlsDF.loc[itemRow.values[0],'Result']

                #print(f'{itemCol.values[0]} value is {colCorrelationVal}')
                #print(f'{itemRow.values[0]} value is {rowCorrelationVal}')

                #Find the col or row value that has more correlation on the Target. Dro
                if rowCorrelationVal > colCorrelationVal :
                    #print(f'Will be droping the feature {itemCol.values[0]}')
                    if ((colIndx+1) not in dropColumns) :
                        dropColumns.append(colIndx+1) # +1 because in urlsDF the first 

dropColumns
```

Out[10]:
```
[5, 8, 9, 10, 11, 12, 14, 17, 18, 22, 23]
```

In [11]:
```
#Step 4. Elimate the least correlated columns from DF

urlsDF.drop(urlsDF.columns[dropColumns], axis=1, inplace=True)
```

In [12]:
```
print(f"Number of samples are {urlsDF.shape}")

#DF is ready now for building the model
```

```
Number of samples are (11055, 21)
```

In [13]:
```
#Split the DF into X and Y
x = urlsDF.iloc[:,1:-1].values
x.shape
```

Out[13]:
```
(11055, 19)
```

In [14]:
```
y = urlsDF.iloc[:,-1:].values
y.shape
```

Out[14]:
```
(11055, 1)
```

In [15]:
```
# split x and y into training and testing sets
```

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=123)
```

Task is to build classification models using a binary classifier to detect malicious or phishing URLs.

Here am using Logistic regression and KNN and will be comparing the models to infer which model is a better one.

Step 1 : Create and logistic regression model

Step 2 : Create a KNN model

In [16]:

```
#Step 1 : Create and logistic regression model

from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(X_train, y_train.ravel())

y_pred=logreg.predict(X_test)
```

In [17]:

```
# Import the necessary modules
from sklearn.metrics import confusion_matrix, classification_report

print(confusion_matrix(y_test, y_pred))

print(classification_report(y_test, y_pred))
```

```
[[1011  262]
 [ 270 1221]]
              precision    recall  f1-score   support

          -1       0.79      0.79      0.79      1273
           1       0.82      0.82      0.82      1491

    accuracy                           0.81      2764
   macro avg       0.81      0.81      0.81      2764
weighted avg       0.81      0.81      0.81      2764
```

In [18]:

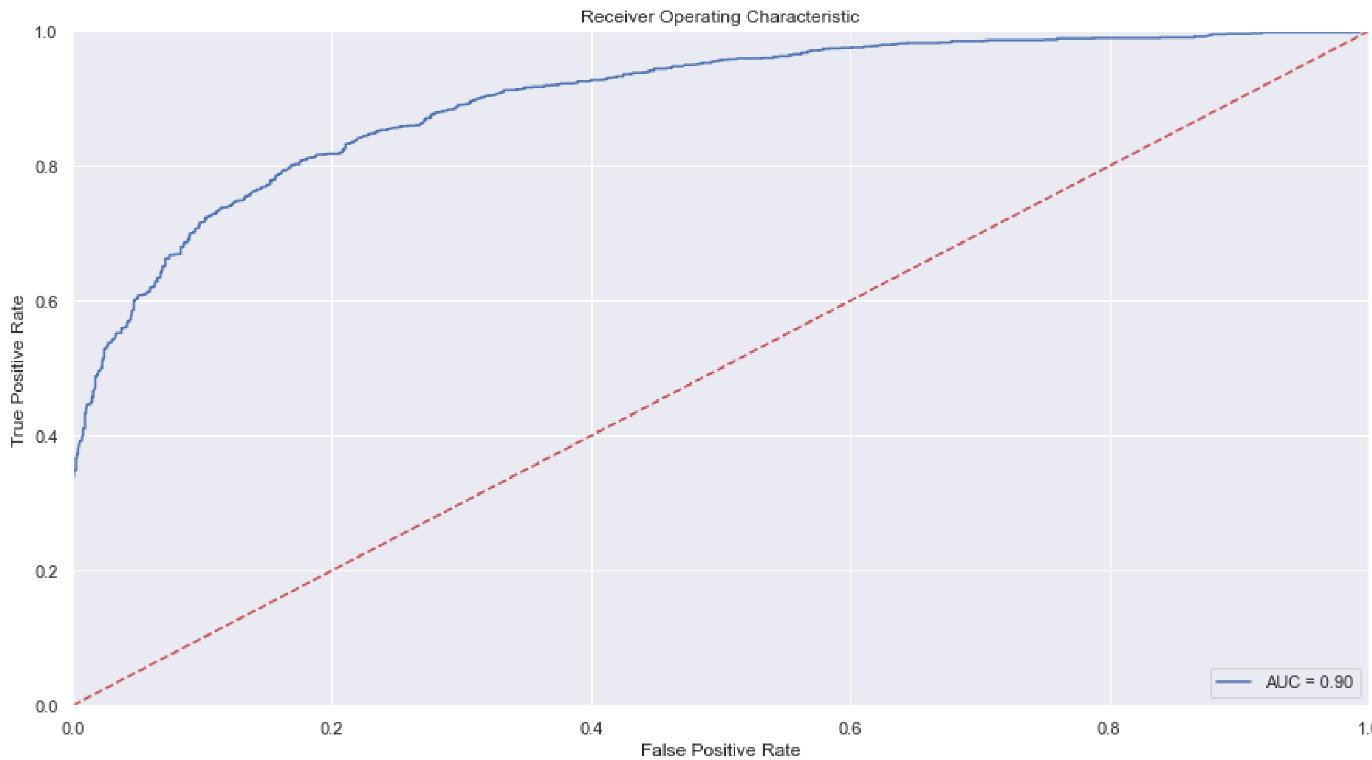
```
#Plot the ROC with AUC after calculating the same

import sklearn.metrics as metrics

# calculate the fpr and tpr for all thresholds of the classification
probabilities = logreg.predict_proba(X_test)
predicts = probabilities[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, predicts)
aucMetric = metrics.auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % aucMetric)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
```

```
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [19]:

```
#Step 2 : Create a KNN model
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)
# Here I tried a few values for the hyper param n like 3, 5, 8, 13 and then zeroed in on
# Better way is to tune the param to get the optimal value by using methods like Grid Se

knn.fit(X_train, y_train.ravel())
y_pred = knn.predict(X_test)

knn.score(X_test, y_test)
```

Out[19]:

```
0.877713458755427
```

In [20]:

```
print(confusion_matrix(y_test, y_pred))

print(classification_report(y_test, y_pred))
```

```
[[1113  160]
 [ 178 1313]]
              precision    recall  f1-score   support

          -1       0.86      0.87      0.87      1273
           1       0.89      0.88      0.89      1491

    accuracy                           0.88      2764
   macro avg       0.88      0.88      0.88      2764
weighted avg       0.88      0.88      0.88      2764
```

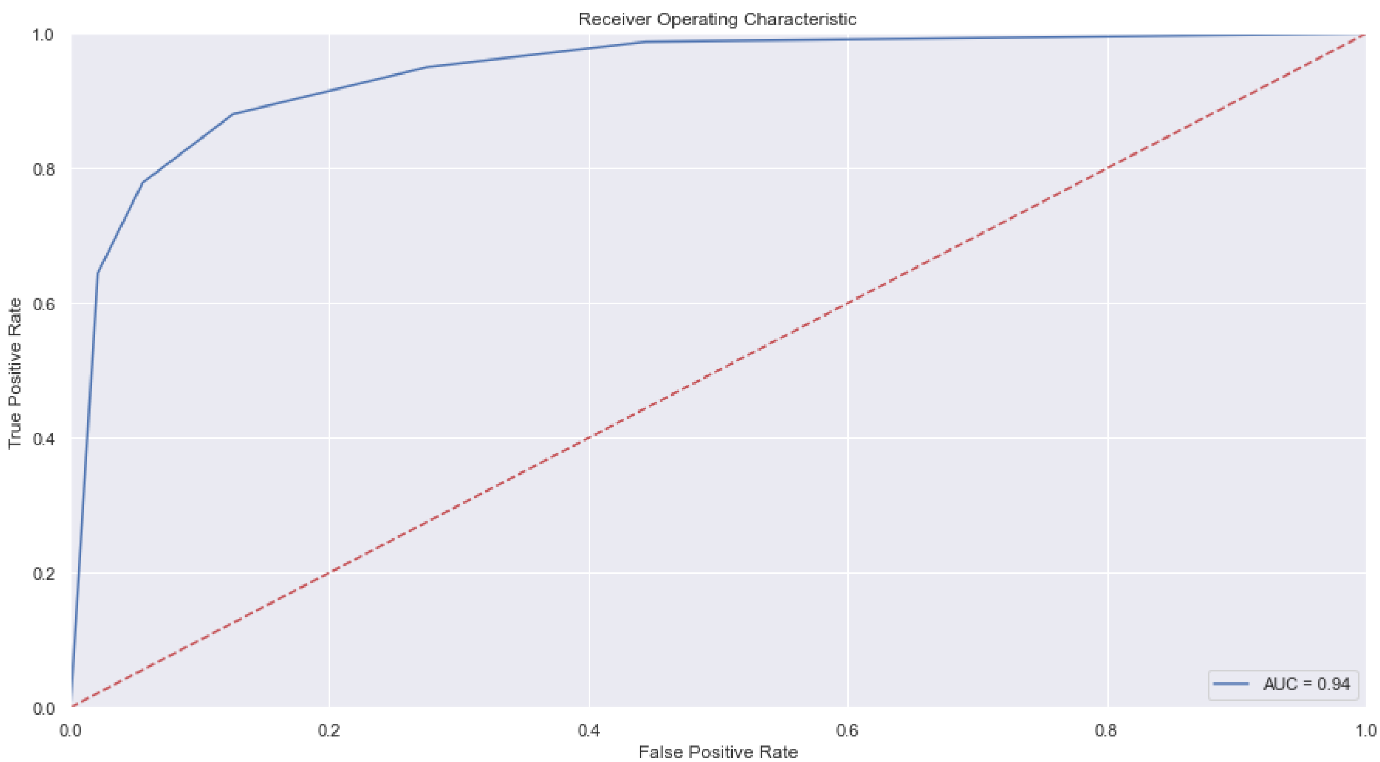
In [21]:

```
#Plot the ROC with AUC after calculating the same

import sklearn.metrics as metrics
```

```
# calculate the fpr and tpr for all thresholds of the classification
probabilities = knn.predict_proba(X_test)
predicts = probabilities[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, predicts)
aucMetric = metrics.auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % aucMetric)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



ROC (Receiver Operating Characterics) and AUC (Area Under The Curve) are the main evaluating metrics of classification models.

The above 2 ROC with AUC plot are for the Logistic Regression model and KNN model accordingly.

Higher the AUC better is the model at predicting the correct values. The ROC curve is plotted with TPR (True Positive Rate) on the y-axiz against the FPR (False Positive Rate) on the x-axis.A perfect model has AUC 1. Closer the AUC is to 1, better the model is.

AUC in case of Logistic regression is 0.90 and in case of KNN it is 0.94. From the ROC and AUC it is evident that KNN is a better model.

In [22]:

```
#Validate the accuracy of data by the K-Fold cross-validation technique.

from sklearn.model_selection import cross_val_score
import numpy as np

cv_results = cross_val_score(logreg, x, y.ravel(), cv=10)
#print(cv_results)
```

```
print('For logestic regression model - CV accuracy: %.3f +/- %.3f' % (np.mean(cv_results

cv_results = cross_val_score(knn, x, y.ravel(), cv=10)

print('For KNN model - CV accuracy: %.3f +/- %.3f' % (np.mean(cv_results),np.std(cv_resu
```

```
For logestic regression model - CV accuracy: 0.806 +/- 0.028
For KNN model - CV accuracy: 0.872 +/- 0.023
```

The value of K = 10 is considered above as it is the standard value of K, generally used. But if the data set size is too huge then we consider K value as 5. In this case a K value of 10 is suitable.

From the above print statement it can be seen that the KNN model is having a higher accuracy as compared to the Logistic regression model.

Recommendation as per the above details is to go by the KNN model