Machine Learning Phishing Project
Phases 1 to 5

Roger Asquith / rsa5249@psu.edu / rogersamudaasquith@gmail.com

Jayvian Pettit /jup625@psu.edu

Phase 1: Data Preprocessing

```
from google.colab import files
uploaded = files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable it.
Saving Phishing_Legitimate_train_missing_data.csv to Phishing_Legitimate_train_missing_data.csv

In [2]:

```
import pandas as pd
import numpy as np
df = pd.read_csv('/content/Phishing_Legitimate_train_missing_data.csv', index_col='id', na_values=['','','n/a'])
print(df)
```

```
      NumDots  SubdomainLevel  PathLevel  UrlLength  NumDash  \
id
1         3.0             1.0        5.0       81.0      1.0
2         2.0             0.0        5.0       78.0      1.0
3         3.0             0.0        4.0       53.0      1.0
4         3.0             1.0        6.0       68.0      0.0
5         3.0             0.0        3.0       61.0      0.0
...       ...             ...        ...        ...      ...
4996      3.0             1.0        1.0       67.0      3.0
4997      1.0             0.0        2.0       36.0      1.0
4998      3.0             2.0        0.0       33.0      0.0
4999      3.0             1.0        2.0       47.0      0.0
5000      1.0             0.0        2.0       37.0      0.0

      NumDashInHostname  AtSymbol  TildeSymbol  NumUnderscore  NumPercent  \
id
1                   1.0       0.0          0.0            1.0         0.0
2                   1.0       0.0          0.0            3.0         0.0
3                   0.0       0.0          0.0            0.0         0.0
4                   0.0       0.0          0.0            0.0         0.0
5                   0.0       0.0          0.0            0.0         0.0
```

```
...              ...      ...        ...          ...        ...
4996             0.0      0.0        0.0          0.0        0.0
4997             0.0      0.0        0.0          0.0        0.0
4998             0.0      0.0        0.0          0.0        0.0
4999             0.0      0.0        0.0          0.0        0.0
5000             0.0      0.0        0.0          0.0        0.0

      ...  InsecureForms  RelativeFormAction  ExtFormAction  \
id    ...
1     ...          1.0                 0.0            0.0
2     ...          1.0                 1.0            0.0
3     ...          1.0                 0.0            1.0
4     ...          1.0                 0.0            0.0
5     ...          1.0                 0.0            0.0
...   ...          ...                 ...            ...
4996  ...          1.0                 0.0            0.0
4997  ...          0.0                 0.0            0.0
4998  ...          1.0                 0.0            1.0
4999  ...          1.0                 1.0            0.0
5000  ...          1.0                 0.0            0.0

      AbnormalFormAction  RightClickDisabled  PopUpWindow  IframeOrFrame  \
id
1                    0.0                 0.0          0.0            0.0
2                    0.0                 0.0          0.0            0.0
3                    0.0                 0.0          0.0            1.0
4                    0.0                 0.0          0.0            0.0
5                    0.0                 0.0          0.0            1.0
...                  ...                 ...          ...            ...
4996                 0.0                 0.0          0.0            1.0
4997                 0.0                 0.0          0.0            1.0
4998                 0.0                 0.0          0.0            1.0
4999                 0.0                 0.0          0.0            0.0
5000                 0.0                 0.0          0.0            1.0

      MissingTitle  ImagesOnlyInForm  CLASS_LABEL
id
1              0.0               0.0            0
2              0.0               0.0            1
3              0.0               0.0            1
4              0.0               0.0            1
5              0.0               0.0            1
...            ...               ...          ...
4996           0.0               0.0            0
4997           0.0               0.0            0
4998           0.0               0.0            0
4999           1.0               0.0            1
5000           0.0               0.0            1
```

[5000 rows x 38 columns]

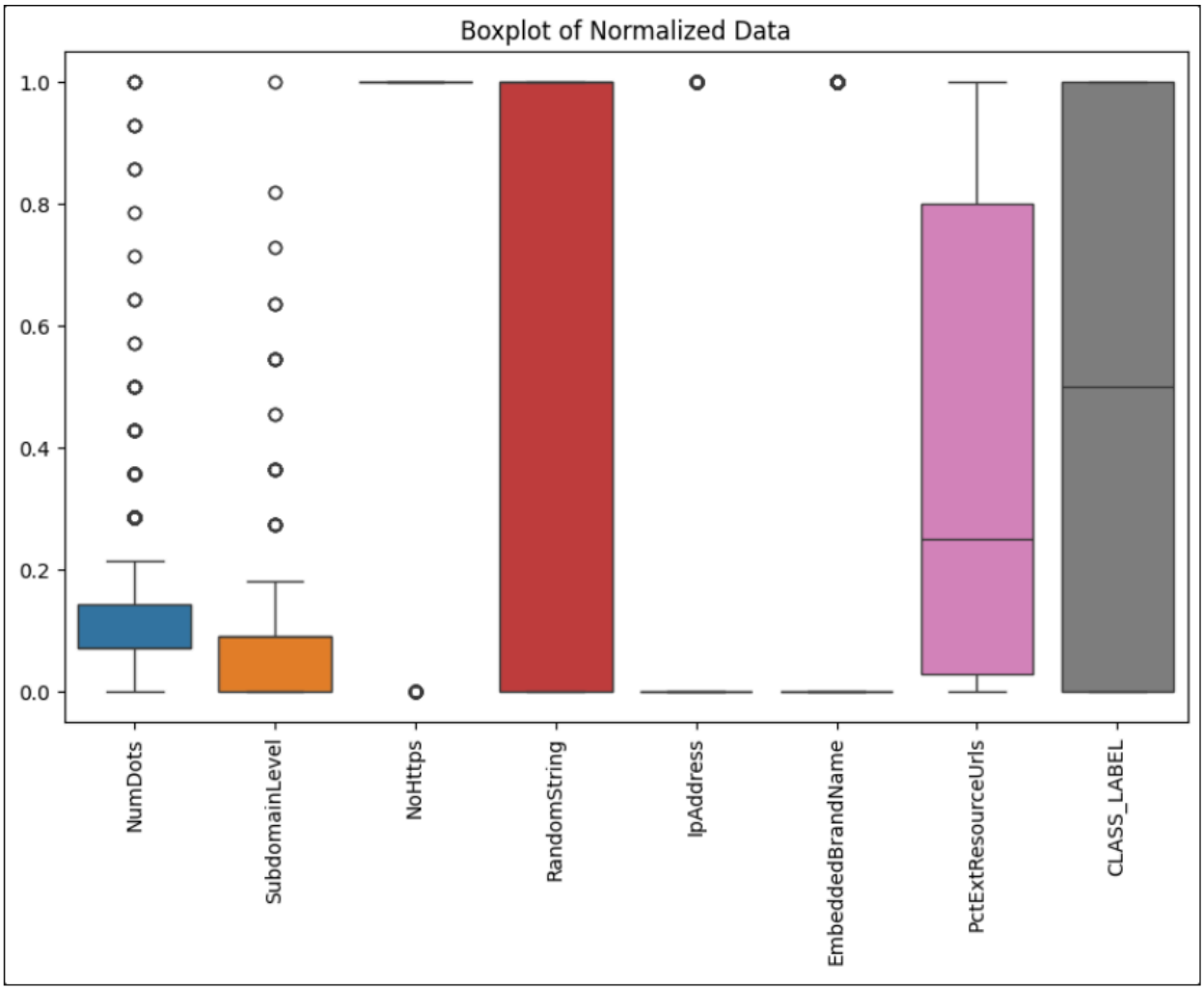
```
# Adding all of the libraries so don't have any problems working on the data cleaning part
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, LabelEncoder

#Normalization
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df_normalized = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
print(df_normalized)
```

```
      NumDots  SubdomainLevel  NoHttps  RandomString  IpAddress  \
0    0.142857        0.090909      1.0           1.0        0.0
1    0.071429        0.000000      1.0           1.0        0.0
2    0.142857        0.000000      1.0           0.0        0.0
3    0.142857        0.090909      1.0           0.0        0.0
4    0.142857        0.000000      1.0           1.0        0.0
...       ...             ...      ...           ...        ...
4995 0.142857        0.090909      1.0           0.0        0.0
4996 0.000000        0.000000      1.0           0.0        0.0
4997 0.142857        0.181818      1.0           0.0        0.0
4998 0.142857        0.090909      1.0           0.0        0.0
4999 0.000000        0.000000      1.0           1.0        0.0

      EmbeddedBrandName  PctExtResourceUrls  CLASS_LABEL
0                   0.0            0.000000          0.0
1                   0.0            1.000000          1.0
2                   0.0            1.000000          1.0
3                   1.0            1.000000          1.0
4                   0.0            0.285714          1.0
...                 ...                 ...          ...
4995                0.0            0.674419          0.0
4996                0.0            0.352941          0.0
4997                0.0            0.647059          0.0
4998                0.0            0.000000          1.0
4999                0.0            0.142857          1.0
```

[5000 rows x 8 columns]

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_normalized)
```

```
plt.title("Boxplot of Normalized Data")
plt.xticks(rotation=90)
plt.show()
```



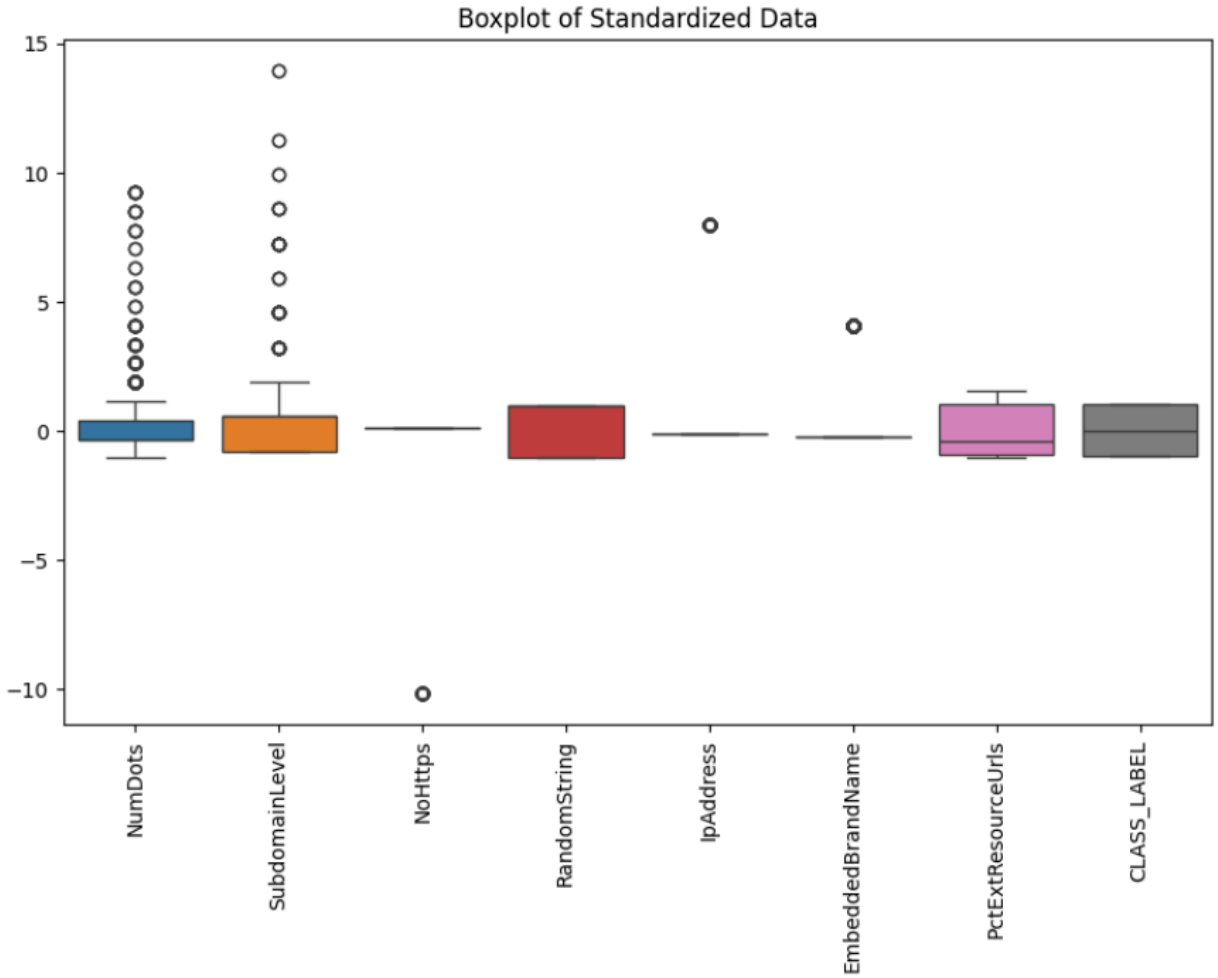

```
#Standardization
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
print(df_scaled)
```

```
    NumDots  SubdomainLevel   NoHttps  RandomString  IpAddress  \
0  0.407766        0.551384  0.098473      0.954993  -0.125089
1 -0.328127       -0.787959  0.098473      0.954993  -0.125089
2  0.407766       -0.787959  0.098473     -1.047128  -0.125089
3  0.407766        0.551384  0.098473     -1.047128  -0.125089
4  0.407766       -0.787959  0.098473      0.954993  -0.125089
```

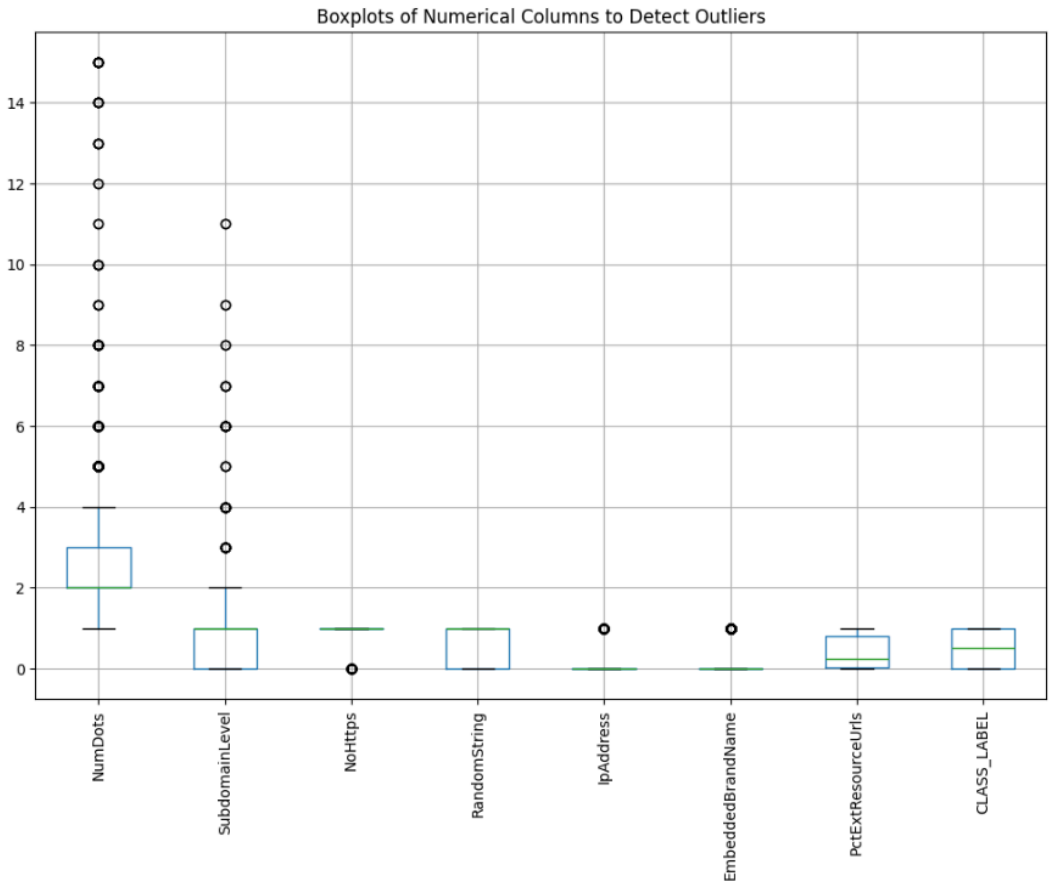
```
...       ...           ...       ...          ...       ...
4995  0.407766       0.551384  0.098473    -1.047128  -0.125089
4996 -1.064019      -0.787959  0.098473    -1.047128  -0.125089
4997  0.407766       1.890727  0.098473    -1.047128  -0.125089
4998  0.407766       0.551384  0.098473    -1.047128  -0.125089
4999 -1.064019      -0.787959  0.098473     0.954993  -0.125089

     EmbeddedBrandName  PctExtResourceUrls  CLASS_LABEL
0            -0.246901           -1.019686         -1.0
1            -0.246901            1.539210          1.0
2            -0.246901            1.539210          1.0
3             4.050208            1.539210          1.0
4            -0.246901           -0.288573          1.0
...                ...                 ...          ...
4995         -0.246901            0.706081         -1.0
4996         -0.246901           -0.116546         -1.0
4997         -0.246901            0.636070         -1.0
4998         -0.246901           -1.019686          1.0
4999         -0.246901           -0.654129          1.0

[5000 rows x 8 columns]
```

In [11]:

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_scaled)
plt.title("Boxplot of Standardized Data")
plt.xticks(rotation=90)
plt.show()
```




```
# Showcases the  outliers using boxplots
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns
plt.figure(figsize=(12, 8))
df[numerical_cols].boxplot()
plt.xticks(rotation=90)
plt.title('Boxplots of Numerical Columns to Detect Outliers')
plt.show()
```




```
# This removes the outliers which cleans
#The Z-score method is used to solve this problem
from scipy.stats import z score
df = df[(np.abs(zscore(df[numerical_cols])) < 3).all(axis=1)]
print("The data set after removing the outliers is ", df.shape)
```

Dataset shape after removing outliers: (0, 8)

Phase 2 and 3: Exploratory Data Analysis & Feature Selection

Our Target

The data frame has 38 columns and 5000 rows. We wanted to split the data frame and see if we can predict anything with the Subdomain Level. So we can slowly understand how to detect a phishing or legitimate website.(The Subdomain Level is a dependent variable in the case study) The reason why we picked the Subdomain level as the variable to check is becasue phishing sites use extra subdomians to impersonate legitmate domains.

After spliting the frame. We noticed that 0.0 and 1.0 has the most counts are from 0.0. Now we need to figure out which one is legitimate and which one is a scam.

df['SubdomainLevel'].value_counts()

Out[ ]:

SubdomainLevel	count
0.0	2411
1.0	2396
2.0	139
4.0	15
3.0	14
6.0	11
7.0	4
5.0	3
11.0	2
8.0	2
9.0	2

dtype: int64

We wanted to check the values within 0.0. Dropping all of the other values so it can be easier to understand.

#df = df.loc[df['SubdomainLevel']== 0.0]

#df['SubdomainLevel'].value_counts()
#It returns 0 - 2411

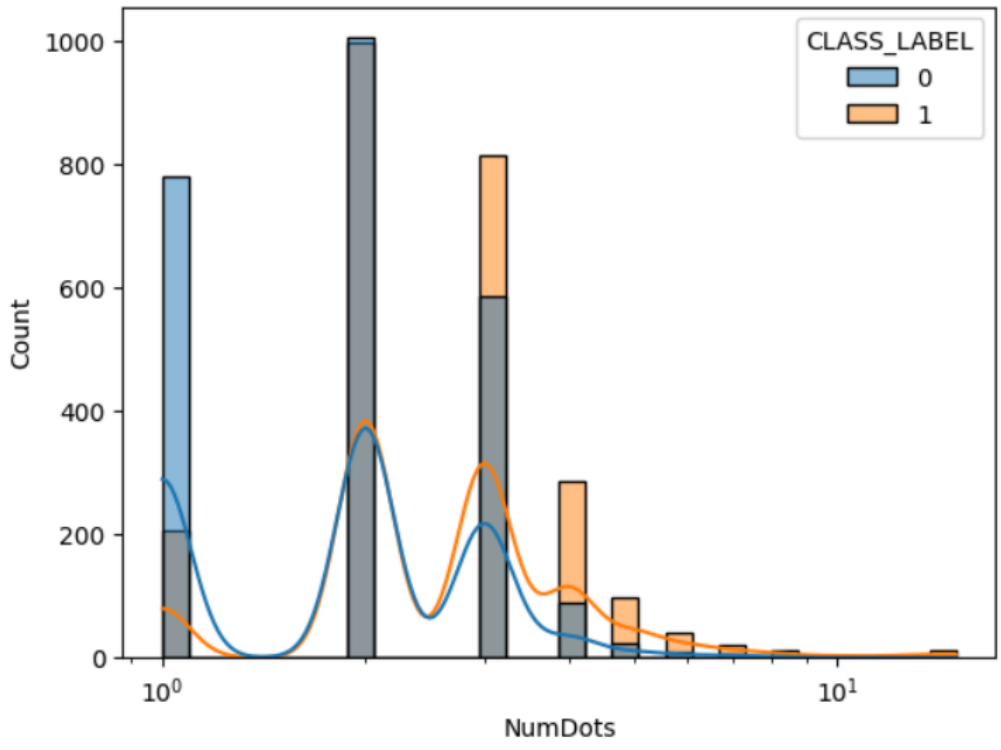
Histograms

After creating Histograms for the other features we used in phase 0/1 we learned that NumDots was the only one that could be properly plotted using a histogram. So we are going to use it as a stepping point for the other graphs.

In [ ]:
import seaborn **as** sns
sns.histplot(x='NumDots', bins=30, kde=**True**,log_scale=**True**, hue = 'CLASS_LABEL', data=df)

Out[ ]:

<Axes: xlabel='NumDots', ylabel='Count'>

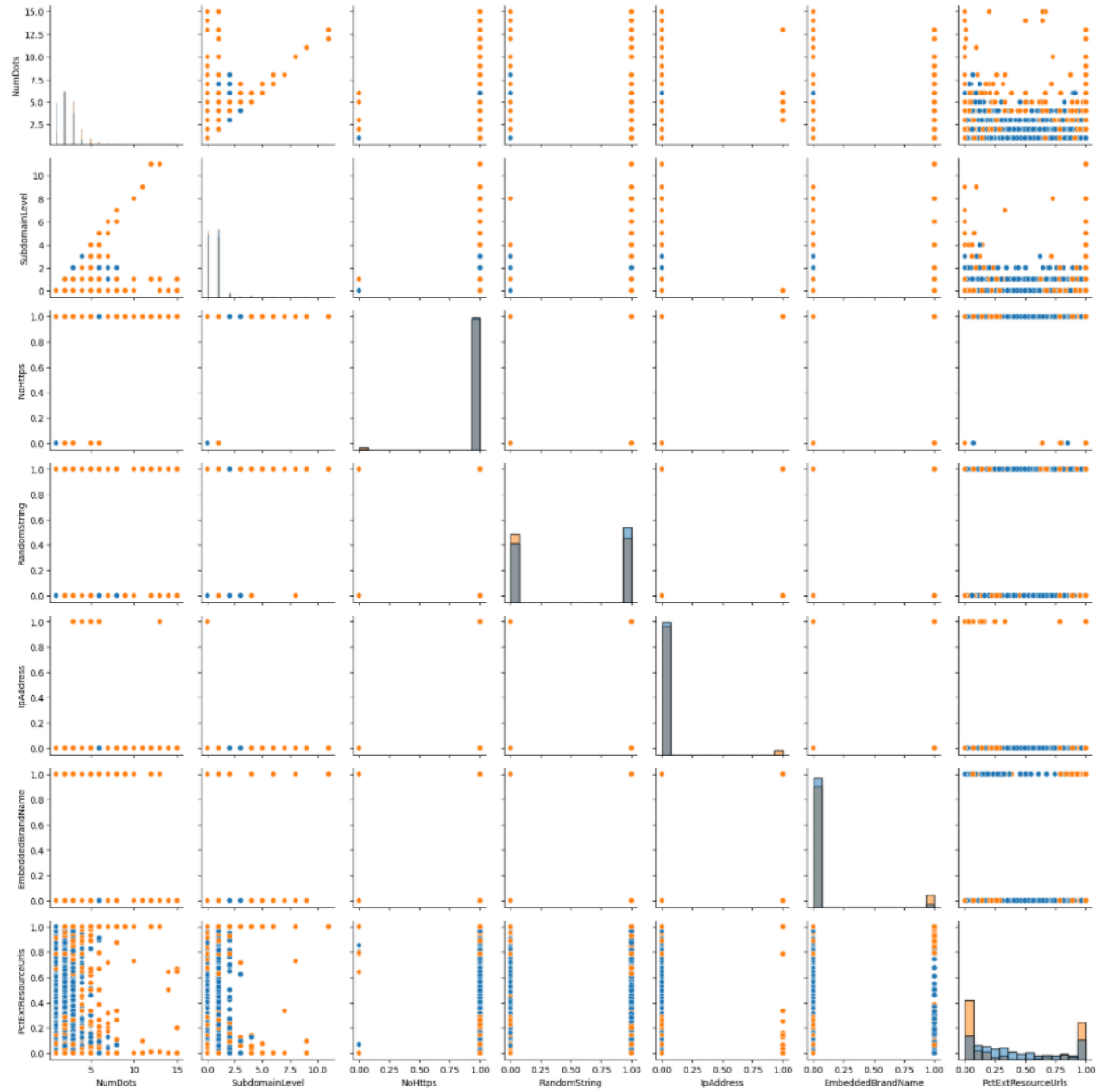


Visualization

#We are looking to see if there are any correlations between the 7 features we picked for phase 0/1
import seaborn as sns

```
columns_to_plot=[ 'CLASS_LABEL','NumDots','SubdomainLevel','NoHttps', 'RandomString','IpAddress',
'EmbeddedBrandName', 'PctExtResourceUrls']
g=sns.PairGrid(df[columns_to_plot], hue="CLASS_LABEL")
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
```

```
<seaborn.axisgrid.PairGrid at 0x78ebee9e3b50>
```



Based on the information we receive from checking the correlation we found out that no value correlates with each other which proves that we can include them on the same model.

We have decided to now check the correlation between columns under the same name. So Nums, Domains, Path/Level, Length, Symbol, HTTPS, and HostName will be checked for visualization.

#Nums

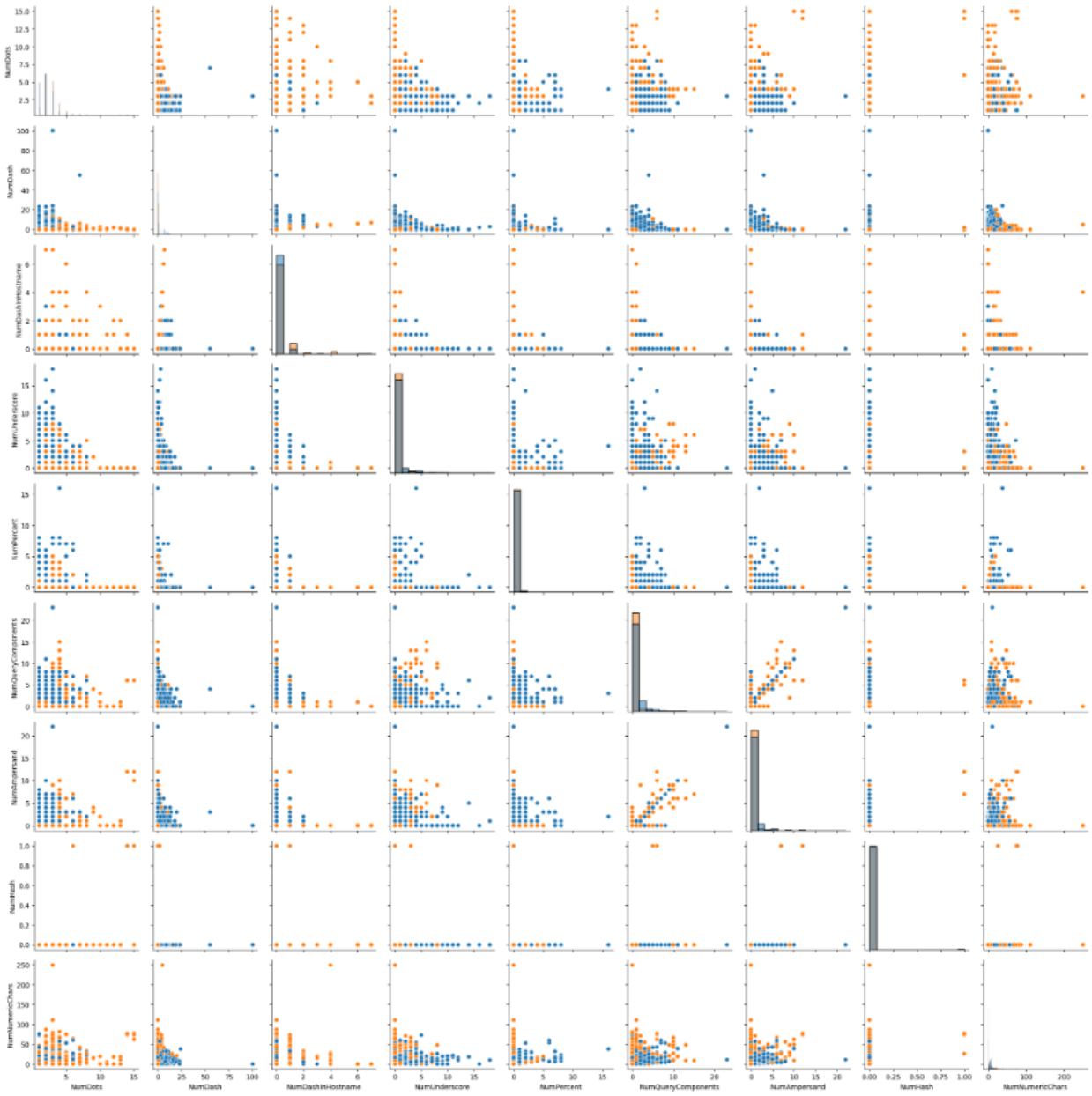
columns_to_plot=['CLASS_LABEL','NumDots','NumDash','NumDashInHostname','NumUnderscore','NumPercent','NumQueryComponents','NumAmpersand','NumHash','NumNumericChars']

g=sns.PairGrid(df[columns_to_plot], hue="CLASS_LABEL")

g.map_diag(sns.histplot)

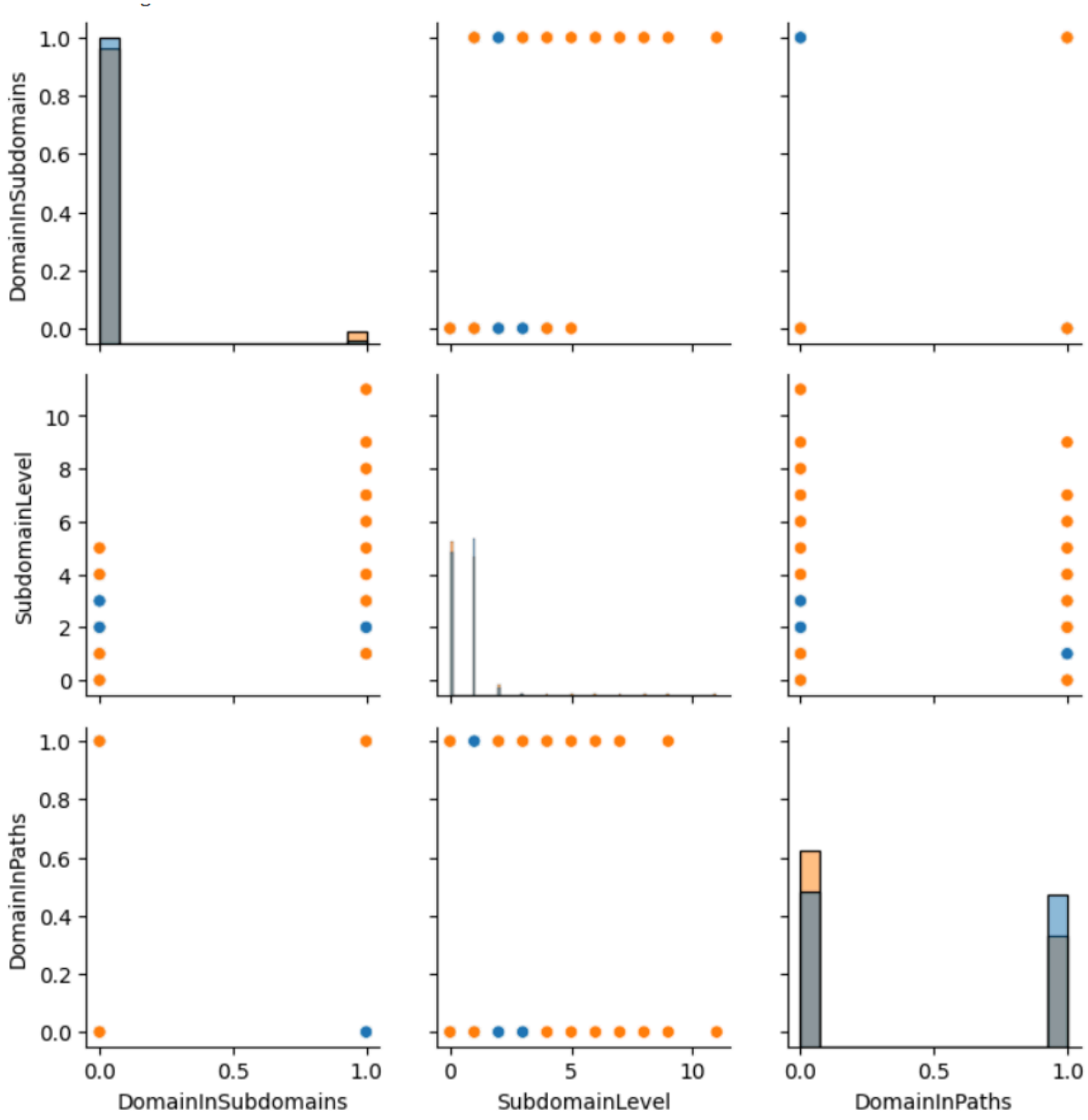
g.map_offdiag(sns.scatterplot)

Out[ ]:



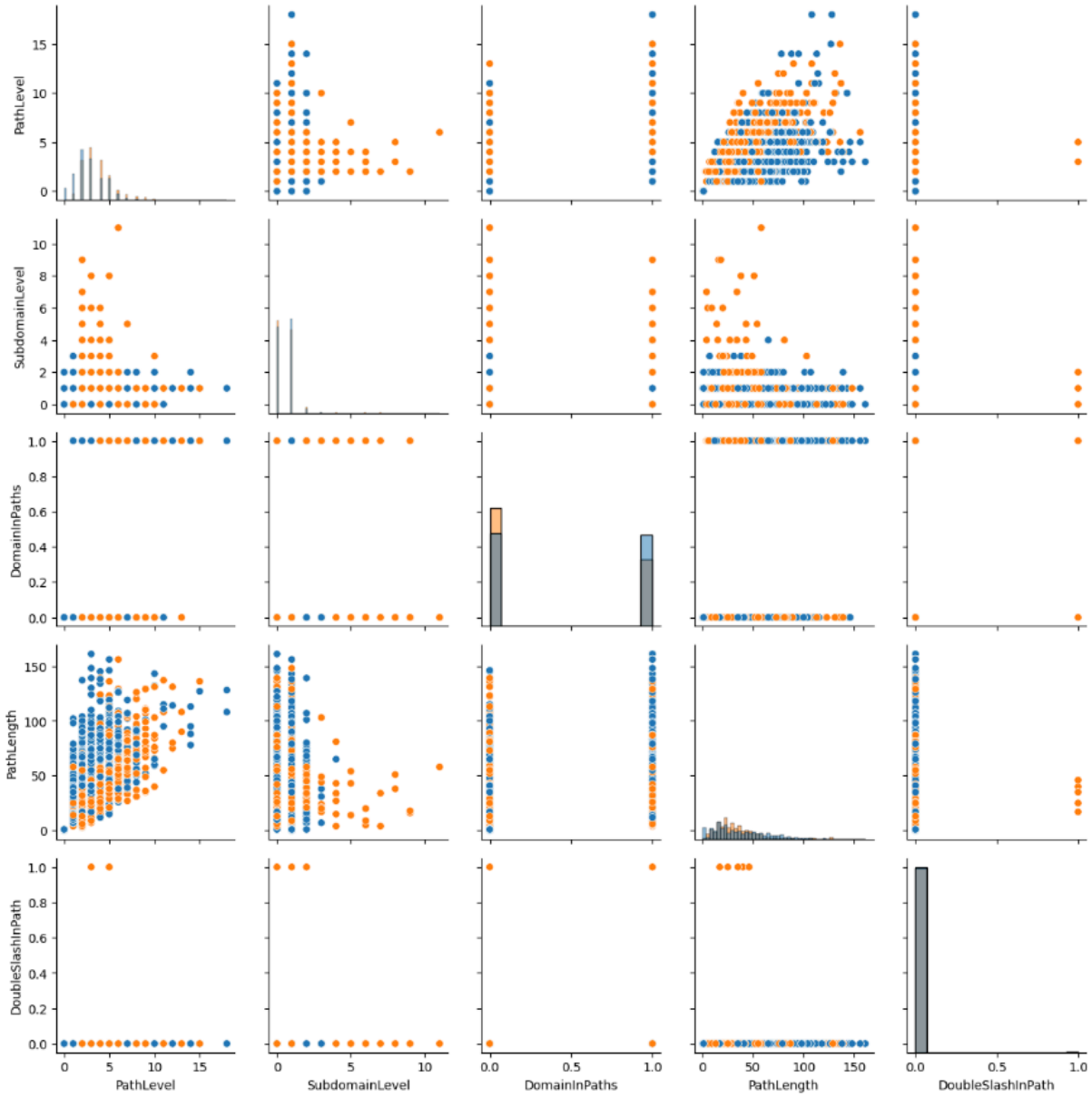
#Domains

```
columns_to_plot=['CLASS_LABEL','DomainInSubdomains','SubdomainLevel','DomainInPaths']
g=sns.PairGrid(df[columns_to_plot], hue="CLASS_LABEL")
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
```

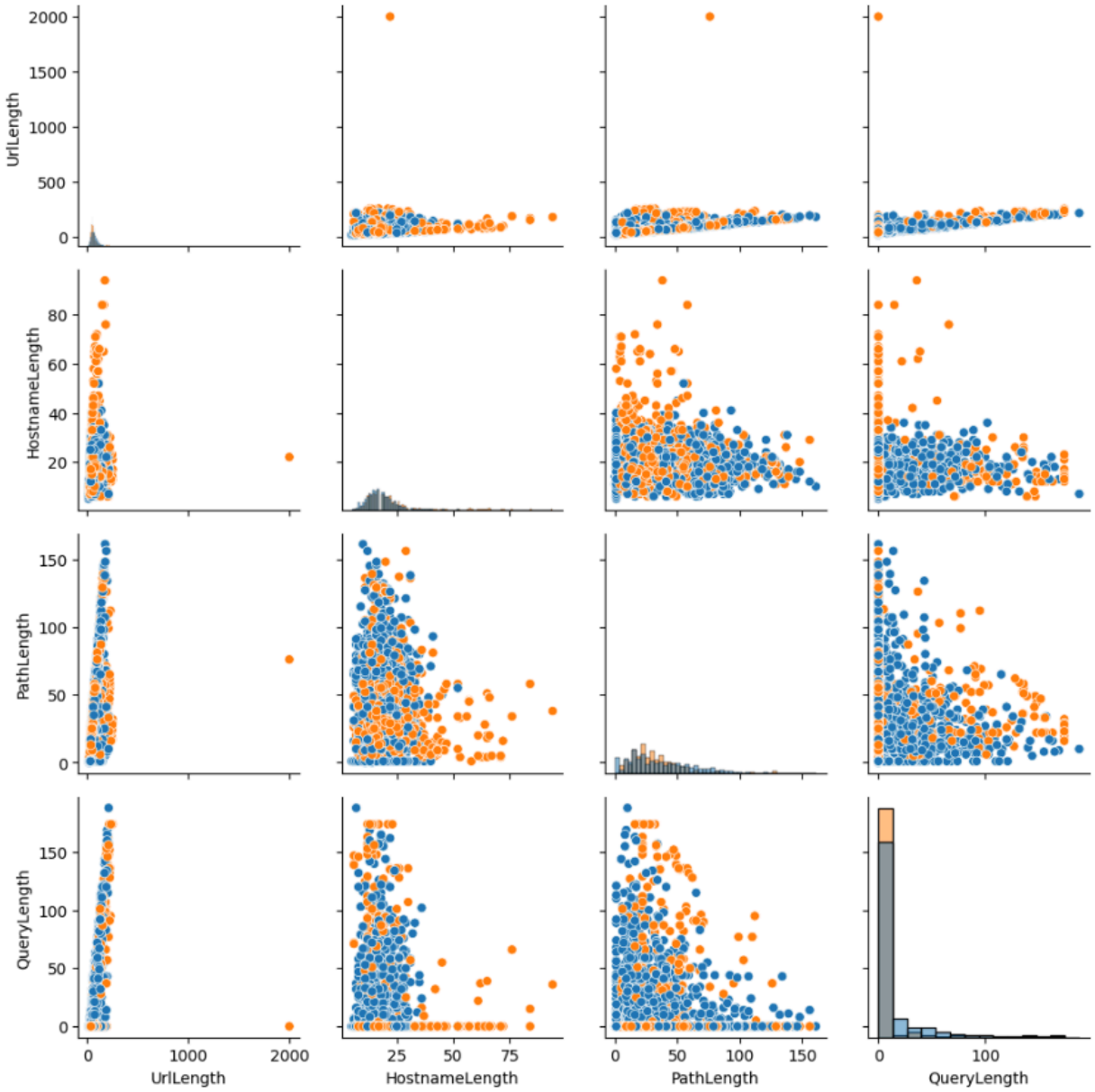


#Path/Level

```
columns_to_plot=['CLASS_LABEL','PathLevel','SubdomainLevel','DomainInPaths','PathLength','DoubleSl
ashInPath']
g=sns.PairGrid(df[columns_to_plot], hue="CLASS_LABEL")
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
```

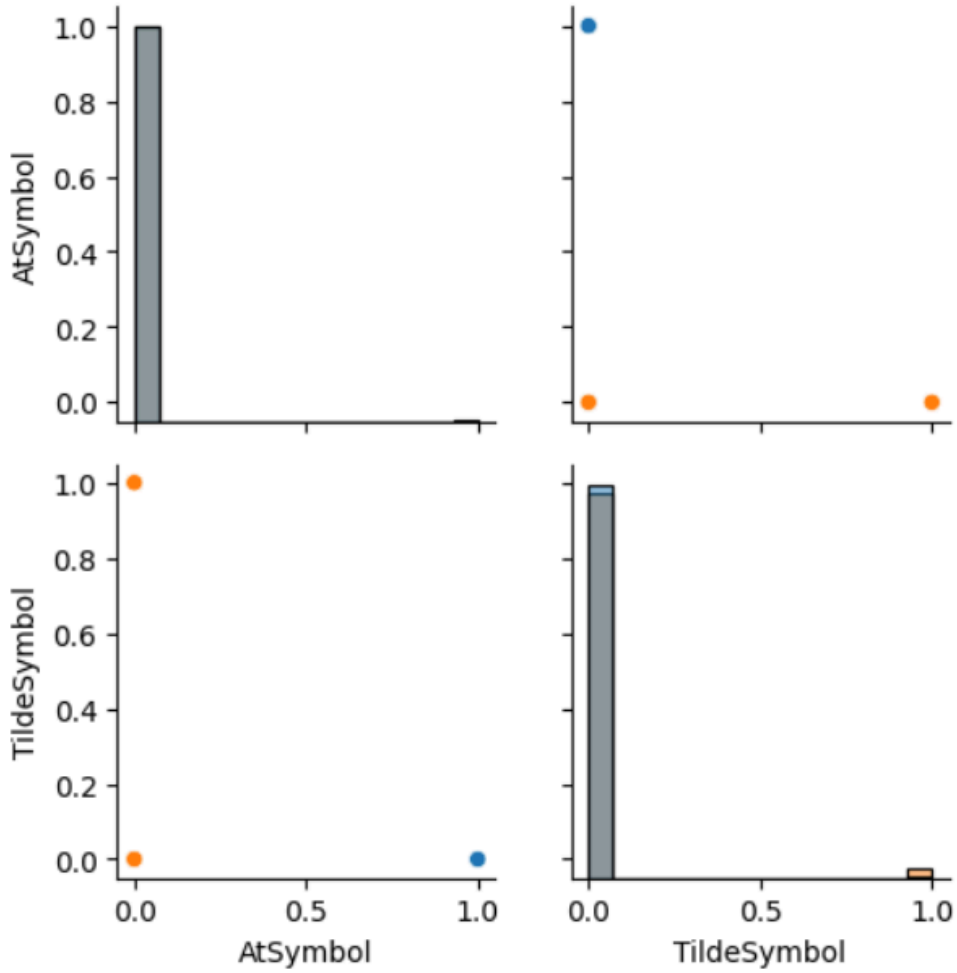



```
#Lengths
columns_to_plot=['CLASS_LABEL','UrlLength','HostnameLength','PathLength','QueryLength']
g=sns.PairGrid(df[columns_to_plot], hue="CLASS_LABEL")
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
```

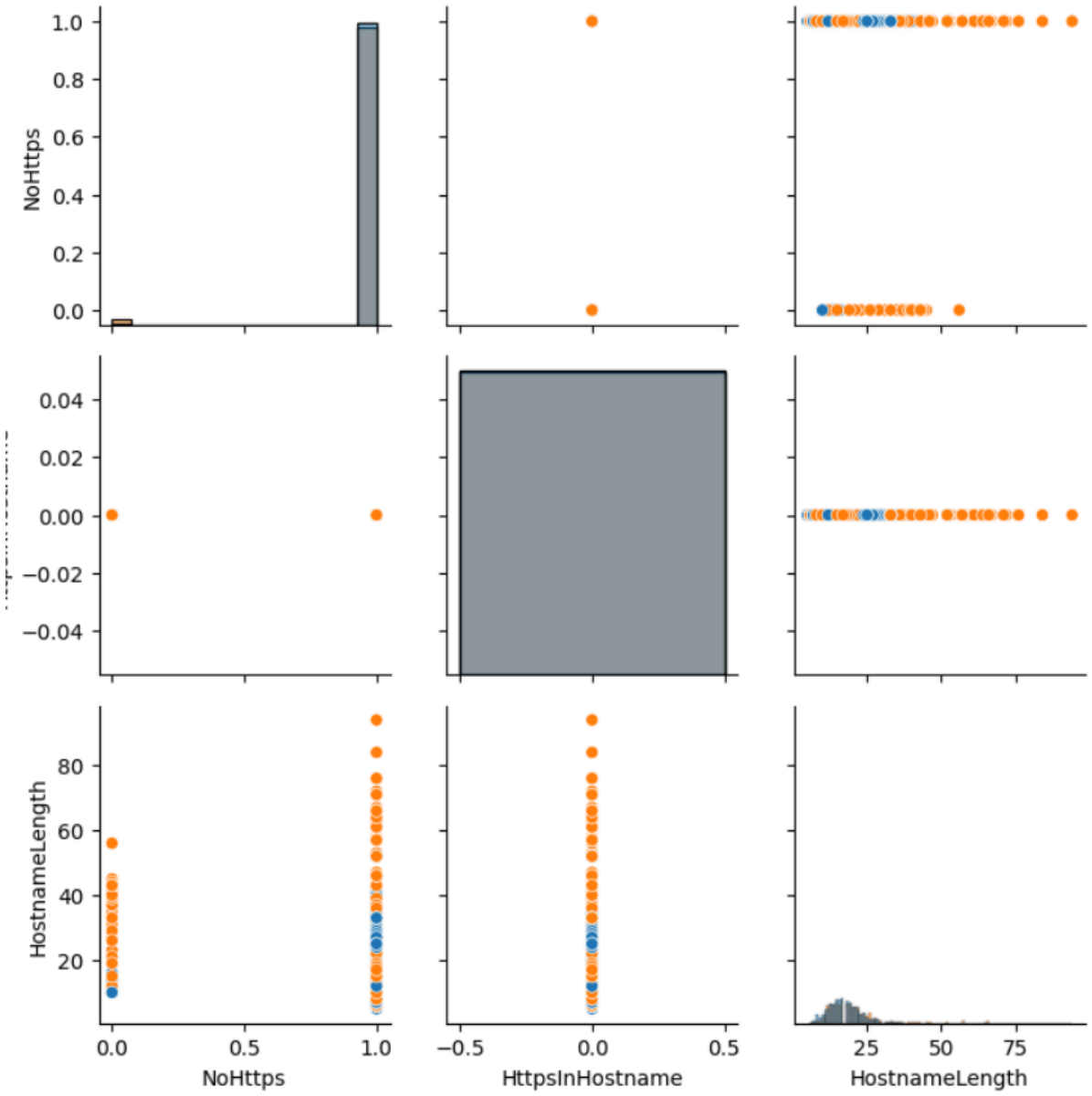


#Symbol

```
columns_to_plot=['CLASS_LABEL','AtSymbol','TildeSymbol']
g=sns.PairGrid(df[columns_to_plot], hue="CLASS_LABEL")
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
```

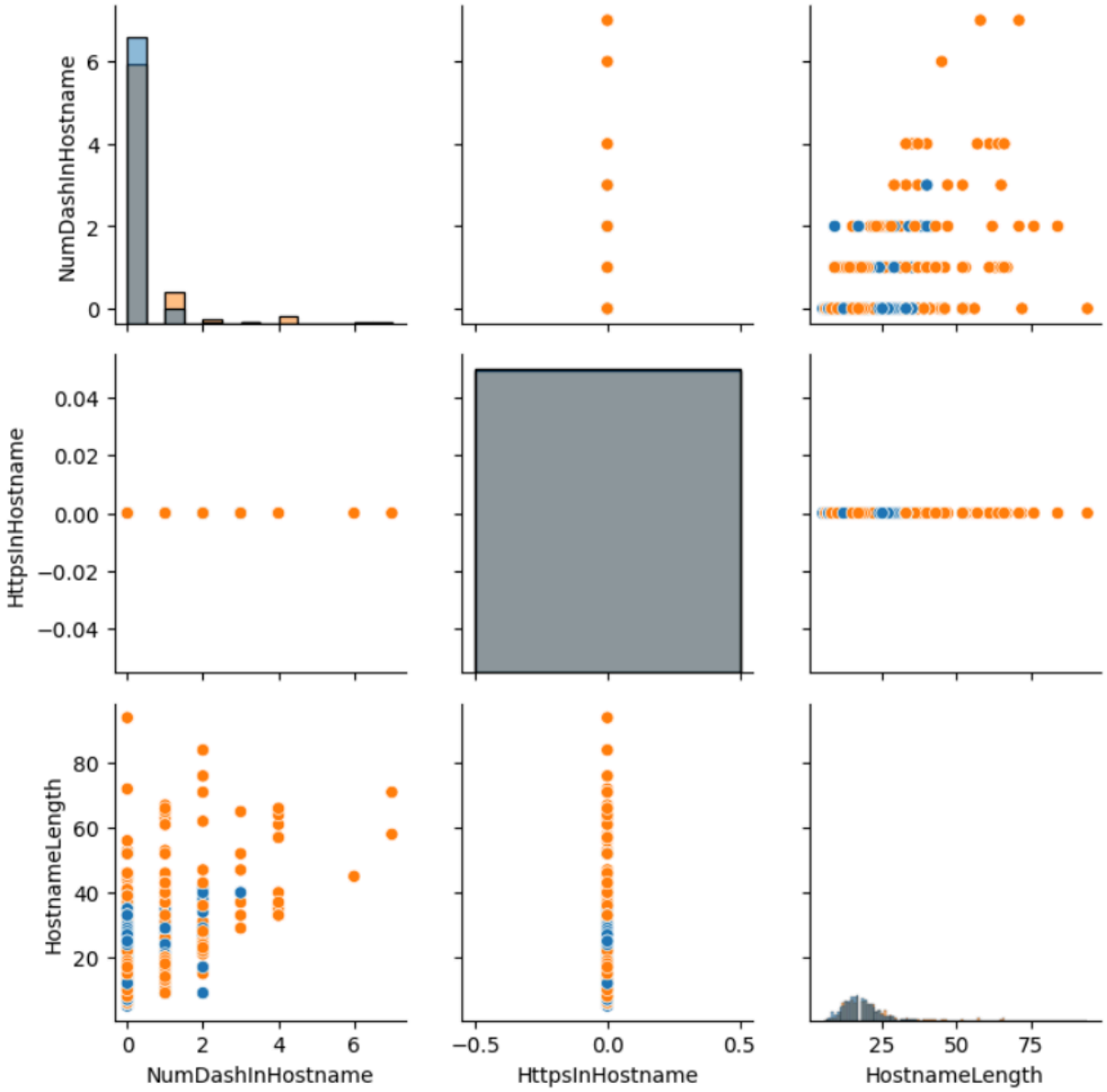



```
#HTTPs
columns_to_plot=['CLASS_LABEL','NoHttps','HttpsInHostname','HostnameLength' ]
g=sns.PairGrid(df[columns_to_plot], hue="CLASS_LABEL")
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
```



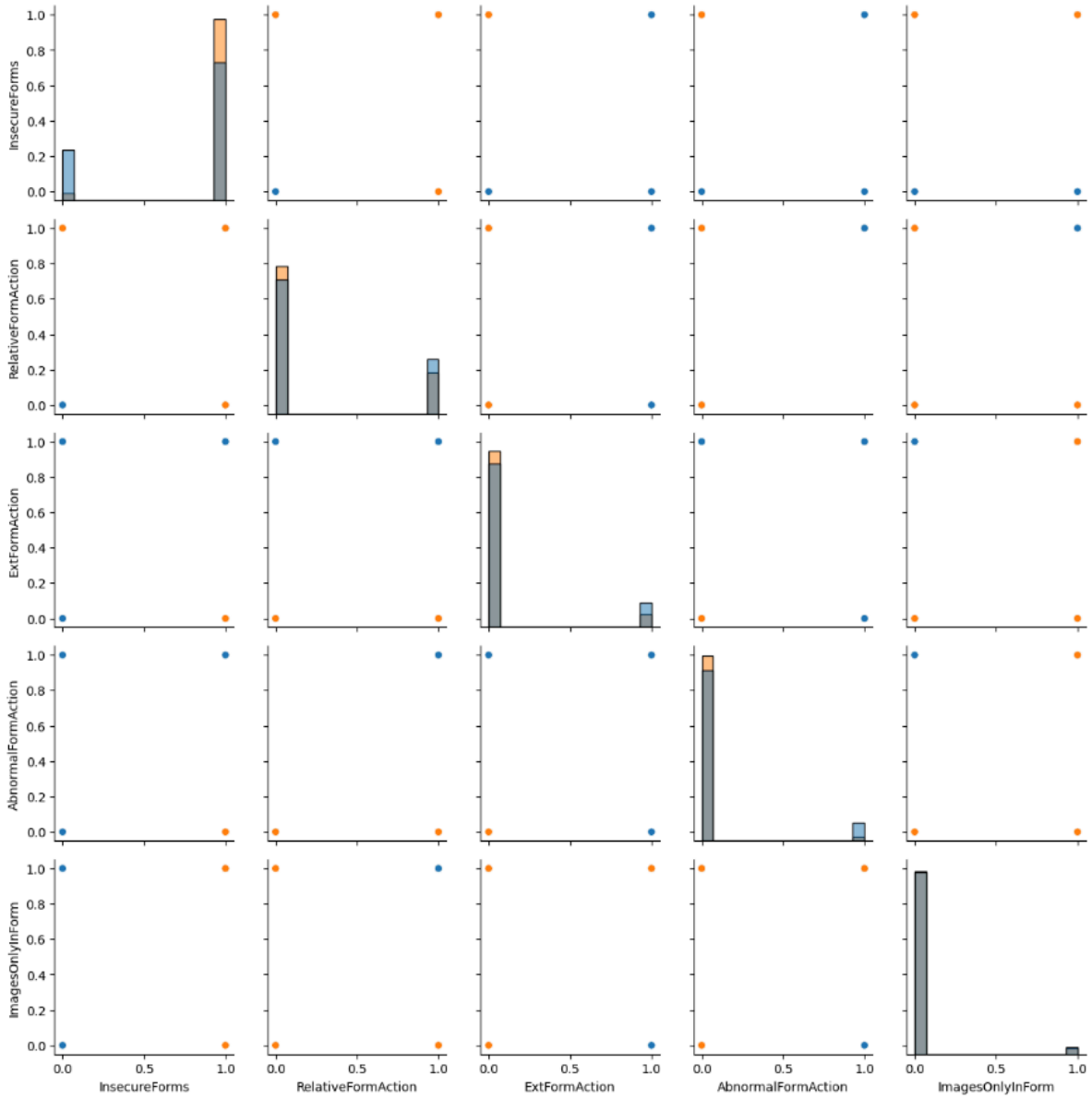
#HostName

```
columns_to_plot=['CLASS_LABEL','NumDashInHostname','HttpsInHostname', 'HostnameLength' ]
g=sns.PairGrid(df[columns_to_plot], hue="CLASS_LABEL")
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
```



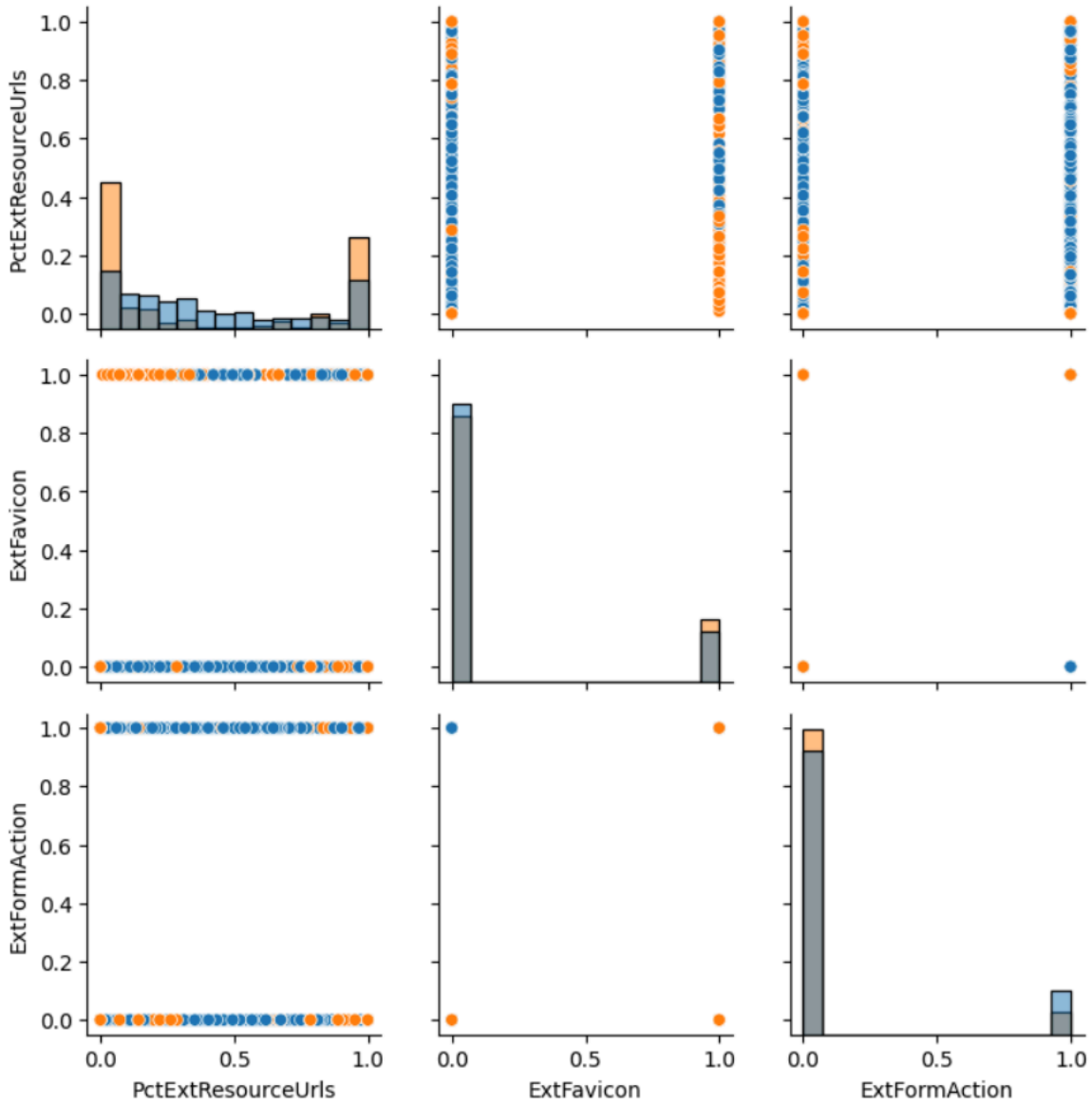
import seaborn **as** sns

In [ ]:
#Forms
columns_to_plot=['CLASS_LABEL','InsecureForms','RelativeFormAction','ExtFormAction','AbnormalFormAction','ImagesOnlyInForm']
g=sns.PairGrid(df[columns_to_plot], hue="CLASS_LABEL")
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)



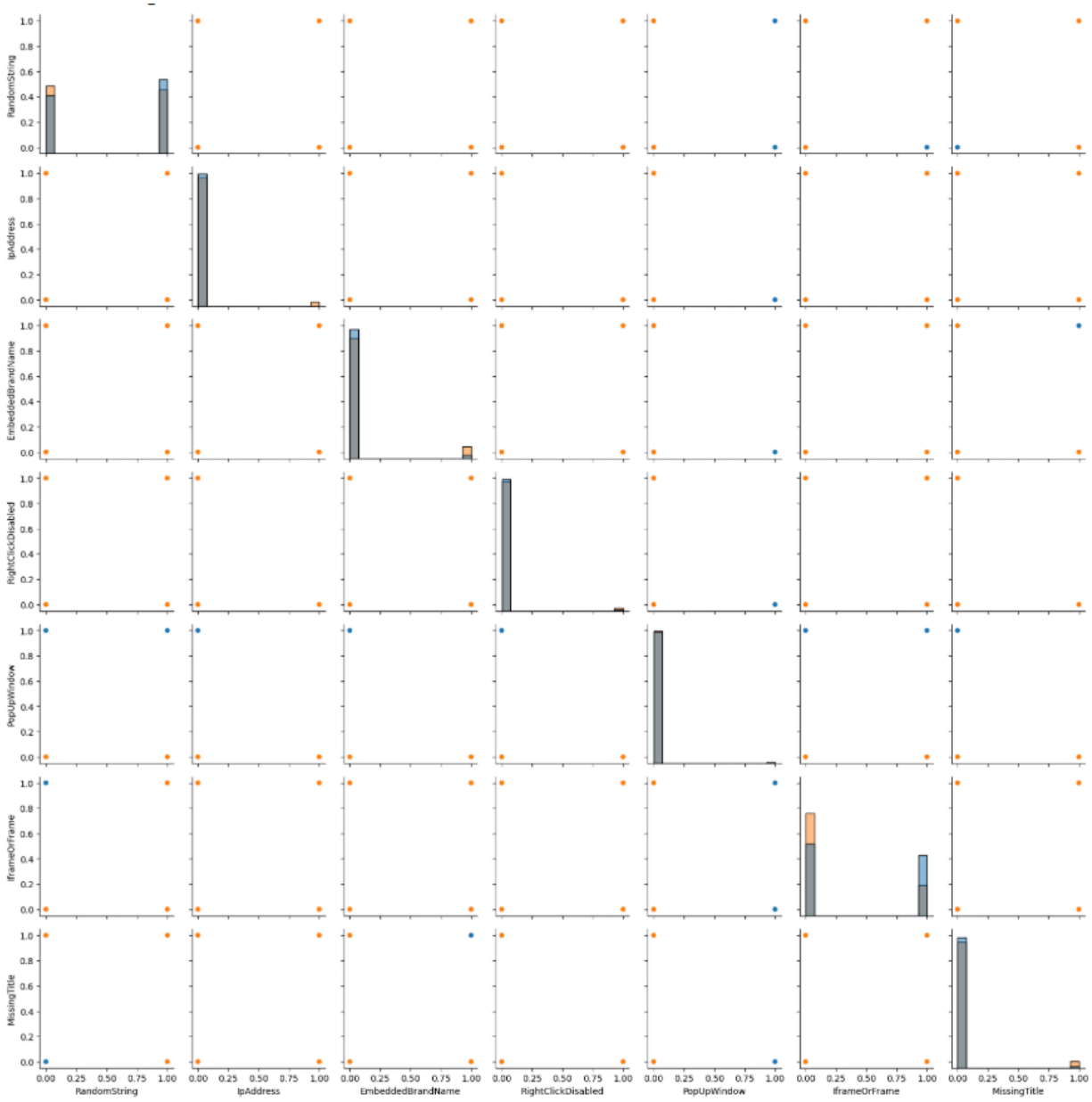
#Ext

```
columns_to_plot=['CLASS_LABEL','PctExtResourceUrls','ExtFavicon','ExtFormAction']
g=sns.PairGrid(df[columns_to_plot], hue="CLASS_LABEL")
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
```



#No phase correlation in words (Columns)

```
columns_to_plot=['CLASS_LABEL','RandomString','IpAddress','EmbeddedBrandName','RightClickDisabl
ed','PopUpWindow','IframeOrFrame','MissingTitle']
g=sns.PairGrid(df[columns_to_plot], hue="CLASS_LABEL")
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
```



Feature Selection

#All of the columns

from sklearn.feature_selection **import** RFE
from sklearn.linear_model **import** LogisticRegression
import pandas **as** pd

 # Define the columns to select
Num_C **=** (['NumDots', 'SubdomainLevel', 'PathLevel', 'UrlLength', 'NumDash', 'NumDashInHostname',

```
        'AtSymbol', 'TildeSymbol', 'NumUnderscore', 'NumPercent', 'NumQueryComponents',
'NumAmpersand', 'NumHash',
        'NumNumericChars', 'NoHttps', 'RandomString', 'IpAddress', 'DomainInSubdomains',
'DomainInPaths', 'HttpsInHostname',
        'HostnameLength', 'PathLength', 'QueryLength', 'DoubleSlashInPath', 'NumSensitiveWords',
'EmbeddedBrandName', 'PctExtResourceUrls',
        'ExtFavicon', 'InsecureForms', 'RelativeFormAction',
        'ExtFormAction', 'AbnormalFormAction', 'RightClickDisabled', 'PopUpWindow', 'IframeOrFrame',
'MissingTitle', 'ImagesOnlyInForm'])

df_cleaned = df.dropna(subset=Num_C + ["CLASS_LABEL"])

# Run RFE on data without missing values
rfe_selector = RFE(estimator=LogisticRegression(), n_features_to_select=7, step=1)
rfe_selector.fit(df_cleaned[Num_C], df_cleaned['CLASS_LABEL'])

# Print the selected columns
selected_columns = df_cleaned[Num_C].loc[:, rfe_selector.get_support()].columns
print(selected_columns)

Index(['NoHttps', 'IpAddress', 'NumSensitiveWords', 'InsecureForms',
       'AbnormalFormAction', 'PopUpWindow', 'MissingTitle'],
      dtype='object')
```

Plotting what we received from RFE

Overall conclusion:

Based on the given graphs from the rfe selected features and PairGrid there is relatively no correlation. We believe this is due to the features being predominantly numerical from 0 to 1. However, the 0 and 1 may indicate individual classes consisting of 0 and 1. 0 being not spam and 1 being spam. This may help us within the next phase to identify whether spam is legitimate or not. There are as well diagonal plots which are groupings of clusters that are now stored within a numerical barplot. Suggesting that the more values within 0 may not be spam while the values stored in 1 could be spam.

How do those charts collaborate with your final selected variables?

Our 7 selected variables: 'NumDots', 'SubdomainLevel', 'NoHttps', 'RandomString', 'IpAddress','EmbeddedBrandName', 'PctExtResourceUrls'

Given 7 from rfe: 'NoHttps', 'IpAddress', 'NumSensitiveWords', 'InsecureForms','AbnormalFormAction', 'PopUpWindow', 'MissingTitle'

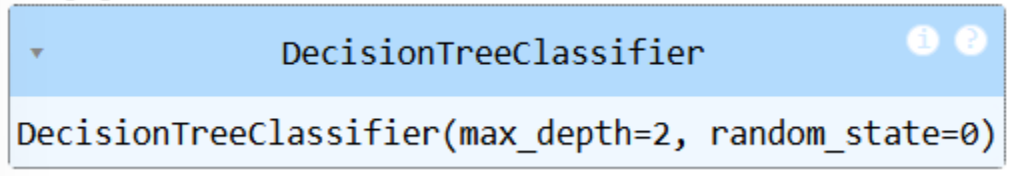
We have two selected variables that are the same as the given output of rfe. Those being NoHttps and IpAddress. These can help us determine if there is spam or not. This we have based off of the

visualization section. Within the visualization section NoHttps displays a small clustering within the barplot for not spam (0) and there being a high clustering for spam (1). Whereas with IpAddress the opposite can be said, the IpAddress may look legitimate, however, it could be utilized for spam. Even with being posed as legitimate.

Phase 4: Exploratory Data Analysis & Feature selection Using Decision Trees

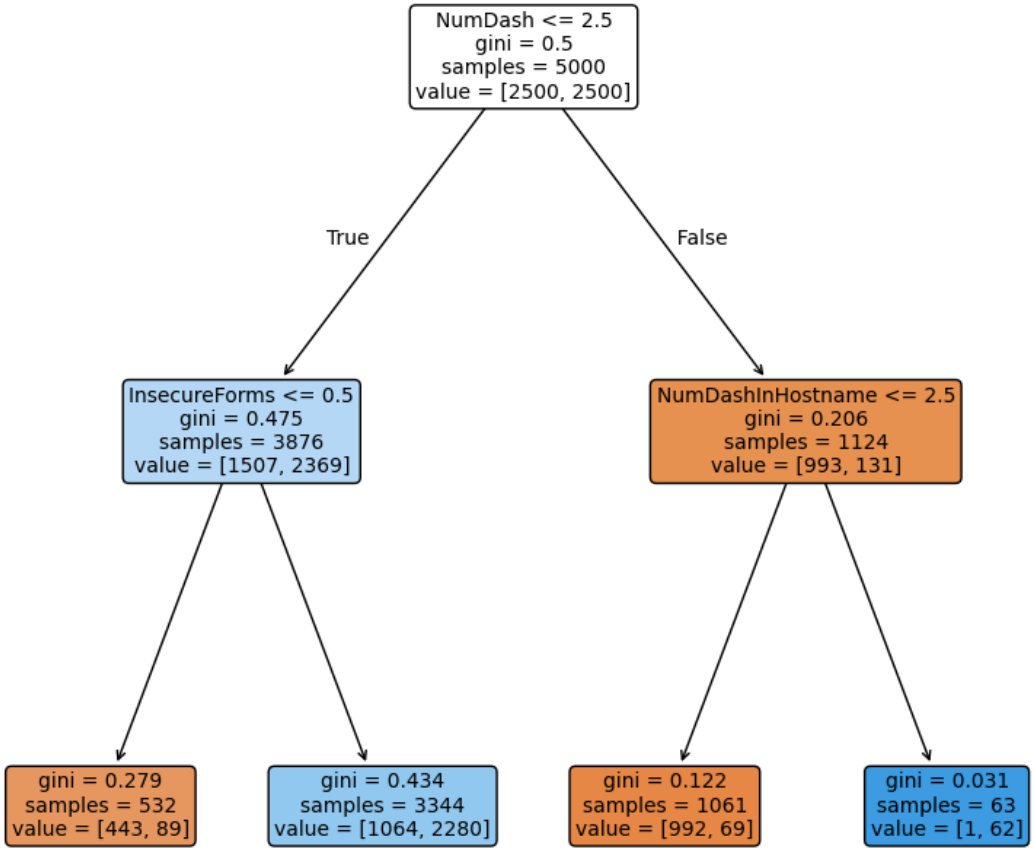
Part A:

```
#Tree
from re import X
from sklearn.tree import DecisionTreeClassifier
#define a tree model with maximum of two levels
clf = DecisionTreeClassifier(max_depth=2,random_state=0)
#train the model

#          X          Y
clf.fit(df[features], df['CLASS_LABEL'])
```




```
import matplotlib.pyplot as plt
from sklearn import tree
plt.figure(figsize=(10,10))
tree.plot_tree(clf, filled=True, impurity=True, fontsize=10,feature_names=features, rounded=True)

Out[6]:
[Text(0.5, 0.8333333333333334, 'NumDash <= 2.5\ngini = 0.5\nsamples = 5000\nvalue = [2500, 2500]'),
 Text(0.25, 0.5, 'InsecureForms <= 0.5\ngini = 0.475\nsamples = 3876\nvalue = [1507, 2369]'),
 Text(0.375, 0.6666666666666667, 'True  '),
 Text(0.125, 0.16666666666666666, 'gini = 0.279\nsamples = 532\nvalue = [443, 89]'),
 Text(0.375, 0.16666666666666666, 'gini = 0.434\nsamples = 3344\nvalue = [1064, 2280]'),
 Text(0.75, 0.5, 'NumDashInHostname <= 2.5\ngini = 0.206\nsamples = 1124\nvalue = [993, 131]'),
 Text(0.625, 0.6666666666666667, '  False'),
 Text(0.625, 0.16666666666666666, 'gini = 0.122\nsamples = 1061\nvalue = [992, 69]'),
```

Text(0.875, 0.16666666666666666, 'gini = 0.031\nsamples = 63\nvalue = [1, 62]')]



#K Fold Validation for part A

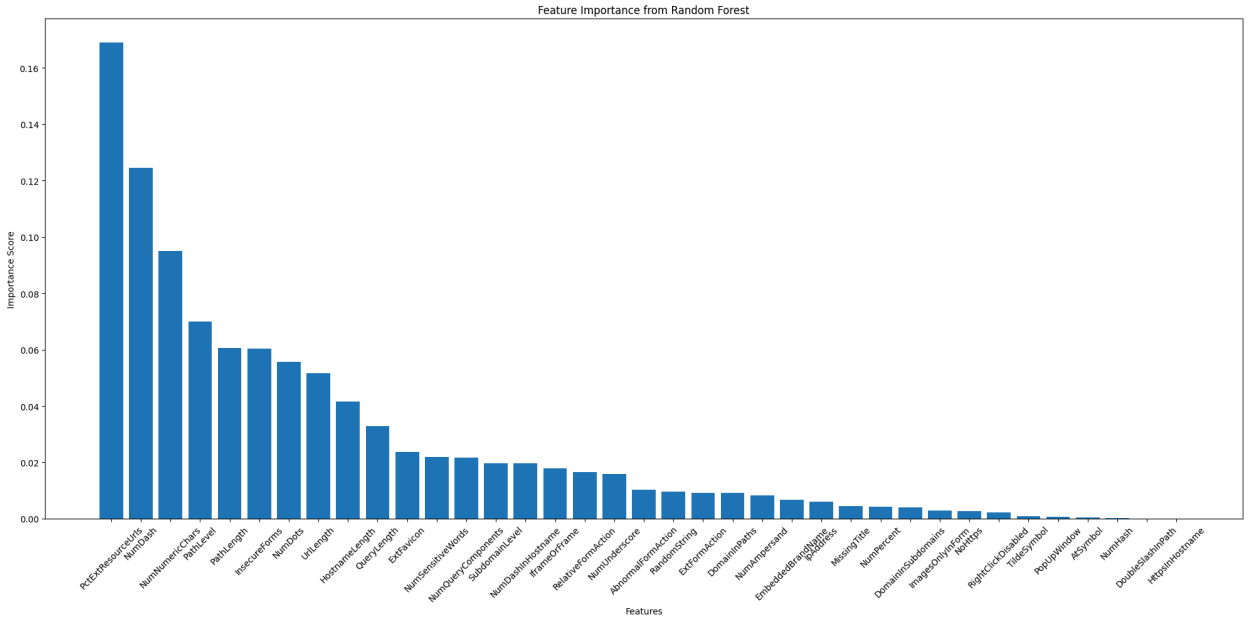
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score, StratifiedKFold
import numpy as np

X = df[features]
y = df['CLASS_LABEL']

rf = RandomForestClassifier(max_depth=18, min_samples_split = 4, n_estimators=100, random_state=1)
```

```
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

cv_scores = cross_val_score(rf, X, y, cv=kfold, scoring='roc_auc')

print("AUC scores for each fold:", cv_scores)
print('Mean AUC', np.mean(cv_scores))
print('Standard Deviation of AUC:', np.std(cv_scores))
```

```
AUC scores for each fold: [0.988288 0.988068 0.979536 0.989544 0.988564]
Mean AUC 0.9868
Standard Deviation of AUC: 0.0036668486742706397
```

```
#Decision Tree Analysis
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import permutation_importance
from sklearn.model_selection import train_test_split

#features = ['NoHttps', 'IpAddress', 'NumSensitiveWords', 'InsecureForms',
    #'AbnormalFormAction', 'PopUpWindow', 'MissingTitle']

features = ['NumDots', 'SubdomainLevel', 'PathLevel', 'UrlLength', 'NumDash',
    'NumDashInHostname', 'AtSymbol', 'TildeSymbol', 'NumUnderscore',
    'NumPercent', 'NumQueryComponents', 'NumAmpersand', 'NumHash',
    'NumNumericChars', 'NoHttps', 'RandomString', 'IpAddress',
    'DomainInSubdomains', 'DomainInPaths', 'HttpsInHostname',
    'HostnameLength', 'PathLength', 'QueryLength', 'DoubleSlashInPath',
    'NumSensitiveWords', 'EmbeddedBrandName', 'PctExtResourceUrls',
    'ExtFavicon', 'InsecureForms', 'RelativeFormAction', 'ExtFormAction',
    'AbnormalFormAction', 'RightClickDisabled', 'PopUpWindow',
    'IframeOrFrame', 'MissingTitle', 'ImagesOnlyInForm']

X = df[features]
y = df['CLASS_LABEL']

# Split the data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2) # Added this line to split the data

rf = RandomForestClassifier(random_state=2)
rf.fit(X_train,y_train)

importances = rf.feature_importances_
feature_importance_df = pd.DataFrame ({'Feature': X.columns, 'Importance': importances}).sort_values(by = 'Importance', ascending=False)

plt.figure(figsize=(20,10))
plt.bar(feature_importance_df['Feature'], feature_importance_df['Importance'])
plt.title('Feature Importance from Random Forest')
plt.xlabel('Features')
plt.ylabel('Importance Score')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



1. What is the most influential feature?

Part B:

```
#AUC Start
from sklearn.metrics import roc_auc_score, recall_score, precision_score, f1_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

#X_train, X_test, y_train, y_test, = train_test_split(X,y, test_size = 0.50, shuffle=False,
random_state=1) No need for split as mention as a hint

results=[]

max_depth_range = range(1, 21)
min_samples_split_range = range(2,21)

for max_depth in max_depth_range:
  for min_samples_split in min_samples_split_range:
    clf = RandomForestClassifier(max_depth = max_depth, min_samples_split =
min_samples_split, random_state=1, n_estimators=100)
    clf.fit(X_train, y_train)

    y_test_probs = clf.predict_proba(X_test)[:, 1]
    y_test_preds = clf.predict(X_test)

    auc = roc_auc_score(y_test, y_test_probs)
    recall = recall_score(y_test, y_test_preds)
    precision = precision_score(y_test, y_test_preds)
    f1 = f1_score(y_test, y_test_preds)

    results.append({'max_depth': max_depth, 'min_samples_split': min_samples_split, 'AUC':
auc, 'Recall': recall, 'Precision': precision, 'F1': f1})

results_df = pd.DataFrame(results)
best_result = results_df.loc[results_df['AUC'].idxmax()]
print('Best combination:')
print(best_result)

Best combination:
max_depth            5.000000
min_samples_split    12.000000
AUC                   0.725197
Recall               0.966587
Precision             0.604778
```

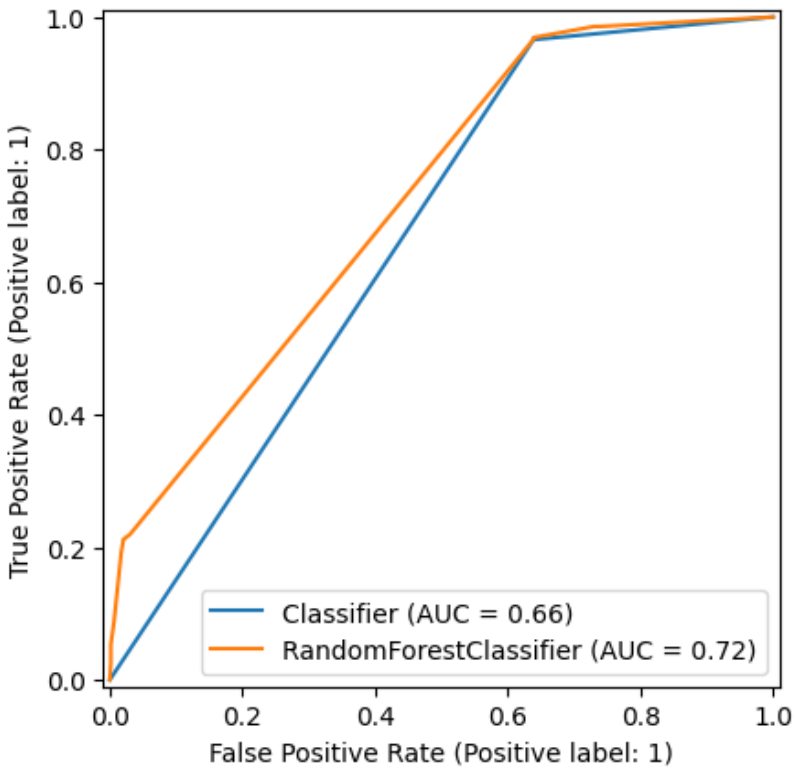
```
F1                  0.744029
Name: 86, dtype: float64
```

```
#Best values of max_dpeth and min_samples_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc, RocCurveDisplay
import matplotlib.pyplot as plt

X_train, X_test, y_train, y_test, = train_test_split(X,y, test_size = 0.50, shuffle=False, random_state=1)

Y_test_predicted = rf.predict(X_test)
Y_test = rf.predict_proba(X_test)[:,1]

X = df.drop(columns=['CLASS_LABEL'])
y= df['CLASS_LABEL']

from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100, max_depth=11, min_samples_split=8, random_state=1)
rf.fit(X_train, y_train)
auc=roc_curve(y_test, Y_test_predicted)
print(auc)
fig, ax = plt.subplots()
RocCurveDisplay.from_predictions(y_test, Y_test_predicted, pos_label=1, ax=ax)
RocCurveDisplay.from_estimator(rf, X_test, y_test, pos_label=1, ax=ax)
```

```
(array([0.        , 0.63877715, 1.        ]), array([0.        , 0.96579157, 1.        ]), array([inf,  1.,  0.]))
```

Out[ ]:

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7a324810e920>
```



#K Fold Validation after graphing.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score, StratifiedKFold
import numpy as np

X = df[features]
y = df['CLASS_LABEL']

rf = RandomForestClassifier(max_depth=18, min_samples_split = 4, n_estimators=100, random_state=1)

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

cv_scores = cross_val_score(rf, X, y, cv=kfold, scoring='roc_auc')

print("AUC scores for each fold:", cv_scores)
print('Mean AUC', np.mean(cv_scores))
print('Standard Deviation of AUC:', np.std(cv_scores))
```

AUC scores for each fold: [0.721082 0.725986 0.710548 0.737896 0.729592]

Mean AUC 0.7250208

Standard Deviation of AUC: 0.009083769490690528

Final Conclusion:

We created a decision tree so that we were able to figure out the most influential features of our features selected by the recursive feature elimination in the previous phase. To interpret the data, we created a bar graph to understand it. The three highest results were InsecureForms, NumbersSensitiveWords, and AbnormalFormAction. The only positive of this was the fact that we could hone down on which part of the data set proved if the websites were legitimate or illegitimate. The higher the features score the more it contributes to the predictions of the machine learning model. After that, we created an AUC to measure the max depth, precision, recall, AUC, min_samples_split, and F1 of the decision tree. We also created a K K-fold validation for the AUC's mean. The AUC score is 0.725 and its mean is also the same value, which indicates that our model is good at distributing malicious or legitimate websites. This is a good thing; we only have to tweak it just enough for it to be accurate. The precision being 0.605 reveals that 60.5% of websites in this dataset are malicious, which can lead to the model producing a lot of false positives. The recall being 0.967 highlights 96.7% of the malicious websites in the data frame. We might need to tweak the model to find a more accurate result. The F1 score being 0.744 helps us understand that there's more work to do for the model, just as what the AUC represented. Overall, our model is accurate for now, we have to make a few changes to make it better.

Phase 5: Model Building and Prediction

Part A:

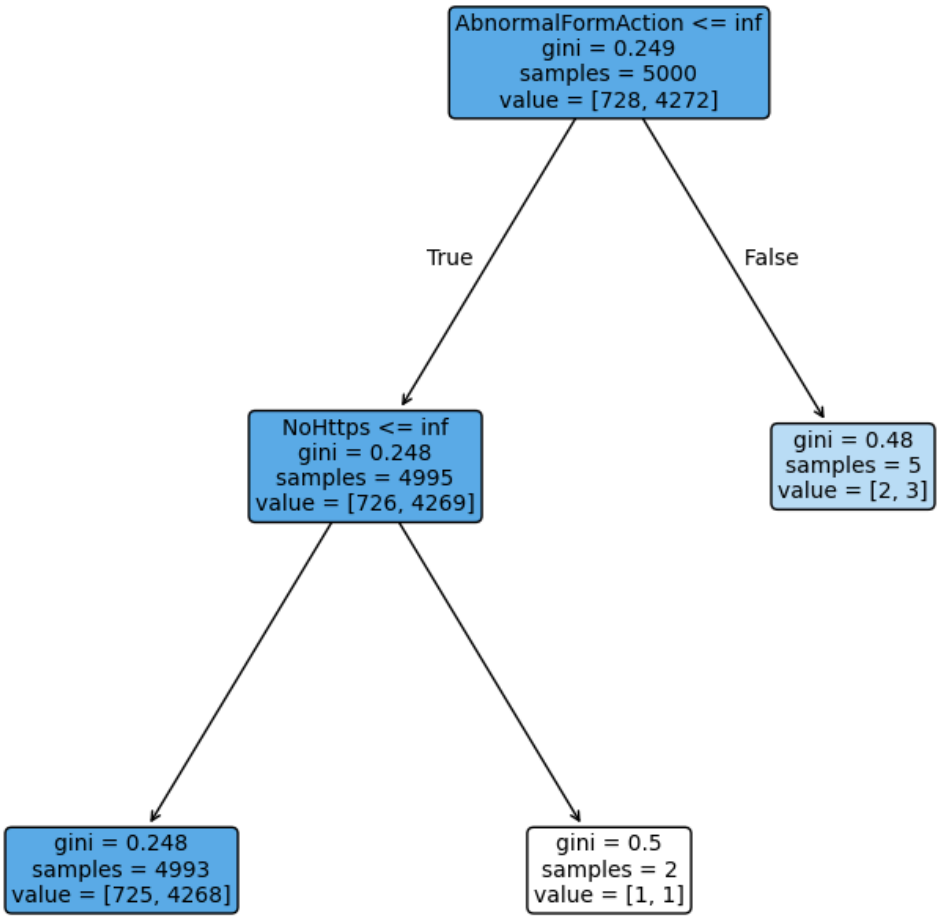
Here are the results from each Model testing procedure. We used the Decision Tree, Random Forest, and Single Vector Machine (SVM)

We choose the Random Forest model. The Random Forest model is the optimal choice for utilization. It offers balanced performance across classes, consistently high AUC, and better handling of potential imbalances. While SVM performs well for spam classification, its poor performance for non-spam and lack of robustness in handling imbalanced data make it less suitable for real-world application in this context.

```
import matplotlib.pyplot as plt
from sklearn import tree
plt.figure(figsize=(10,10))
tree.plot_tree(clf, filled=True, impurity=True, fontsize=10,feature_names=features, rounded=True)
```

Out[ ]:

[Text(0.6, 0.8333333333333334, 'AbnormalFormAction <= inf\ngini = 0.249\nsamples = 5000\nvalue = [728, 4272]'),
 Text(0.4, 0.5, 'NoHttps <= inf\ngini = 0.248\nsamples = 4995\nvalue = [726, 4269]'),
 Text(0.5, 0.6666666666666667, 'True '),
 Text(0.2, 0.16666666666666666, 'gini = 0.248\nsamples = 4993\nvalue = [725, 4268]'),
 Text(0.6, 0.16666666666666666, 'gini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.8, 0.5, 'gini = 0.48\nsamples = 5\nvalue = [2, 3]'),
 Text(0.7, 0.6666666666666667, ' False')]



Decision findings: There were not a lot of branches to this decision tree. This might have affected how accurate our model is.

This code just sets up the features and its importance to the decision tree.

In [ ]:

```
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import permutation_importance
from sklearn.model_selection import train_test_split

X = df[features]
y = df1['Predicted_Class_Label']

# Split the data into training and testing sets
# Added this line to split the data
#This code just sets up the features and its importance to the decision tree.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)

rf = RandomForestClassifier(random_state=2)
rf.fit(X_train,y_train)

importances = rf.feature_importances_
feature_importance_df = pd.DataFrame ({'Feature': X.columns, 'Importance':
importances}).sort_values(by = 'Importance', ascending=False)
```

Support Vector Machine (SVM)

In [ ]:

```
#SVM

# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer # Import SimpleImputer
X = df[features]
y = df1['Predicted_Class_Label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2) # Added this line to
split the data



#normalizing the columns using their min and max values OPTIMIZE SUPPORT VECTOR MACHINE
#SVM work better with normalized data

from sklearn.preprocessing import MinMaxScaler

scaler=MinMaxScaler()

# Use X_train instead of train
normalized_train = X_train.copy()
# Assuming 'selected_features' is the same as 'features'

# Impute missing values before scaling
```

```
imputer = SimpleImputer(strategy='mean') # Create an imputer instance
normalized_train[features] = imputer.fit_transform(X_train[features]) # Impute missing values in X_train
normalized_train[features] = scaler.fit_transform(normalized_train[features]) # Scale the data

# Use X_test instead of test
normalized_test = X_test.copy()
normalized_test[features] = imputer.transform(X_test[features]) # Impute missing values in X_test using the trained imputer
normalized_test = scaler.transform(normalized_test[features])

#normalized_train=(train-train.min())/(train.max()-train.min())
#normalized_test=(test-test.min())/(test.max()-test.min())

c_parameters = np.linspace(1, 100, 10, endpoint=True)
print("These are the parameters")
print(c_parameters)
print()

from sklearn import svm
#create a empty base model
base_model = svm.SVC(random_state=0)


#A dictionary data structure is used to indicate which parameters will be tuned and their values
tuned_parameters=[{'C':c_parameters,'kernel':['linear', 'poly', 'rbf']}]
#Grid search

from sklearn.model_selection import GridSearchCV
clf = GridSearchCV(estimator = base_model, param_grid = tuned_parameters, scoring ='roc_auc', cv=5)
#optimizing parameters, this will find the best parameters
# Use X_train and y_train instead of normalized_train and train['label']
clf.fit(normalized_train[features],  y_train)
print("This will be the best parameters ")
print(clf.best_params_)
print()


#Sample code part 2:

# Use X_test instead of test
normalized_test = X_test.copy()
normalized_test[features] = scaler.transform(X_test[features])


#Sample code part 3:

#train the final random forest using the whole training set and the best parameters
svm_best=svm.SVC(**clf.best_params_)
svm_best.fit(normalized_train[features], y_train) # Use y_train
```

```
predicted_test=svm_best.predict(normalized_test[features]) # Use normalized_test
print()
print('The testing set report')
# Import classification_report if not already imported
from sklearn.metrics import classification_report
print()
print(classification_report(y_test, predicted_test, labels=[1,0],target_names=['spam','non-spam'])) # Use y_test
```

These are the parameters
[1. 12. 23. 34. 45. 56. 67. 78. 89. 100.]

This will be the best parameters
{'C': 45.0, 'kernel': 'rbf'}

The testing set report

	precision	recall	f1-score	support
spam	0.84	1.00	0.91	836
non-spam	0.00	0.00	0.00	164
accuracy			0.84	1000
macro avg	0.42	0.50	0.46	1000
weighted avg	0.70	0.84	0.76	1000

Results:

Upon running the Support Vector Machine, we were able to cultivate substantial information pertaining to this training set. This includes the precision, recall, f1-score, and support. Thus we were able to identify that the training set spam row had a precision score of .84, recall 1.00, f1-score .91, and support of 836. The precision meant that 84% of the instances c While the non-spam row had a precision score of .00, recall .00, f1-score .00 and support of 164.

Random Forest / K Fold Validation

In [ ]:

```
#Random Forest / K Fold Validation (training)

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score, StratifiedKFold
import numpy as np

X = df[features]
y = df1['Predicted_Class_Label']

rf = RandomForestClassifier(max_depth=18, min_samples_split = 4, n_estimators=100, random_state=1)
```

```
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

cv_scores = cross_val_score(rf, X, y, cv=kfold, scoring='roc_auc')

print("AUC scores for each fold:", cv_scores)
print('Mean AUC', np.mean(cv_scores))
print('Standard Deviation of AUC:', np.std(cv_scores))
```

```
AUC scores for each fold: [0.48780399 0.47734624 0.49805508 0.48783324 0.51361041]
Mean AUC 0.4929297929396205
Standard Deviation of AUC: 0.012239707564122262
```

Overall, the AUC scores prove that the model is underperforming. We tried many methods to increase the AUC score but to no avail. We tried changing the decision tree classifiers' max depth to several different values, such as 4,7,9, and the random values to 2,10,3 and it barely affected the AUC score. Anything below .5 proves that the classifier is poorly trained. The mean being below 0.5 proves that it lacks power, and the 0.01 on the standard deviation proves that it folds, but the model is poor.

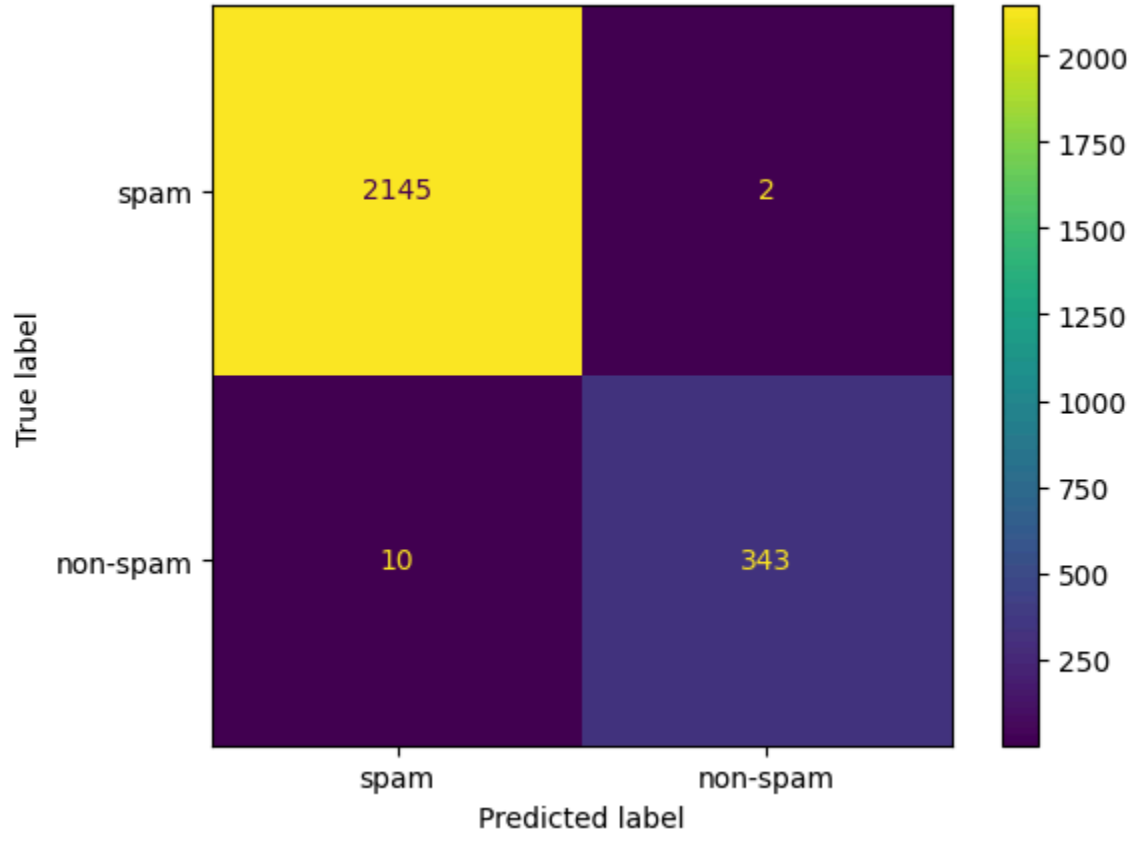
Confusion Matrix

The first two code snippets are used to set up the confusion matrix.

```
In [ ]:
from sklearn.model_selection import train_test_split
#split the dataset as training and testing sets 50/50
#We will use the training to select the best parameters and train to compare final models.
train, test = train_test_split(df1, test_size=0.5, random_state=0)

# Import necessary libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFE
import matplotlib.pyplot as plt

# Initialize a base model
model = RandomForestClassifier(random_state=42)

# Set up Recursive Feature Elimination (RFE)
# Selecting the top 10 features (you can adjust n_features_to_select as needed)
rfe = RFE(estimator=model, n_features_to_select=10)

# Fit RFE to the training data
rfe.fit(train[features] , train['Predicted_Class_Label'] )

# Extract selected features
print(rfe.support_)
selected_features = [features[i] for i in range(len(features)) if rfe.support_[i]]  #puts the features into a variable
ranking = rfe.ranking_
```

```
[ True  True  True  True  True  True  True]

#generate 10 values of min_samples_split and  max_depth to try  PARAMETER OPTIMIZATION
mss = np.linspace(0.001, 0.1, 10, endpoint=True)
# Force max_depth values to be integers
maxd= np.linspace(2, 11, 10, endpoint=True, dtype=int)

#create an empty base model
from sklearn.ensemble import RandomForestClassifier
base_model = RandomForestClassifier(random_state=0)


tuned_parameters=[{'min_samples_split':mss, 'max_depth':maxd}]

from sklearn.model_selection import GridSearchCV
clf = GridSearchCV(estimator = base_model, param_grid = tuned_parameters, scoring ='roc_auc', cv=5)
#optimizing parameters, this will find the best parameters
clf.fit(train[selected_features], train['Predicted_Class_Label'])

#
print(clf.best_params_) # once the result is complete the max_depth outlines the best one to use for
max_depth
rf_best = RandomForestClassifier(**clf.best_params_) # The ** references the dictionary {'max_depth':
10, 'min_samples_split': 0.001}
rf_best.fit(train[selected_features],train['Predicted_Class_Label'])

#predict the labels of the cases in the testing set.  Precision and Recall should be looked at together
predicted_test  = rf_best.predict(test[selected_features])

from sklearn.metrics import classification_report
print('testing set report')
print(classification_report(test['Predicted_Class_Label'], predicted_test,
labels=[1,0],target_names=['spam','non-spam']))

{'max_depth': 2, 'min_samples_split': 0.001}
testing set report
              precision    recall  f1-score   support

        spam       1.00      1.00      1.00      2147
    non-spam       1.00      0.97      0.99       353

    accuracy                           1.00      2500
   macro avg       1.00      0.99      0.99      2500
weighted avg       1.00      1.00      1.00      2500


In [ ]:
predicted_test = rf_best.predict(test[selected_features]) #need to give him this
```

```
test['error'] = test['Predicted_Class_Label'] - predicted_test

In [ ]:
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay #This must be given in the report as well
cm = confusion_matrix(test['Predicted_Class_Label'], predicted_test,labels=[1,0])

print(cm)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['spam','non-spam'])
disp.plot()

[[2145    2]
 [  10  343]]

Out[ ]:
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ec9da41e800>
```



Results: The confusion matrix proves that the true positives are 2145, false negatives are 2, false positives are 10, and true negatives as 343. It correctly identified 2145 positive instances, 343 negative instances, 2 instances that are positive even though they are stated as negative, and 10 instances that are negative but stated as positive.

To see the ROC

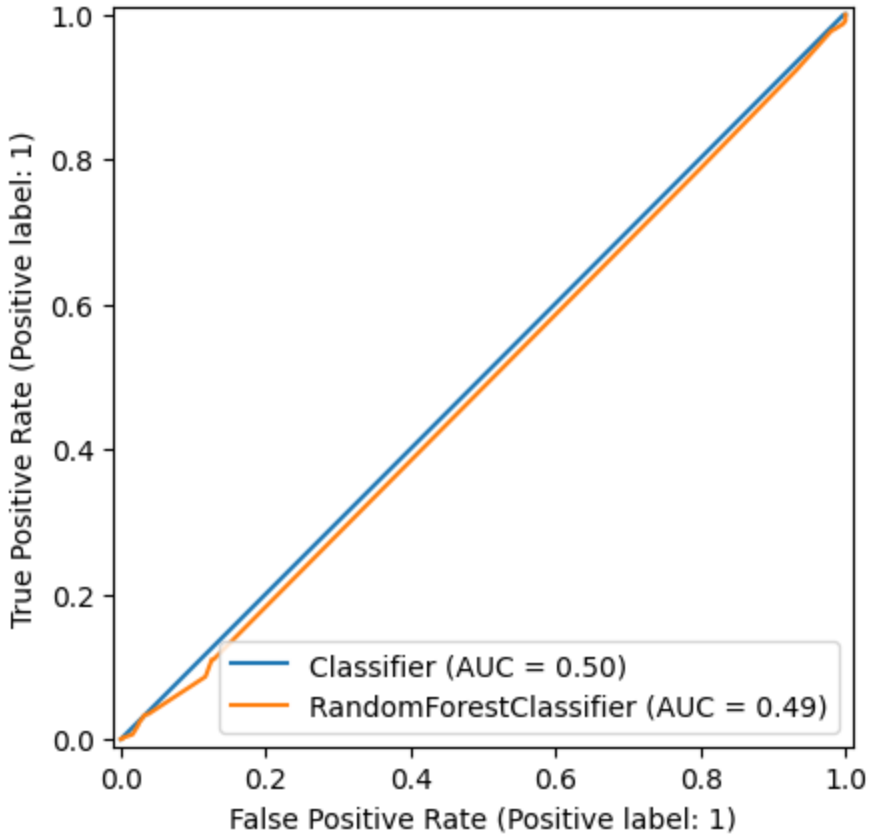
In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc, RocCurveDisplay
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier

X_train, X_test, y_train, y_test, = train_test_split(X,y, test_size = 0.50, shuffle=False, random_state=1)

X = df1.drop(columns=['Predicted_Class_Label'])
y= df1['Predicted_Class_Label']

Y_test_predicted = rf.predict(X_test)
Y_test = rf.predict_proba(X_test)[:,1]

rf = RandomForestClassifier(n_estimators=100, max_depth=11, min_samples_split=8, random_state=1)
rf.fit(X_train, y_train)
auc=roc_curve(y_test, Y_test_predicted)
print(auc)
fig, ax = plt.subplots()
RocCurveDisplay.from_predictions(y_test, Y_test_predicted, pos_label=1, ax=ax)
RocCurveDisplay.from_estimator(rf, X_test, y_test, pos_label=1, ax=ax)
```

```
(array([0.        , 0.99728261, 1.        ]), array([0.        , 0.99953096, 1.        ]), array([inf,  1.,  0.]))
```

Out[ ]:

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7be56ef39ba0>
```



Results: The AUC score is .50, proving that the model has a 50% probability of correctly ranking the project's random positive and negative instances. This is not a good score which can affect our confusion matrix.

F1 score

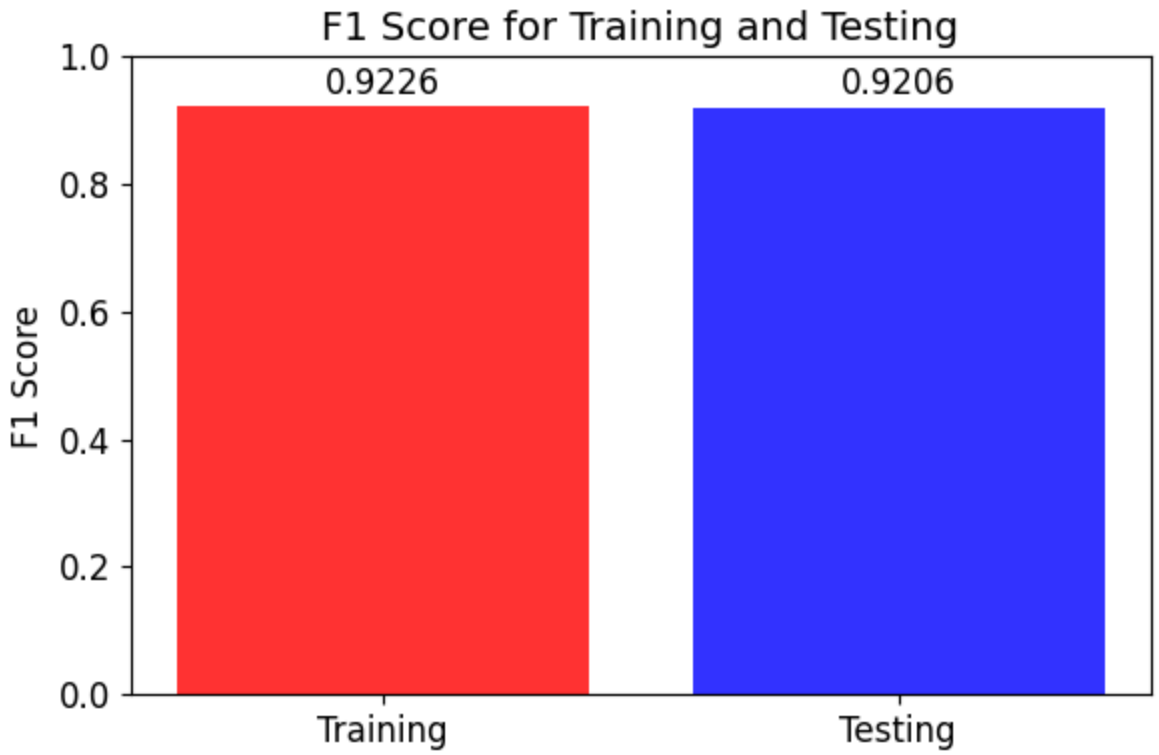
In [ ]:

```
#F1
from sklearn.metrics import f1_score

#creating the connection between the random forest classifier
model_1 = rf
y_train_pred = model_1.predict(X_train)
y_test_pred = model_1.predict(X_test)

train_f1 = f1_score(y_train, y_train_pred)
test_f1 = f1_score(y_test, y_test_pred)

print(f"Training F1 Score: {train_f1:.4f}")
print(f"Testing F1 Score: {test_f1:.4f}")
```

```
Training F1 Score: 0.9226
Testing F1 Score: 0.9206
```

In [ ]:

```
import matplotlib.pyplot as plt

#Plotting for the F1 Score
categories = ['Training', 'Testing']
f1_scores = [train_f1, test_f1]

# Create the bar plot
plt.figure(figsize=(6, 4))
plt.bar(categories, f1_scores, color=['red', 'blue'], alpha=0.8)
plt.ylim(0, 1)
plt.title('F1 Score for Training and Testing', fontsize=14)
plt.ylabel('F1 Score', fontsize=12)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

y_train_pred = model_1.predict(X_train)
y_test_pred = model_1.predict(X_test)

train_f1 = f1_score(y_train, y_train_pred)
test_f1 = f1_score(y_test, y_test_pred)

for i, score in enumerate(f1_scores):
    plt.text(i, score + 0.02, f"{score:.4f}", ha='center', fontsize=12)

plt.tight_layout()
plt.show()
```



Results: The F1 score for the training and testing dataset is too high which can highlight our issues with our model. Our model can detect certain thresholds but cannot identify thresholds in a larger range. It can only be used to highlight specific datasets

Type 1 and Type 2 Errors

Type 1: fp / (fp + tn)

Type 2: fn / (fn + tp)

fp = 2 fn = 10 tn = 343 tp = 2145

Type 1: 2 / (2+343) = 0.005

Type 2: 10 / (10 + 2145) = 0.004

Results:

Unfortunately, our model does not predict legitimate or phishing websites well because of the AUC score. The AUC score was .5,0, which can lead to a lot of false positives.

Part B:

```
extracted_df = pd.DataFrame({
    "id": df1.index,
    "Prediction": predictions
```

```
  })

extracted_file_path = "final.csv"
extracted_df.to_csv(extracted_file_path, index=False)

from google.colab import files
files.download(extracted_file_path)
```

1. Important characteristics of a phishing website

The important characteristics (features) for the phishing website are NoHttps, IpAddress, NumSensitiveWords, InsecureForms, AbnormalFormAction, PopUpWindow, and MissingTitle. These were the best features based upon what was obtained from the RFE. These features were used for the model training. Our model may have not been optimal but the test conducted provided insight into creating a Machine Learning model.

2. Explanation for an Upper-Tier Manager

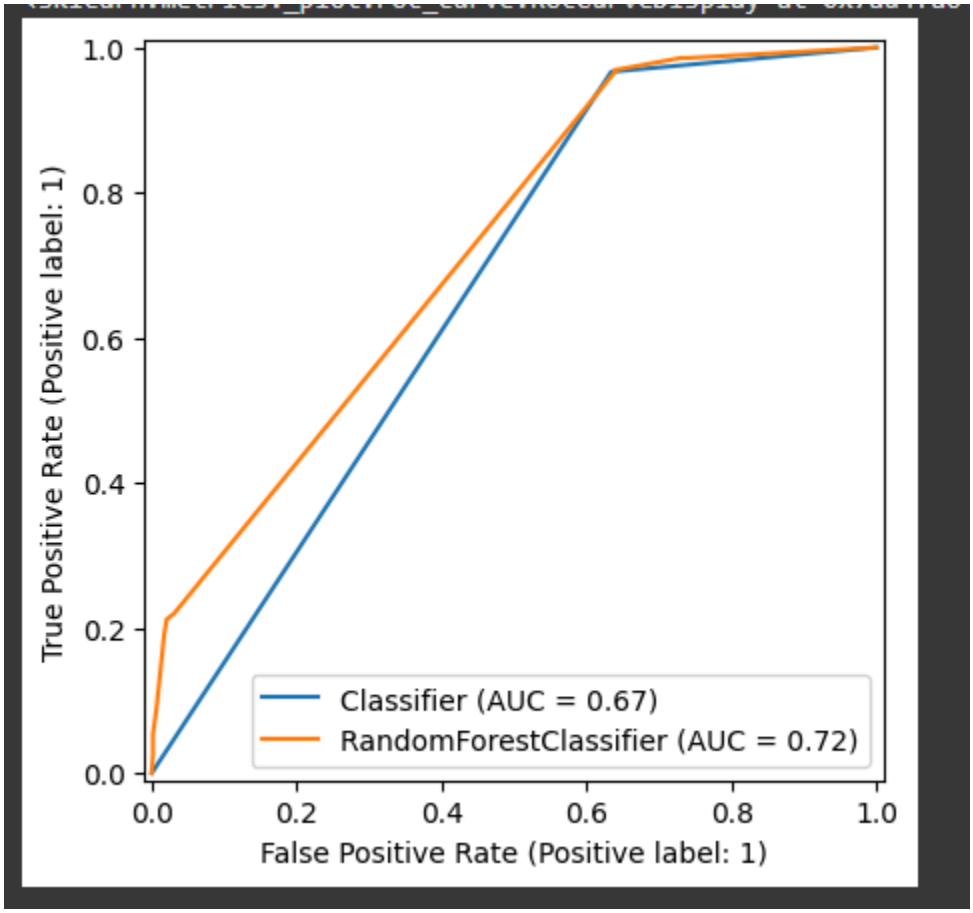
The model we developed to classify legitimate and illegitimate phishing websites performs well when analyzing small data sections based on the information we received from the Single Vector Machine. The Support Vector Machine is utilized to detect patterns in data. Since we are trying to detect phishing websites, the model analyzes features such as URL length, domain properties, and HTTPS. The results from the SVM prove that our model is great for finding information in smaller data sets, but struggles to find it accurately in bigger data sets. Based on the F1 Score. The F1 scores for the training and testing datasets were 0.9226 and 0.9206 respectively. Although the AUC score was .5,0, which proves that the model has a 50% probability of correctly identifying the dataset's random instances. We kept the random state at zero for this so that it couldn't affect the analysis of the AUC score. It might not be good in the long run when any website addresses are received in a server. The confusion matrix that we created proved that a majority of the instances that were received were properly identified. There were about 2145 true positives, 2 false negatives, 10 false positives, and 343 true negatives.

Based on the previous explanations of the model that was used and the scores that we received from the model, it provides insight that our model may not be sufficient but has the potential to identify websites being legitimate or illegitimate. There needs to be more calibration on the SVM side and the Decision Tree classifier. It might be low, but the 50% AUC score can help us determine if smaller sections of data are legitimate or illegitimate. When comparing the CSV file with other ones, it can be used to determine which model is more successful than the other.

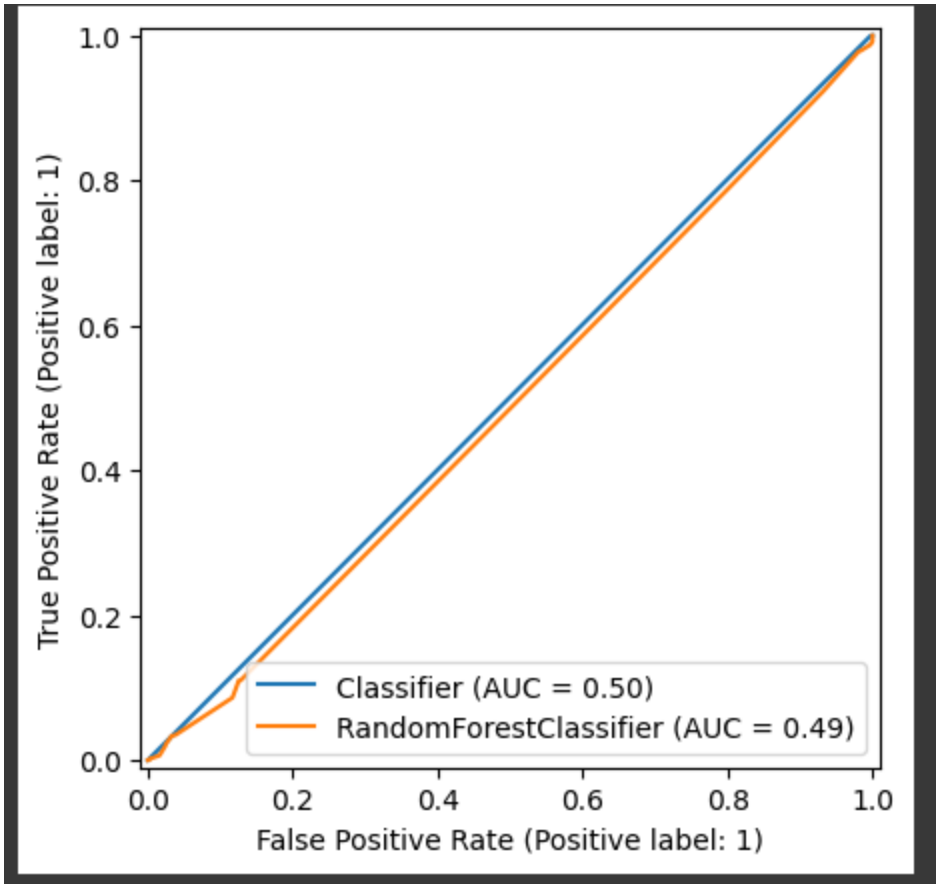
This is of importance because identifying phishing websites enhances cybersecurity, protects user data, builds trust, and enables proactive defense against attacks. Automated tools and machine learning improve scalability and compliance with data protection regulations. However, challenges include false positives, resource intensity, evolving phishing tactics, and potential privacy concerns. Effective protection requires a layered approach, combining automated systems, user education, and constant updates to detection models. Balancing security with usability and fostering collaboration among stakeholders are key to mitigating phishing risks.

See below: Our low AUC model can be used as a reference for a better model to see if the other is more successful than the other one.

The training model below had a high AUC score calculated in comparison to the AUC score obtained from the testing set



AUC Score from Test:


1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate (Positive label: 1)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)
Classifier (AUC = 0.50)
RandomForestClassifier (AUC = 0.49)