Phishing Website Detection

CAPSTONE 2

ELIZABETH ROGERS

Data

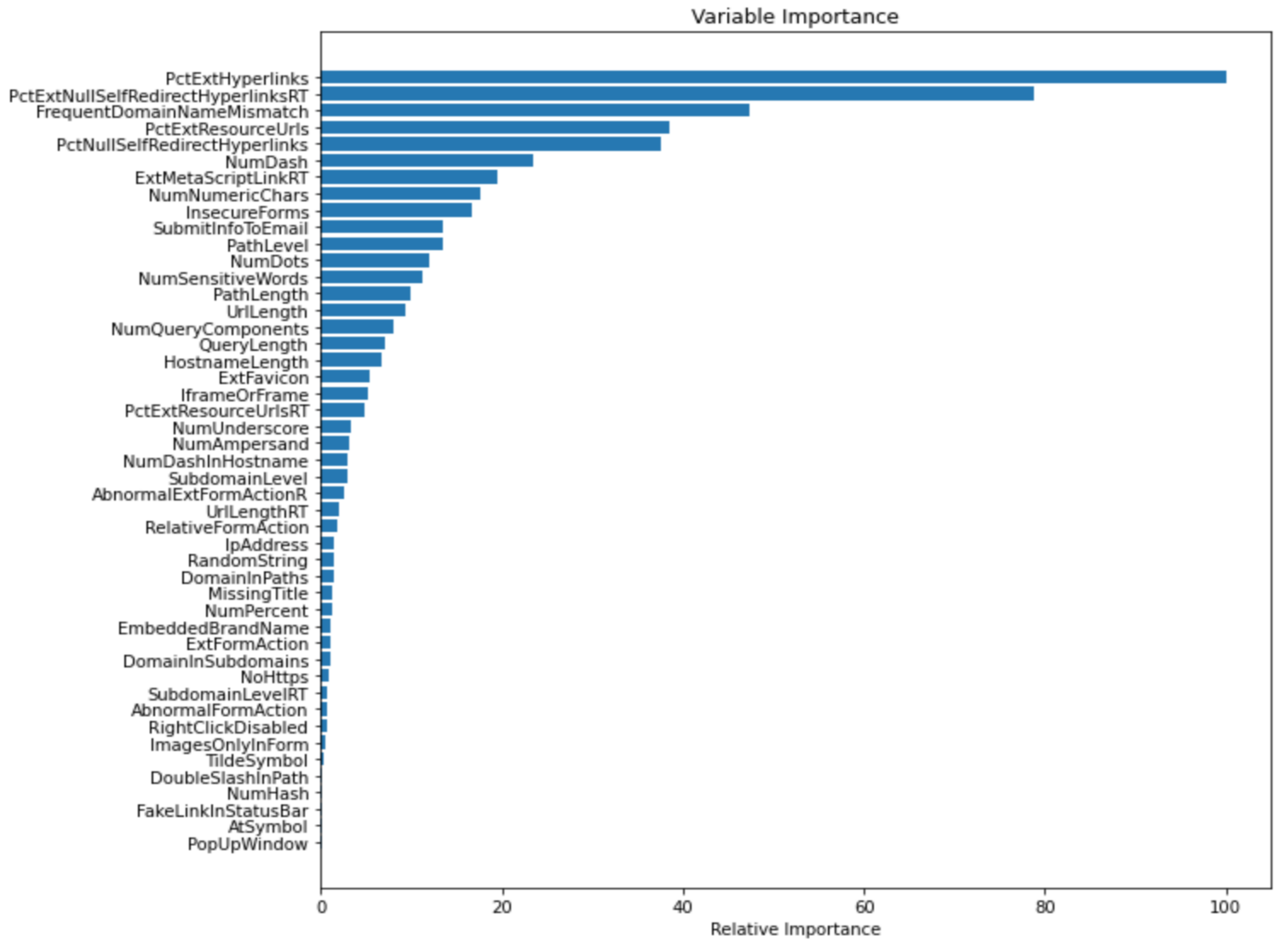
- 5,000 PHISHING WEBSITES
- 5,000 NON-PHISHING WEBSITES
- 48 NUMERICAL PARAMETERS
- AROUND HALF OF PARAMETERS BOOLEANS
- MANY INTS
- 3 PERCENTAGE PARAMETERS ARE FLOATS

Data Cleaning

- 2 OF THE COLUMNS DROPPED
- "ID" CONTAINED A REDUNDANT AND NOT VALUABLE INDEX
- "HTTPSINHOSTNAME" CONTAINED ONLY ONE VALUE
- DATA WAS ALL NUMERIC WITH NO MISSING VALUES
- ONE HOT ENCODING WAS PERFORMED EXPERIMENTALLY ON THE MOST CATEGORICAL-SEEMING INTS, BUT ULTIMATELY THIS PROVED UN-USEFUL

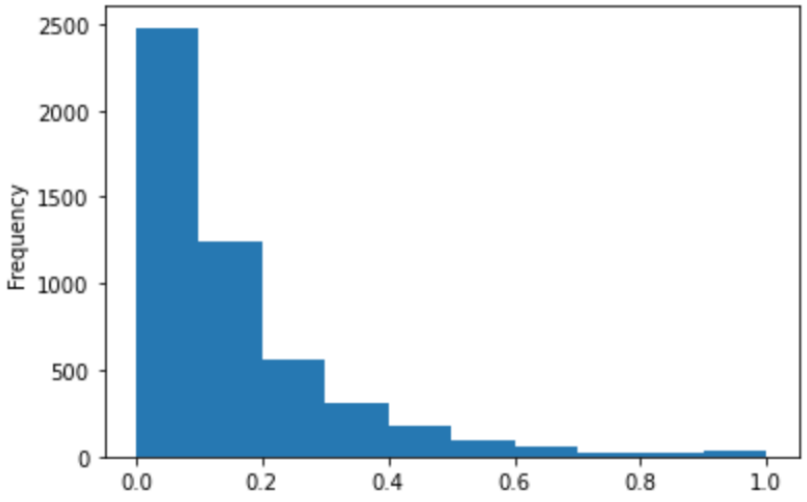
EDA

- HISTOGRAMS, VIOLIN PLOTS AND HEATMAPS USED TO VISUALIZE DATA
- THE FOLLOWING CATEGORIES PROVED MOST IMPORTANT IN MODEL PREDICTION: “PCTEXTHYPERLINKS”, “PCTEXTNULLSELFREDIRECTHYPERLINKSRT”, “FREQUENTDOMAINNAMEMISMATCH”, “PCTEXTRESOURCEURLS”, “PCTNULLSELFREDIRECTHYPERLINKS”, “NUMDASH”


Variable Importance
PctExtHyperlinks
PctExtNullSelfRedirectHyperlinksRT
FrequentDomainNameMismatch
PctExtResourceUrls
PctNullSelfRedirectHyperlinks
NumDash
ExtMetaScriptLinkRT
NumNumericChars
InsecureForms
SubmitInfoToEmail
PathLevel
NumDots
NumSensitiveWords
PathLength
UrlLength
NumQueryComponents
QueryLength
HostnameLength
ExtFavicon
IframeOrFrame
PctExtResourceUrlsRT
NumUnderscore
NumAmpersand
NumDashInHostname
SubdomainLevel
AbnormalExtFormActionR
UrlLengthRT
RelativeFormAction
IpAddress
RandomString
DomainInPaths
MissingTitle
NumPercent
EmbeddedBrandName
ExtFormAction
DomainInSubdomains
NoHttps
SubdomainLevelRT
AbnormalFormAction
RightClickDisabled
ImagesOnlyInForm
TildeSymbol
DoubleSlashInPath
NumHash
FakeLinkInStatusBar
AtSymbol
PopUpWindow
0
20
40
60
80
100
Relative Importance

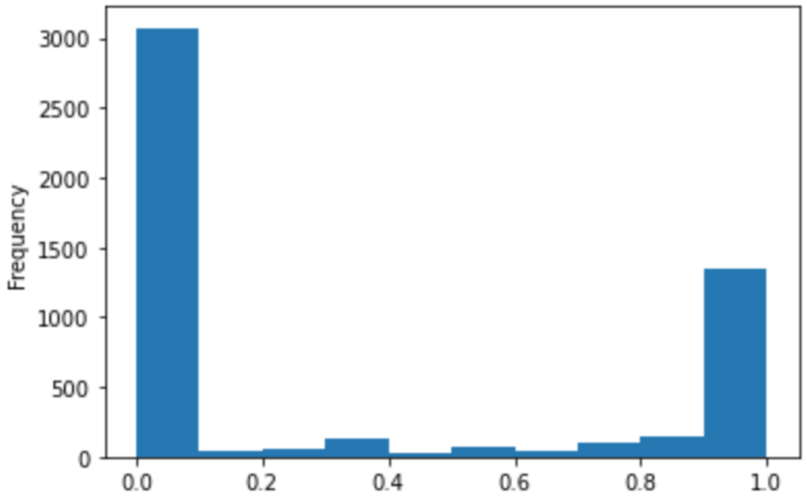
```
nefarious['PctExtHyperlinks'].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



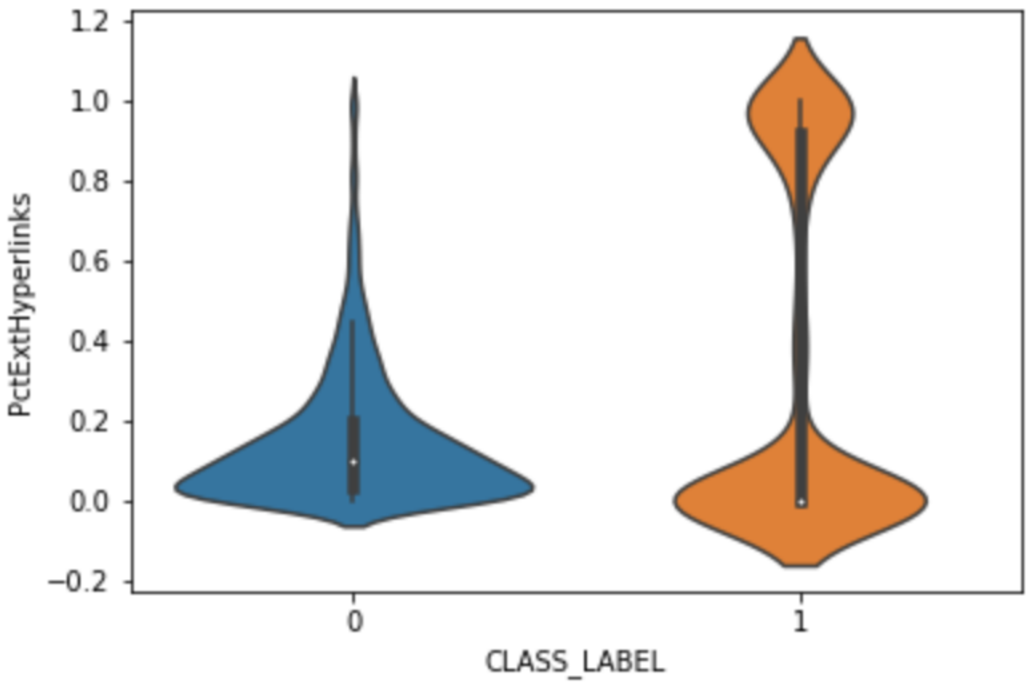

```
non_nefarious['PctExtHyperlinks'].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



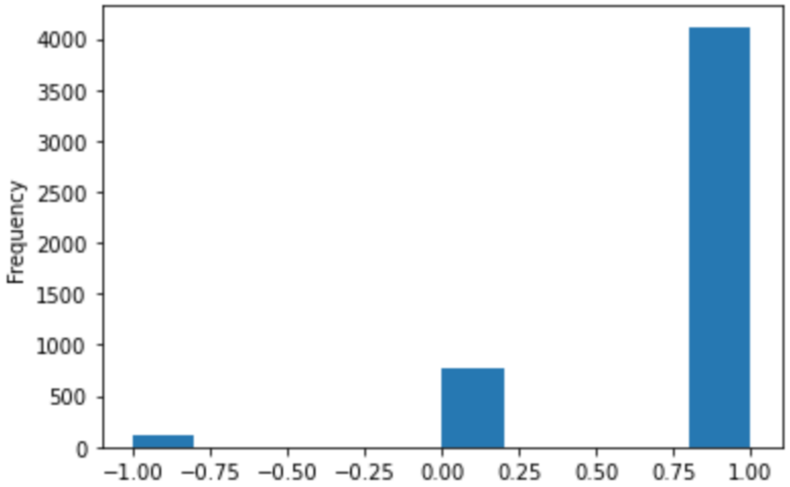

```
sns.violinplot(data=data, x='CLASS_LABEL',y=important_array[0])
```

```
<AxesSubplot:xlabel='CLASS_LABEL', ylabel='PctExtHyperlinks'>
```



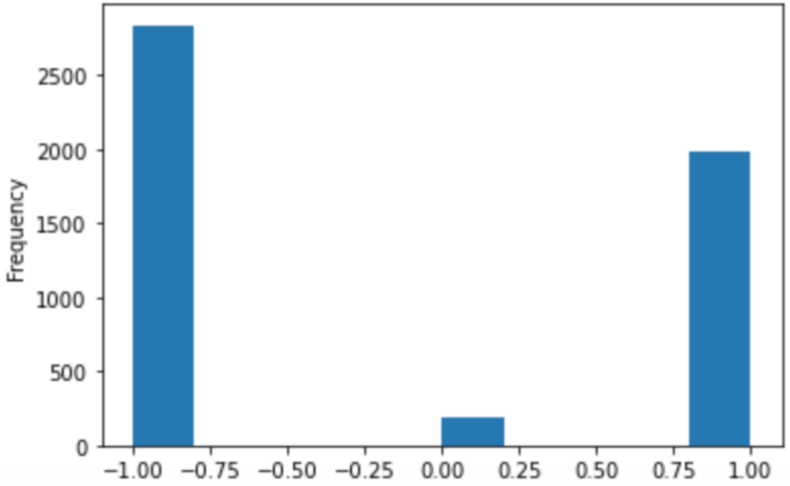
```
nefarious['PctExtNullSelfRedirectHyperlinksRT'].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



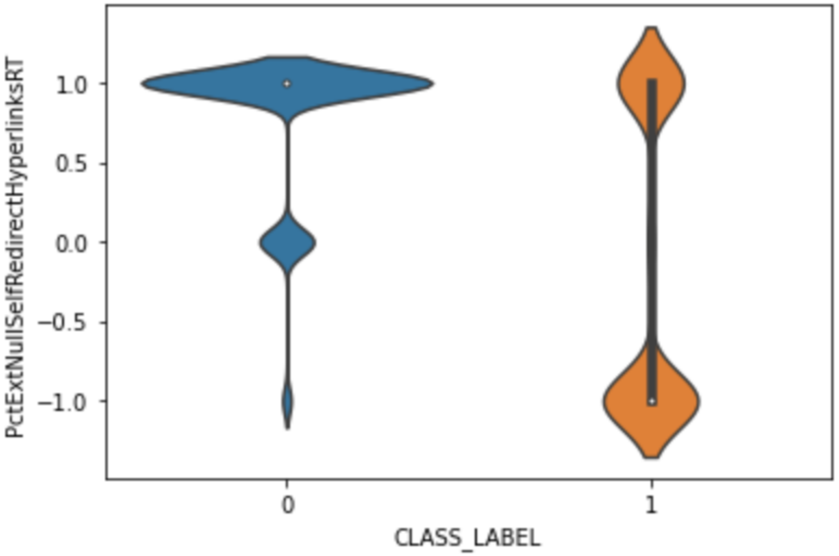

```
non_nefarious['PctExtNullSelfRedirectHyperlinksRT'].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



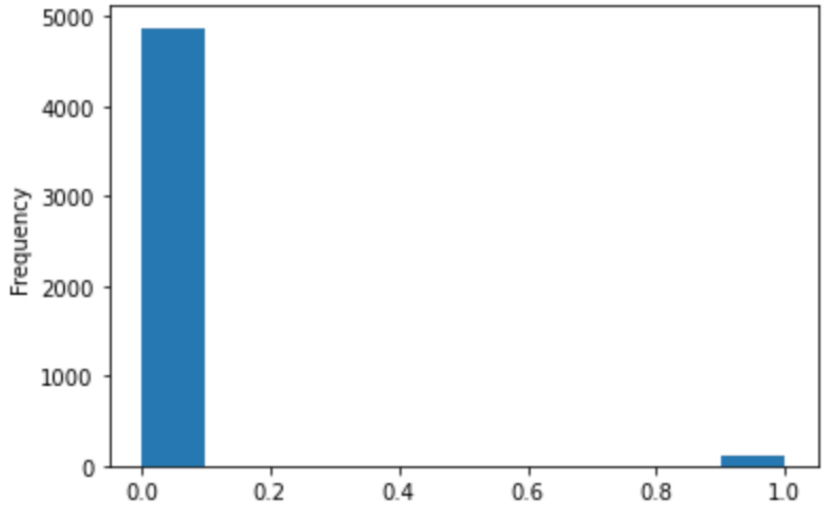

```
sns.violinplot(data=data, x='CLASS_LABEL',y=important_array[1])
```

```
<AxesSubplot:xlabel='CLASS_LABEL', ylabel='PctExtNullSelfRedirectHyperlinksRT
```



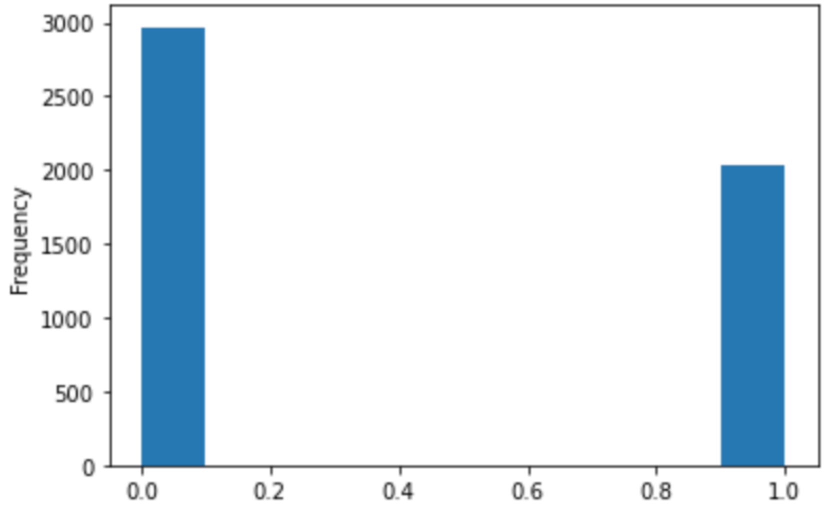
```
nefarious['FrequentDomainNameMismatch'].plot.hist()
```

<AxesSubplot:ylabel='Frequency'>



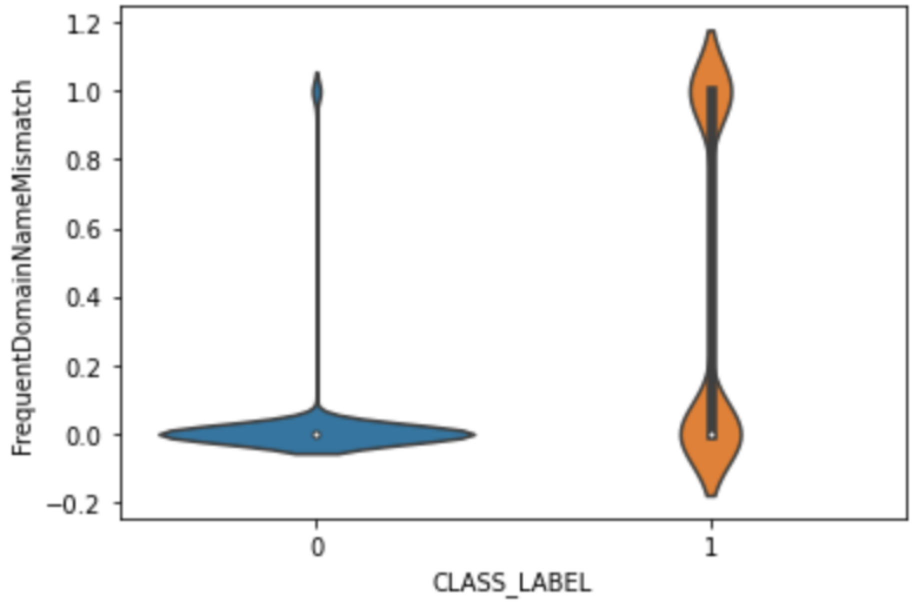

```
non_nefarious['FrequentDomainNameMismatch'].plot.hist()
```

<AxesSubplot:ylabel='Frequency'>



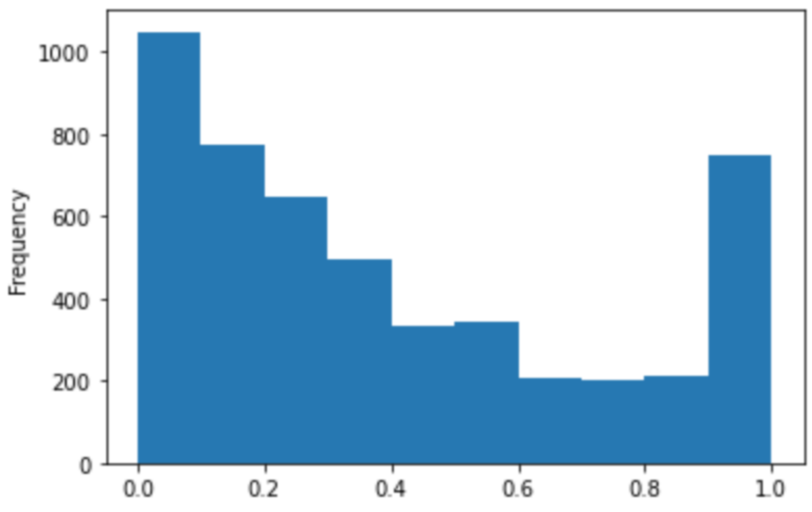

```
sns.violinplot(data=data, x='CLASS_LABEL',y=important_array[2])
```

<AxesSubplot:xlabel='CLASS_LABEL', ylabel='FrequentDomainNameMismatch'>



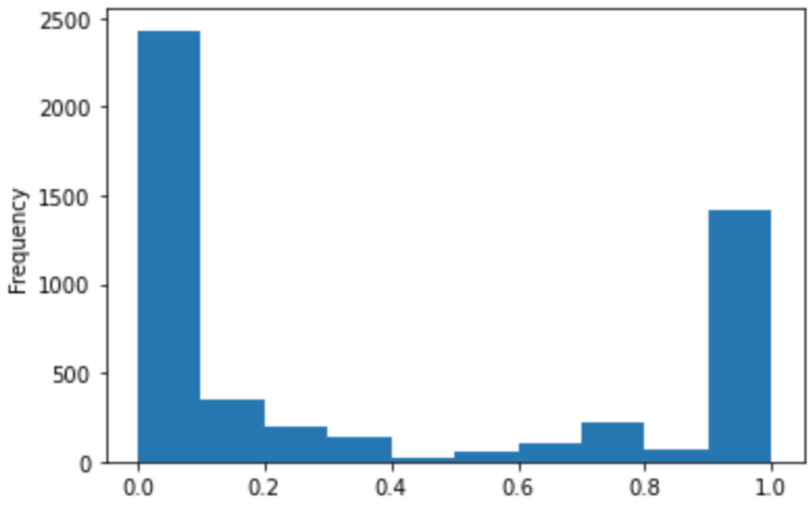
```
nefarious['PctExtResourceUrls'].plot.hist()
```

<AxesSubplot:ylabel='Frequency'>



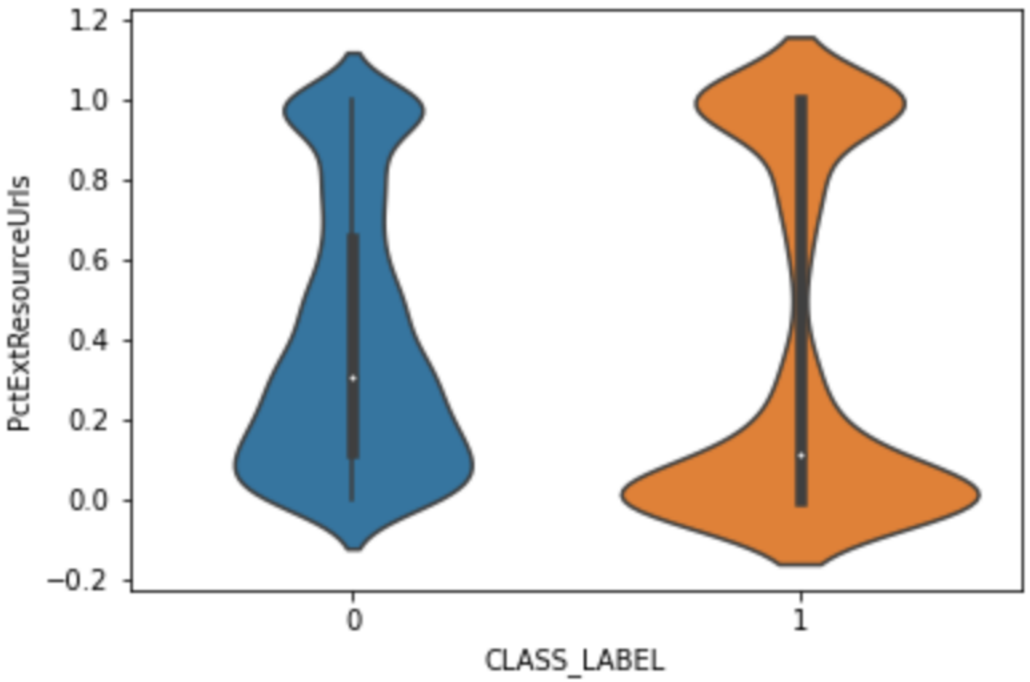

```
non_nefarious['PctExtResourceUrls'].plot.hist()
```

<AxesSubplot:ylabel='Frequency'>



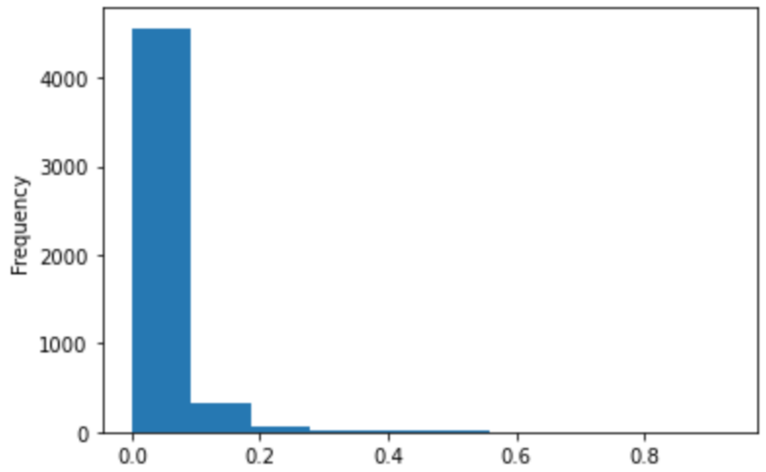

```
sns.violinplot(data=data, x='CLASS_LABEL',y=important_array[3])
```

<AxesSubplot:xlabel='CLASS_LABEL', ylabel='PctExtResourceUrls'>



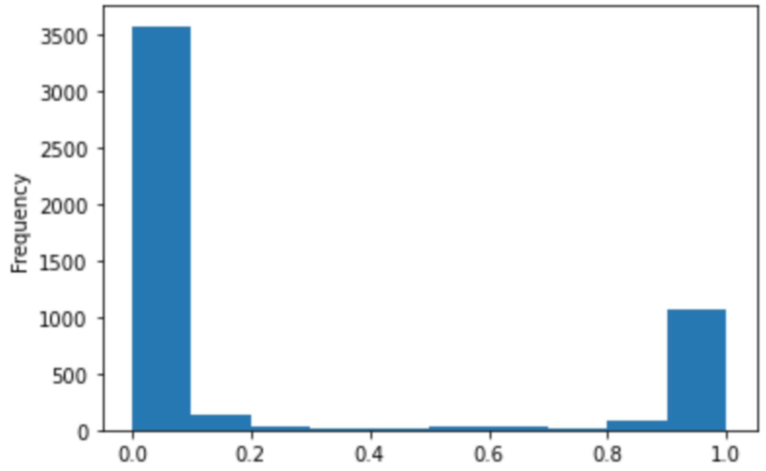
```
nefarious['PctNullSelfRedirectHyperlinks'].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



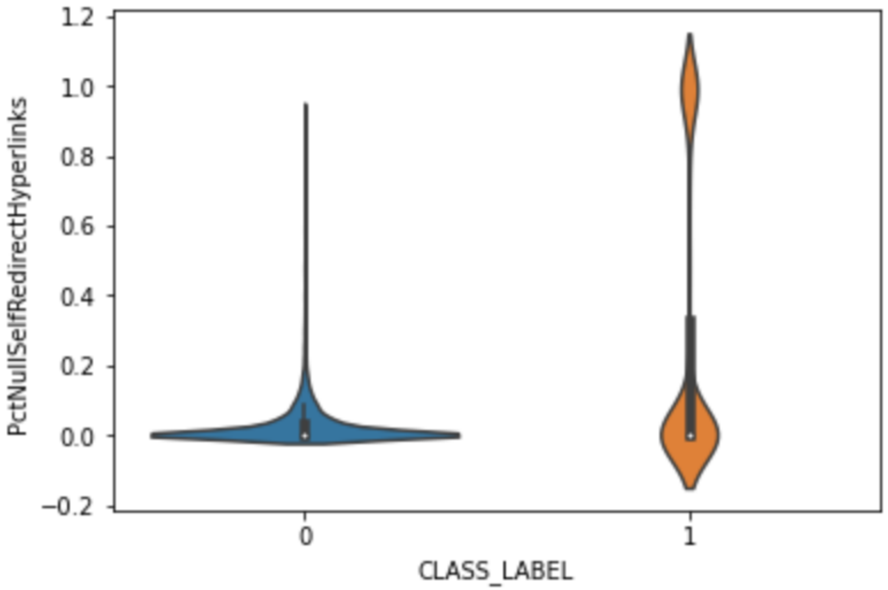

```
non_nefarious['PctNullSelfRedirectHyperlinks'].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



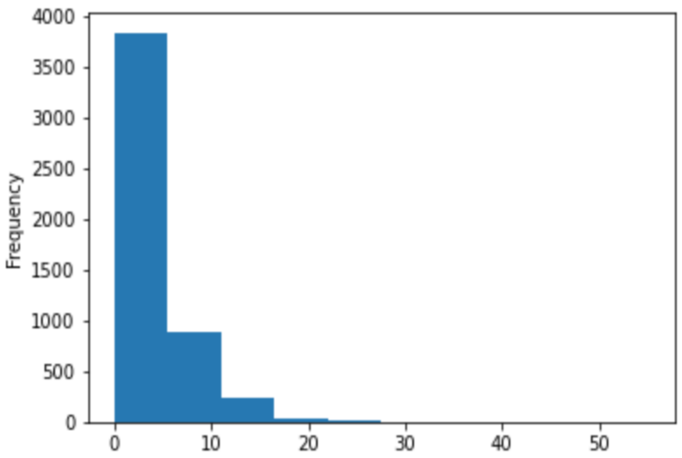

```
sns.violinplot(data=data, x='CLASS_LABEL',y=important_array[4])
```

```
<AxesSubplot:xlabel='CLASS_LABEL', ylabel='PctNullSelfRedirectHyperlinks'>
```



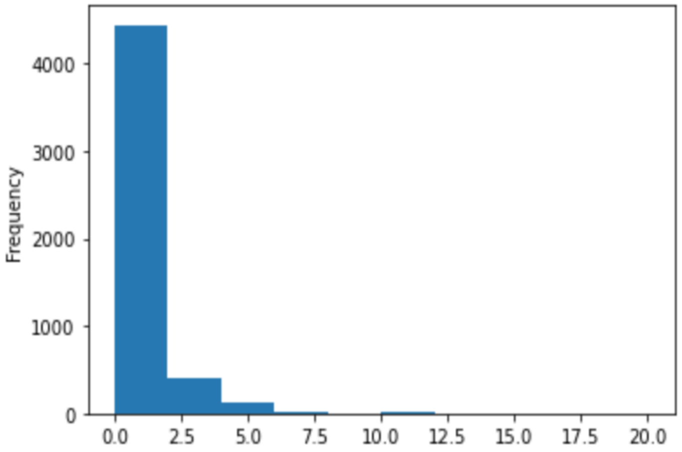
```
nefarious['NumDash'].plot.hist()
```

<AxesSubplot:ylabel='Frequency'>



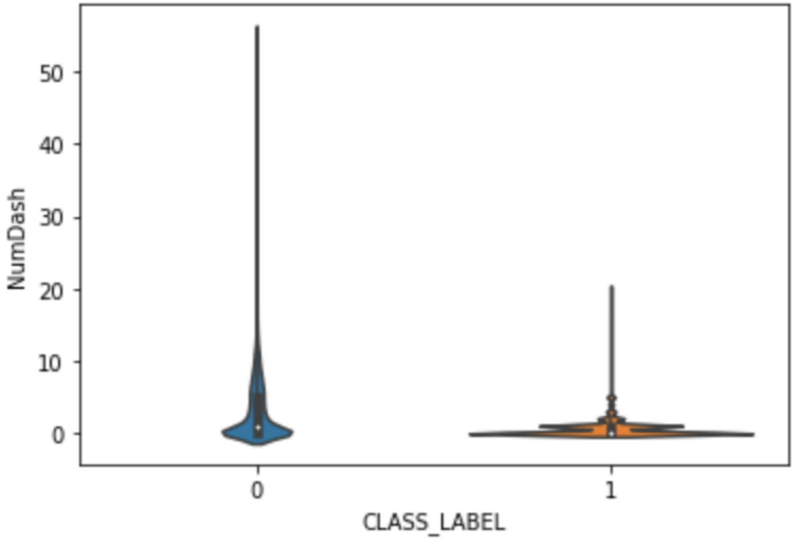

```
non_nefarious['NumDash'].plot.hist()
```

<AxesSubplot:ylabel='Frequency'>



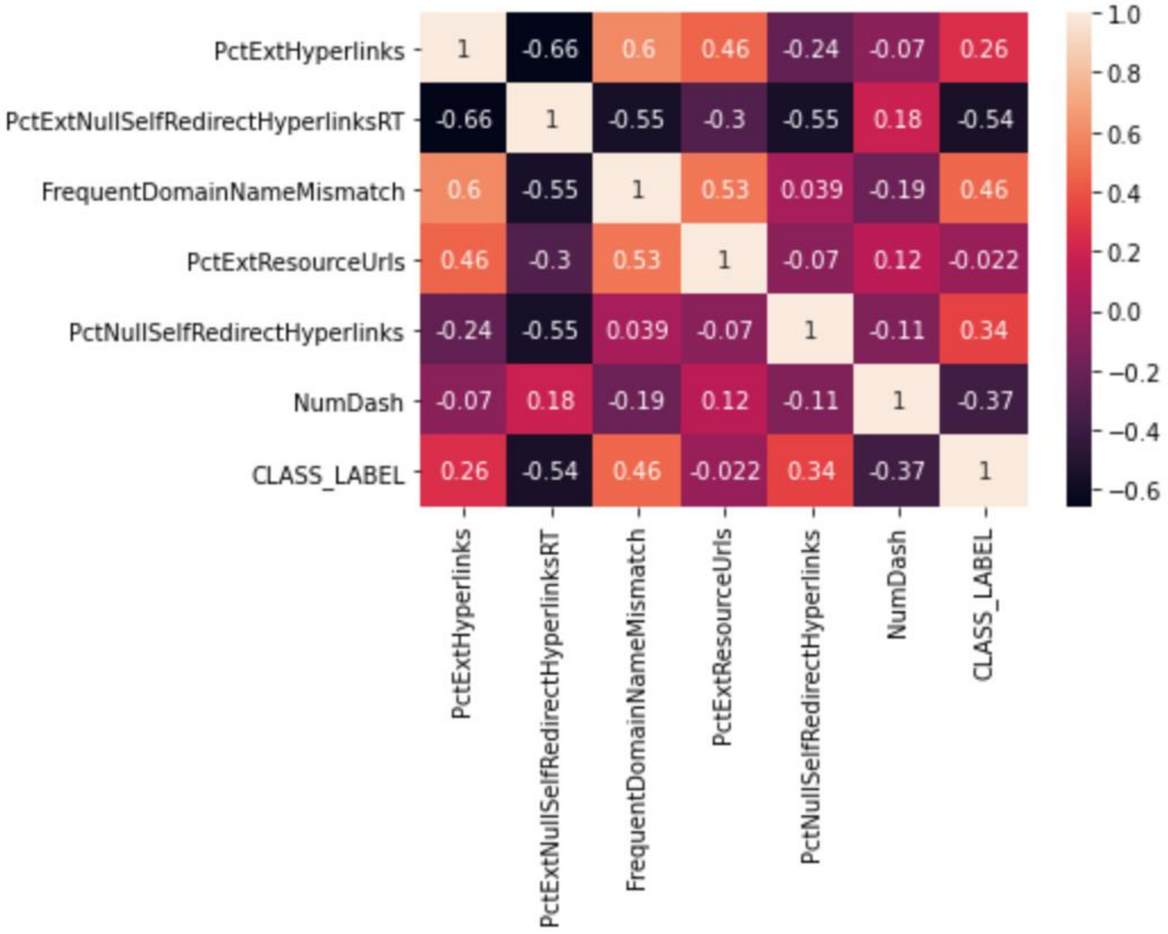

```
sns.violinplot(data=data, x='CLASS_LABEL',y=important_array[5])
```

<AxesSubplot:xlabel='CLASS_LABEL', ylabel='NumDash'>



```python
sns.heatmap(data[important_array].corr(), annot=True)
```

```
<AxesSubplot:>
```



Modeling

- RANDOM FOREST MODEL WITH GRID SEARCH
- KNN WITH GRID SEARCH
- DECISION TREE ENTROPY MODEL
- DECISION TREE GINI MODEL
- GRADIENT BOOSTING CLASSIFIER
- LIGHT GRADIENT BOOSTING WITH BAYESIAN OPTIMIZATION

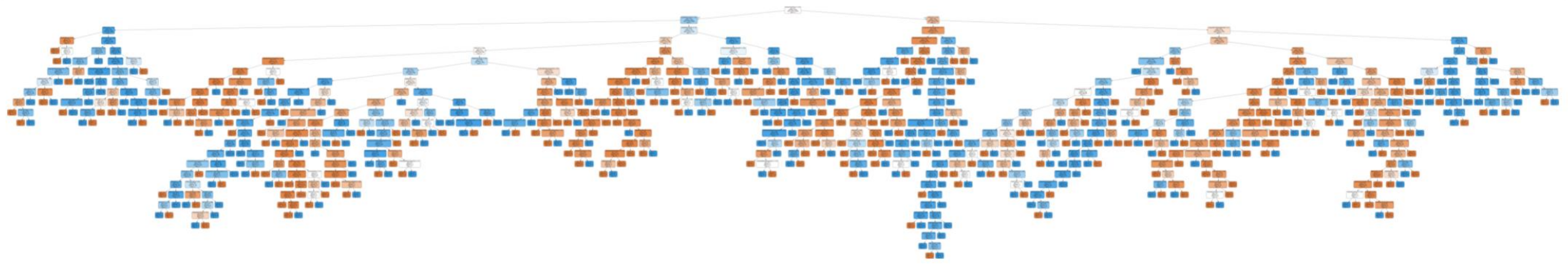
Random Forest:

Used grid search to determine optimal n_estimators of 201

```
Random Forest: Accuracy=0.982
Further Random Forest Metrics:
Balanced accuracy: 0.9815023425330571
Precision score: 0.9823529411764705
Recall score: 0.9813907933398629
```

Random Forest:

Used grid search to determine optimal n_estimators of 201



KNN:

Used grid search to determine optimal n_neighbors

```
Best Score:0.9536250000000001
Best Parameters: {'n_neighbors': 1}
```

```
neigh = KNeighborsClassifier(n_neighbors=1)
neigh.fit(X_train_scaled, y_train)
y_pred = neigh.predict(X_test_scaled)
print("KNN model:")
print("Accuracy:", metrics.accuracy_score(y_test,y_pred))
print("Balanced accuracy:", metrics.balanced_accuracy_score(y_test,y_pred))
print('Precision score:', metrics.precision_score(y_test,y_pred))
print('Recall score:', metrics.recall_score(y_test,y_pred))
```

```
KNN model:
Accuracy: 0.9625
Balanced accuracy: 0.962451941306116
Precision score: 0.9619140625
Recall score: 0.9647404505386875
```

Gini and Entropy Decision Trees

```
print("Model Gini impurity model")
print("Accuracy:", metrics.accuracy_score(y_test,y_pred))
print("Balanced accuracy:", metrics.balanced_accuracy_score(y_test,y_pred))
print('Precision score for:' , metrics.precision_score(y_test,y_pred))
print('Recall score:' , metrics.recall_score(y_test,y_pred))
```

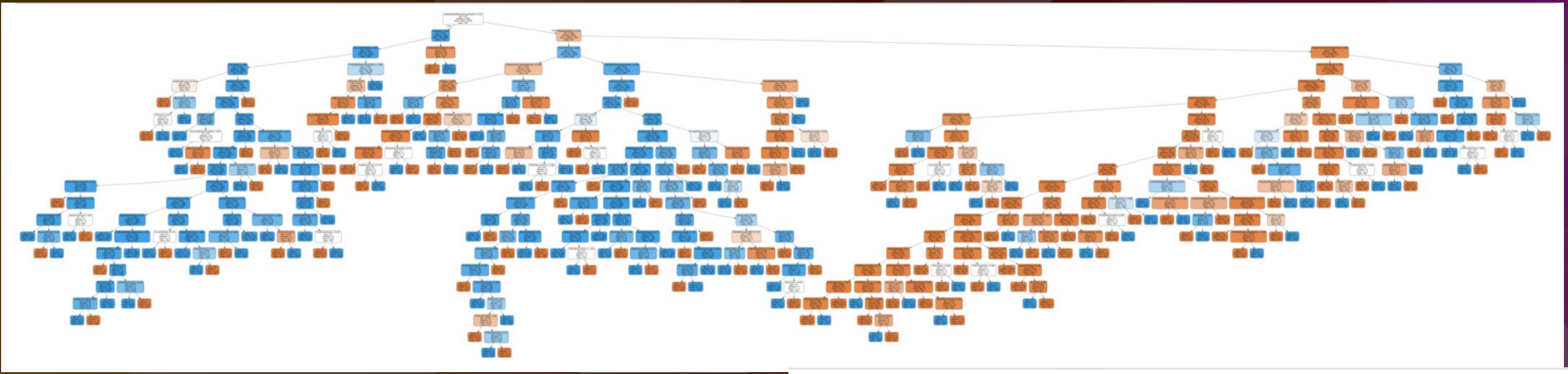
```
Model Gini impurity model
Accuracy: 0.9655
Balanced accuracy: 0.9655162926850741
Precision score for: 0.9675834970530451
Recall score: 0.9647404505386875
```

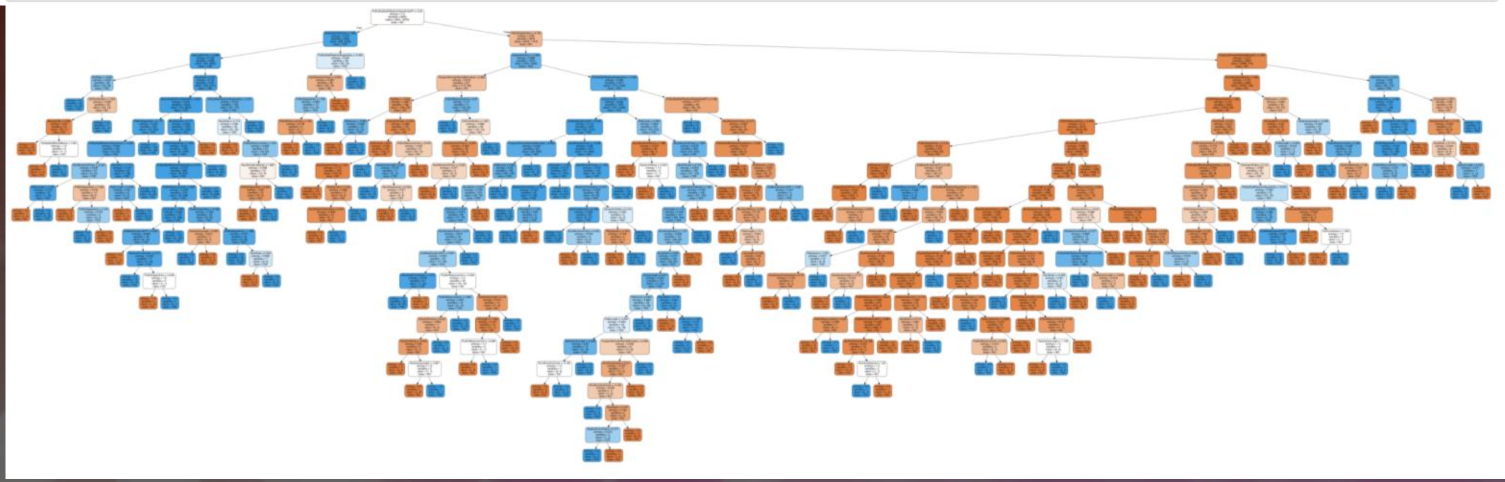
```
#The gini model is graphed below, but its accuracy, along with that of the entropy model, is below what was achieved
#with Random Forest.
Image(graph.create_png())
```

```
#Below, the Metrics for the Entropy Decision Tree Model, based on the prediction compared to the y_test value:
#Note: No Max Depth has been set:
print("Model Entropy - no max depth")
print("Accuracy:", metrics.accuracy_score(y_test,y_pred))
print("Balanced accuracy:", metrics.balanced_accuracy_score(y_test,y_pred))
print('Precision score:' , metrics.precision_score(y_test,y_pred))
print('Recall score:' , metrics.recall_score(y_test,y_pred))
```

```
Model Entropy - no max depth
Accuracy: 0.9625
Balanced accuracy: 0.9623468949806865
Precision score: 0.9574468085106383
Recall score: 0.9696376101860921
```

Gini and Entropy Decision Trees





Gradient Boosting Classifier:

```
print("Gradient Boosting Model:")
print("Accuracy:", metrics.accuracy_score(y_test,y_pred))
print("Balanced accuracy:", metrics.balanced_accuracy_score(y_test,y_pred))
print('Precision score' , metrics.precision_score(y_test,y_pred))
print('Recall score' , metrics.recall_score(y_test,y_pred))
```

```
Gradient Boosting Model:
Accuracy: 0.954
Balanced accuracy: 0.9538956679895834
Precision score 0.9514091350826045
Recall score 0.9588638589618022
```

Light Gradient Boosting:

Used Bayesian optimization to tune parameters

```
y_pred = model.predict(X_test_scaled, num_iteration=model.best_iteration)
y_pred = y_pred.round(0)
```

```
#The best model previous to this LGBM model was the Random Forest Model.
#This one shows an improvement over Random Forest in all four of the metrics being monitored.
print("Light GBM with Bayesian Parameter Tuning:")
print("Accuracy:", metrics.accuracy_score(y_test,y_pred))
print("Balanced accuracy:", metrics.balanced_accuracy_score(y_test,y_pred))
print('Precision score' , metrics.precision_score(y_test,y_pred))
print('Recall score' , metrics.recall_score(y_test,y_pred))
```

```
Light GBM with Bayesian Parameter Tuning:
Accuracy: 0.9855
Balanced accuracy: 0.9854620887811525
Precision score 0.984375
Recall score 0.9872673849167483
```

Predictions

	Accuracy:	Balanced Accuracy:	Precision Score:	Recall Score:
Random Forest (optimized by a grid search) - fluctuates	0.982	0.9815	0.9823	0.9813
KNN (optimized by grid search	0.9625	0.9624	0.9619	0.9647
Entropy Decision Tree	0.9625	0.9623	0.9574	0.9696
Gini Decision Tree	0.9655	0.9655	0.9675	0.9647
Gradient Boosting Classifier	0.954	0.9538	0.9514	0.9588
Light Gradient Boosting Model (LGBM - optimized by Bayesian Optimization) - fluctuates	0.9855	0.9854	0.9843	0.9872

- The Random Forest Model performed second best overall.
- The Light GBM performed best in accuracy, precision and recall when making predictions.
- Recall is an important indicator given the importance of low false negatives in phishing detection.