```
#importing basic packages
import pandas as pd
import numpy as np
import seaborn as sns
import re
import joblib
import matplotlib.pyplot as plt
```

```
#Loading the data
from google.colab import files
uploaded = files.upload()
```

Choose Files Phishing_U…Dataset.csv

- **Phishing_URL_Dataset.csv**(text/csv) - 56854345 bytes, last modified: 3/24/2025 - 100% done

```
Saving Phishing_URL_Dataset.csv to Phishing_URL_Dataset.csv
```

```
#Reading the uploaded file
df = pd.read_csv('Phishing_URL_Dataset.csv', encoding='latin-1', on_bad_lines='skip')
```

Obtained from; https://archive.ics.uci.edu/dataset/967/phiusiil+phishing+url+dataset.

PhiUSIIL Phishing URL Dataset is a substantial dataset comprising 134,850 legitimate and 100,945 phishing URLs. Most of the URLs we analyzed, while constructing the dataset, are the latest URLs. Features are extracted from the source code of the webpage and URL. Features such as CharContinuationRate, URLTitleMatchScore, URLCharProb, and TLDLegitimateProb are derived from existing features.

```
df.head()
```

	ï»¿FILENAME	URL	URLLength	Domain	Doma:
0	521848.txt	https://www.southbankmosaics.com	31	www.southbankmosaics.com	
1	31372.txt	https://www.uni-mainz.de	23	www.uni-mainz.de	
2	597387.txt	https://www.voicefmradio.co.uk	29	www.voicefmradio.co.uk	
3	554095.txt	https://www.sfnmjournal.com	26	www.sfnmjournal.com	
4	151578.txt	https://www.rewildingargentina.org	33	www.rewildingargentina.org	

5 rows × 56 columns

```
df.sample(20)
```

	ï»¿FILENAME	URL	URLLength	
49105	133965.txt	https://www.iwra.org	19	
148223	103499.txt	https://www.incredibuild.com	27	w
189235	720286.txt	https://www.pjd.ma	17	
188225	42574.txt	https://www.mountain-forecast.com	32	www.mo
157956	461706.txt	https://www.darts1.de	20	
217266	8118962.txt	https://u33635890.ct.sendgrid.net/ls/click?upn...	748	u3363
156672	8043731.txt	https://olipichinch--pinchechiolin.repl.co/	43	olipichinch--
8420	161599.txt	https://www.reed.senate.gov	26	w
147356	8061210.txt	https://aol-104204.weeblysite.com/	34	aol-104
202021	mw42588.txt	http://www.uswest.cpufan.club	28	www
194400	8130504.txt	https://objectstorage.ap-tokyo-1.oraclecloud.c...	173	obje
74163	8114319.txt	https://site.appmarketing.com.br/wp-content/th...	82	site.a
204745	808424.txt	https://www.strother-nuckels.com	31	www.s
114886	8082586.txt	http://www.rakoten-cacd.xjeoiyl.cn/	35	www.rak
159712	120530.txt	https://www.wsipp.wa.gov	23	
34577	mw24258.txt	http://www.kvm1.j963289.n5zdn.vps.myjino.ru	42	www.kvm1.j963289.
173372	8109484.txt	https://www.drect-smtb.jp.ap1.ib.xuanglasses.c...	72	smtb.jp.ap1.
74295	401134.txt	https://www.cineytele.com	24	
211322	848901.txt	https://www.nfdw.com	19	
194165	8046716.txt	https://guiacpf.com.br/	23	

20 rows × 56 columns

```
df.shape
```

```
(235795, 56)
```

```
df.info()
```

```
 2   URLLength                   235795 non-null  int64
 3   Domain                      235795 non-null  object
 4   DomainLength                235795 non-null  int64
 5   IsDomainIP                  235795 non-null  int64
 6   TLD                         235795 non-null  object
 7   URLSimilarityIndex          235795 non-null  float64
 8   CharContinuationRate        235795 non-null  float64
 9   TLDLegitimateProb           235795 non-null  float64
 10  URLCharProb                 235795 non-null  float64
 11  TLDLength                   235795 non-null  int64
 12  NoOfSubDomain               235795 non-null  int64
 13  HasObfuscation              235795 non-null  int64
 14  NoOfObfuscatedChar          235795 non-null  int64
 15  ObfuscationRatio            235795 non-null  float64
 16  NoOfLettersInURL            235795 non-null  int64
 17  LetterRatioInURL            235795 non-null  float64
 18  NoOfDegitsInURL             235795 non-null  int64
 19  DegitRatioInURL             235795 non-null  float64
 20  NoOfEqualsInURL             235795 non-null  int64
 21  NoOfQMarkInURL              235795 non-null  int64
 22  NoOfAmpersandInURL          235795 non-null  int64
 23  NoOfOtherSpecialCharsInURL  235795 non-null  int64
 24  SpacialCharRatioInURL       235795 non-null  float64
 25  IsHTTPS                     235795 non-null  int64
 26  LineOfCode                  235795 non-null  int64
 27  LargestLineLength           235795 non-null  int64
 28  HasTitle                    235795 non-null  int64
 29  Title                       235795 non-null  object
 30  DomainTitleMatchScore       235795 non-null  float64
 31  URLTitleMatchScore          235795 non-null  float64
 32  HasFavicon                  235795 non-null  int64
 33  Robots                      235795 non-null  int64
 34  IsResponsive                235795 non-null  int64
 35  NoOfURLRedirect             235795 non-null  int64
 36  NoOfSelfRedirect            235795 non-null  int64
 37  HasDescription              235795 non-null  int64
 38  NoOfPopup                   235795 non-null  int64
 39  NoOfiFrame                  235795 non-null  int64
 40  HasExternalFormSubmit       235795 non-null  int64
 41  HasSocialNet                235795 non-null  int64
 42  HasSubmitButton             235795 non-null  int64
 43  HasHiddenFields             235795 non-null  int64
 44  HasPasswordField            235795 non-null  int64
 45  Bank                        235795 non-null  int64
 46  Pay                         235795 non-null  int64
 47  Crypto                      235795 non-null  int64
 48  HasCopyrightInfo            235795 non-null  int64
 49  NoOfImage                   235795 non-null  int64
 50  NoOfCSS                     235795 non-null  int64
 51  NoOfJS                      235795 non-null  int64
 52  NoOfSelfRef                 235795 non-null  int64
 53  NoOfEmptyRef                235795 non-null  int64
 54  NoOfExternalRef             235795 non-null  int64
 55  label                       235795 non-null  int64
dtypes: float64(10), int64(41), object(5)
memory usage: 100.7+ MB
```

```
df.columns
```

```
Index(['ï»¿FILENAME', 'URL', 'URLLength', 'Domain', 'DomainLength',
       'IsDomainIP', 'TLD', 'URLSimilarityIndex', 'CharContinuationRate',
       'TLDLegitimateProb', 'URLCharProb', 'TLDLength', 'NoOfSubDomain',
       'HasObfuscation', 'NoOfObfuscatedChar', 'ObfuscationRatio',
       'NoOfLettersInURL', 'LetterRatioInURL', 'NoOfDegitsInURL',
       'DegitRatioInURL', 'NoOfEqualsInURL', 'NoOfQMarkInURL',
       'NoOfAmpersandInURL', 'NoOfOtherSpecialCharsInURL',
       'SpacialCharRatioInURL', 'IsHTTPS', 'LineOfCode', 'LargestLineLength',
       'HasTitle', 'Title', 'DomainTitleMatchScore', 'URLTitleMatchScore',
       'HasFavicon', 'Robots', 'IsResponsive', 'NoOfURLRedirect',
       'NoOfSelfRedirect', 'HasDescription', 'NoOfPopup', 'NoOfiFrame',
       'HasExternalFormSubmit', 'HasSocialNet', 'HasSubmitButton',
       'HasHiddenFields', 'HasPasswordField', 'Bank', 'Pay', 'Crypto',
       'HasCopyrightInfo', 'NoOfImage', 'NoOfCSS', 'NoOfJS', 'NoOfSelfRef',
       'NoOfEmptyRef', 'NoOfExternalRef', 'label'],
      dtype='object')
```

Data Cleaning

```
# 1. Remove non-numeric irrelevant columns that can't be used for model training
df_cleaned = df.drop(columns=['ï»¿FILENAME', 'URL', 'Domain', 'Title'])
```

Irrelevant Columns: Removed columns such as FILENAME, URL, Domain, and Title, which are not useful for classification.

```
# For simplicity, we'll drop rows with missing values. Alternatively, you can fill missing v
df_cleaned = df_cleaned.dropna()  # Remove rows with missing values
```

```
# 2. Check for and remove duplicate rows
df_cleaned = df_cleaned.drop_duplicates()
```

Duplicates: Duplicates were identified and removed, ensuring that each observation in the dataset is unique. This avoids any bias in model training caused by repeated data.

```
# 3. Check for missing values
missing_values = df_cleaned.isnull().sum()
```

```
# 5. Ensure proper data types
df_cleaned.dtypes
```

	0
URLLength	int64
DomainLength	int64
IsDomainIP	int64
TLD	object
URLSimilarityIndex	float64
CharContinuationRate	float64
TLDLegitimateProb	float64
URLCharProb	float64
TLDLength	int64
NoOfSubDomain	int64
HasObfuscation	int64
NoOfObfuscatedChar	int64
ObfuscationRatio	float64
NoOfLettersInURL	int64
LetterRatioInURL	float64
NoOfDegitsInURL	int64
DegitRatioInURL	float64
NoOfEqualsInURL	int64
NoOfQMarkInURL	int64
NoOfAmpersandInURL	int64
NoOfOtherSpecialCharsInURL	int64
SpacialCharRatioInURL	float64
IsHTTPS	int64
LineOfCode	int64
LargestLineLength	int64
HasTitle	int64
DomainTitleMatchScore	float64
URLTitleMatchScore	float64
HasFavicon	int64
Robots	int64

IsResponsive	int64
NoOfURLRedirect	int64
NoOfSelfRedirect	int64
HasDescription	int64
NoOfPopup	int64
NoOfiFrame	int64
HasExternalFormSubmit	int64
HasSocialNet	int64
HasSubmitButton	int64
HasHiddenFields	int64
HasPasswordField	int64
Bank	int64
Pay	int64
Crypto	int64
HasCopyrightInfo	int64
NoOfImage	int64
NoOfCSS	int64
NoOfJS	int64
NoOfSelfRef	int64
NoOfEmptyRef	int64
NoOfExternalRef	int64
label	int64

dtype: object

Data Types: Ensured that all columns have the correct data type (integers for numerical features, etc.).

```
# Display the cleaned dataframe and missing values
df_cleaned.head(), missing_values
```

```
2      7          42           2             5      1
3     15          22           1            31      1
4     34          72           1            85      1

[5 rows x 52 columns],
URLLength                   0
DomainLength                0
IsDomainIP                  0
TLD                         0
URLSimilarityIndex          0
CharContinuationRate        0
TLDLegitimateProb           0
URLCharProb                 0
TLDLength                   0
NoOfSubDomain               0
HasObfuscation              0
NoOfObfuscatedChar          0
ObfuscationRatio            0
NoOfLettersInURL            0
LetterRatioInURL            0
NoOfDegitsInURL             0
DegitRatioInURL             0
NoOfEqualsInURL             0
```

```
Pay                     0
Crypto                  0
HasCopyrightInfo        0
NoOfImage               0
NoOfCSS                 0
NoOfJS                  0
NoOfSelfRef             0
NoOfEmptyRef            0
NoOfExternalRef         0
label                   0
dtype: int64)
```

Missing Values: The dataset was checked for missing values, and since there were no missing entries, we moved forward with no imputation or removal of rows due to missing values.

```
df['label']
```

	label
0	1
1	1
2	1
3	1
4	1
...	...
235790	1
235791	1
235792	1
235793	0
235794	1

235795 rows × 1 columns

dtype: int64

```
df.sample(10)
```

	ï»¿FILENAME	URL	URLLength	
156568	146314.txt	https://www.nic.sh	17	www
128468	8090896.txt	https://aol-mail-109688.weeblysite.com/	39	a 109688.weeblys
68411	oph12157.txt	http://organisasi.bulungan.go.id/public/wjesho...	72	organisasi.bulunga
60620	mw131783.txt	http://www.cena-iran.ml	23	www.cena
104841	mw179866.txt	http://www.51she.info	21	www.51s
117207	8092122.txt	https://valeu-lojas-online.myshopify.com/produ...	120	vale online.myshop
63750	mw73571.txt	http://www.arsels.info	21	www.ars
105756	mw68727.txt	http://www.atlantisads.com	25	www.atlantisa
29859	697851.txt	https://www.gothamgreens.com	27	www.gothamgree
42771	8135560.txt	https://quickrectifier.vercel.app/wallets	41	quickrectifier.ver

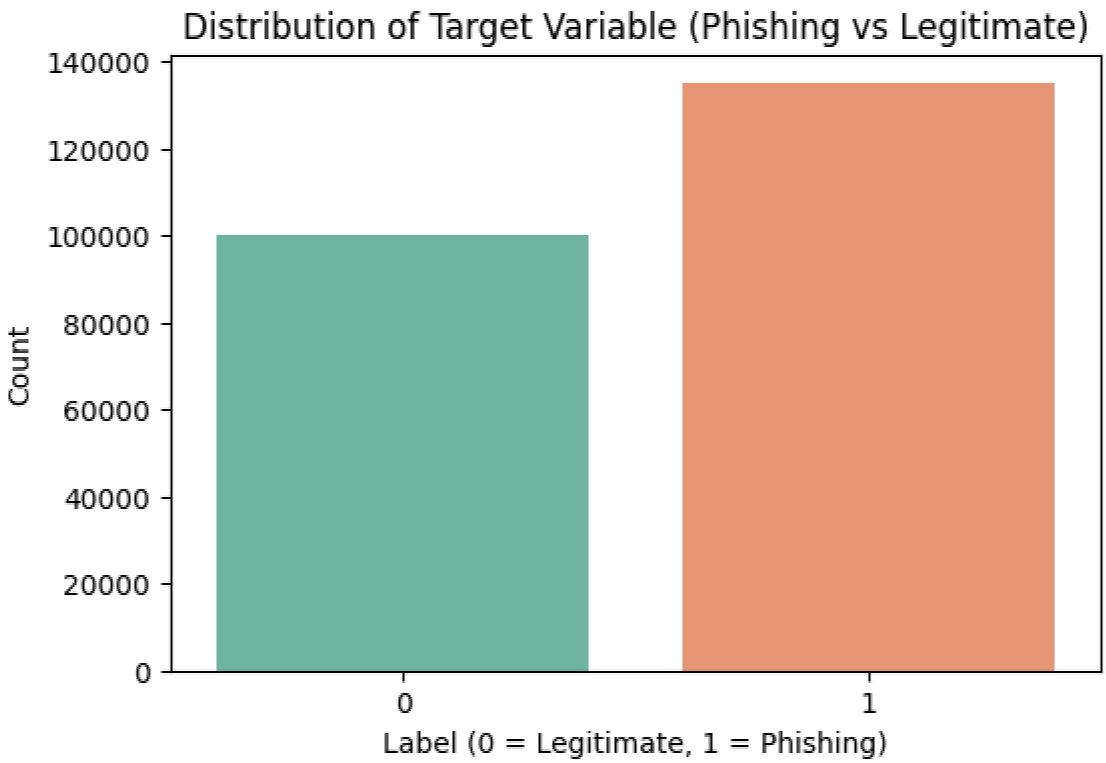
10 rows × 56 columns

Exploratory Data Analysis (EDA) *Explore key features and relationships.*

Target Distribution: The label column, which classifies URLs as phishing (1) or legitimate (0), shows an imbalanced distribution. Phishing URLs (labeled 1) dominate the dataset. This class imbalance might affect model performance, requiring techniques like oversampling or adjusting class weights to improve performance for the minority class (legitimate URLs).

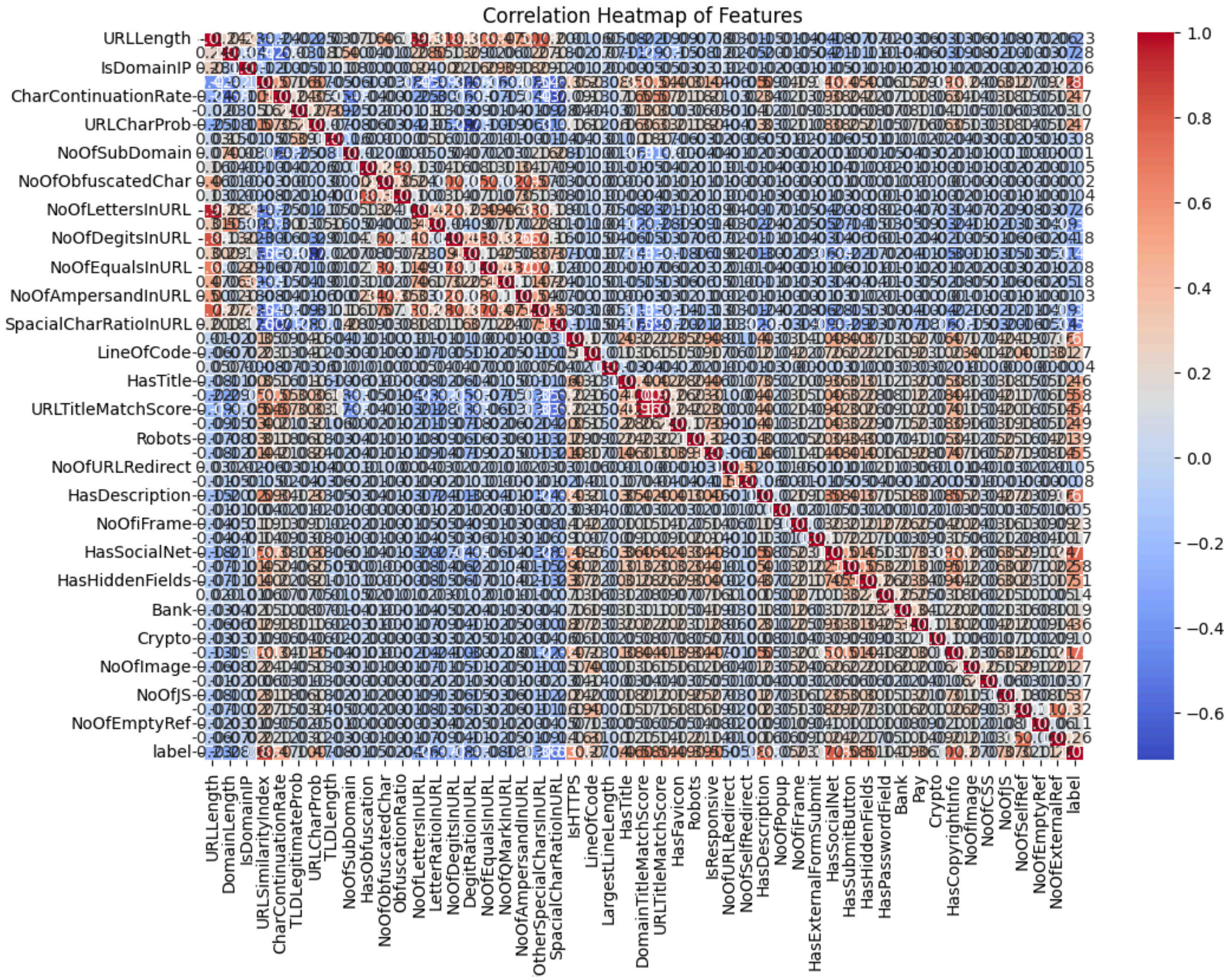
```
# Distribution of the target variable:
plt.figure(figsize=(6, 4))
sns.countplot(x='label', data=df_cleaned, palette='Set2')
plt.title('Distribution of Target Variable (Phishing vs Legitimate)')
plt.xlabel('Label (0 = Legitimate, 1 = Phishing)')
plt.ylabel('Count')
plt.show()
```

```
<ipython-input-23-ef37b1277484>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

  sns.countplot(x='label', data=df_cleaned, palette='Set2')
```



Correlation Heatmap: The correlation heatmap revealed several strong correlations between numerical features, particularly features related to URL length and domain characteristics. This suggests that certain features may provide redundant information, and careful feature selection could help streamline the model and reduce overfitting.

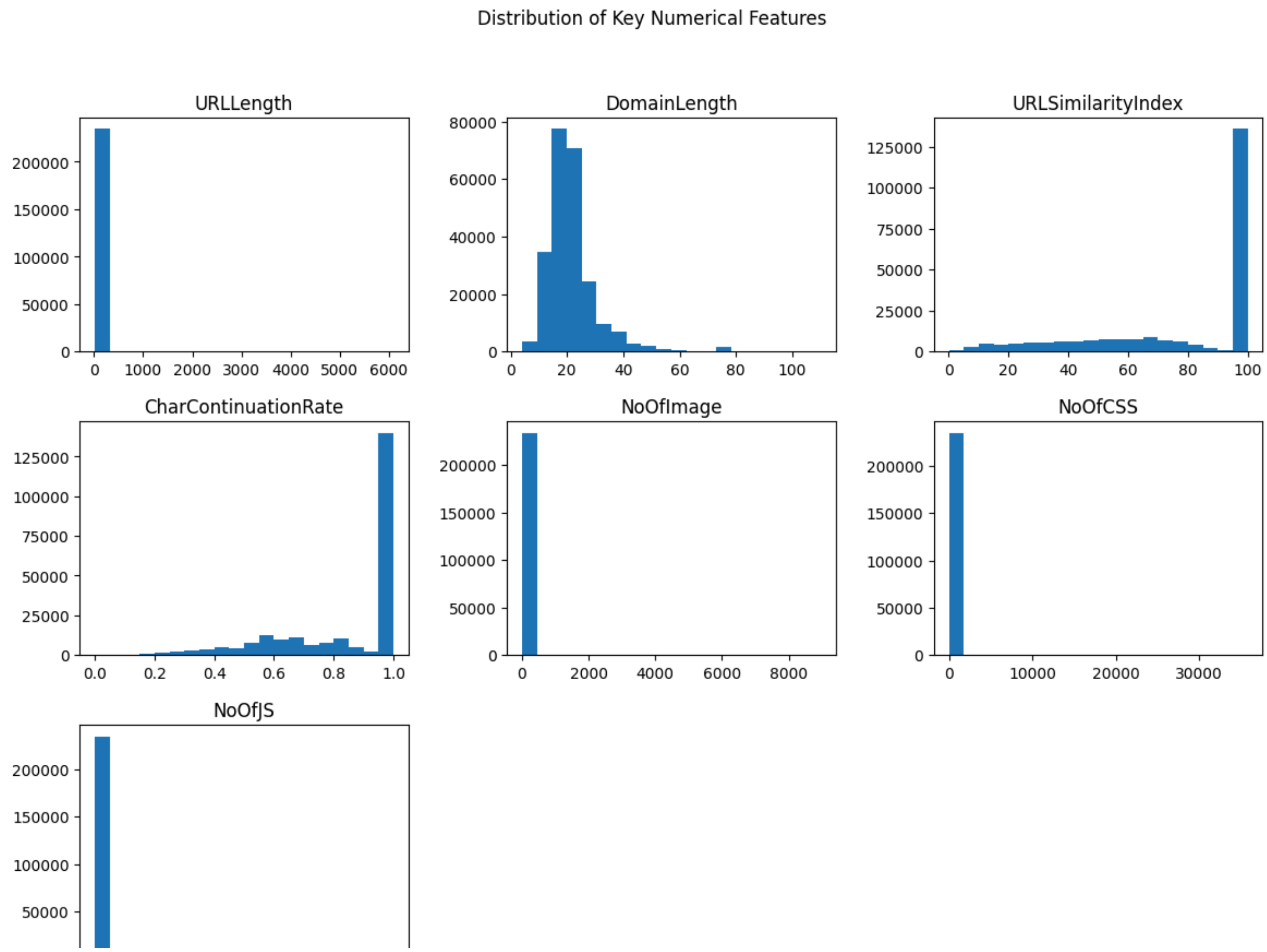
```
# Compute the correlation matrix
# Select only numeric columns before calculating correlation
correlation_matrix = df_cleaned.select_dtypes(include=np.number).corr()
# Plot the correlation matrix as a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidth
plt.title('Correlation Heatmap of Features')
plt.show()
```



Feature Distribution: URLLength and DomainLength: These features exhibited different ranges across phishing and legitimate URLs, which could be useful in distinguishing between the two classes. Longer URLs and domain names may be indicative of phishing attempts.

NoOfImage, NoOfCSS, NoOfJS: The presence of images, CSS, and JavaScript files showed variability across phishing and legitimate URLs, indicating that phishing websites may use more complex designs, which is a common strategy to appear legitimate.

```
#Distribution of key numerical features
key_features = ['URLLength', 'DomainLength', 'URLSimilarityIndex', 'CharContinuat
df_cleaned[key_features].hist(bins=20, figsize=(14, 10), grid=False)
plt.suptitle('Distribution of Key Numerical Features')
plt.show()
```

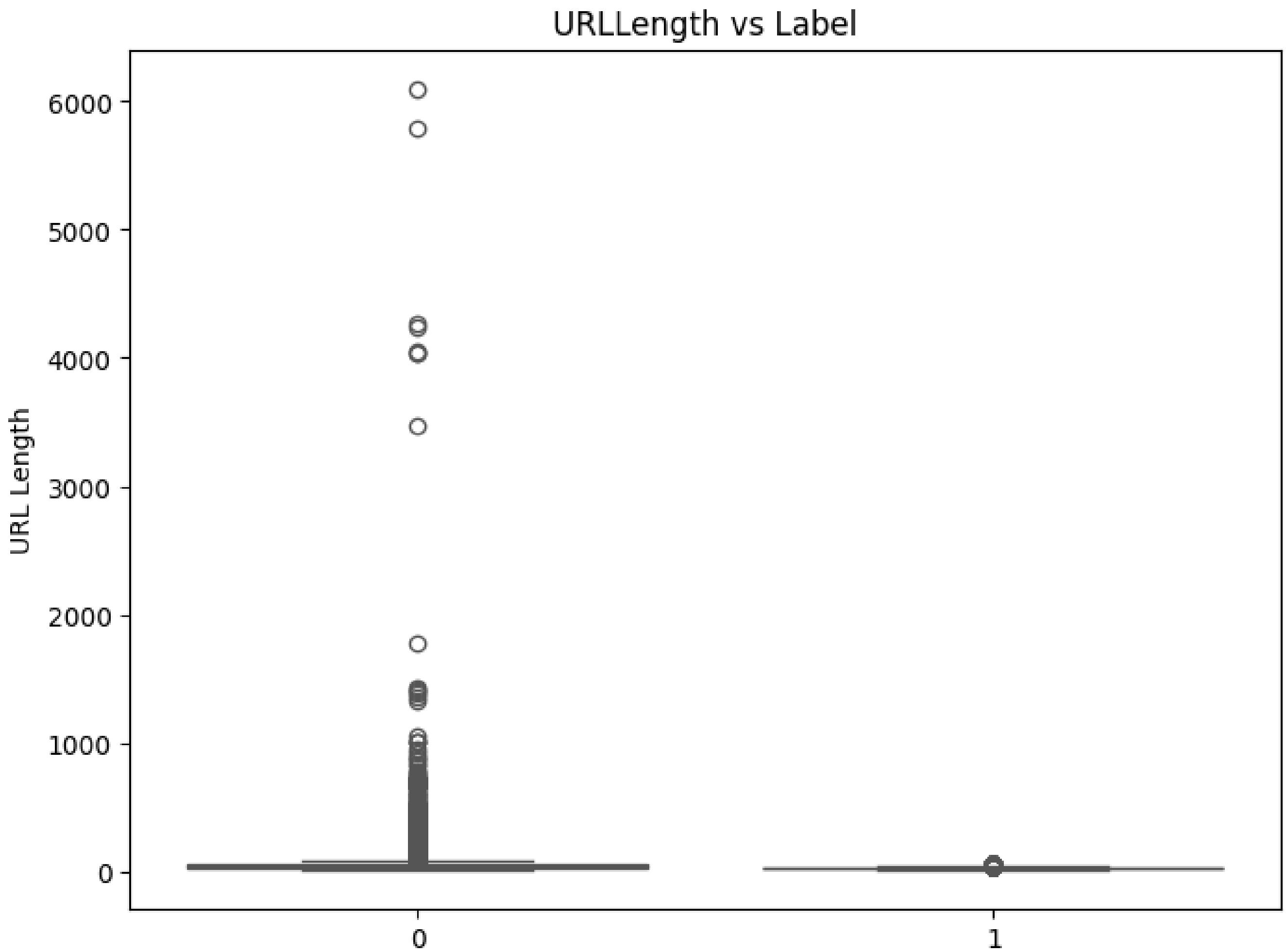


Feature vs Target Visualization: Boxplots for URLLength and DomainLength: These plots show that phishing URLs tend to have longer URL lengths and domain names than legitimate ones. This could suggest that phishing websites often use more complex URLs to confuse users.

Boxplot for NoOfImage: The number of images appears to vary between phishing and legitimate URLs, with phishing sites using more images. This aligns with common strategies used in phishing sites to mimic real websites.

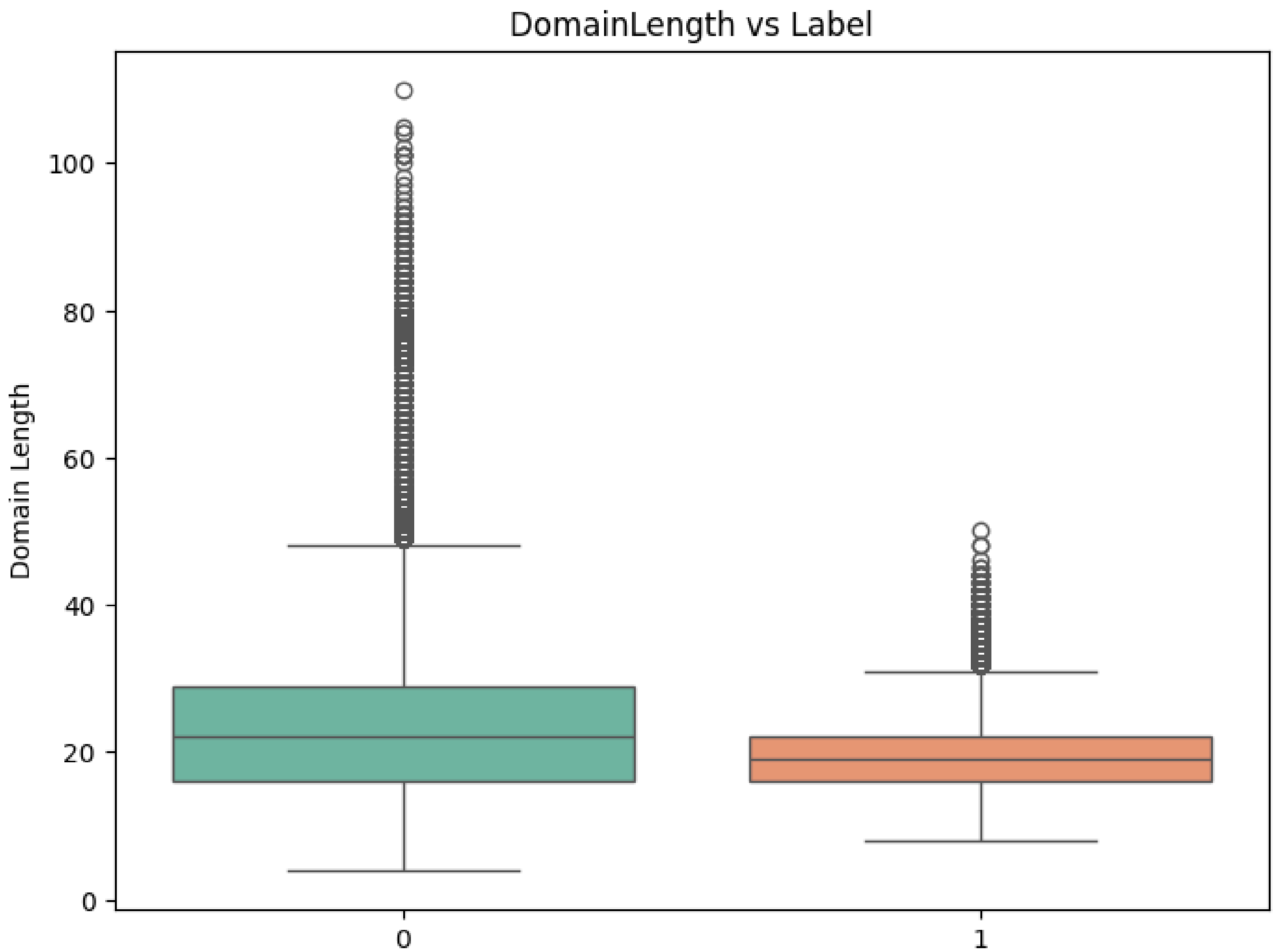
```
# Visualizing relationships between features and the target variable (label)
# Visualizing 'URLLength' vs 'label'
plt.figure(figsize=(8, 6))
sns.boxplot(x='label', y='URLLength', data=df_cleaned, palette='Set2')
plt.title('URLLength vs Label')
plt.xlabel('Label (0 = Legitimate, 1 = Phishing)')
plt.ylabel('URL Length')
plt.show()
```

```
<ipython-input-26-0aede6df14f2>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

  sns.boxplot(x='label', y='URLLength', data=df_cleaned, palette='Set2')
```



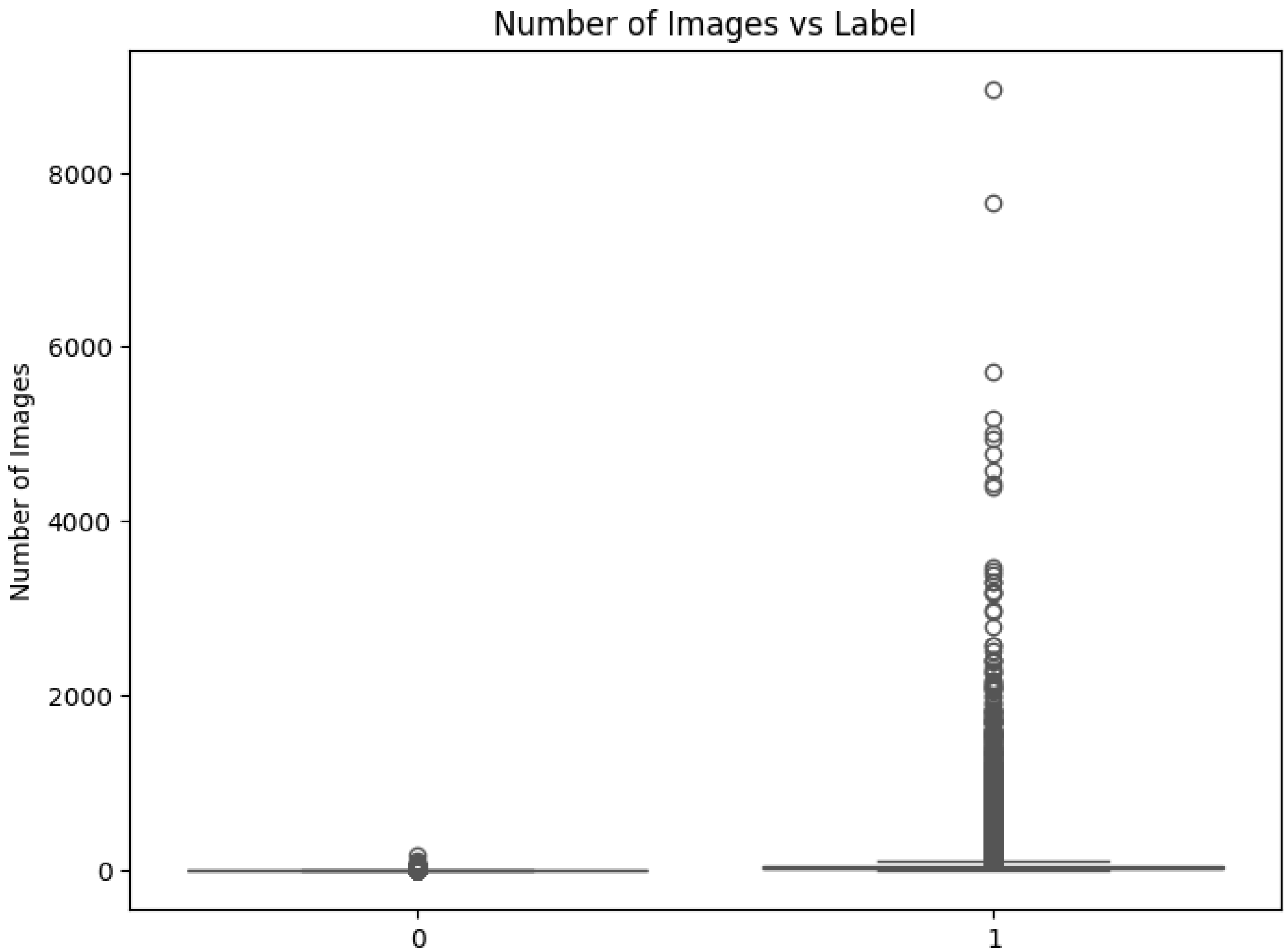

```
# Visualizing 'DomainLength' vs 'label'
plt.figure(figsize=(8, 6))
sns.boxplot(x='label', y='DomainLength', data=df_cleaned, palette='Set2')
plt.title('DomainLength vs Label')
plt.xlabel('Label (0 = Legitimate, 1 = Phishing)')
plt.ylabel('Domain Length')
plt.show()
```

```
<ipython-input-27-1afb258e9acd>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

  sns.boxplot(x='label', y='DomainLength', data=df_cleaned, palette='Set2')
```




```
plt.figure(figsize=(8, 6))
sns.boxplot(x='label', y='NoOfImage', data=df_cleaned, palette='Set2')
plt.title('Number of Images vs Label')
plt.xlabel('Label (0 = Legitimate, 1 = Phishing)')
plt.ylabel('Number of Images')
plt.show()
```

```
<ipython-input-28-f4bb8e8c2b18>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

  sns.boxplot(x='label', y='NoOfImage', data=df_cleaned, palette='Set2')
```



Data Preprocessing

Data preprocessing is a crucial step before model training. It includes tasks such as handling categorical features, scaling numerical data, dealing with class imbalance, splitting the dataset into training and testing sets, and more. Here's how we'll proceed with preprocessing:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import resample


scaler = StandardScaler()


# In this dataset, we already removed non-numeric columns, so no categorical encoding is nee
# Based on EDA, we can keep all the numerical features for now, but we could remove highly c
```

```
# We'll continue using the cleaned dataset from the previous steps.
# Identify numeric columns (we only have numerical columns remaining after cleaning)
numeric_columns = df_cleaned.select_dtypes(include=['int64', 'float64']).columns

# Standardize the numerical features


# Standardize the numerical features
X = df_cleaned[numeric_columns]
y = df_cleaned['label']

X_scaled = scaler.fit_transform(X)



# 4. Addressing Class Imbalance
# We will use random oversampling to balance the classes.
# First, concatenate the features and target for easy manipulation
df_balanced = pd.concat([pd.DataFrame(X_scaled), y], axis=1)


# Separate the minority and majority classes
df_majority = df_balanced[df_balanced['label'] == 1]
df_minority = df_balanced[df_balanced['label'] == 0]


# Upsample the minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,     # Sample with replacement
                                 n_samples=len(df_majority),  # Match the majority class siz
                                 random_state=42)  # For reproducibility


# Combine the majority class with the upsampled minority class
df_balanced_upsampled = pd.concat([df_majority, df_minority_upsampled])


# Separate the features and target again after upsampling
X_balanced = df_balanced_upsampled.drop(columns=['label'])
y_balanced = df_balanced_upsampled['label']



# 5. Train-Test Split
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_balanced, y_balanced, test_size=0.2, r


# 3. Scaling Numerical Features (Standardization)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)  # Use the same scaler to transform the test set
```

```
#6. Outlier Handling (if needed)
# We already used z-scores during EDA to handle outliers, so this step has been h

# Display the processed data information
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((215760, 51), (53940, 51), (215760,), (53940,))
```

This preprocessing ensures that the data is properly prepared for model training, with balanced classes and standardized features. Key Steps: Handling Categorical Data: Since the dataset consists of only numerical features, we don't need to encode categorical columns. All features are ready for model use.

Feature Selection: We retain all features for now as we found useful information in the features during EDA. Feature engineering can be applied later if necessary.

Scaling Numerical Features: All numerical features are standardized using StandardScaler to ensure they have a mean of 0 and a standard deviation of 1.

Class Imbalance: The dataset has an imbalance, with phishing URLs (1) being more frequent than legitimate URLs (0). To address this, we oversample the minority class (legitimate URLs) using resample to ensure the classes are balanced in the training data.

Train-Test Split: The dataset is split into 80% training data and 20% testing data to evaluate the model effectively.

Outlier Handling: Outliers were previously handled in the EDA step using z-scores, so we don't need additional handling here.

Model Training

Now that the data preprocessing is complete, we can move forward with the model training phase. This step involves choosing the appropriate machine learning model, training it using the prepared dataset, evaluating its performance, and fine-tuning the model as needed. Below are the steps involved in the model training process:

Random Forest

For a binary classification task like this one (phishing vs legitimate URLs), we can choose from several algorithms. In this case, we will use a Random Forest Classifier, which is a powerful ensemble learning method based on decision trees. Random Forest is a great choice for handling both classification and regression tasks, as it performs well with imbalanced data, is robust to overfitting, and can handle both numerical and categorical features.

```
import time


from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC


from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error
from sklearn.metrics import classification_report
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import precision_recall_curve
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve, roc_auc_score , auc
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer




# 1. Train and Evaluate Random Forest Classifier
rfc_model = RandomForestClassifier(n_estimators=100, random_state=42)
rfc_model.fit(X_train_scaled, y_train)
rfc_y_pred = rfc_model.predict(X_test_scaled)


# 2. Train and Evaluate XGBoost Classifier
xgb_model = XGBClassifier(random_state=42)
xgb_model.fit(X_train_scaled, y_train)
xgb_y_pred = xgb_model.predict(X_test_scaled)
```

```
# Assuming X and y are your original data
# 1. Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean')  # Replace with your preferred strategy
X_imputed = imputer.fit_transform(X)

# 2. Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X_imputed, y, test_size=0.2, random_state=42, stratify=y
)

# 3. Scale numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 4. Train the SVM model
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train_scaled, y_train)
```

```
SVC
SVC(kernel='linear', random_state=42)
```

```
# Function to compute metrics and plot confusion matrix
def evaluate_model(y_test, y_pred, model_name):
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Display metrics
    print(f"{model_name} - Accuracy: {accuracy:.2f}")
    print(f"{model_name} - Precision: {precision:.2f}")
    print(f"{model_name} - Recall: {recall:.2f}")
    print(f"{model_name} - F1: {f1:.2f}")

    # Plot confusion matrix
    plt.figure(figsize=(6, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Legitimate', '
    plt.title(f"{model_name} - Confusion Matrix")
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    return accuracy, precision, recall, f1, conf_matrix
```

```
 # Evaluate Random Forest Classifier
# Assuming rfc_model, xgb_model, and svm_model are trained
```

```
# and X_test_scaled, y_test are from the SVM preprocessing (cell 57)

# Predictions for Random Forest
rfc_y_pred = rfc_model.predict(X_test_scaled)
evaluate_model(y_test, rfc_y_pred, "Random Forest")

# Predictions for XGBoost
xgb_y_pred = xgb_model.predict(X_test_scaled)
evaluate_model(y_test, xgb_y_pred, "XGBoost")

# Predictions for SVM (already done in cell 57)
# ... (no need to predict again)
evaluate_model(y_test, svm_model.predict(X_test_scaled), "SVM")
```

```
Random Forest - Accuracy: 0.51
Random Forest - Precision: 0.56
Random Forest - Recall: 0.70
Random Forest - F1: 0.62
```