PhishGuar2

September 22, 2024

[11]:
```python
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Machine learning libraries
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    roc_auc_score,
    roc_curve,
    accuracy_score,
)

# Feature selection
from sklearn.feature_selection import SelectKBest, chi2

# Handling imbalanced data
from imblearn.over_sampling import SMOTE

# For loading ARFF files
from scipy.io import arff

# Suppress warnings
import warnings
warnings.filterwarnings('ignore')

#----------------------------------------------------
# 1. Load the ARFF file
#----------------------------------------------------

# Load the ARFF file
```

```python
data, meta = arff.loadarff('Training Dataset.arff')
df = pd.DataFrame(data)

# Decode byte strings to regular strings for object columns
for col in df.select_dtypes([object]).columns:
    df[col] = df[col].str.decode('utf-8')

# Preview the data
print("First few rows of the dataset:")
print(df.head())

#----------------------------------------------------
# 2. Data Preprocessing
#----------------------------------------------------

# Check for missing values
print("\nMissing values in each column:")
print(df.isnull().sum())

# Since missing values are minimal, we can fill them (if any)
df.fillna(method='ffill', inplace=True)

# Convert target variable 'Result' to integer if necessary
df['Result'] = df['Result'].astype(int)

# Separate features and target variable
X = df.drop('Result', axis=1)
y = df['Result']

# Convert -1 to 0 for binary classification
y = y.replace(-1, 0)

#----------------------------------------------------
# 3. Handling Categorical Variables
#----------------------------------------------------

# Identify categorical columns
categorical_columns = X.select_dtypes(include=['object']).columns

# Apply Label Encoding to categorical columns
label_encoder = LabelEncoder()
for col in categorical_columns:
    X[col] = label_encoder.fit_transform(X[col])

#----------------------------------------------------
# 4. Handle class imbalance with SMOTE
#----------------------------------------------------
```

```python
# Handle class imbalance with SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Check class distribution after SMOTE
print("\nClass distribution after SMOTE:")
print(pd.Series(y_resampled).value_counts())

#-----------------------------------------------------
# 5. Feature Selection
#-----------------------------------------------------

from sklearn.feature_selection import SelectKBest, f_classif

# Using SelectKBest with f_classif to select top 15 features
selector = SelectKBest(f_classif, k=15)
selector.fit(X_resampled, y_resampled)
selected_features = X.columns[selector.get_support(indices=True)]
print("\nSelected Features:")
print(selected_features)

# Update X with selected features
X_resampled = X_resampled[selected_features]

#-----------------------------------------------------
# 6. Split the data into training and testing sets
#-----------------------------------------------------

X_train, X_test, y_train, y_test = train_test_split(
    X_resampled, y_resampled, test_size=0.2, random_state=42
)

#-----------------------------------------------------
# 7. Random Forest Classifier
#-----------------------------------------------------

# Initialize the Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Hyperparameter tuning using GridSearchCV
param_grid_rf = {
    'n_estimators': [100, 150, 200],
    'max_depth': [10, 15, 20],
    'min_samples_split': [2, 5],
    'criterion': ['gini', 'entropy'],
}
```

```python
grid_rf = GridSearchCV(
    estimator=rf_model,
    param_grid=param_grid_rf,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
)
grid_rf.fit(X_train, y_train)

# Best parameters
print("\nBest Parameters for Random Forest:")
print(grid_rf.best_params_)

# Train the model with best parameters
best_rf = grid_rf.best_estimator_
best_rf.fit(X_train, y_train)

#-----------------------------------------------------
# 8. XGBoost Classifier
#-----------------------------------------------------

# Initialize the XGBoost model
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss',␣
 ↪random_state=42)

# Hyperparameter tuning
param_grid_xgb = {
    'n_estimators': [100, 150, 200],
    'max_depth': [5, 10, 15],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.6, 0.8, 1.0],
}

grid_xgb = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid_xgb,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
)
grid_xgb.fit(X_train, y_train)

# Best parameters
print("\nBest Parameters for XGBoost:")
print(grid_xgb.best_params_)
```

```
# Train the model with best parameters
best_xgb = grid_xgb.best_estimator_
best_xgb.fit(X_train, y_train)

#----------------------------------------------------
# 9. Model Evaluation
#----------------------------------------------------

# Function to evaluate model performance
def evaluate_model(model, X_test, y_test, model_name):
    # Predictions
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]

    # Classification report
    print(f"\n{model_name} Classification Report:")
    print(classification_report(y_test, y_pred))

    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    print(f"{model_name} Confusion Matrix:")
    print(cm)

    # ROC AUC score
    roc_auc = roc_auc_score(y_test, y_proba)
    print(f"{model_name} ROC AUC Score: {roc_auc:.3f}")

    return y_pred, y_proba, cm, roc_auc

# Evaluate Random Forest
y_pred_rf, y_proba_rf, cm_rf, roc_auc_rf = evaluate_model(
    best_rf, X_test, y_test, "Random Forest"
)

# Evaluate XGBoost
y_pred_xgb, y_proba_xgb, cm_xgb, roc_auc_xgb = evaluate_model(
    best_xgb, X_test, y_test, "XGBoost"
)

#----------------------------------------------------
# 10. Feature Importance
#----------------------------------------------------

# Random Forest Feature Importance
importances_rf = best_rf.feature_importances_
indices_rf = np.argsort(importances_rf)[::-1]
```

```python
plt.figure(figsize=(10, 6))
plt.title("Feature Importances - Random Forest")
sns.barplot(
    x=importances_rf[indices_rf],
    y=selected_features[indices_rf],
    palette="viridis",
)
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.tight_layout()
plt.show()

# XGBoost Feature Importance
importances_xgb = best_xgb.feature_importances_
indices_xgb = np.argsort(importances_xgb)[::-1]

plt.figure(figsize=(10, 6))
plt.title("Feature Importances - XGBoost")
sns.barplot(
    x=importances_xgb[indices_xgb],
    y=selected_features[indices_xgb],
    palette="magma",
)
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.tight_layout()
plt.show()

#-----------------------------------------------------
# 11. ROC Curve Plotting
#-----------------------------------------------------

# Plot ROC Curves
plt.figure(figsize=(8, 6))

# Random Forest ROC Curve
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_proba_rf)
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {roc_auc_rf:.3f})')

# XGBoost ROC Curve
fpr_xgb, tpr_xgb, _ = roc_curve(y_test, y_proba_xgb)
plt.plot(fpr_xgb, tpr_xgb, label=f'XGBoost (AUC = {roc_auc_xgb:.3f})')

# Plot settings
plt.plot([0, 1], [0, 1], 'k--')
plt.title('Receiver Operating Characteristic')
plt.xlabel('False Positive Rate')
```

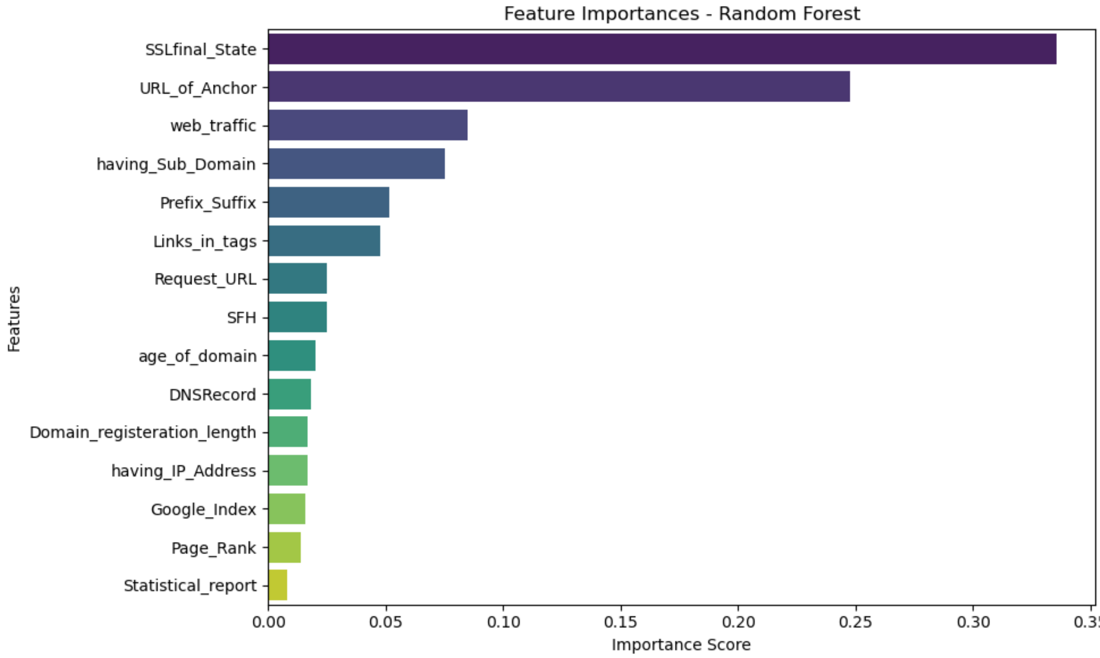
```python
plt.ylabel('True Positive Rate')
plt.legend()
plt.tight_layout()
plt.show()

#-----------------------------------------------------
# 12. Confusion Matrix Heatmaps
#-----------------------------------------------------

# Random Forest Confusion Matrix Heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix - Random Forest')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# XGBoost Confusion Matrix Heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(cm_xgb, annot=True, fmt='d', cmap='Greens')
plt.title('Confusion Matrix - XGBoost')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

#-----------------------------------------------------
# 13. Additional Visualizations
#-----------------------------------------------------

# Create a figure with multiple subplots
fig, axs = plt.subplots(3, 2, figsize=(15, 15))
fig.suptitle('PhishGuard Model - Visualizations', fontsize=16)

# Histogram of URL Length
sns.histplot(df['URL_Length'], bins=30, ax=axs[0, 0], kde=True)
axs[0, 0].set_title('Distribution of URL Length')

# Distribution of Domain Age
sns.histplot(df['Domain_Age'], bins=30, ax=axs[0, 1], kde=True)
axs[0, 1].set_title('Distribution of Domain Age')

# Countplot of SSLfinal_State
sns.countplot(x='SSLfinal_State', data=df, ax=axs[1, 0])
axs[1, 0].set_title('SSL Final State Counts')

# Boxplot of web_traffic vs. Result
sns.boxplot(x='Result', y='web_traffic', data=df, ax=axs[1, 1])
```

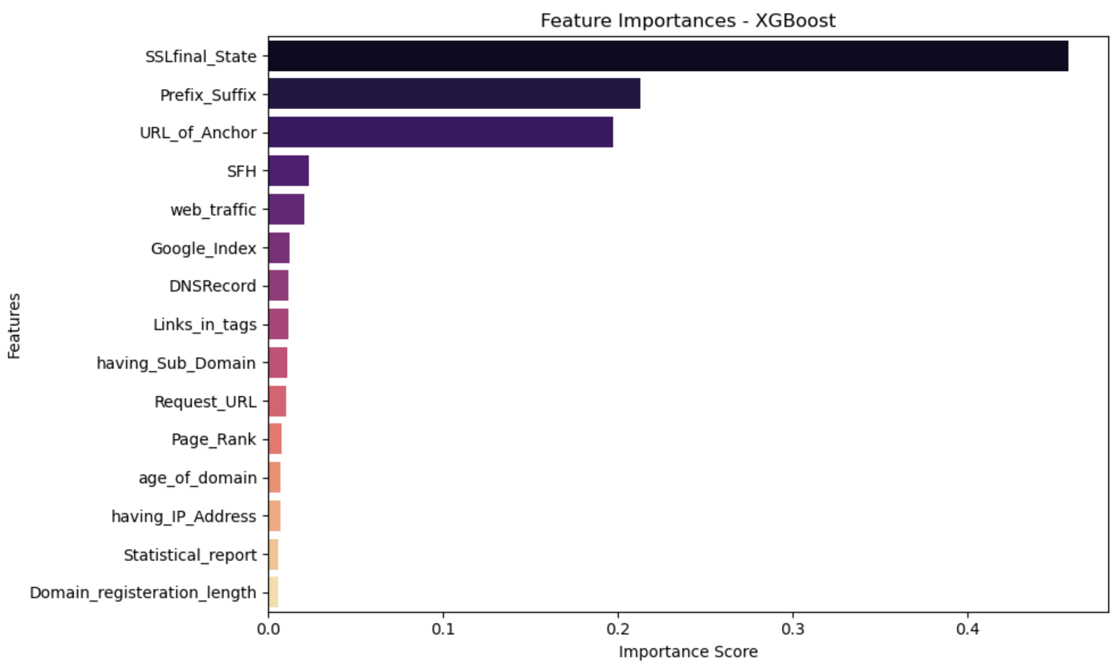
```
axs[1, 1].set_title('Web Traffic vs. Result')

# Heatmap of Correlation Matrix
corr_matrix = df.corr()
sns.heatmap(corr_matrix, ax=axs[2, 0], cmap='coolwarm')
axs[2, 0].set_title('Correlation Matrix')

# Pie Chart of Class Distribution
class_counts = df['Result'].value_counts()
axs[2, 1].pie(
    class_counts,
    labels=['Legitimate', 'Phishing'],
    autopct='%1.1f%%',
    startangle=90,
    colors=['#66b3ff', '#ff6666'],
)
axs[2, 1].set_title('Class Distribution')

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

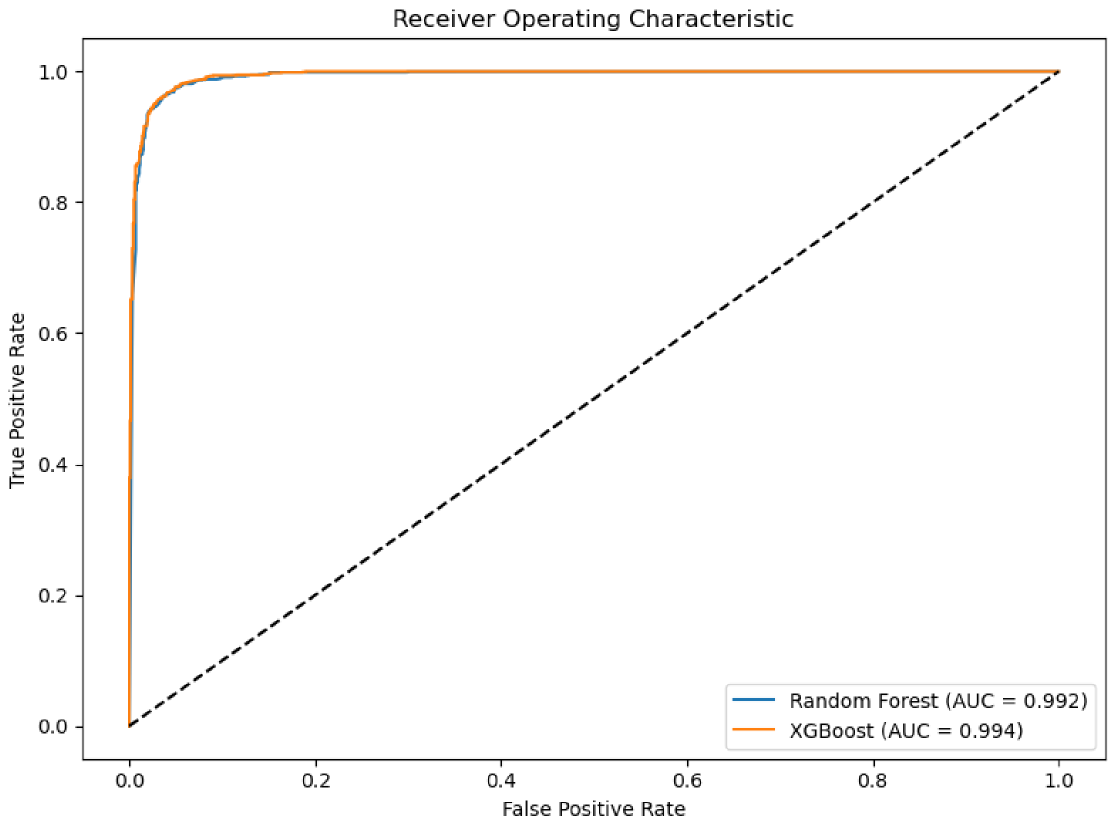
```
First few rows of the dataset:
   having_IP_Address  URL_Length  Shortining_Service  having_At_Symbol  \
0                 -1           1                   1                 1
1                  1           1                   1                 1
2                  1           0                   1                 1
3                  1           0                   1                 1
4                  1           0                  -1                 1

   double_slash_redirecting  Prefix_Suffix  having_Sub_Domain  SSLfinal_State  \
0                        -1             -1                 -1              -1
1                         1             -1                  0               1
2                         1             -1                 -1              -1
3                         1             -1                 -1              -1
4                         1             -1                  1               1

   Domain_registeration_length  Favicon  …  popUpWidnow  Iframe  age_of_domain  \
0                           -1        1  …            1       1             -1
1                           -1        1  …            1       1             -1
2                           -1        1  …            1       1              1
3                            1        1  …            1       1             -1
4                           -1        1  …           -1       1             -1

   DNSRecord  web_traffic  Page_Rank  Google_Index  Links_pointing_to_page  \
0         -1           -1         -1             1                       1
1         -1            0         -1             1                       1
2         -1            1         -1             1                       0
3         -1            1         -1             1                      -1
```

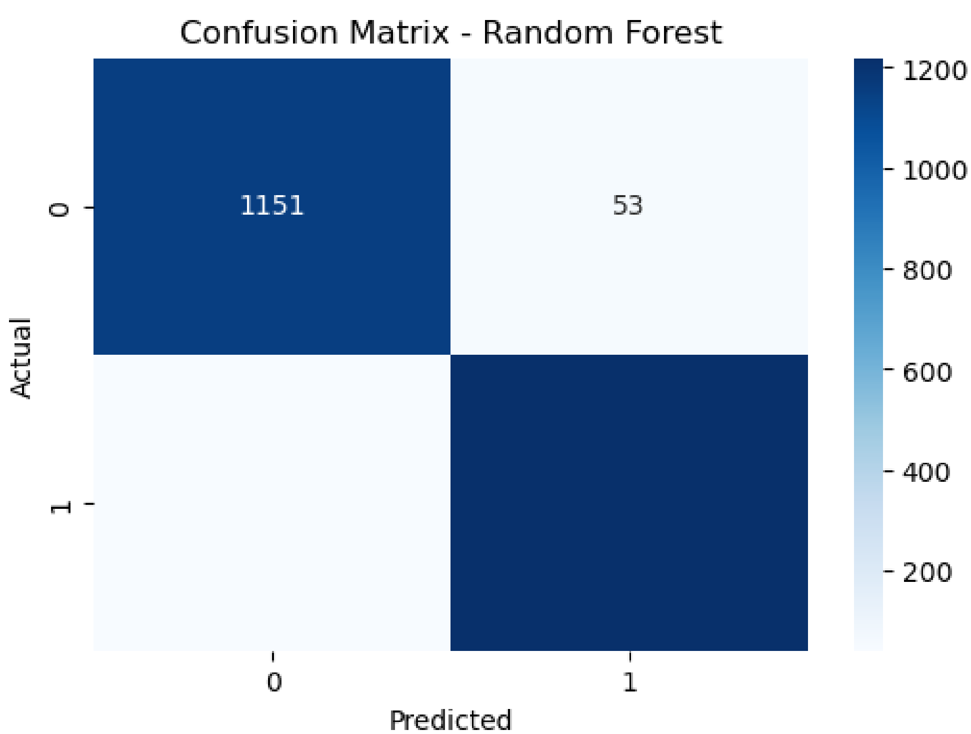
```
4       -1           0       -1            1                       1

  Statistical_report Result
0                 -1     -1
1                  1     -1
2                 -1     -1
3                  1     -1
4                  1      1

[5 rows x 31 columns]

Missing values in each column:
having_IP_Address              0
URL_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

Class distribution after SMOTE:
Result
0    6157
```

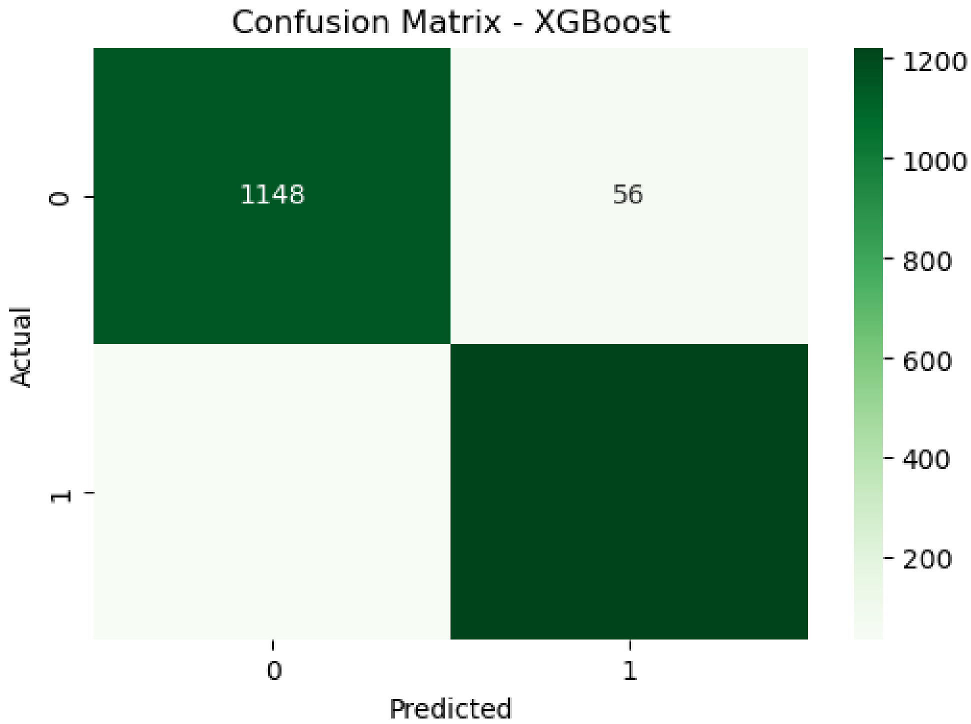
```
1    6157
Name: count, dtype: int64

Selected Features:
Index(['having_IP_Address', 'Prefix_Suffix', 'having_Sub_Domain',
       'SSLfinal_State', 'Domain_registeration_length', 'Request_URL',
       'URL_of_Anchor', 'Links_in_tags', 'SFH', 'age_of_domain', 'DNSRecord',
       'web_traffic', 'Page_Rank', 'Google_Index', 'Statistical_report'],
      dtype='object')

Best Parameters for Random Forest:
{'criterion': 'gini', 'max_depth': 20, 'min_samples_split': 2, 'n_estimators':
100}

Best Parameters for XGBoost:
{'learning_rate': 0.2, 'max_depth': 10, 'n_estimators': 150, 'subsample': 1.0}

Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.96      0.96      1204
           1       0.96      0.97      0.96      1259

    accuracy                           0.96      2463
   macro avg       0.96      0.96      0.96      2463
weighted avg       0.96      0.96      0.96      2463

Random Forest Confusion Matrix:
[[1151   53]
 [  41 1218]]
Random Forest ROC AUC Score: 0.992

XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.95      0.96      1204
           1       0.96      0.97      0.96      1259

    accuracy                           0.96      2463
   macro avg       0.96      0.96      0.96      2463
weighted avg       0.96      0.96      0.96      2463

XGBoost Confusion Matrix:
[[1148   56]
 [  37 1222]]
XGBoost ROC AUC Score: 0.994
```


Feature Importances - Random Forest
SSLfinal_State
URL_of_Anchor
web_traffic
having_Sub_Domain
Prefix_Suffix
Links_in_tags
Request_URL
SFH
age_of_domain
DNSRecord
Domain_registeration_length
having_IP_Address
Google_Index
Page_Rank
Statistical_report
Features
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Importance Score


Feature Importances - XGBoost
SSLfinal_State
Prefix_Suffix
URL_of_Anchor
SFH
web_traffic
Google_Index
DNSRecord
Links_in_tags
having_Sub_Domain
Request_URL
Page_Rank
age_of_domain
having_IP_Address
Statistical_report
Domain_registeration_length
Features
0.0
0.1
0.2
0.3
0.4
Importance Score


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Random Forest (AUC = 0.992)
XGBoost (AUC = 0.994)

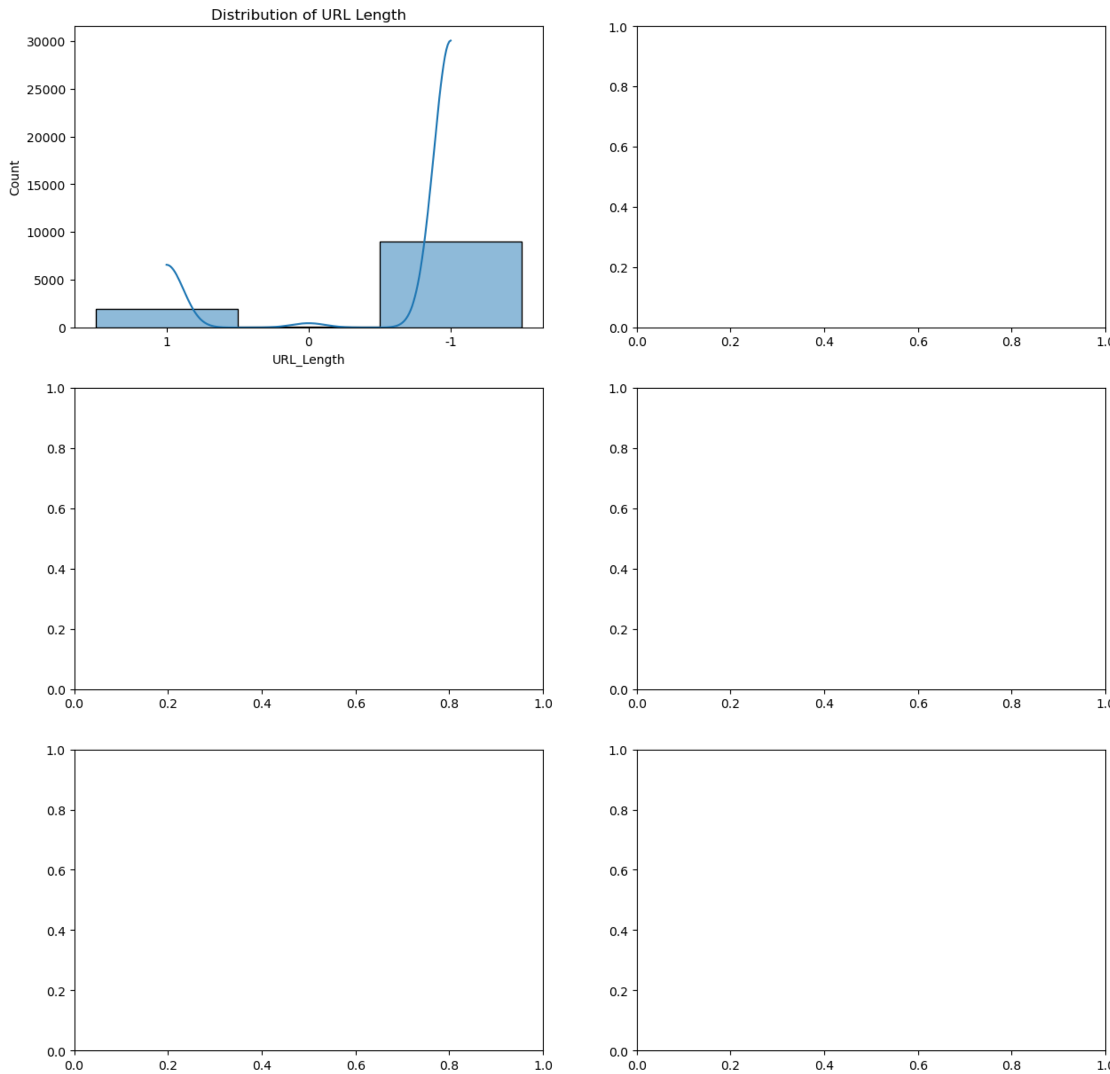

Confusion Matrix - Random Forest
1151
53
Actual
Predicted
0
1
1200
1000
800
600
400
200




```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File ~\anaconda3\Lib\site-packages\pandas\core\indexes\base.py:3791, in Index.
↪get_loc(self, key)
   3790 try:
-> 3791     return self._engine.get_loc(casted_key)
   3792 except KeyError as err:

File index.pyx:152, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:181, in pandas._libs.index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()

KeyError: 'Domain_Age'

The above exception was the direct cause of the following exception:
```

```
KeyError                                  Traceback (most recent call last)
Cell In[11], line 310
    307 axs[0, 0].set_title('Distribution of URL Length')
    309 # Distribution of Domain Age
--> 310 sns.histplot(df['Domain_Age'], bins=30, ax=axs[0, 1], kde=True)
    311 axs[0, 1].set_title('Distribution of Domain Age')
    313 # Countplot of SSLfinal_State

File ~\anaconda3\Lib\site-packages\pandas\core\frame.py:3893, in DataFrame.
 ↪__getitem__(self, key)
   3891 if self.columns.nlevels > 1:
   3892     return self._getitem_multilevel(key)
-> 3893 indexer = self.columns.get_loc(key)
   3894 if is_integer(indexer):
   3895     indexer = [indexer]

File ~\anaconda3\Lib\site-packages\pandas\core\indexes\base.py:3798, in Index.
 ↪get_loc(self, key)
   3793     if isinstance(casted_key, slice) or (
   3794         isinstance(casted_key, abc.Iterable)
   3795         and any(isinstance(x, slice) for x in casted_key)
   3796     ):
   3797         raise InvalidIndexError(key)
-> 3798     raise KeyError(key) from err
   3799 except TypeError:
   3800     # If we have a listlike key, _check_indexing_error will raise
   3801     #  InvalidIndexError. Otherwise we fall through and re-raise
   3802     #  the TypeError.
   3803     self._check_indexing_error(key)

KeyError: 'Domain_Age'
```

PhishGuard Model - Visualizations




```
[13]: # Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Machine learning libraries
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

```python
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    roc_auc_score,
    roc_curve,
    accuracy_score,
)

# Feature selection
from sklearn.feature_selection import SelectKBest, chi2

# Handling imbalanced data
from imblearn.over_sampling import SMOTE

# For loading ARFF files
from scipy.io import arff

# Suppress warnings
import warnings
warnings.filterwarnings('ignore')

#-----------------------------------------------------
# 1. Load the ARFF file
#-----------------------------------------------------

# Load the ARFF file
data, meta = arff.loadarff('Training Dataset.arff')
df = pd.DataFrame(data)

# Decode byte strings to regular strings for object columns
for col in df.select_dtypes([object]).columns:
    df[col] = df[col].str.decode('utf-8')

# Preview the data
print("First few rows of the dataset:")
print(df.head())

#-----------------------------------------------------
# 2. Data Preprocessing
#-----------------------------------------------------

# Check for missing values
print("\nMissing values in each column:")
print(df.isnull().sum())

# Since missing values are minimal, we can fill them (if any)
df.fillna(method='ffill', inplace=True)
```

```python
# Convert target variable 'Result' to integer if necessary
df['Result'] = df['Result'].astype(int)

# Separate features and target variable
X = df.drop('Result', axis=1)
y = df['Result']

# Convert -1 to 0 for binary classification
y = y.replace(-1, 0)

#--------------------------------------------------
# 3. Handling Categorical Variables
#--------------------------------------------------

# Identify categorical columns
categorical_columns = X.select_dtypes(include=['object']).columns

# Apply Label Encoding to categorical columns
label_encoder = LabelEncoder()
for col in categorical_columns:
    X[col] = label_encoder.fit_transform(X[col])

#--------------------------------------------------
# 4. Handle class imbalance with SMOTE
#--------------------------------------------------

# Handle class imbalance with SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Check class distribution after SMOTE
print("\nClass distribution after SMOTE:")
print(pd.Series(y_resampled).value_counts())

#--------------------------------------------------
# 5. Feature Selection
#--------------------------------------------------

from sklearn.feature_selection import SelectKBest, f_classif

# Using SelectKBest with f_classif to select top 15 features
selector = SelectKBest(f_classif, k=15)
selector.fit(X_resampled, y_resampled)
selected_features = X.columns[selector.get_support(indices=True)]
print("\nSelected Features:")
print(selected_features)
```

```python
# Update X with selected features
X_resampled = X_resampled[selected_features]

#----------------------------------------------------
# 6. Split the data into training and testing sets
#----------------------------------------------------

X_train, X_test, y_train, y_test = train_test_split(
    X_resampled, y_resampled, test_size=0.2, random_state=42
)

#----------------------------------------------------
# 7. Random Forest Classifier
#----------------------------------------------------

# Initialize the Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Hyperparameter tuning using GridSearchCV
param_grid_rf = {
    'n_estimators': [100, 150, 200],
    'max_depth': [10, 15, 20],
    'min_samples_split': [2, 5],
    'criterion': ['gini', 'entropy'],
}

grid_rf = GridSearchCV(
    estimator=rf_model,
    param_grid=param_grid_rf,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
)
grid_rf.fit(X_train, y_train)

# Best parameters
print("\nBest Parameters for Random Forest:")
print(grid_rf.best_params_)

# Train the model with best parameters
best_rf = grid_rf.best_estimator_
best_rf.fit(X_train, y_train)

#----------------------------------------------------
# 8. XGBoost Classifier
#----------------------------------------------------
```

```
# Initialize the XGBoost model
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss',␣
 ↪random_state=42)

# Hyperparameter tuning
param_grid_xgb = {
    'n_estimators': [100, 150, 200],
    'max_depth': [5, 10, 15],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.6, 0.8, 1.0],
}

grid_xgb = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid_xgb,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
)
grid_xgb.fit(X_train, y_train)

# Best parameters
print("\nBest Parameters for XGBoost:")
print(grid_xgb.best_params_)

# Train the model with best parameters
best_xgb = grid_xgb.best_estimator_
best_xgb.fit(X_train, y_train)

#-----------------------------------------------------
# 9. Model Evaluation
#-----------------------------------------------------

# Function to evaluate model performance
def evaluate_model(model, X_test, y_test, model_name):
    # Predictions
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]

    # Classification report
    print(f"\n{model_name} Classification Report:")
    print(classification_report(y_test, y_pred))

    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    print(f"{model_name} Confusion Matrix:")
```

```python
    print(cm)

    # ROC AUC score
    roc_auc = roc_auc_score(y_test, y_proba)
    print(f"{model_name} ROC AUC Score: {roc_auc:.3f}")

    return y_pred, y_proba, cm, roc_auc

# Evaluate Random Forest
y_pred_rf, y_proba_rf, cm_rf, roc_auc_rf = evaluate_model(
    best_rf, X_test, y_test, "Random Forest"
)

# Evaluate XGBoost
y_pred_xgb, y_proba_xgb, cm_xgb, roc_auc_xgb = evaluate_model(
    best_xgb, X_test, y_test, "XGBoost"
)

#----------------------------------------------------
# 10. Feature Importance
#----------------------------------------------------

# Random Forest Feature Importance
importances_rf = best_rf.feature_importances_
indices_rf = np.argsort(importances_rf)[::-1]

plt.figure(figsize=(10, 6))
plt.title("Feature Importances - Random Forest")
sns.barplot(
    x=importances_rf[indices_rf],
    y=selected_features[indices_rf],
    palette="viridis",
)
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.tight_layout()
plt.show()

# XGBoost Feature Importance
importances_xgb = best_xgb.feature_importances_
indices_xgb = np.argsort(importances_xgb)[::-1]

plt.figure(figsize=(10, 6))
plt.title("Feature Importances - XGBoost")
sns.barplot(
    x=importances_xgb[indices_xgb],
    y=selected_features[indices_xgb],
```

```
    palette="magma",
)
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.tight_layout()
plt.show()

#----------------------------------------------------
# 11. ROC Curve Plotting
#----------------------------------------------------

# Plot ROC Curves
plt.figure(figsize=(8, 6))

# Random Forest ROC Curve
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_proba_rf)
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {roc_auc_rf:.3f})')

# XGBoost ROC Curve
fpr_xgb, tpr_xgb, _ = roc_curve(y_test, y_proba_xgb)
plt.plot(fpr_xgb, tpr_xgb, label=f'XGBoost (AUC = {roc_auc_xgb:.3f})')

# Plot settings
plt.plot([0, 1], [0, 1], 'k--')
plt.title('Receiver Operating Characteristic')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.tight_layout()
plt.show()

#----------------------------------------------------
# 12. Confusion Matrix Heatmaps
#----------------------------------------------------

# Random Forest Confusion Matrix Heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix - Random Forest')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# XGBoost Confusion Matrix Heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(cm_xgb, annot=True, fmt='d', cmap='Greens')
plt.title('Confusion Matrix - XGBoost')
```

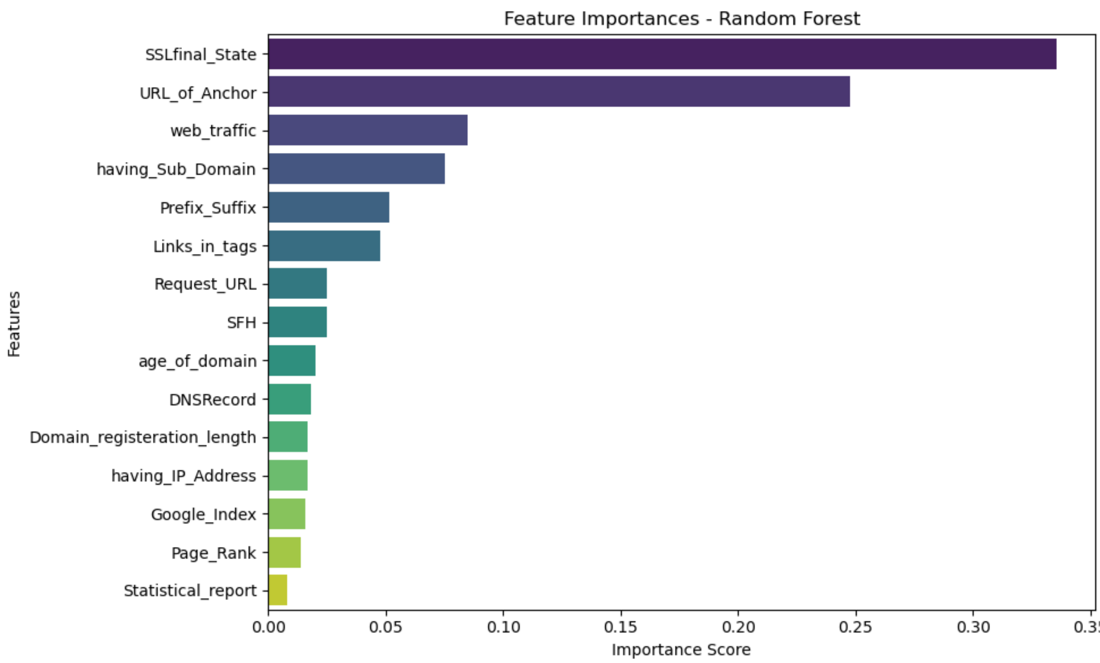
```python
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

#---------------------------------------------------
# 13. Additional Visualizations
#---------------------------------------------------

# Check for actual column names
print("\nColumn names:", df.columns)

# Create a figure with multiple subplots
fig, axs = plt.subplots(3, 2, figsize=(15, 15))
fig.suptitle('PhishGuard Model - Visualizations', fontsize=16)

# Histogram of URL Length
sns.histplot(df['URL_Length'], bins=30, ax=axs[0, 0], kde=True)
axs[0, 0].set_title('Distribution of URL Length')

# Check if 'age_of_domain' is present instead of 'Domain_Age'
if 'age_of_domain' in df.columns:
    sns.histplot(df['age_of_domain'], bins=30, ax=axs[0, 1], kde=True)
    axs[0, 1].set_title('Distribution of Domain Age')
else:
    print("'age_of_domain' or 'Domain_Age' column not found.")

# Countplot of SSLfinal_State
sns.countplot(x='SSLfinal_State', data=df, ax=axs[1, 0])
axs[1, 0].set_title('SSL Final State Counts')

# Boxplot of web_traffic vs. Result
sns.boxplot(x='Result', y='web_traffic', data=df, ax=axs[1, 1])
axs[1, 1].set_title('Web Traffic vs. Result')

# Heatmap of Correlation Matrix
corr_matrix = df.corr()
sns.heatmap(corr_matrix, ax=axs[2, 0], cmap='coolwarm')
axs[2, 0].set_title('Correlation Matrix')

# Pie Chart of Class Distribution
class_counts = df['Result'].value_counts()
axs[2, 1].pie(
    class_counts,
    labels=['Legitimate', 'Phishing'],
    autopct='%1.1f%%',
    startangle=90,
    colors=['#66b3ff', '#ff6666'],
```

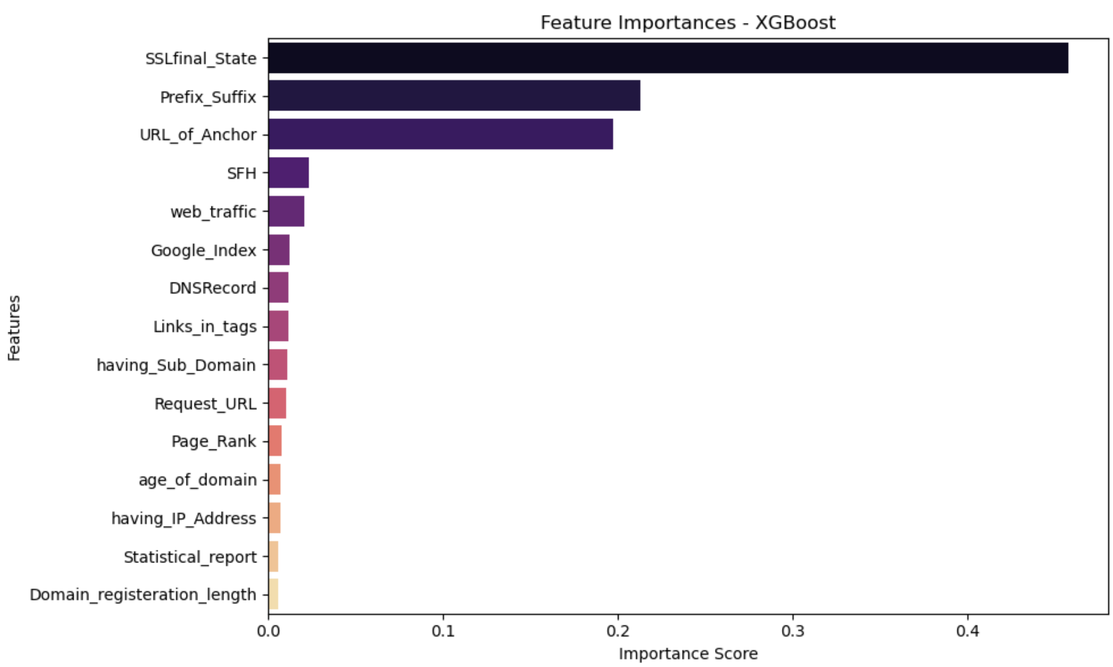
```
)
axs[2, 1].set_title('Class Distribution')

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

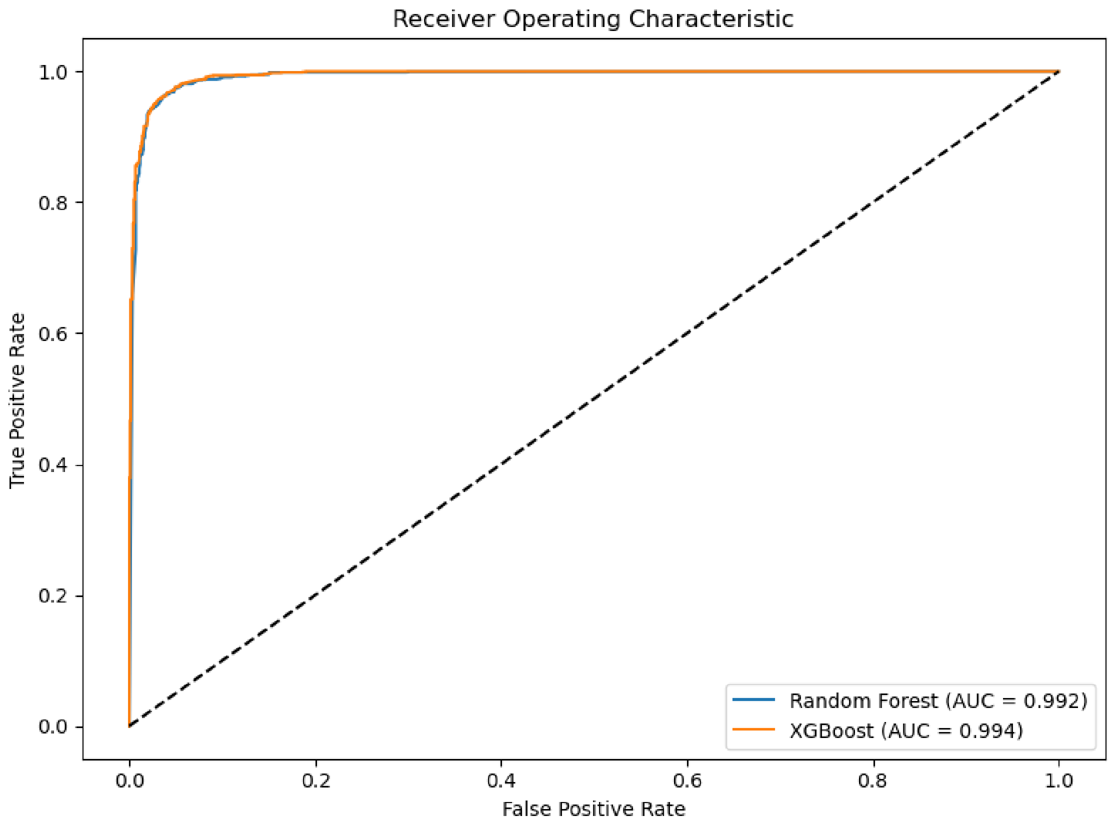
```
First few rows of the dataset:
   having_IP_Address  URL_Length  Shortining_Service  having_At_Symbol  \
0                 -1           1                   1                 1
1                  1           1                   1                 1
2                  1           0                   1                 1
3                  1           0                   1                 1
4                  1           0                  -1                 1

   double_slash_redirecting  Prefix_Suffix  having_Sub_Domain  SSLfinal_State  \
0                        -1             -1                 -1              -1
1                         1             -1                  0               1
2                         1             -1                 -1              -1
3                         1             -1                 -1              -1
4                         1             -1                  1               1

   Domain_registeration_length  Favicon  …  popUpWidnow  Iframe  age_of_domain  \
0                           -1        1  …            1       1             -1
1                           -1        1  …            1       1             -1
2                           -1        1  …            1       1              1
3                            1        1  …            1       1             -1
4                           -1        1  …           -1       1             -1

   DNSRecord  web_traffic  Page_Rank  Google_Index  Links_pointing_to_page  \
0         -1           -1         -1             1                       1
1         -1            0         -1             1                       1
2         -1            1         -1             1                       0
3         -1            1         -1             1                      -1
4         -1            0         -1             1                       1

   Statistical_report  Result
0                  -1      -1
1                   1      -1
2                  -1      -1
3                   1      -1
4                   1       1

[5 rows x 31 columns]

Missing values in each column:
having_IP_Address              0
URL_Length                     0
Shortining_Service             0
```

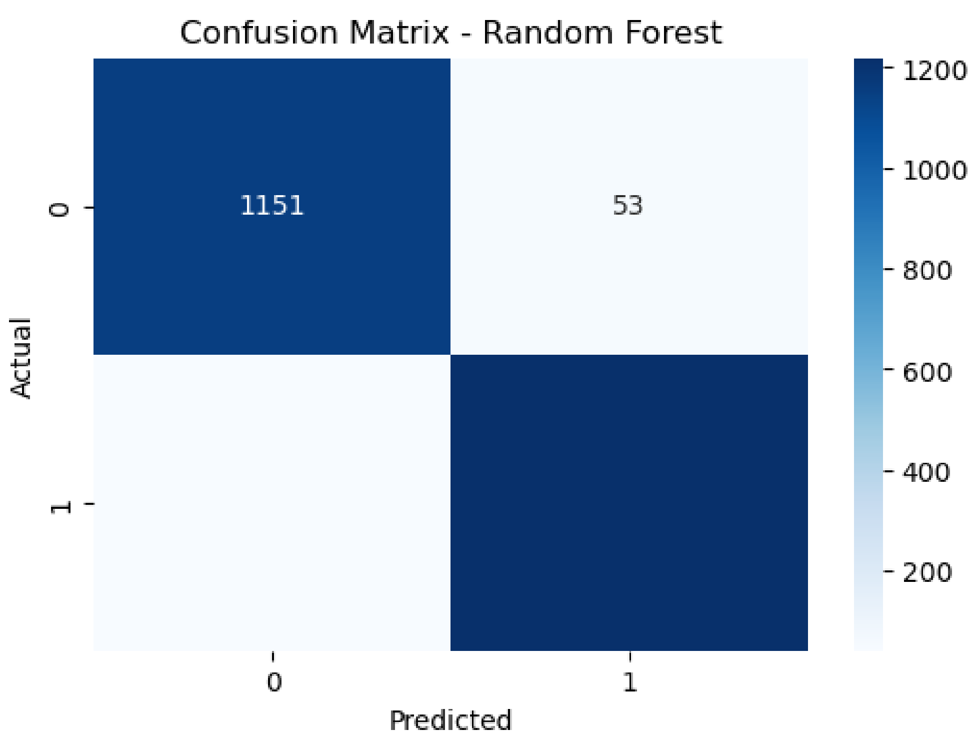
```
having_At_Symbol              0
double_slash_redirecting      0
Prefix_Suffix                 0
having_Sub_Domain             0
SSLfinal_State                0
Domain_registeration_length   0
Favicon                       0
port                          0
HTTPS_token                   0
Request_URL                   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

Class distribution after SMOTE:
Result
0    6157
1    6157
Name: count, dtype: int64

Selected Features:
Index(['having_IP_Address', 'Prefix_Suffix', 'having_Sub_Domain',
       'SSLfinal_State', 'Domain_registeration_length', 'Request_URL',
       'URL_of_Anchor', 'Links_in_tags', 'SFH', 'age_of_domain', 'DNSRecord',
       'web_traffic', 'Page_Rank', 'Google_Index', 'Statistical_report'],
      dtype='object')

Best Parameters for Random Forest:
{'criterion': 'gini', 'max_depth': 20, 'min_samples_split': 2, 'n_estimators':
100}

Best Parameters for XGBoost:
```

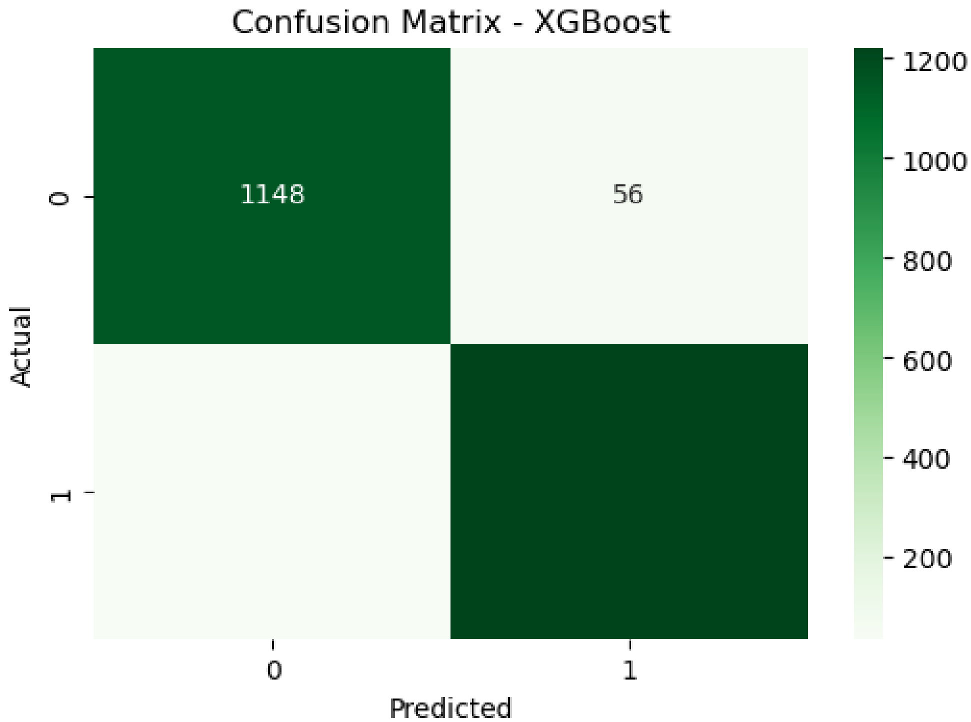
```
{'learning_rate': 0.2, 'max_depth': 10, 'n_estimators': 150, 'subsample': 1.0}

Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.96      0.96      1204
           1       0.96      0.97      0.96      1259

    accuracy                           0.96      2463
   macro avg       0.96      0.96      0.96      2463
weighted avg       0.96      0.96      0.96      2463

Random Forest Confusion Matrix:
[[1151   53]
 [  41 1218]]
Random Forest ROC AUC Score: 0.992

XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.95      0.96      1204
           1       0.96      0.97      0.96      1259

    accuracy                           0.96      2463
   macro avg       0.96      0.96      0.96      2463
weighted avg       0.96      0.96      0.96      2463

XGBoost Confusion Matrix:
[[1148   56]
 [  37 1222]]
XGBoost ROC AUC Score: 0.994
```


Feature Importances - Random Forest
SSLfinal_State
URL_of_Anchor
web_traffic
having_Sub_Domain
Prefix_Suffix
Links_in_tags
Request_URL
SFH
age_of_domain
DNSRecord
Domain_registeration_length
having_IP_Address
Google_Index
Page_Rank
Statistical_report
Features
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Importance Score

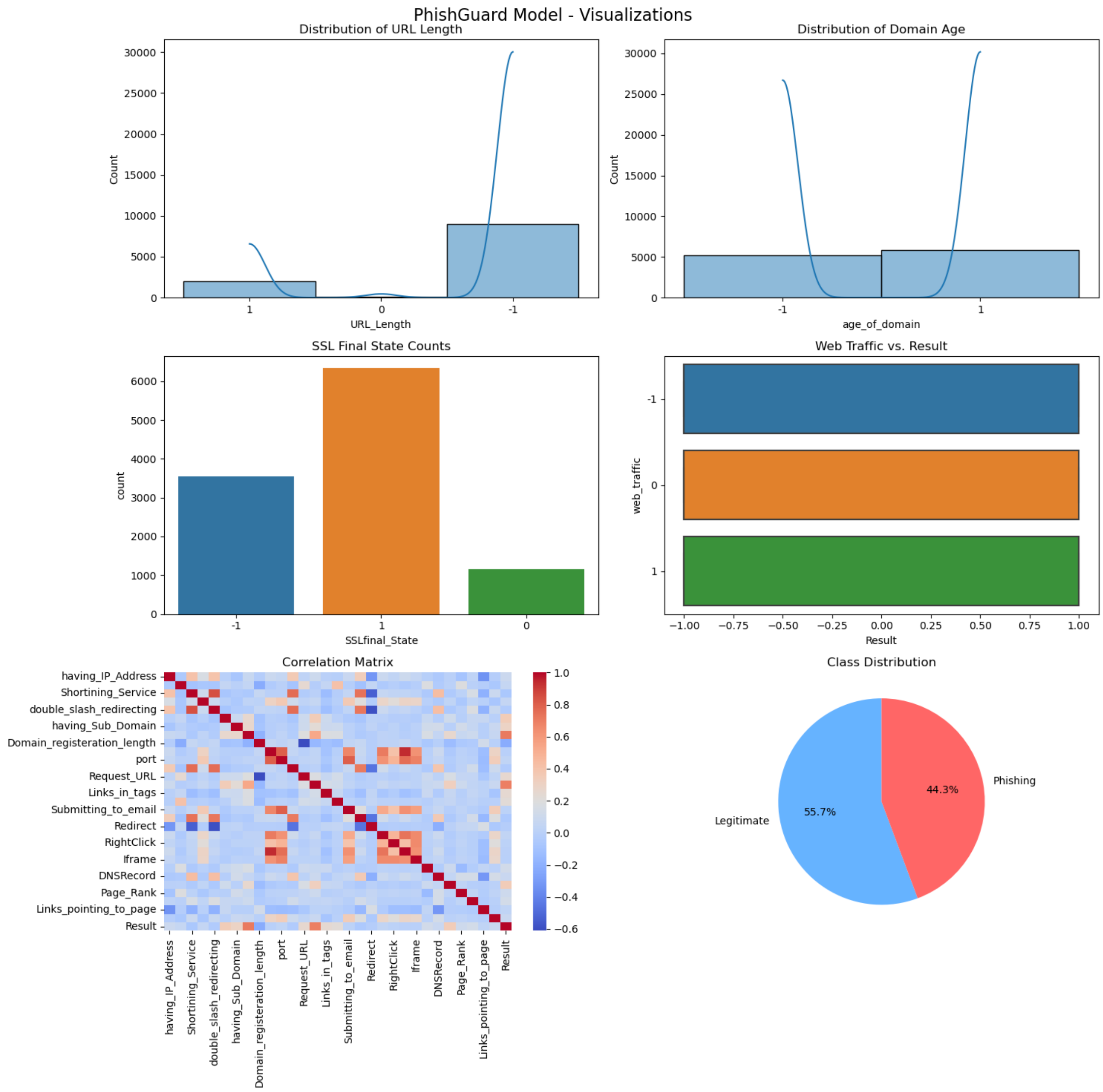

Feature Importances - XGBoost
SSLfinal_State
Prefix_Suffix
URL_of_Anchor
SFH
web_traffic
Google_Index
DNSRecord
Links_in_tags
having_Sub_Domain
Request_URL
Page_Rank
age_of_domain
having_IP_Address
Statistical_report
Domain_registeration_length
Features
0.0
0.1
0.2
0.3
0.4
Importance Score


Receiver Operating Characteristic
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
Random Forest (AUC = 0.992)
XGBoost (AUC = 0.994)


Confusion Matrix - Random Forest
1151
53
Actual
0
1
Predicted
1200
1000
800
600
400
200




```
Column names: Index(['having_IP_Address', 'URL_Length', 'Shortining_Service',
       'having_At_Symbol', 'double_slash_redirecting', 'Prefix_Suffix',
       'having_Sub_Domain', 'SSLfinal_State', 'Domain_registeration_length',
       'Favicon', 'port', 'HTTPS_token', 'Request_URL', 'URL_of_Anchor',
       'Links_in_tags', 'SFH', 'Submitting_to_email', 'Abnormal_URL',
       'Redirect', 'on_mouseover', 'RightClick', 'popUpWidnow', 'Iframe',
       'age_of_domain', 'DNSRecord', 'web_traffic', 'Page_Rank',
       'Google_Index', 'Links_pointing_to_page', 'Statistical_report',
       'Result'],
      dtype='object')
```


PhishGuard Model - Visualizations
Distribution of URL Length
Count
URL_Length
Distribution of Domain Age
Count
age_of_domain
SSL Final State Counts
count
SSLfinal_State
Web Traffic vs. Result
web_traffic
Result
Correlation Matrix
having_IP_Address
Shortining_Service
double_slash_redirecting
having_Sub_Domain
Domain_registeration_length
port
Request_URL
Links_in_tags
Submitting_to_email
Redirect
RightClick
Iframe
DNSRecord
Page_Rank
Links_pointing_to_page
Result
Class Distribution
Phishing
44.3%
Legitimate
55.7%

[ ]: