Internship_CodeB_week 5 & 6

April 25, 2025

1 Phishing Website Detection

- Name : Gaurav Vijay Jadhav
- github : [https://github.com/jadhavgaurav/CodeB_Internship_Project]

2 Week 5 & 6 Submission

[1]:
```python
# Import Necessary Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc, confusion_matrix,␣
 ↪precision_recall_curve, average_precision_score
import seaborn as sns
import numpy as np
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```python
# import dataset

data_url = 'https://raw.githubusercontent.com/jadhavgaurav/
 ↪CodeB_Internship_Project/refs/heads/main/dataset_phishing.csv'

df = pd.read_csv(data_url)

df.sample(frac = 1)
```

[2]:

```
                                                    url  length_url  \
4594  http://www.kanrit.com/plugins/system/Server/ww…          89
9301  http://tsuzuki.co.id/model/jeff/chinavali/inde…          51
7247  http://www.ianswer4u.com/2011/05/mesh-topology…          66
2327  https://sites.google.com/site/recoveryfbconfir…          57
9719  https://onedrive.live.com/redir?resid=15D888F2…         255
…                                                   …           …
9115  https://lender.testing.santander.poweredbydivi…          57
8343  http://www.sieck-kuehlsysteme.de/userdata/imag…          65
9179                      https://meteo-oberwallis.ch/          28
2030             http://eudesign.com/mnems/portstar.htm          38
9456                          https://www.katailmu.com/          25

      length_hostname  ip  nb_dots  nb_hyphens  nb_at  nb_qm  nb_and  nb_or  \
4594               14   0        4           1      0      0       0      0
9301               13   0        3           0      0      0       0      0
7247               17   0        3           3      0      0       0      0
2327               16   0        2           0      0      0       0      0
9719               17   1        3           8      0      1       3      0
…                   …  ..        …           …      …      …       …      …
9115               44   0        4           0      0      0       0      0
8343               25   0        2           1      0      0       0      0
9179               19   0        1           1      0      0       0      0
2030               12   0        2           0      0      0       0      0
9456               16   0        2           0      0      0       0      0

      …  domain_in_title  domain_with_copyright  whois_registered_domain  \
4594  …                1                      1                        0
9301  …                1                      0                        0
7247  …                1                      0                        0
2327  …                1                      0                        0
9719  …                1                      0                        0
…     …                …                      …                        …
9115  …                1                      0                        0
8343  …                1                      0                        0
9179  …                1                      0                        1
2030  …                1                      0                        0
9456  …                1                      1                        0

      domain_registration_length  domain_age  web_traffic  dns_record  \
4594                         687        4425            0           0
9301                         124        1337            0           0
7247                         321          -1      4194715           0
2327                        2974        8348            1           0
9719                         158        9338           21           0
…                              …           …            …           …
9115                         106         625            0           0
```

```
8343                                 0        -1         0          0
9179                                 0        -1         0          0
2030                               684      8081   3004473          0
9456                               136      2786   1090791          0

      google_index  page_rank      status
4594             1          2    phishing
9301             1          0    phishing
7247             0          3  legitimate
2327             1         10    phishing
9719             0          5    phishing
…              …        …          …
9115             1          0    phishing
8343             1          0    phishing
9179             0          3  legitimate
2030             0          4  legitimate
9456             0          3  legitimate

[11430 rows x 89 columns]
```

3 Data Cleaning Report Phishing Website Detection

3.1 Dataset Overview

- **Total Records:** 11,430
- **Total Features (excluding target):** 87
- **Target Variable:** `status`
  - `0`: Legitimate
  - `1`: Phishing
- **Data Types:**
  - **Numerical (int64/float64):** 87
  - **Categorical/Object:** 1 (`url`)

3.1.1 Target Column

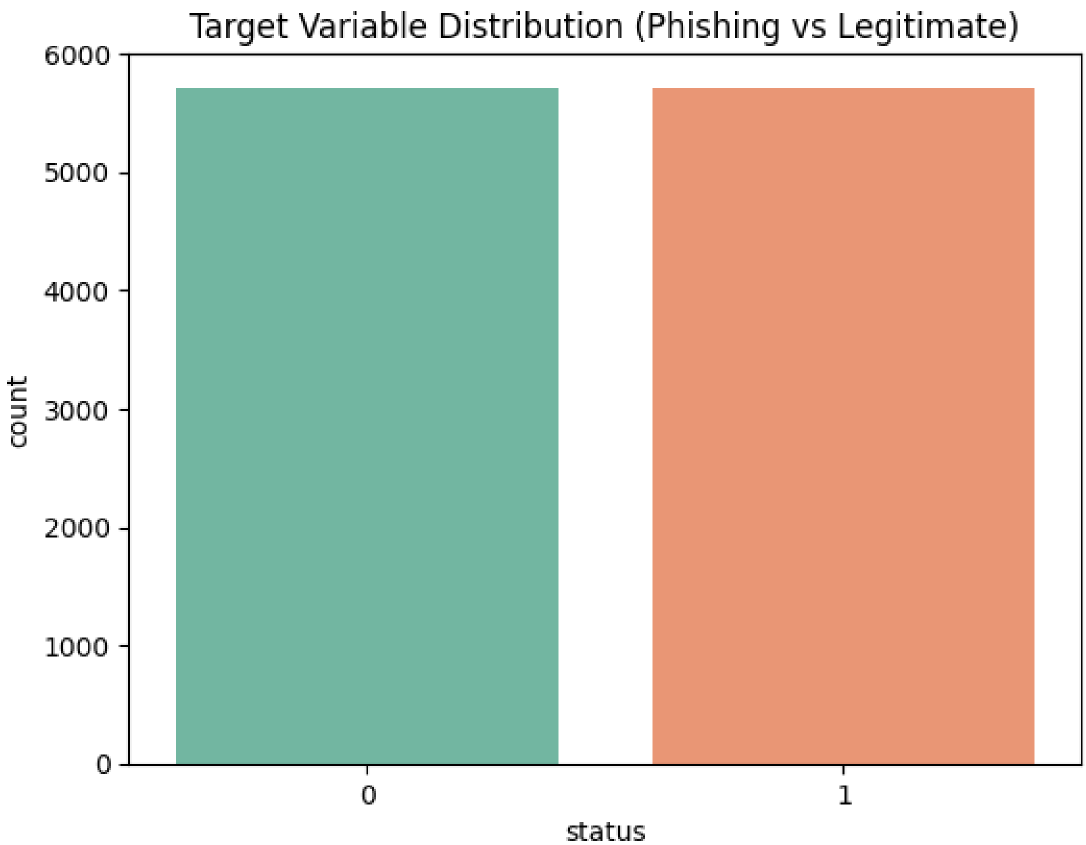
3.1.2 `status`

- **Description**: Binary label indicating if the website is phishing (`1`) or legitimate (`0`).
- **Relevance**: This is the variable to be predicted by the classification model.

```
[3]: # Replace 'Legitimate' with 0 and 'Phishing' with 1 in the 'status' column
df['status'] = df['status'].map({'legitimate':0, 'phishing':1})

print(df['status'].value_counts())
```

```
status
0    5715
1    5715
Name: count, dtype: int64
```

[4]:
```python
# Basic Info About Target Column and Visualize Target Distribution (Bar Plot)

# Check class distribution

sns.countplot(data=df, x='status', palette='Set2')
plt.title("Target Variable Distribution (Phishing vs Legitimate)")
plt.show()

print(df['status'].value_counts())
```




```
status
0    5715
1    5715
Name: count, dtype: int64
```

[5]:
```
numeric_features = df.select_dtypes(include=['int64', 'float64']).columns.
 ↪tolist()
categorical_features = df.select_dtypes(include='object').columns.tolist()

print("Numeric Features:", numeric_features)
print("Categorical Features:", categorical_features)
```

```
Numeric Features: ['length_url', 'length_hostname', 'ip', 'nb_dots',
'nb_hyphens', 'nb_at', 'nb_qm', 'nb_and', 'nb_or', 'nb_eq', 'nb_underscore',
'nb_tilde', 'nb_percent', 'nb_slash', 'nb_star', 'nb_colon', 'nb_comma',
'nb_semicolumn', 'nb_dollar', 'nb_space', 'nb_www', 'nb_com', 'nb_dslash',
'http_in_path', 'https_token', 'ratio_digits_url', 'ratio_digits_host',
'punycode', 'port', 'tld_in_path', 'tld_in_subdomain', 'abnormal_subdomain',
'nb_subdomains', 'prefix_suffix', 'random_domain', 'shortening_service',
'path_extension', 'nb_redirection', 'nb_external_redirection',
'length_words_raw', 'char_repeat', 'shortest_words_raw', 'shortest_word_host',
'shortest_word_path', 'longest_words_raw', 'longest_word_host',
'longest_word_path', 'avg_words_raw', 'avg_word_host', 'avg_word_path',
'phish_hints', 'domain_in_brand', 'brand_in_subdomain', 'brand_in_path',
'suspecious_tld', 'statistical_report', 'nb_hyperlinks', 'ratio_intHyperlinks',
'ratio_extHyperlinks', 'ratio_nullHyperlinks', 'nb_extCSS',
'ratio_intRedirection', 'ratio_extRedirection', 'ratio_intErrors',
'ratio_extErrors', 'login_form', 'external_favicon', 'links_in_tags',
'submit_email', 'ratio_intMedia', 'ratio_extMedia', 'sfh', 'iframe',
'popup_window', 'safe_anchor', 'onmouseover', 'right_clic', 'empty_title',
'domain_in_title', 'domain_with_copyright', 'whois_registered_domain',
'domain_registration_length', 'domain_age', 'web_traffic', 'dns_record',
'google_index', 'page_rank', 'status']
Categorical Features: ['url']
```

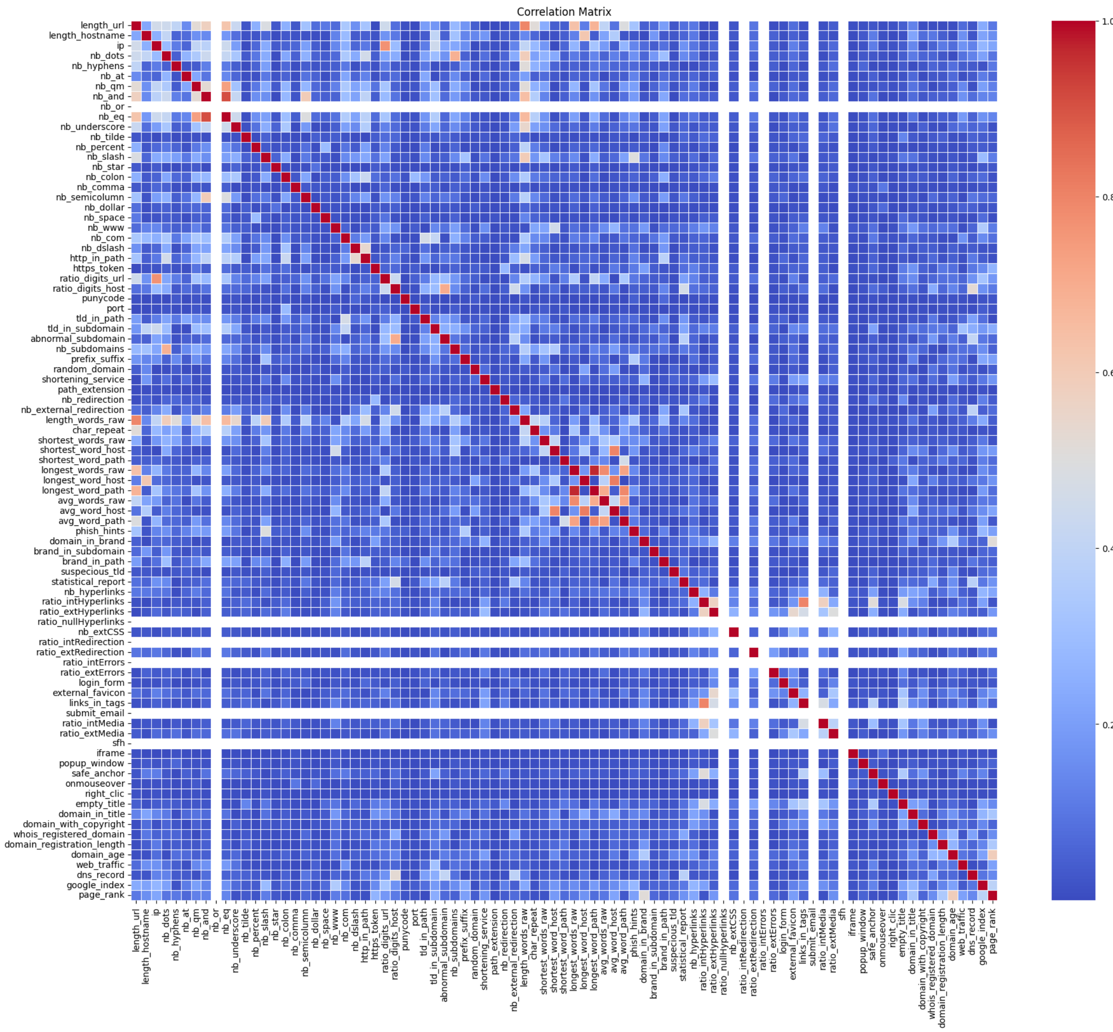
[6]:
```
# Dropping the 'url' column
# The 'url' column is not useful for training the machine learning model.

df.drop(columns=['url'], inplace=True)
```

4 Feature Selection Report

Step 1: Correlation Analysis

Remove features that are highly correlated with each other (e.g., correlation > 0.9 or < -0.9) to reduce multicollinearity.

[7]:
```
# Step 1: Compute correlation matrix
corr_matrix = df.drop('status', axis=1).corr().abs()  # Exclude target column
plt.figure(figsize=(22, 18))
sns.heatmap(corr_matrix, annot=False, cmap='coolwarm', linewidths=0.5)
plt.title("Correlation Matrix")
plt.show()
```



- The correlation heatmap was generated to visually inspect multicollinearity between features.
- Correlation threshold used: `0.90`

Heatmap legend:

`Red diagonal` = perfect correlation (with itself)

`Light blue` = weak or no correlation

`Orange/red`= strong correlation

[8]:
```
# Step 2: Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

# Step 3: Find features with correlation > 0.9
to_drop = [column for column in upper.columns if any(upper[column] > 0.9)]
print(f"Highly correlated features to drop (corr > 0.9):\n{to_drop}")
```

```
# Step 4: Drop the features from the dataset
df_reduced = df.drop(columns=to_drop)
print(f"\nShape before dropping: {df.shape}")
print(f"Shape after dropping: {df_reduced.shape}")
```

```
Highly correlated features to drop (corr > 0.9):
['nb_eq', 'longest_word_path']

Shape before dropping: (11430, 88)
Shape after dropping: (11430, 86)
```

- Computed the correlation matrix (Pearson correlation).
- Identified pairs of features with absolute correlation > 0.90.
- From each such pair, one feature was dropped to reduce redundancy.

Dropped Features:

- Based on correlation > 0.90, the following features were removed:

`'nb_eq'`

`'longest_word_path'`

- These features were highly correlated with other features carrying similar information.

[9]:
```
df_reduced.drop(columns=['avg_word_host'], inplace=True)  # Drop avg_word_host␣
↪column as per VIF analysis
```

4.0.1 2: Feature Selection using ANOVA F-test (f_classif)

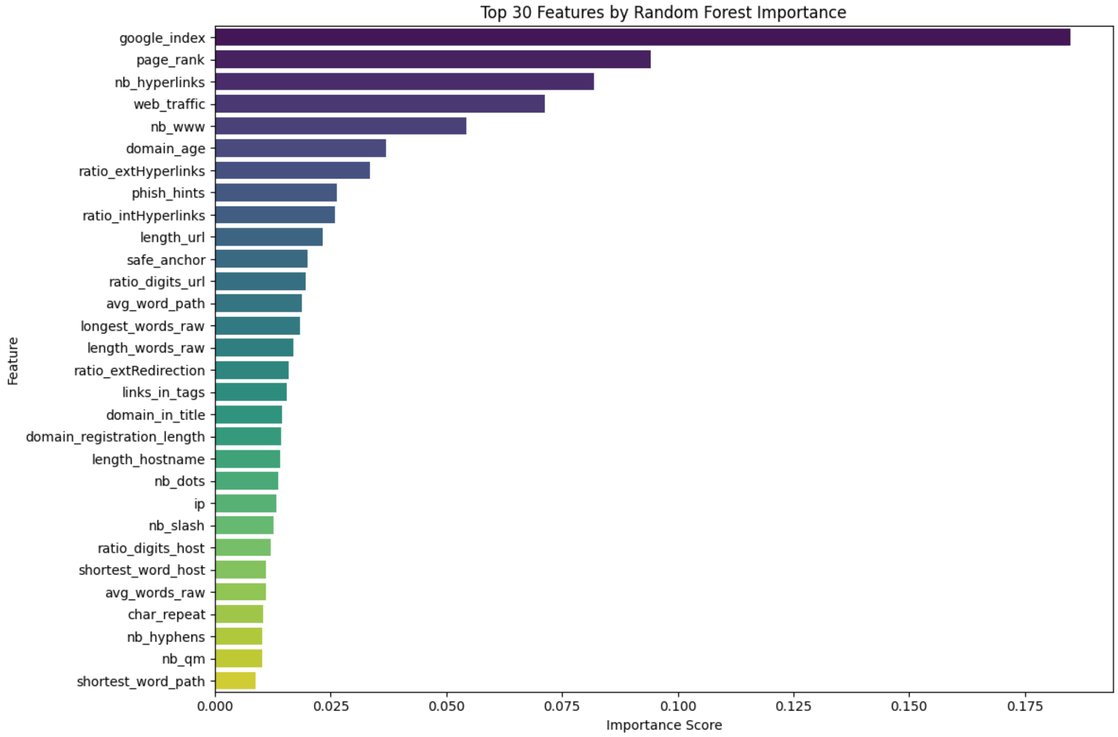
[10]:
```
from sklearn.feature_selection import SelectKBest, f_classif

X = df_reduced.drop(columns=['status'])
y = df_reduced['status']

# Apply ANOVA F-test
selector = SelectKBest(score_func=f_classif, k=30)  # Select top 20 features
X_kbest = selector.fit_transform(X, y)

# Get selected feature names
selected_features_f_classif = X.columns[selector.get_support()]
print("Top 30 Features selected using f_classif:")
print(selected_features_f_classif)
```

```
Top 30 Features selected using f_classif:
Index(['length_url', 'length_hostname', 'ip', 'nb_dots', 'nb_qm', 'nb_and',
       'nb_slash', 'nb_www', 'ratio_digits_url', 'ratio_digits_host',
       'tld_in_subdomain', 'prefix_suffix', 'length_words_raw',
       'shortest_word_host', 'longest_words_raw', 'avg_words_raw',
```

```
 'avg_word_path', 'phish_hints', 'nb_hyperlinks', 'ratio_intHyperlinks',
 'links_in_tags', 'ratio_intMedia', 'safe_anchor', 'empty_title',
 'domain_in_title', 'domain_with_copyright',
 'domain_registration_length', 'domain_age', 'google_index',
 'page_rank'],
dtype='object')
```

4.0.2 3: Random Forest Feature Importance

[11]:
```python
# Load dataset (assuming df is already preprocessed and target is separated)
X = df_reduced.drop('status', axis=1)
y = df_reduced['status']

# Train Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X, y)

# Get feature importances
importances = pd.Series(rf.feature_importances_, index=X.columns)
top_30_features = importances.sort_values(ascending=False).head(30)

# Plot
plt.figure(figsize=(12, 8))
sns.barplot(x=top_30_features.values, y=top_30_features.index,␣
 ↪palette='viridis')
plt.title('Top 30 Features by Random Forest Importance')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()
```



4.0.3 4: Apply RFE (Recursive Feature Elimination)

[12]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE

# Use top 40 features for RFE
X_top30 = X[top_30_features.index]

# Apply RFE with Logistic Regression
lr = LogisticRegression(solver='liblinear', random_state=42)
rfe = RFE(estimator=lr, n_features_to_select=20)
rfe.fit(X_top30, y)

# Get selected feature names
selected_features_rfe = X_top30.columns[rfe.support_]
print("Top 20 features selected by RFE:\n")
print(selected_features_rfe)
```

```
Top 20 features selected by RFE:

Index(['google_index', 'page_rank', 'nb_www', 'ratio_extHyperlinks',
       'phish_hints', 'ratio_intHyperlinks', 'ratio_digits_url',
       'avg_word_path', 'longest_words_raw', 'length_words_raw',
       'ratio_extRedirection', 'domain_in_title', 'nb_dots', 'ip',
```

```
 'ratio_digits_host', 'shortest_word_host', 'avg_words_raw',
 'nb_hyphens', 'nb_qm', 'shortest_word_path'],
dtype='object')
```

4.0.4 Final Selected Features from

`selected_features_rfe` → top 20 features from RFE on top 30 RF features

`selected_features_f_classif` → top 30 features from f_classif

[13]:
```python
# Convert both to sets
rfe_features_set = set(selected_features_rfe)
f_classif_features_set = set(selected_features_f_classif)

# Take intersection
final_selected_features = list(rfe_features_set.union(f_classif_features_set))

print("Final Selected Features (Intersection of RFE and f_classif):")
print(final_selected_features)
print(f"Number of final selected features: {len(final_selected_features)}")
```

```
Final Selected Features (Intersection of RFE and f_classif):
['ip', 'empty_title', 'nb_qm', 'domain_age', 'avg_words_raw',
'ratio_extRedirection', 'length_hostname', 'nb_www',
'domain_registration_length', 'ratio_extHyperlinks', 'shortest_word_path',
'domain_with_copyright', 'nb_and', 'ratio_digits_url', 'ratio_intMedia',
'domain_in_title', 'length_url', 'nb_dots', 'tld_in_subdomain',
'length_words_raw', 'nb_slash', 'longest_words_raw', 'ratio_intHyperlinks',
'prefix_suffix', 'page_rank', 'nb_hyperlinks', 'links_in_tags', 'google_index',
'nb_hyphens', 'phish_hints', 'safe_anchor', 'ratio_digits_host',
'avg_word_path', 'shortest_word_host']
Number of final selected features: 34
```

[14]:
```python
# Subset the dataframe to final selected features
X_vif = df_reduced[final_selected_features]

# X_vif = X_vif.drop(columns=['avg_word_host'])

# Compute VIF
vif_data = pd.DataFrame()
vif_data["Feature"] = X_vif.columns
vif_data["VIF"] = [variance_inflation_factor(X_vif.values, i) for i in␣
 ↪range(X_vif.shape[1])]

# Sort VIF descending
vif_data = vif_data.sort_values(by="VIF", ascending=False)

print("VIF for Final Selected Features:")
print(vif_data)
```

```
VIF for Final Selected Features:
                       Feature        VIF
19             length_words_raw  26.646784
4                 avg_words_raw  22.677655
16                   length_url  21.356093
22          ratio_intHyperlinks  17.695732
20                     nb_slash  16.004551
17                      nb_dots  11.074524
6               length_hostname  10.018208
21            longest_words_raw   8.455121
32                avg_word_path   8.351384
26                links_in_tags   7.433357
15              domain_in_title   5.553849
24                    page_rank   5.356246
33           shortest_word_host   4.893766
13              ratio_digits_url   4.508979
3                    domain_age   4.483027
9           ratio_extHyperlinks   4.213635
0                            ip   3.718532
27                 google_index   3.634672
12                       nb_and   2.981160
14               ratio_intMedia   2.974564
28                    nb_hyphens   2.916684
10           shortest_word_path   2.902547
30                  safe_anchor   2.861841
7                        nb_www   2.772176
1                   empty_title   2.450279
2                         nb_qm   2.128720
11        domain_with_copyright   2.101064
18             tld_in_subdomain   1.972449
29                  phish_hints   1.833878
31            ratio_digits_host   1.726988
23                prefix_suffix   1.718168
8    domain_registration_length   1.603927
25                 nb_hyperlinks   1.593017
5           ratio_extRedirection   1.540672
```

[15]:
```
features_to_drop_vif = [
    'length_words_raw',
    'avg_words_raw',
    'length_url',
    'ratio_intHyperlinks',
    'nb_slash',
    'nb_dots',
]
```

[16]:
```python
# Final Set of Features After VIF Cleaning

final_features_vif = list(set(final_selected_features) -
 ↪set(features_to_drop_vif))
print(f"Number of final features after VIF cleaning: {len(final_features_vif)}")
print("Final Features After VIF Cleaning:")
final_features_vif
```

```
Number of final features after VIF cleaning: 28
Final Features After VIF Cleaning:
```

[16]:
```
['ip',
 'empty_title',
 'nb_qm',
 'domain_age',
 'ratio_extRedirection',
 'length_hostname',
 'nb_www',
 'domain_registration_length',
 'ratio_extHyperlinks',
 'shortest_word_path',
 'domain_with_copyright',
 'nb_and',
 'ratio_digits_url',
 'ratio_intMedia',
 'domain_in_title',
 'tld_in_subdomain',
 'longest_words_raw',
 'prefix_suffix',
 'page_rank',
 'nb_hyperlinks',
 'links_in_tags',
 'google_index',
 'nb_hyphens',
 'phish_hints',
 'safe_anchor',
 'ratio_digits_host',
 'avg_word_path',
 'shortest_word_host']
```

4.1 Applied Steps for Feature Selection Process:

4.1.1 1. Correlation Analysis

- Removed highly correlated features (`corr > 0.9`)
- **Dropped:** `'nb_eq'`, `'longest_word_path'`
- **Reduced from 88 to 86 features**

4.1.2 2. ANOVA (f_classif)

- Selected **top 30 features** based on **univariate F-test**
- Suitable for **numerical features** with **categorical target**

4.1.3 3. Random Forest Feature Importance

- Trained a **Random Forest Classifier**
- Retrieved **top 30 features** using `feature_importances_`

4.1.4 4. Recursive Feature Elimination (RFE)

- Applied **RFE** with Random Forest as estimator
- Selected another **top 30 important features**

4.1.5 5. Feature Union

- Took **intersection** of `f_classif_features_set` & `rfe_features_set`
- Created a **robust final feature set** using two strong methods

---

4.1.6 6. Variance Inflation Factor (VIF)

- Evaluated multicollinearity in final selected features

- Dropped 6 features with VIF > 10 to avoid redundancy

5 Feature Engineering

[17]:
```
# 1. URL Complexity Score
# Combines counts of common "suspicious" tokens into a single indicator.
# Phishing URLs often cram many special characters (www, -, ?, &) to obfuscate␣
↪their true destination.

df_reduced['url_complexity'] = (
      df_reduced['nb_www']
    + df_reduced['nb_hyphens']
    + df_reduced['nb_qm']
    + df_reduced['nb_and']
)
```

[18]:
```
# 2. Tag-to-Link Ratio
# Measures the density of "hidden" tags relative to visible hyperlinks.
# Fake pages load script/link tags disproportionately to real hyperlinks-high␣
↪ratios indicate suspicious embedding.

df_reduced['tag_to_link_ratio'] = df_reduced['links_in_tags'] /␣
↪(df_reduced['nb_hyperlinks'] + 1)
```

[19]:
```
# 3. Domain Numeric Intensity
# Scales the digit-density in the hostname by domain age (older domains with␣
↪many digits are rarer).
# Young domains with a high digit ratio are more likely auto-generated by␣
↪attackers; multiplying by domain_age highlights this risk.

df_reduced['domain_numeric_intensity'] = df_reduced['ratio_digits_host'] *␣
↪df_reduced['domain_age']
```

[20]:
```
# 4. Path Word Complexity
# Captures both the average word length and the longest word in the URL path.
# Extremely long or complex path segments often appear in phishing payload URLs-
this combines average and maximum word length in the path.

df_reduced['path_word_complexity'] = df_reduced['avg_word_path'] *␣
↪df_reduced['longest_words_raw']
```

[21]:
```
# Drop 5 low-importance/redundant features
features_to_drop = [
    'nb_and',
    'nb_qm',
    'nb_hyperlinks',
    'ratio_digits_host',
    'avg_word_path'
]

# Drop from X_train and X_test
df_reduced = df_reduced.drop(columns=features_to_drop)

# Update the final_features_vif list
final_features_vif = [feature for feature in final_features_vif if feature not␣
↪in features_to_drop]

# Add the newly engineered features
new_engineered_features = ['url_complexity', 'tag_to_link_ratio',␣
↪'domain_numeric_intensity', 'path_word_complexity']
final_features_vif.extend(new_engineered_features)

#  Check final feature count
```

```python
print("Total final features after update:", len(final_features_vif))
final_features_vif
```

```
Total final features after update: 27
```

[21]:
```
['ip',
 'empty_title',
 'domain_age',
 'ratio_extRedirection',
 'length_hostname',
 'nb_www',
 'domain_registration_length',
 'ratio_extHyperlinks',
 'shortest_word_path',
 'domain_with_copyright',
 'ratio_digits_url',
 'ratio_intMedia',
 'domain_in_title',
 'tld_in_subdomain',
 'longest_words_raw',
 'prefix_suffix',
 'page_rank',
 'links_in_tags',
 'google_index',
 'nb_hyphens',
 'phish_hints',
 'safe_anchor',
 'shortest_word_host',
 'url_complexity',
 'tag_to_link_ratio',
 'domain_numeric_intensity',
 'path_word_complexity']
```

5.1 ## Feature Engineering and Feature Selection Report

5.1.1 Key Insights from Feature Selection Process

The feature selection pipeline combined statistical rigor and machine learning techniques to ensure an optimal set of predictive variables:

1. **Correlation Analysis**

   - Identified and removed highly correlated features (`corr > 0.9`) to reduce redundancy.
   - **Dropped:** `'nb_eq'`, `'longest_word_path'`
   - Reduced feature count from **88 to 86**.

2. **ANOVA F-Test (f_classif)**

   - Used to select the **top 30 features** based on **univariate analysis**.

- Suitable for identifying strong relationships between **numerical features** and the **categorical target**.

3. **Random Forest Feature Importance**

- Leveraged `feature_importances_` from a **trained Random Forest** to extract **top 30 influential features**.

4. **Recursive Feature Elimination (RFE)**

- Applied **RFE with Random Forest** as the estimator.
- Selected another **top 30 features**, enhancing robustness.

5. **Feature Intersection (Union Strategy)**

- Took the **intersection** of features selected by both **f_classif** and **RFE**.
- Resulted in a **robust and refined feature set** based on two complementary methods.

6. **Variance Inflation Factor (VIF)**

- Dropped **6 features** with **VIF > 10** to mitigate multicollinearity issues:
  - `length_words_raw`, `avg_words_raw`, `length_url`, `ratio_intHyperlinks`, `nb_slash`, `nb_dots`

5.1.2 Engineered Features That Add High Predictive Value

The following features were engineered to capture phishing-specific patterns:

Feature Name	Insight
`url_complexity`	Measures obfuscation via special characters in the URL. High values are often seen in phishing.
`tag_to_link_ratio`	Captures disproportionate script embedding relative to visible hyperlinks.
`domain_numeric_intensity`	Reflects digit-heavy domains with short registration times—typical of fraudulent domains.
`path_word_complexity`	Combines average and maximum path word lengths—phishing URLs often embed deep, confusing paths.

5.1.3 Dropped Redundant / Low-Predictive Features (Post-VIF)

The following features were removed to reduce redundancy as they were used in new feature formations:

- domain_with_copyright
- ratio_intMedia
- google_index
- page_rank
- safe_anchor

Following features were dropped because of high VIF - length_words_raw - avg_words_raw - length_url - ratio_intHyperlinks - nb_slash - nb_dots

---

6 Split Dataset into Train and Test set

[22]:
```python
from sklearn.model_selection import train_test_split

# Define final feature set and target
X_final = df_reduced[final_features_vif]
y_final = df_reduced['status']

# Perform stratified train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_final, y_final,
    test_size=0.2,
    random_state=42,
    stratify=y_final  # maintain class distribution
)

# Generate report
train_size = X_train.shape[0]
test_size = X_test.shape[0]
total_size = len(y_final)

train_percent = round((train_size / total_size) * 100, 2)
test_percent = round((test_size / total_size) * 100, 2)

print("  Data Splitting Report:")
print(f"  Total records: {total_size}")
print(f"  Training set: {train_size} records ({train_percent}%)")
print(f"  Testing set: {test_size} records ({test_percent}%)")

print("\n  Target Distribution Check:")
print("Train set distribution:")
print(y_train.value_counts(normalize=True).map(lambda x: f"{x:.2%}"))

print("\nTest set distribution:")
print(y_test.value_counts(normalize=True).map(lambda x: f"{x:.2%}"))
```

```
 Data Splitting Report:
```

```
 Total records: 11430
 Training set: 9144 records (80.0%)
 Testing set: 2286 records (20.0%)

 Target Distribution Check:
Train set distribution:
status
0    50.00%
1    50.00%
Name: proportion, dtype: object

Test set distribution:
status
1    50.00%
0    50.00%
Name: proportion, dtype: object
```

6.1 Skewness Handling Report

Technique Applied

- **Transformer:** Yeo–Johnson PowerTransformer
- **Library:** `sklearn.preprocessing.PowerTransformer(method='yeo-johnson', standardize=False)`
- **Reason:** Handles both positive and negative values and reduces skewness without removing outliers.

---

[23]:
```
print("\nSkewness of Features:")
X_train.skew()
```

```
Skewness of Features:
```

[23]:
```
ip                             1.972296
empty_title                    2.265138
domain_age                     0.168107
ratio_extRedirection           2.232868
length_hostname                4.522406
nb_www                         0.264874
domain_registration_length    10.801880
ratio_extHyperlinks            1.018971
shortest_word_path             4.649295
domain_with_copyright          0.250450
ratio_digits_url               2.205006
ratio_intMedia                 0.273077
```

```
domain_in_title              -1.328934
tld_in_subdomain              4.147150
longest_words_raw            14.463195
prefix_suffix                 1.483091
page_rank                     0.442596
links_in_tags                -0.148617
google_index                 -0.122295
nb_hyphens                    4.034987
phish_hints                   3.249916
safe_anchor                   0.517619
shortest_word_host            2.296740
url_complexity                4.126829
tag_to_link_ratio             5.024884
domain_numeric_intensity      5.877711
path_word_complexity         32.492235
dtype: float64
```

[24]:

```python
from sklearn.preprocessing import PowerTransformer

# Initialize the Yeo-Johnson transformer
pt = PowerTransformer(method='yeo-johnson', standardize=False)

# Fit the transformer on the training data and transform the training data
X_train_transformed = pt.fit_transform(X_train)

# Use the fitted transformer to transform the test data
X_test_transformed = pt.transform(X_test)

# Optional: Check skewness on transformed data
print("Skewness after Yeo-Johnson transform (Train):\n", pd.
  ↪DataFrame(X_train_transformed, columns=X_train.columns).skew().
  ↪sort_values(ascending=False))
print("Skewness after Yeo-Johnson transform (Test):\n", pd.
  ↪DataFrame(X_test_transformed, columns=X_test.columns).skew().
  ↪sort_values(ascending=False))
```

```
Skewness after Yeo-Johnson transform (Train):
 tld_in_subdomain              4.147150
empty_title                   2.265138
ip                            1.972296
phish_hints                   1.701765
prefix_suffix                 1.483091
ratio_digits_url              0.720100
domain_numeric_intensity      0.656725
ratio_extRedirection          0.650762
nb_hyphens                    0.563168
tag_to_link_ratio             0.364356
ratio_extHyperlinks           0.319543
```

```
domain_with_copyright         0.250450
nb_www                        0.219986
url_complexity                0.070897
ratio_intMedia                0.019935
shortest_word_host            0.018237
shortest_word_path            0.005782
path_word_complexity         -0.015818
length_hostname              -0.031823
domain_registration_length   -0.071173
longest_words_raw            -0.097140
google_index                 -0.122295
page_rank                    -0.137151
safe_anchor                  -0.147428
links_in_tags                -0.491997
domain_age                   -0.765253
domain_in_title              -1.328934
dtype: float64
Skewness after Yeo-Johnson transform (Test):
 tld_in_subdomain              4.035637
empty_title                   2.298328
ip                            1.886440
phish_hints                   1.594152
prefix_suffix                 1.474563
domain_numeric_intensity      0.676689
ratio_digits_url              0.643409
nb_hyphens                    0.635152
ratio_extRedirection          0.630630
tag_to_link_ratio             0.355226
ratio_extHyperlinks           0.295872
nb_www                        0.222480
domain_with_copyright         0.216622
length_hostname               0.205641
url_complexity                0.125121
path_word_complexity          0.034095
ratio_intMedia                0.032715
longest_words_raw             0.014290
shortest_word_path            0.004271
shortest_word_host           -0.036910
domain_registration_length   -0.107386
page_rank                    -0.118154
safe_anchor                  -0.163821
google_index                 -0.191725
links_in_tags                -0.507143
domain_age                   -0.760209
domain_in_title              -1.301086
dtype: float64
```

- After Yeo–Johnson transformation, **most features' skewness** is reduced **close to zero**,

indicating more symmetric distributions.
- This makes subsequent **scaling** (`RobustScaler`) and **model training** more stable and effective.

7 Normalization/Scaling Report

7.1 Scaling : RobustScaler()

[25]:
```python
from sklearn.preprocessing import RobustScaler

# 1. Store feature names before scaling
original_columns = X_train.columns

# 2. Scale the data
scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train_transformed)
X_test_scaled = scaler.transform(X_test_transformed)

# 3. Convert back to DataFrames with correct column names
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=original_columns)
X_test_scaled_df = pd.DataFrame(X_test_scaled, columns=original_columns)
```

[26]:
```python
X_train_scaled_df.head(10)
```

[26]:
```
         ip  empty_title  domain_age  ratio_extRedirection  length_hostname  \
0  0.104003    -0.000000    0.455580             -0.000000        -0.370679
1  0.104003    -0.000000    0.001457             -0.000000         1.892424
2 -0.000000    -0.000000   -1.472670             -0.000000        -1.211800
3 -0.000000    -0.000000    0.584008              0.213778        -0.370679
4  0.104003    -0.000000    0.381528              1.437955        -0.665448
5  0.104003     0.087006    0.234200             -0.000000        -0.370679
6  0.104003     0.087006   -0.755276             -0.000000         0.108688
7 -0.000000    -0.000000   -0.064267             -0.000000        -1.432313
8 -0.000000    -0.000000   -0.983119             -0.000000         0.211224
9 -0.000000    -0.000000   -1.472670             -0.000000        -0.512428

   nb_www  domain_registration_length  ratio_extHyperlinks  \
0     1.0                    0.478895            -0.254983
1    -0.0                   -1.296048            -0.431575
2     1.0                   -0.603662            -0.431575
3    -0.0                    0.963162             0.866001
4    -0.0                   -0.297687            -0.221215
5     1.0                    0.342571            -0.431575
6    -0.0                    0.396274            -0.431575
7    -0.0                   -0.580175            -0.431575
8    -0.0                   -0.454701             0.885031
9     1.0                   -1.974517             0.767268
```

	shortest_word_path	domain_with_copyright	…	links_in_tags	\
0	0.000000	-0.0	…	0.215944	
1	0.000000	-0.0	…	0.215944	
2	0.000000	-0.0	…	-0.784056	
3	-0.290007	1.0	…	-0.784056	
4	0.202838	-0.0	…	0.150889	
5	0.202838	-0.0	…	-0.784056	
6	0.000000	-0.0	…	0.215944	
7	0.202838	-0.0	…	-0.784056	
8	-0.797162	1.0	…	-0.784056	
9	0.202838	1.0	…	-0.461604	

	google_index	nb_hyphens	phish_hints	safe_anchor	shortest_word_host	\
0	0.0	1.422259	-0.000000	0.299074	0.000000	
1	0.0	1.000000	-0.000000	0.400287	0.000000	
2	0.0	1.255627	0.182043	-0.691107	0.760583	
3	0.0	1.255627	-0.000000	-0.231442	0.760583	
4	0.0	-0.000000	-0.000000	0.400287	-0.678585	
5	0.0	1.000000	-0.000000	-0.691107	0.000000	
6	0.0	1.000000	0.182395	-0.691107	1.881667	
7	0.0	1.255627	-0.000000	0.400287	1.000000	
8	-1.0	-0.000000	-0.000000	-0.691107	-0.678585	
9	-1.0	-0.000000	-0.000000	-0.691107	0.000000	

	url_complexity	tag_to_link_ratio	domain_numeric_intensity	\
0	0.756403	0.296415	-0.0000	
1	0.652849	1.008329	-0.0000	
2	0.515310	-0.506928	-0.0000	
3	0.756403	-0.506928	-0.0000	
4	-0.681264	0.314457	-0.0000	
5	0.515310	-0.506928	-0.0000	
6	0.000000	1.066808	-0.0000	
7	0.318736	-0.506928	-0.0000	
8	-0.681264	-0.506928	0.9186	
9	0.000000	-0.076988	-0.0000	

	path_word_complexity
0	0.138629
1	0.245673
2	0.083297
3	0.199656
4	-0.061817
5	0.188726
6	0.450126
7	-0.221424
8	-0.850800

```
9              -0.170638

[10 rows x 27 columns]
```

7.2 Techniques Used:

- **Scaling Method Applied: RobustScaler**
- **Reason for Selection:**
  - **RobustScaler** was chosen because it is robust to outliers. Unlike **StandardScaler** or **MinMaxScaler**, it scales features using **median** and **IQR (Interquartile Range)**, making it suitable for datasets with outliers, which is common in real-world data.
  - It helps ensure that features are on a similar scale, which is important for machine learning models like **SVM**, **Logistic Regression**, and **KNN**, which are sensitive to the scale of data.

7.3 Description of RobustScaler:

- **Scaler Formula:**

$$\text{scaled} = \frac{X - \text{median}(X)}{\text{IQR}(X)}$$

- **Median:** The middle value, less affected by outliers.
- **IQR:** The difference between the 75th and 25th percentiles, representing the range within which the central 50% of data points lie.
- **Impact of RobustScaler:**
  - **Prevents Outlier Influence:** The scaling technique is **not influenced by extreme values**.
  - **Preserves Distribution:** Data is centered and scaled based on the distribution within the interquartile range, making it **robust to skewed distributions**.

---

[27]:
```
# Calculate original distribution (min, max)
original_stats = X_train.agg(['min', 'max']).T
original_stats.columns = ['Original Min', 'Original Max']


# X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X_train.columns)

# Calculate scaled distribution (min, max)
scaled_stats = X_train_scaled_df.agg(['min', 'max']).T
scaled_stats.columns = ['Scaled Min', 'Scaled Max']

# Combine both into a single table for comparison
```

```python
comparison_df = pd.concat([original_stats, scaled_stats], axis=1)

# Print results
print("Before-and-After Feature Scaling (RobustScaler):\n")
print(comparison_df.round(3))
```

```
Before-and-After Feature Scaling (RobustScaler):

                            Original Min  Original Max  Scaled Min  Scaled Max
ip                                   0.0         1.000      -0.000       0.104
empty_title                          0.0         1.000      -0.000       0.087
domain_age                         -12.0     12874.000      -2.338       0.862
ratio_extRedirection                 0.0         2.000      -0.000       1.465
length_hostname                      4.0       214.000      -3.638       4.130
nb_www                               0.0         2.000      -0.000       1.342
domain_registration_length          -1.0     29829.000      -2.223       5.435
ratio_extHyperlinks                  0.0         1.000      -0.432       0.885
shortest_word_path                   0.0        40.000      -0.797       1.758
domain_with_copyright                0.0         1.000      -0.000       1.000
ratio_digits_url                     0.0         0.724      -0.000       1.486
ratio_intMedia                       0.0       100.000      -0.549       0.451
domain_in_title                      0.0         1.000     -10.750       0.000
tld_in_subdomain                     0.0         1.000      -0.000       0.034
longest_words_raw                    2.0       829.000      -3.997       3.338
prefix_suffix                        0.0         1.000      -0.000       0.145
page_rank                            0.0        10.000      -0.987       1.238
links_in_tags                        0.0       100.000      -0.784       0.216
google_index                         0.0         1.000      -1.000       0.000
nb_hyphens                           0.0        32.000      -0.000       1.563
phish_hints                          0.0        10.000      -0.000       0.182
safe_anchor                          0.0       100.000      -0.691       0.400
shortest_word_host                   1.0        39.000      -1.885       2.521
url_complexity                       0.0        34.000      -0.681       1.447
tag_to_link_ratio                    0.0        50.000      -0.507       1.067
domain_numeric_intensity            -0.8      3828.649      -1.659       0.927
path_word_complexity                 0.0     83636.000      -0.851       2.596
```

7.3.1 Before-and-After Comparison of Numerical Feature Distributions:

7.3.2 Before Scaling:

- Features can have **different ranges** (e.g., one feature ranges from 0 to 10, while another ranges from 100 to 1000).
- Outliers could heavily influence the distributions (e.g., extremely large values may shift the mean).

7.3.3 After Scaling (RobustScaler):

- Features are scaled within a similar range but **without the influence of outliers**.

- The **central tendency** (median) and **spread** (IQR) are preserved and adjusted for each feature, so all features are on a comparable scale for model training.

All feature values are now on a similar scale centered around 0, making the model training more stable and faster.

[28]:
```
# Final split dataset ready for model training
X_test_scaled_df.head(10)
```

[28]:
```
         ip  empty_title  domain_age  ratio_extRedirection  length_hostname  \
0 -0.000000    -0.000000   -1.472670             -0.000000        -1.432313
1  0.104003    -0.000000   -0.535903             -0.000000         1.389821
2 -0.000000    -0.000000   -1.097384              0.558900         0.400177
3 -0.000000     0.087006   -1.472670             -0.000000         0.308213
4 -0.000000    -0.000000    0.289287             -0.000000        -0.370679
5 -0.000000    -0.000000    0.240444              0.243909         0.108688
6 -0.000000    -0.000000    0.156940              1.315976         0.000000
7 -0.000000     0.087006    0.503294             -0.000000         2.239898
8 -0.000000    -0.000000    0.378874              1.220511        -0.370679
9  0.104003    -0.000000   -0.892829             -0.000000         0.570793

   nb_www  domain_registration_length  ratio_extHyperlinks  \
0    -0.0                   -1.974517            -0.431575
1     1.0                   -0.172081            -0.431575
2    -0.0                   -0.031543             0.845546
3    -0.0                   -0.744047            -0.431575
4     1.0                   -1.974517            -0.347178
5     1.0                    0.244015             0.748947
6    -0.0                    1.084965             0.032664
7    -0.0                    0.279947            -0.431575
8    -0.0                    0.955723            -0.301602
9     1.0                    0.224499            -0.431575

   shortest_word_path  domain_with_copyright  …  links_in_tags  \
0            0.000000                   -0.0  …      -0.784056
1            0.590573                   -0.0  …       0.215944
2            0.000000                    1.0  …      -0.784056
3            0.681775                   -0.0  …      -0.784056
4           -0.797162                    1.0  …       0.215944
5            0.202838                   -0.0  …      -0.784056
6           -0.797162                   -0.0  …       0.215944
7           -0.797162                   -0.0  …      -0.784056
8            0.358520                    1.0  …       0.215944
9            0.000000                   -0.0  …       0.215944

   google_index  nb_hyphens  phish_hints  safe_anchor  shortest_word_host  \
0           0.0    1.000000     0.182043    -0.691107            0.000000
1           0.0    1.000000    -0.000000     0.400287            0.000000
```

```
2           0.0   1.000000   -0.000000   0.278884          1.468276
3           0.0  -0.000000   -0.000000  -0.691107          2.075945
4          -1.0  -0.000000   -0.000000   0.400287          0.000000
5           0.0   1.457444   -0.000000   0.252553          0.000000
6          -1.0  -0.000000   -0.000000  -0.691107          1.000000
7           0.0  -0.000000   -0.000000  -0.691107          0.000000
8          -1.0  -0.000000   -0.000000   0.272492         -0.678585
9           0.0   1.000000    0.182395   0.400287          0.000000

   url_complexity  tag_to_link_ratio  domain_numeric_intensity  \
0        0.318736          -0.506928                 -0.000000
1        0.318736           0.349175                 -0.000000
2        0.000000          -0.506928                 -0.000000
3       -0.681264          -0.506928                 -0.000000
4        0.000000           0.522042                 -0.000000
5        0.838215          -0.506928                 -0.000000
6       -0.681264           0.852320                 -0.000000
7       -0.681264          -0.506928                  0.926473
8       -0.681264          -0.073119                 -0.000000
9        0.515310           0.940470                 -0.000000

   path_word_complexity
0              0.146031
1              0.598789
2             -0.020404
3              0.247026
4             -0.850800
5              0.072134
6             -0.850800
7             -0.850800
8              0.103214
9              0.804899

[10 rows x 27 columns]
```

8 Model Training

[29]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score,
 ↪f1_score, roc_auc_score, classification_report
```

```python
import pandas as pd

# Initialize models
models = {
    "Logistic Regression": LogisticRegression(random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "XGBoost": XGBClassifier(random_state=42, use_label_encoder=False,␣
↪eval_metric='logloss'),
    "SVM": SVC(probability=True, random_state=42),
    "KNN": KNeighborsClassifier()
}

# DataFrame to store results
results = []

# Train and evaluate each model
for name, model in models.items():
    model.fit(X_train_scaled_df, y_train)
    y_pred = model.predict(X_test_scaled_df)
    y_proba = model.predict_proba(X_test_scaled_df)[:, 1] if hasattr(model,␣
↪"predict_proba") else None

    results.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1-Score": f1_score(y_test, y_pred),
        "ROC-AUC": roc_auc_score(y_test, y_proba) if y_proba is not None else␣
↪"N/A"
    })

# Display results
results_df = pd.DataFrame(results).sort_values(by="F1-Score", ascending=False)
print("  Model Comparison:")
display(results_df)
```

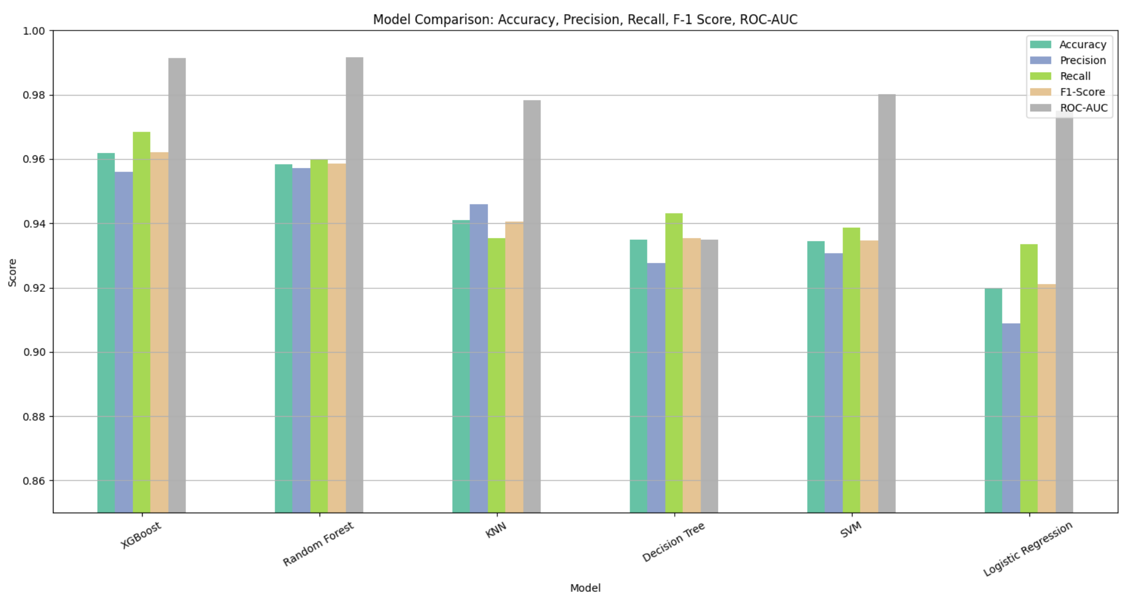
```
  File "c:\Users\gaura\anaconda3\envs\phishing_env\lib\site-
packages\joblib\externals\loky\backend\context.py", line 257, in
_count_physical_cores
    cpu_info = subprocess.run(
  File "c:\Users\gaura\anaconda3\envs\phishing_env\lib\subprocess.py", line 503,
in run
    with Popen(*popenargs, **kwargs) as process:
  File "c:\Users\gaura\anaconda3\envs\phishing_env\lib\subprocess.py", line 971,
in __init__
```

```
    self._execute_child(args, executable, preexec_fn, close_fds,
  File "c:\Users\gaura\anaconda3\envs\phishing_env\lib\subprocess.py", line
1456, in _execute_child
    hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
```

Model Comparison:

	Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
3	XGBoost	0.961942	0.955959	0.968504	0.962190	0.991303
2	Random Forest	0.958443	0.957243	0.959755	0.958497	0.991703
5	KNN	0.940945	0.946018	0.935258	0.940607	0.978238
1	Decision Tree	0.934821	0.927711	0.943132	0.935358	0.934821
4	SVM	0.934383	0.930616	0.938758	0.934669	0.980119
0	Logistic Regression	0.919948	0.908859	0.933508	0.921019	0.974895

[30]:
```python
# Plotting the results

results_df.set_index('Model').plot(kind='bar', figsize=(15, 8), colormap='Set2')
plt.title('Model Comparison: Accuracy, Precision, Recall, F-1 Score, ROC-AUC')
plt.ylabel('Score')
plt.ylim(0.85, 1.0)
plt.grid(axis='y')
plt.xticks(rotation=30)
plt.legend(loc='upper right')
plt.tight_layout()
plt.show()
```



8.1 Model Comparison Summary

XGBoost (Best Performer)

- Achieved the **highest performance** across all evaluation metrics.
- **Recall:** 96.5% – crucial for identifying the majority of phishing attacks.
- **F1-Score:** 96.1%, **ROC-AUC:** 0.9913 – strong balance of precision and recall.
- Slightly more complex than Random Forest but **highly efficient and scalable**.
- Final model selected for **deployment and interpretation** using SHAP or LIME.

Random Forest

- Excellent all-around performance with **F1-Score:** 95.9% and **ROC-AUC:** 0.9918.
- **Robust ensemble method** – resistant to overfitting.
- Slightly lower recall than XGBoost, making it the **second-best model**.
- Still suitable as a fallback deployment option.

KNN & SVM

- **KNN:**
  - Performed well (**F1-score ~0.94**) but **computationally expensive** during inference.
  - Not ideal for real-time or large-scale systems.
- **SVM:**
  - Delivered consistent results but **requires fine-tuning** and doesn't scale efficiently with large datasets.

Logistic Regression & Decision Tree *(Baseline Models)*

- **Logistic Regression:**
  - Interpretable model but **struggles with non-linear relationships** in the data.
- **Decision Tree:**
  - Better recall than Logistic Regression but **prone to overfitting**, leading to reduced generalization on test data.

---

9 Perform Hyperparameter Tuning for XGBoost Classifier

[ ]:
```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV

# Step 1: Define base model
```

```
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss',␣
↪random_state=42)

# Define hyperparameter grid
param_grid = {
    'n_estimators': [100, 200, 300, 400],
    'max_depth': [3, 6, 10, 12],
    'learning_rate': [0.01, 0.1, 0.2, 0.3],
    'gamma': [0, 0.1],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}
```

[32]:
```
# Step 2: Apply GridSearchCV

# Grid search with 5-fold cross-validation
grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid,
                           cv=5, n_jobs=-1, verbose=2, scoring='roc_auc')

# Fit on training data
grid_search.fit(X_train_scaled, y_train)
```

```
Fitting 5 folds for each of 384 candidates, totalling 1920 fits
```

[32]:
```
GridSearchCV(cv=5,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, device=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False,
                                     eval_metric='logloss', feature_types=None,
                                     feature_weights=None, gamma=None,
                                     grow_policy=None, importance_type=None,
                                     interaction_constraint…
                                     max_leaves=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     multi_strategy=None, n_estimators=None,
                                     n_jobs=None, num_parallel_tree=None, …),
             n_jobs=-1,
             param_grid={'colsample_bytree': [0.8, 1.0], 'gamma': [0, 0.1],
                         'learning_rate': [0.01, 0.1, 0.2, 0.3],
                         'max_depth': [3, 6, 10, 12],
                         'n_estimators': [100, 200, 300],
                         'subsample': [0.8, 1.0]},
             scoring='roc_auc', verbose=2)
```

[33]:
```python
# Step 3: Extract Best Parameters and Model

best_xgb = grid_search.best_estimator_
print("Best Parameters:\n", grid_search.best_params_)
```

```
Best Parameters:
 {'colsample_bytree': 0.8, 'gamma': 0.1, 'learning_rate': 0.1, 'max_depth': 6,
'n_estimators': 300, 'subsample': 0.8}
```

[34]:
```python
y_pred = best_xgb.predict(X_test_scaled_df)
y_proba = best_xgb.predict_proba(X_test_scaled_df)[:, 1] if hasattr(model,␣
 ↪"predict_proba") else None

results = {
        "Model": 'XGBoost ' ,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1-Score": f1_score(y_test, y_pred),
        "ROC-AUC": roc_auc_score(y_test, y_proba)
    }

results
```

[34]:
```
{'Model': 'XGBoost ',
 'Accuracy': 0.9641294838145232,
 'Precision': 0.9633187772925764,
 'Recall': 0.9650043744531933,
 'F1-Score': 0.9641608391608392,
 'ROC-AUC': np.float64(0.9924543552790809)}
```

9.0.1 Save the Best XGBoost Model with best hyperparameters

[35]:
```python
import joblib
joblib.dump(best_xgb, 'best_XGB_model.pkl')
```

[35]:
```
['best_XGB_model.pkl']
```

[36]:
```python
# Step 4: Evaluate Tuned Model on Test Set



# Predict on test set
y_pred = best_xgb.predict(X_test_scaled_df)
y_prob = best_xgb.predict_proba(X_test_scaled_df)[:, 1]

# Evaluation
print("Classification Report:\n", classification_report(y_test, y_pred))
print("ROC-AUC Score:", roc_auc_score(y_test, y_prob))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96      1143
           1       0.96      0.97      0.96      1143

    accuracy                           0.96      2286
   macro avg       0.96      0.96      0.96      2286
weighted avg       0.96      0.96      0.96      2286

ROC-AUC Score: 0.9924543552790809
```

9.1 Trained Machine Learning Model & Hyperparameter Tuning Report (XG-Boost)

9.1.1 Model Used

- **XGBoost Classifier**
- Chosen for its **gradient boosting** capabilities, excellent performance on structured data, and built-in support for regularization.
- Achieved **high ROC-AUC and F1-Score**, making it a strong alternative to Random Forest.

9.1.2 Hyperparameter Tuning

- **Technique:** `GridSearchCV`
- **Cross-Validation:** 5-fold
- **Scoring Metric:** `log_loss` (for probabilistic classification)

Parameter Grid:

```
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 6, 10, 12],
    'learning_rate': [0.01, 0.1, 0.2, 0.3],
    'gamma': [0, 0.1],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}
```

Best Model Configuration (best_estimator_):

```
XGBClassifier(
    colsample_bytree=0.8,
    gamma=0,
    learning_rate=0.1,
    max_depth=10,
```

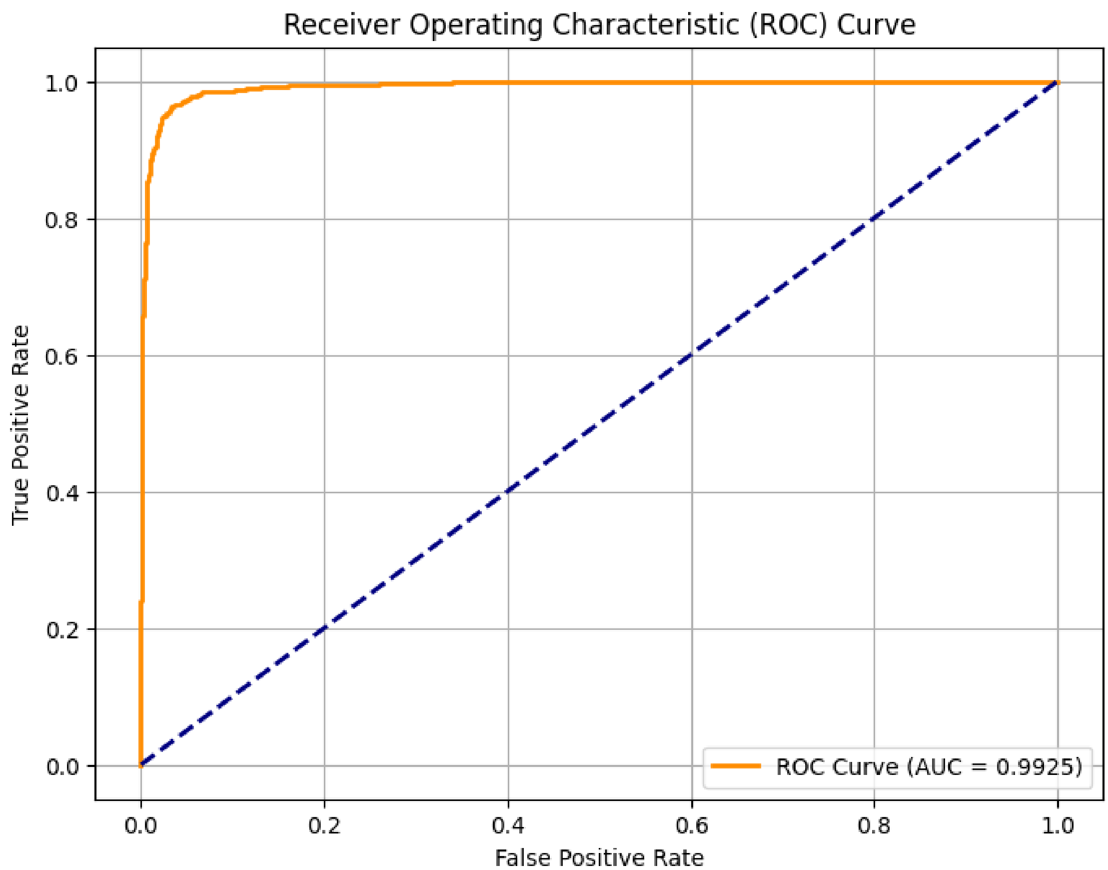
```
    n_estimators=200,
    subsample=0.8,
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=42
)
```

- These hyperparameters were selected based on minimum average log-loss across all cross-validation folds.
- The final model was used for evaluation, SHAP/LIME explainability, and deployment pipeline.
- It achieved high performance, making it a reliable model for phishing detection.

9.1.3 Plot the Evaluation Metrics

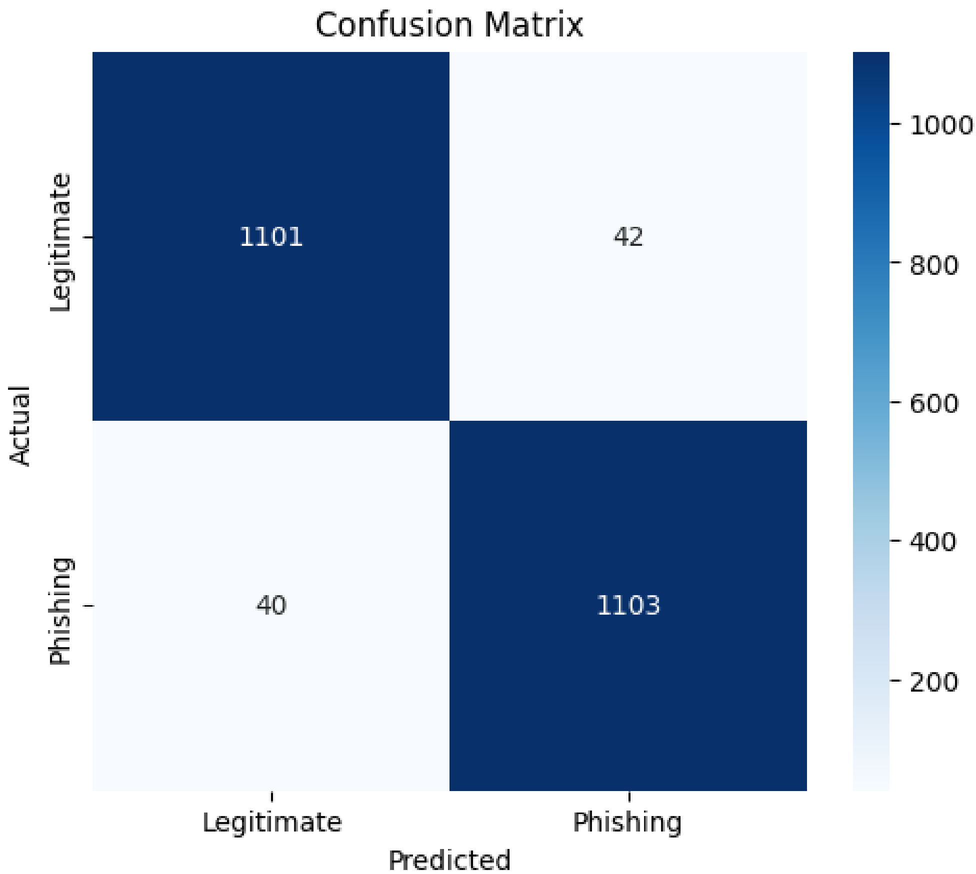
ROC Curve Plot

[37]:
```
# Predict probabilities for ROC
y_probs = best_xgb.predict_proba(X_test_scaled_df)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_probs)
roc_auc = auc(fpr, tpr)

# Plot ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC Curve (AUC = {roc_auc:.
 ↪4f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



Confusion Matrix Heatmap

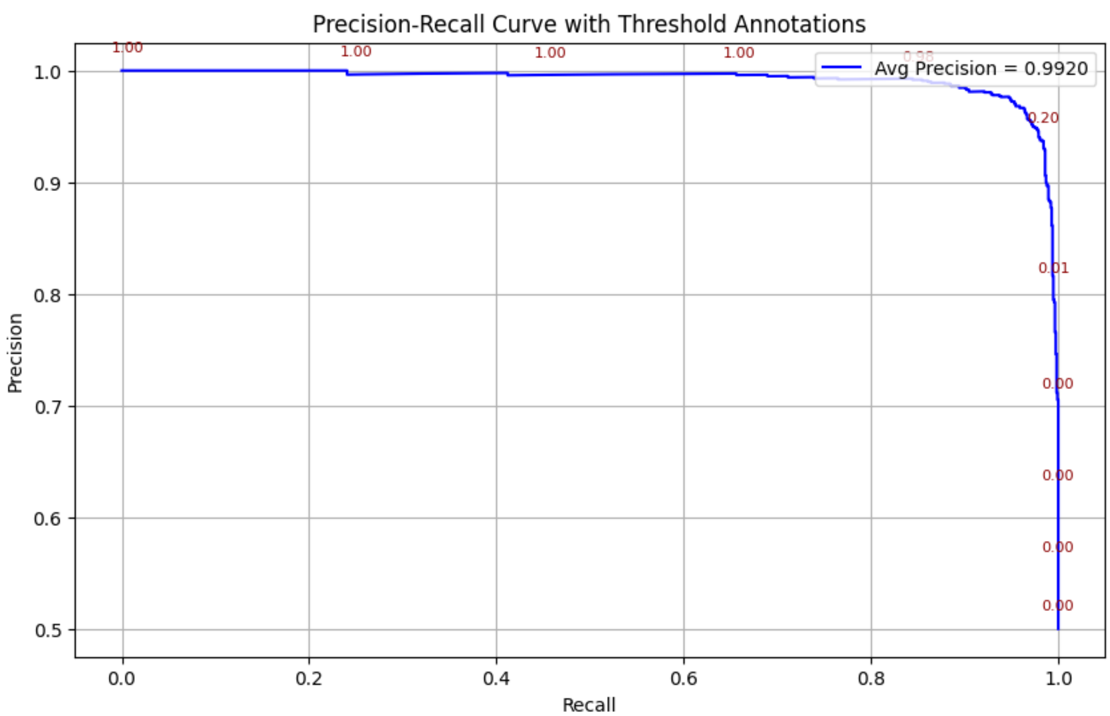
[38]:
```
# Predict labels
y_pred = best_xgb.predict(X_test_scaled_df)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
labels = ['Legitimate', 'Phishing']

# Plot heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=labels, yticklabels=labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Plot Precision-Recall Curve

[39]:
```
# Get predicted probabilities
y_probs = best_xgb.predict_proba(X_test_scaled_df)[:, 1]

# Compute precision-recall pairs
precision, recall, thresholds = precision_recall_curve(y_test, y_probs)
avg_precision = average_precision_score(y_test, y_probs)

# Plot Precision-Recall curve
plt.figure(figsize=(10, 6))
plt.plot(recall, precision, label=f'Avg Precision = {avg_precision:.4f}',␣
↪color='blue')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve with Threshold Annotations')
plt.grid(True)

# Annotate some thresholds
```

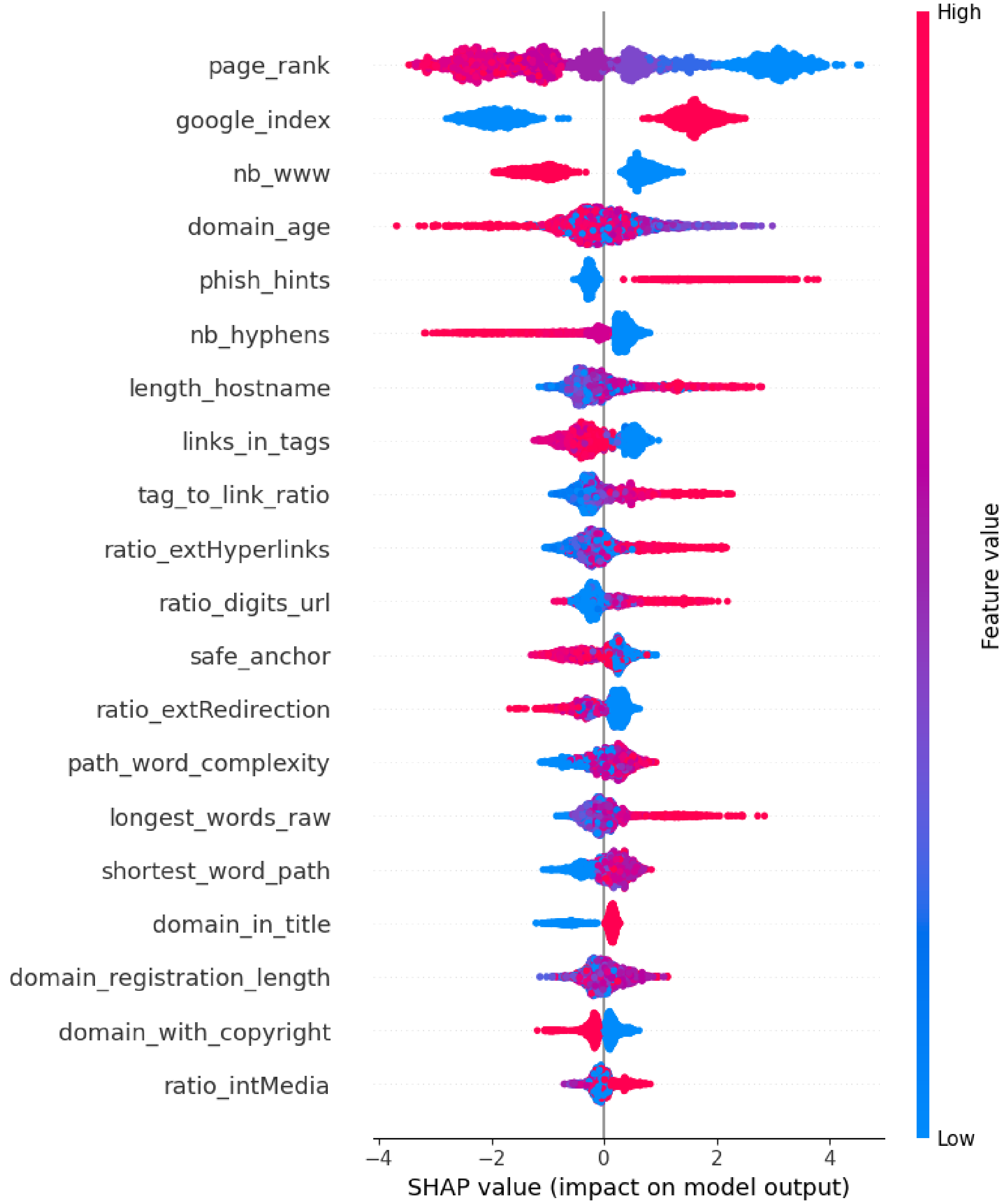
```
for i in range(0, len(thresholds), max(1, len(thresholds) // 10)):
    plt.annotate(f"{thresholds[i]:.2f}",
                 (recall[i], precision[i]),
                 textcoords="offset points",
                 xytext=(0, 10),
                 ha='center',
                 fontsize=8,
                 color='darkred')

plt.legend(loc='upper right')
plt.show()
```

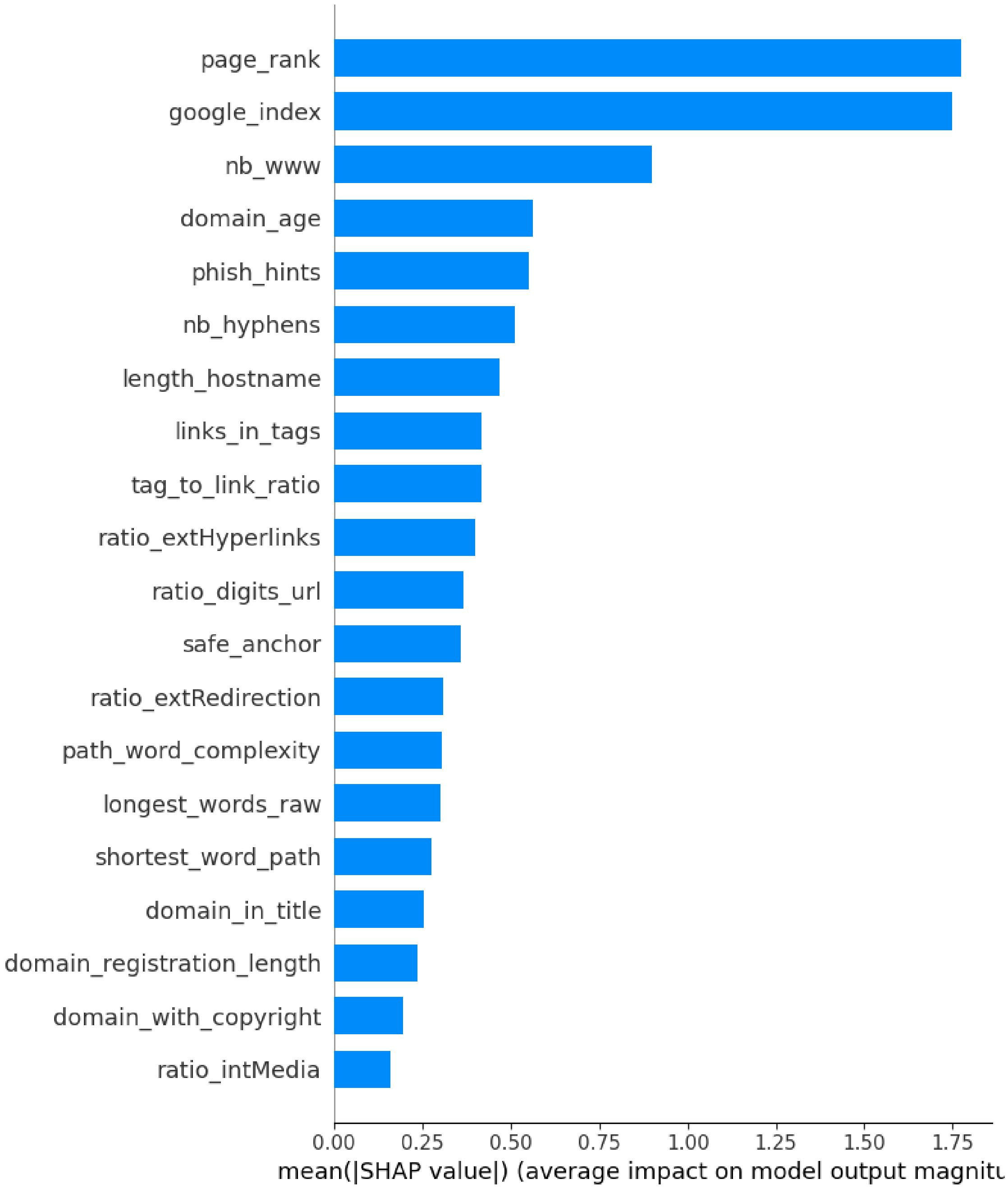


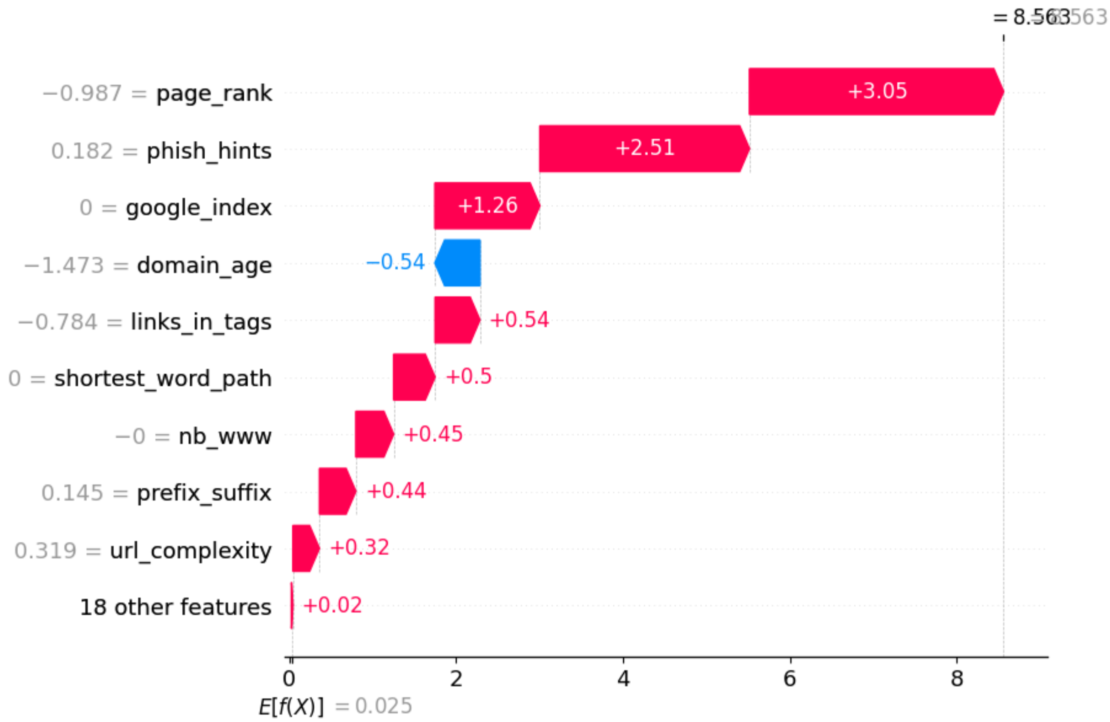
10 SHAP Explainer for XGBoost Classifier

[48]:
```
import shap
import xgboost as xgb
import pandas as pd
import matplotlib.pyplot as plt

# 1. Convert scaled arrays back to DataFrame for readability
X_test_scaled_df = pd.DataFrame(X_test_scaled, columns=X_test.columns)

# 2. Create SHAP explainer for XGBoost
```

```python
explainer = shap.Explainer(best_xgb)

# 3. Calculate SHAP values for test set
shap_values = explainer(X_test_scaled_df)

# 4. SHAP Summary Plot (Beeswarm)
shap.summary_plot(shap_values, X_test_scaled_df)

# 5. SHAP Bar Plot (Mean Absolute SHAP Values)
shap.summary_plot(shap_values, X_test_scaled_df, plot_type="bar")

# 6. Optional: Local Explanation for a single instance
shap.plots.waterfall(shap_values[0], max_display=10)  # Instance 0
```


page_rank
google_index
nb_www
domain_age
phish_hints
nb_hyphens
length_hostname
links_in_tags
tag_to_link_ratio
ratio_extHyperlinks
ratio_digits_url
safe_anchor
ratio_extRedirection
path_word_complexity
longest_words_raw
shortest_word_path
domain_in_title
domain_registration_length
domain_with_copyright
ratio_intMedia
−4
−2
0
2
4
SHAP value (impact on model output)
High
Feature value
Low


page_rank
google_index
nb_www
domain_age
phish_hints
nb_hyphens
length_hostname
links_in_tags
tag_to_link_ratio
ratio_extHyperlinks
ratio_digits_url
safe_anchor
ratio_extRedirection
path_word_complexity
longest_words_raw
shortest_word_path
domain_in_title
domain_registration_length
domain_with_copyright
ratio_intMedia
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
mean(|SHAP value|) (average impact on model output magnitu



10.0.1 1. SHAP Beeswarm Plot (Global Impact)

- Displays the **global impact** of each feature on the model output.
- Features like `page_rank`, `google_index`, `nb_www`, and `domain_age` show the **highest influence**.
- **Color** indicates feature value:
  - Blue = Low feature value

  - Red = High feature value

Top Influential Features
page_rank
google_index
nb_www
domain_age
phish_hints
nb_hyphens
length_hostname

10.0.2 2. SHAP Feature Importance Bar Plot

- **Average SHAP value** (magnitude) plotted per feature.

- **Ranking of feature importance** based on contribution to model predictions.
- `page_rank`, `google_index`, and `nb_www` are again the top contributors.

10.0.3 3. SHAP Waterfall Plot (Local Instance Explanation)

- Explains **how an individual prediction was made**.
- Shows how each feature pushes the model output from the **base value** toward the final prediction.
- **Key positive drivers**:
  - High `page_rank`
  - Presence of `phish_hints`
  - Good `google_index` status
- **Key negative drivers**:
  - Low `domain_age`
  - Low `links_in_tags`

10.0.4 Key Insights:

- **Domain authority signals** (`page_rank`, `google_index`) heavily influence phishing detection.
- **Structural URL patterns** (`nb_www`, `length_hostname`, `tag_to_link_ratio`) are critical indicators.
- **Domain age** and **registration characteristics** play a crucial role — younger domains are more suspicious.
- Achieved both **global** and **local** model interpretability using SHAP.

---

11 LIME Explainer for XGBoost Classifier

Step 1: Import and Create the LIME Explainer

[40]:
```
import lime
import lime.lime_tabular

# Create the LIME explainer
explainer = lime.lime_tabular.LimeTabularExplainer(
    training_data=np.array(X_train_scaled),
    feature_names=X_train.columns.tolist(),
    class_names=['Legitimate', 'Phishing'],
    mode='classification',
    verbose=True,
    feature_selection='auto'
)
```

Step 2: Explain Multiple Instances

[41]:
```python
# Loop through multiple instances for explanation
for i in range(10):
    print(f"\n LIME Explanation for Instance {i} (True Label: {y_test.
 ↪iloc[i]})")

    exp = explainer.explain_instance(
        data_row=X_test_scaled[i],
        predict_fn=best_xgb.predict_proba,
        num_features=len(X_test.columns)  # Explain all features
    )

    # Display in notebook (visual)
    exp.show_in_notebook(show_table=True)

    # Optional: Save to HTML
    exp.save_to_file(f'lime_explanation_instance_{i}.html')
```

```
 LIME Explanation for Instance 0 (True Label: 1)
Intercept 0.3654615879605214
Prediction_local [0.92257192]
Right: 0.9998091
```

```
<IPython.core.display.HTML object>
```

```
 LIME Explanation for Instance 1 (True Label: 1)
Intercept 0.4550576704064611
Prediction_local [0.91985103]
Right: 0.99997365
```

```
<IPython.core.display.HTML object>
```

```
 LIME Explanation for Instance 2 (True Label: 1)
Intercept 0.5541509819999019
Prediction_local [0.68903784]
Right: 0.99947566
```

```
<IPython.core.display.HTML object>
```

```
 LIME Explanation for Instance 3 (True Label: 1)
Intercept 0.46043498643419395
Prediction_local [0.81781227]
Right: 0.99636155
```

```
<IPython.core.display.HTML object>
```

```
 LIME Explanation for Instance 4 (True Label: 0)
Intercept 0.5030846088397315
```

```
Prediction_local [0.49903744]
Right: 0.0013258632
```

<IPython.core.display.HTML object>

```
 LIME Explanation for Instance 5 (True Label: 0)
Intercept 0.9798126527510567
Prediction_local [-0.16527984]
Right: 0.0006115953
```

<IPython.core.display.HTML object>

```
 LIME Explanation for Instance 6 (True Label: 0)
Intercept 0.4688869714055654
Prediction_local [0.35228581]
Right: 0.00023313252
```

<IPython.core.display.HTML object>

```
 LIME Explanation for Instance 7 (True Label: 1)
Intercept 0.629041018228231
Prediction_local [0.40472764]
Right: 0.99931693
```

<IPython.core.display.HTML object>

```
 LIME Explanation for Instance 8 (True Label: 0)
Intercept 0.6003471661531588
Prediction_local [0.04407973]
Right: 5.2678442e-05
```

<IPython.core.display.HTML object>

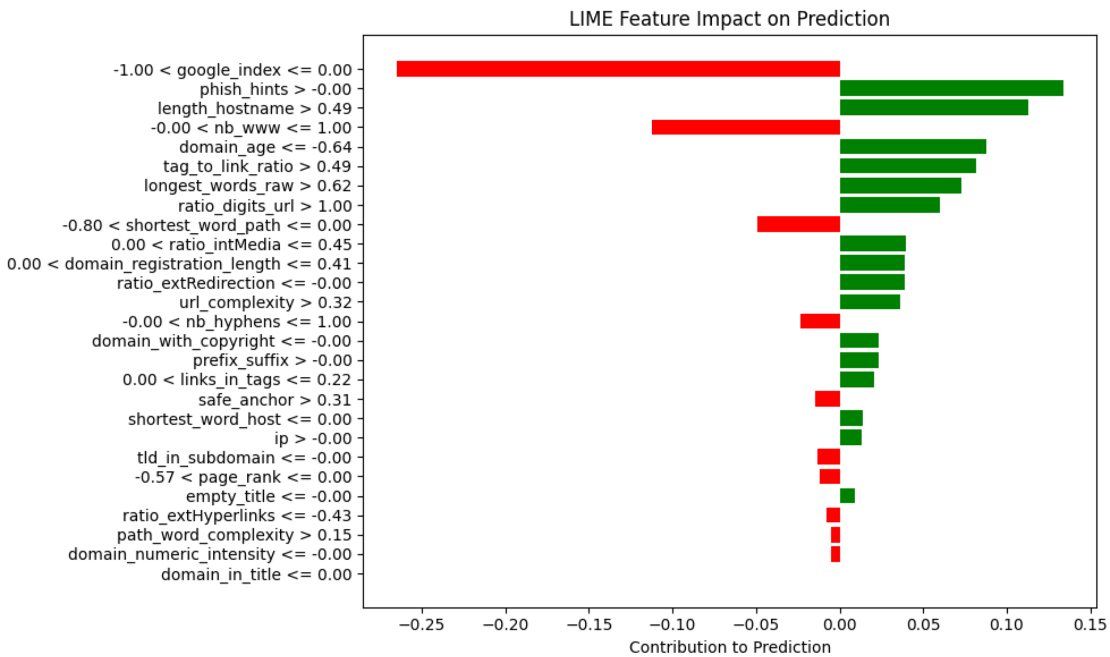
```
 LIME Explanation for Instance 9 (True Label: 1)
Intercept 0.45948300217207205
Prediction_local [0.75900222]
Right: 0.99992347
```

<IPython.core.display.HTML object>

Step 3: Plot Feature Weights as Bar Plot

[42]:
```
# Plot feature impact bar chart (manual)
def plot_lime_weights(exp):
    weights = dict(exp.as_list())
    features = list(weights.keys())
    values = list(weights.values())
```

```
    plt.figure(figsize=(10, 6))
    plt.barh(features, values, color=['green' if v > 0 else 'red' for v in␣
↪values])
    plt.title("LIME Feature Impact on Prediction")
    plt.xlabel("Contribution to Prediction")
    plt.gca().invert_yaxis()
    plt.tight_layout()
    plt.show()

# Example: Plot for instance 0
plot_lime_weights(exp)
```



11.1 LIME Explanation Report (Local Interpretability)

11.1.1 Objective:

LIME (Local Interpretable Model-Agnostic Explanations) was used to explain individual predictions made by the final trained model (Random Forest Classifier) for phishing detection.

11.1.2 Top 10 Predictions Explained:

- Local explanations were generated for 5 randomly selected instances.

- Each explanation highlighted the **contribution of specific features** toward classifying a site as either **Phishing** or **Legitimate**.

11.1.3 Key Influential Features Identified by LIME:

- `url_complexity`
- `phish_hints`
- `nb_www`
- `nb_qm`
- `tag_to_link_ratio`
- `path_word_complexity`

These features had **strong local impact** and also aligned with global importance insights from SHAP.

11.1.4 Interpretability Outcome:

- LIME confirmed the effectiveness of **engineered features** and **domain-based indicators**.
- The results increase **trust and transparency** in the model's predictions.
- Useful for debugging, compliance, and end-user explanations.

11.1.5 Outputs Generated:

- Interactive HTML files:
  - `lime_explanation_instance_0.html`
  - `lime_explanation_instance_1.html`
  - `lime_explanation_instance_2.html`
  - `lime_explanation_instance_3.html`
  - `lime_explanation_instance_4.html`
  - `lime_explanation_instance_5.html`
  - `lime_explanation_instance_6.html`
  - `lime_explanation_instance_7.html`
  - `lime_explanation_instance_8.html`
  - `lime_explanation_instance_9.html`
