Internship_CodeB_week 3

April 14, 2025

1 Phishing Website Detection

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

2 Week 3 Submission

[118]:
```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
from sklearn.preprocessing import StandardScaler, LabelEncoder
```

[119]:
```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)
```

[119]:
```
                                                   url  length_url  \
8645               http://www.yourdictionary.com/dynasty          37
10106  http://work.chron.com/yearly-salary-cnc-machin…          59
7866   https://support-appleld.com.secureupdate.duila…         128
11080                     http://www.hospitalitydnb.com/          30
6509                            https://www.mcmakler.de/          24
…                                                    …           …
7020                 http://mifiboltrisman.blogspot.com/          35
10497  http://office365-0nedrive-portal.el.r.appspot…          60
2694   https://onedrive.live.com/?authkey=%21ADgDSBYl…         135
```

```
8066   https://www.ausa.org/sites/default/files/us-ar…          85
9557   https://www.tripadvisor.com/Restaurant_Review-…         111

       length_hostname  ip  nb_dots  nb_hyphens  nb_at  nb_qm  nb_and  nb_or  \
8645                22   0        2           0      0      0       0      0
10106               14   0        3           4      0      0       0      0
7866                50   1        4           1      0      1       2      0
11080               22   0        2           0      0      0       0      0
6509                15   0        2           0      0      0       0      0
…                    …  ..      …         …     …     …      …     …
7020                27   0        2           0      0      0       0      0
10497               42   0        4           3      0      0       0      0
2694                17   1        2           0      0      1       4      0
8066                12   0        3           5      0      0       0      0
9557                19   1        3           5      0      0       0      0

       …  domain_in_title  domain_with_copyright  whois_registered_domain  \
8645   …                1                      1                        0
10106  …                0                      1                        0
7866   …                1                      1                        0
11080  …                1                      0                        0
6509   …                0                      1                        0
…    …              …                    …                      …
7020   …                1                      0                        0
10497  …                1                      1                        0
2694   …                1                      0                        0
8066   …                1                      1                        0
9557   …                0                      1                        0

       domain_registration_length  domain_age  web_traffic  dns_record  \
8645                          448        7587          988           0
10106                         275       11049          549           0
7866                           25        3993      5707171           0
11080                        1788        3691      8098260           0
6509                            0          -1       428294           0
…                             …         …          …         …
7020                          370        7299            0           0
10497                         217        5627            0           0
2694                          157        9339           21           0
8066                           46        9085       399658           0
9557                          609        7792          557           0

       google_index  page_rank      status
8645              0          5  legitimate
10106             0          6  legitimate
7866              1          0    phishing
11080             0          3  legitimate
```

```
6509               0          3  legitimate
…              …         …          …
7020               0          5  legitimate
10497              1          5    phishing
2694               0          5    phishing
8066               1          5  legitimate
9557               0          8  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.

3.1.3 1. Missing Values Handling

[120]:
```
# Check for Missing Values
missing_values = df.isnull().sum()
missing_values[missing_values > 0]
```

[120]:
```
Series([], dtype: int64)
```

Observation: The dataset contains no missing values.

Action Taken: No imputation was required as all entries across features were complete.

Conclusion: No imputation techniques (mean, median, mode, etc.) were applied.

3.1.4 2. Duplicate Records

[121]:
```
#Check for Duplicate Rows
duplicate_count = df.duplicated().sum()
print(f"Number of duplicate rows: {duplicate_count}")
```

```
Number of duplicate rows: 0
```

Observation: There are no duplicate rows present in the dataset.

Action Taken: No deduplication process was necessary.

Duplicate Percentage: 0% of the original dataset.

3.1.5 3. Feature Encoding

Categorical Features:

`url`: This column represents the full URL and is not useful for training the machine learning model.

- Action: Dropped from the dataset.

`status`: This is the target variable with two categories: 'legitimate' and 'phishing'.

- Encoding Applied: Label Encoding
- Mapping: 'legitimate' → 0, 'phishing' → 1
- Method Used: .map() function

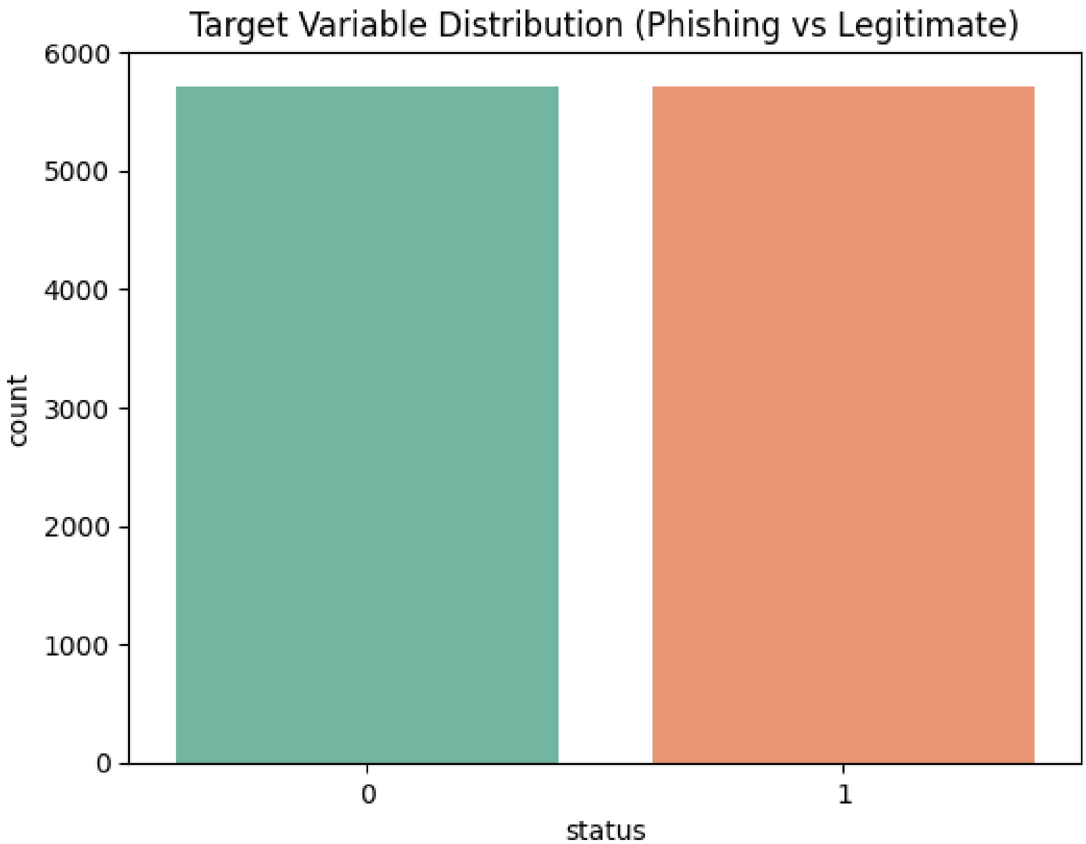
Remaining Features:

- All other features are numerical and did not require encoding.

[122]:
```
# 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
```

[123]:
```
# 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
```

[124]:

```
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']
```

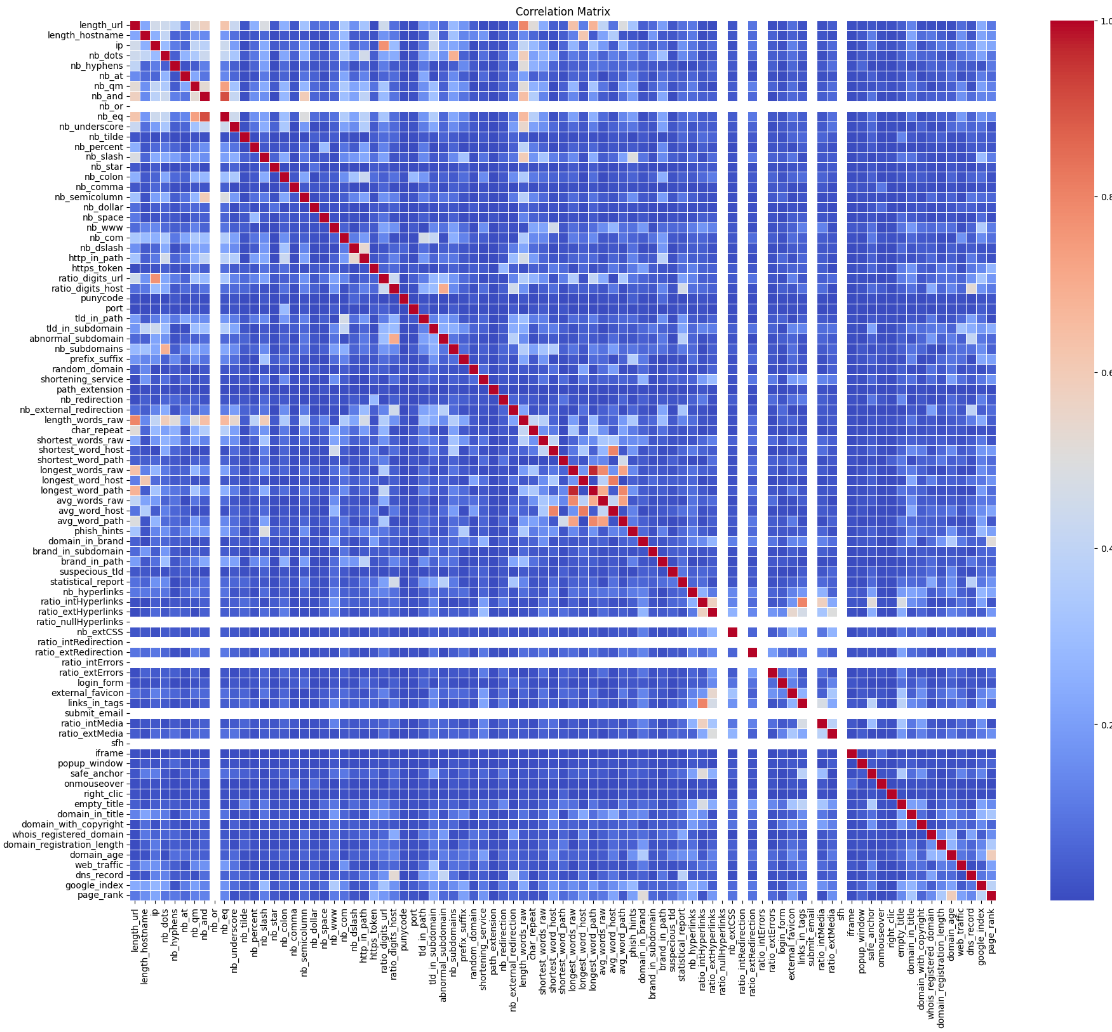
[125]:
```
# Dropping the 'url' column
# The 'url' column is not useful for training the machine learning model.

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

3.2 Feature Selection

Step 1: Correlation Analysis

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

[126]:
```
# 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

[127]:
```python
# 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}")
```

```python
# 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.

[128]:
```python
df_reduced.drop(columns=['avg_word_host'], inplace=True)  # Drop avg_word_host␣
↪column as per VIF analysis
```

3.2.1 2: Feature Selection using ANOVA F-test (f_classif)

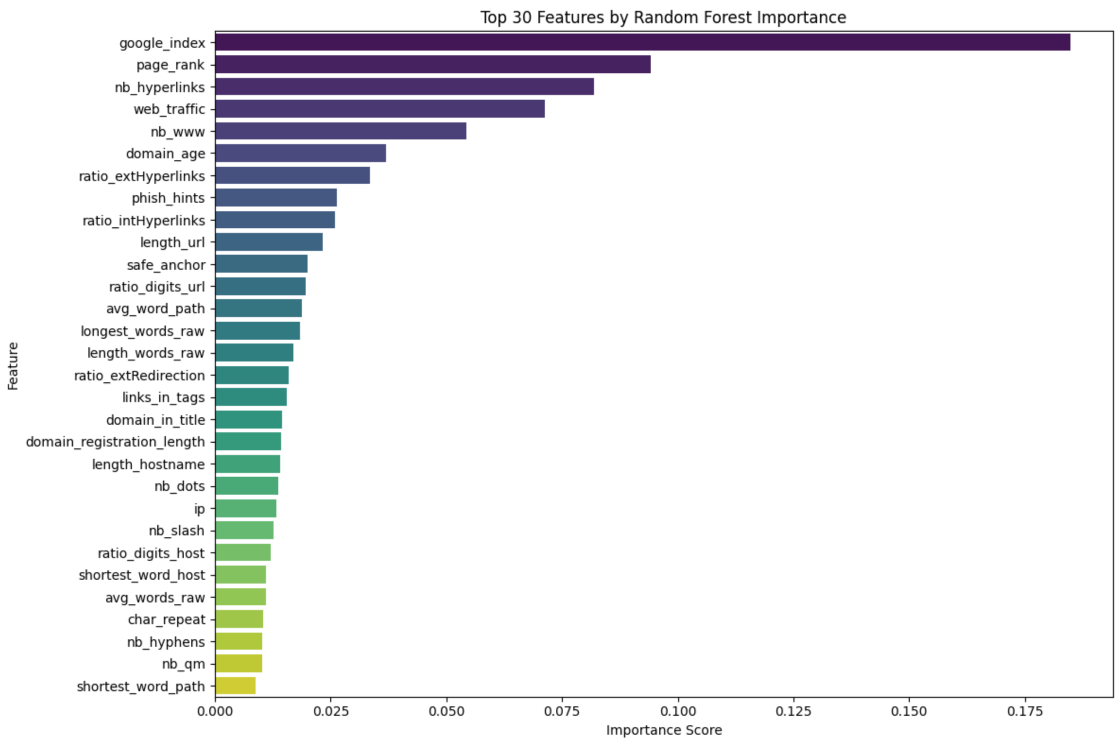
[129]:
```python
from sklearn.feature_selection import SelectKBest, f_classif

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')
```

3.2.2 3: Random Forest Feature Importance

[130]:
```python
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# 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()
```



3.2.3 4: Apply RFE (Recursive Feature Elimination)

[131]:
```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')
```

3.2.4 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

```
[132]: # 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):
['shortest_word_path', 'links_in_tags', 'nb_www', 'ratio_digits_url',
'tld_in_subdomain', 'shortest_word_host', 'avg_word_path', 'nb_hyphens',
'domain_with_copyright', 'avg_words_raw', 'ratio_intMedia',
'ratio_intHyperlinks', 'ratio_extRedirection', 'length_url', 'domain_in_title',
'google_index', 'nb_dots', 'nb_qm', 'longest_words_raw', 'ip',
'ratio_digits_host', 'nb_and', 'domain_age', 'nb_slash', 'safe_anchor',
'length_hostname', 'nb_hyperlinks', 'page_rank', 'ratio_extHyperlinks',
'length_words_raw', 'empty_title', 'domain_registration_length', 'phish_hints',
'prefix_suffix']
Number of final selected features: 34
```

```
[133]: from statsmodels.stats.outliers_influence import variance_inflation_factor
import pandas as pd

# Subset the dataframe to final selected features
X_vif = df[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)
```

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

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

[134]:
```python
features_to_drop_vif = [
    'length_words_raw',
    'avg_words_raw',
    'length_url',
    'ratio_intHyperlinks',
    'nb_slash',
```

```
    'nb_dots',
]
```

[135]:
```
# 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:
```

[135]:
```
['shortest_word_path',
 'links_in_tags',
 'nb_www',
 'ratio_digits_url',
 'tld_in_subdomain',
 'shortest_word_host',
 'avg_word_path',
 'nb_hyphens',
 'domain_with_copyright',
 'ratio_intMedia',
 'ratio_extRedirection',
 'domain_in_title',
 'google_index',
 'nb_qm',
 'longest_words_raw',
 'ip',
 'ratio_digits_host',
 'nb_and',
 'domain_age',
 'safe_anchor',
 'length_hostname',
 'nb_hyperlinks',
 'page_rank',
 'ratio_extHyperlinks',
 'empty_title',
 'domain_registration_length',
 'phish_hints',
 'prefix_suffix']
```

3.3 Applied Steps for Feature Selection Process:

3.3.1 1. Correlation Analysis

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

3.3.2 2. ANOVA (f_classif)

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

3.3.3 3. Random Forest Feature Importance

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

3.3.4 4. Recursive Feature Elimination (RFE)

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

3.3.5 5. Feature Union

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

---

3.3.6 6. Variance Inflation Factor (VIF)

- Evaluated multicollinearity in final selected features
- Dropped 6 features with VIF > 10 to avoid redundancy

3.3.7 Outliers Detection and Handling

- `quantile(0.25)` and `quantile(0.75)`: Calculate the 25th and 75th percentiles for each feature (Q1 and Q3).
- `IQR`: Compute the IQR (difference between Q3 and Q1).
- `Outlier Filtering`: Rows that have values outside the calculated bounds are removed.
- `y_no_outliers`: Ensure that the target variable y is aligned with the filtered dataset.

[136]:
```
# import numpy as np
```

```
# # Calculate Q1 (25th percentile) and Q3 (75th percentile) for each feature
# Q1 = X_final.quantile(0.25)
# Q3 = X_final.quantile(0.75)

# # Calculate IQR (Interquartile Range)
# IQR = Q3 - Q1

# # Define outlier thresholds
# lower_bound = Q1 - 1.5 * IQR
# upper_bound = Q3 + 1.5 * IQR

# # Filter out outliers
# X_no_outliers = X_final[~((X_final < lower_bound) | (X_final > upper_bound)).
↪any(axis=1)]
# y_no_outliers = y[X_no_outliers.index]  # Ensure the target column␣
↪corresponds to filtered data

# # Check the shape of the new dataset after removing outliers
# print(f"Original shape: {X_final.shape}")
# print(f"Shape after removing outliers: {X_no_outliers.shape}")
```

3.4 Split Dataset into Train and Test set

[137]:
```
from sklearn.model_selection import train_test_split

# Define final feature set and target
X_final = df[final_features_vif]
y_final = df['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
```

3.5 Scaling : RobustScaler()

[138]:
```
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[139]:
```
X_train.sample(10)
```

[139]:
```
       shortest_word_path  links_in_tags  nb_www  ratio_digits_url  \
7477                    0     100.000000       1          0.000000
8456                    4       0.000000       1          0.000000
5731                    9     100.000000       0          0.006494
2895                    4      37.500000       0          0.126582
752                     2      66.666667       0          0.058140
3389                    0      60.000000       1          0.000000
4097                    0       0.000000       0          0.000000
7116                    0       0.000000       0          0.000000
```

6430	6	0.000000	1	0.000000
10755	0	92.307692	1	0.000000

	tld_in_subdomain	shortest_word_host	avg_word_path	nb_hyphens	\
7477	0	3	0.000000	0	
8456	0	3	6.000000	0	
5731	0	10	21.000000	0	
2895	1	4	6.000000	2	
752	0	4	7.714286	1	
3389	0	3	0.000000	0	
4097	0	9	0.000000	0	
7116	0	2	0.000000	0	
6430	0	3	7.000000	2	
10755	0	3	0.000000	0	

	domain_with_copyright	ratio_intMedia	…	domain_age	safe_anchor	\
7477	1	100.000000	…	-1	66.666667	
8456	0	0.000000	…	7446	0.000000	
5731	1	100.000000	…	4775	33.333333	
2895	0	100.000000	…	1777	0.000000	
752	0	100.000000	…	134	100.000000	
3389	1	35.135135	…	-1	50.000000	
4097	0	0.000000	…	7412	0.000000	
7116	0	0.000000	…	-1	0.000000	
6430	0	0.000000	…	7957	0.000000	
10755	1	0.000000	…	9047	16.666667	

	length_hostname	nb_hyperlinks	page_rank	ratio_extHyperlinks	\
7477	26	30	4	0.033333	
8456	25	12	0	1.000000	
5731	14	466	5	0.004292	
2895	51	19	2	0.473684	
752	17	28	0	0.142857	
3389	22	117	4	0.239316	
4097	22	6	2	1.000000	
7116	9	0	4	0.000000	
6430	20	0	2	0.000000	
10755	17	42	5	0.214286	

	empty_title	domain_registration_length	phish_hints	prefix_suffix
7477	0	382	0	0
8456	0	252	0	0
5731	0	339	1	0
2895	0	0	1	0
752	0	231	0	1
3389	0	0	0	0
4097	0	281	0	0

```
7116                1                          -1          0           0
6430                0                        1173          0           0
10755               0                          88          0           0

[10 rows x 28 columns]
```

4 Normalization/Scaling Report

4.1 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.

4.2 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
```

- **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**.

---

[140]:
```python
# 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']
```

```python
# Combine both into a single table for comparison
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
shortest_word_path                   0.0        40.000      -0.667      12.667
links_in_tags                        0.0       100.000      -0.600       0.400
nb_www                               0.0         2.000       0.000       2.000
ratio_digits_url                     0.0         0.724       0.000       9.257
tld_in_subdomain                     0.0         1.000       0.000       1.000
shortest_word_host                   1.0        39.000      -0.667      12.000
avg_word_path                        0.0       206.000      -0.725      30.175
nb_hyphens                           0.0        32.000       0.000      32.000
domain_with_copyright                0.0         1.000       0.000       1.000
ratio_intMedia                       0.0       100.000      -0.111       0.889
ratio_extRedirection                 0.0         2.000       0.000       8.811
domain_in_title                      0.0         1.000      -1.000       0.000
google_index                         0.0         1.000      -1.000       0.000
nb_qm                                0.0         3.000       0.000       3.000
longest_words_raw                    2.0       829.000      -1.286     116.857
ip                                   0.0         1.000       0.000       1.000
ratio_digits_host                    0.0         0.800       0.000       0.800
nb_and                               0.0        19.000       0.000      19.000
domain_age                         -12.0     12874.000      -0.665       1.474
safe_anchor                          0.0       100.000      -0.322       1.032
length_hostname                      4.0       214.000      -1.667      21.667
nb_hyperlinks                        0.0      4659.000      -0.370      50.272
page_rank                            0.0        10.000      -0.750       1.750
ratio_extHyperlinks                  0.0         1.000      -0.277       1.848
empty_title                          0.0         1.000       0.000       1.000
domain_registration_length          -1.0     29829.000      -0.666      81.060
phish_hints                          0.0        10.000       0.000      10.000
prefix_suffix                        0.0         1.000       0.000       1.000
```

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

4.3.1 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).

4.3.2 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.

[145]:
```
# Final split dataset ready for model training
X_train_scaled
X_test_scaled
```

[145]:
```
array([[ 0.        , -0.6       ,  0.        , …, -0.6630137 ,
         2.        ,  1.        ],
       [ 1.33333333,  0.4       ,  1.        , …, -0.16438356,
         0.        ,  1.        ],
       [ 0.        , -0.6       ,  0.        , …, -0.03287671,
         0.        ,  1.        ],
       …,
       [-0.66666667, -0.31052632,  1.        , …,  7.40273973,
         0.        ,  0.        ],
       [ 0.33333333,  0.32307692,  1.        , …,  0.11232877,
         0.        ,  0.        ],
       [-0.66666667, -0.45507246,  1.        , …, -0.6630137 ,
         0.        ,  1.        ]], shape=(2286, 28))
```