Codes:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

import time

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from nltk.tokenize import RegexpTokenizer

from nltk.stem.snowball import SnowballStemmer

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics import accuracy\_score

import joblib

# Loading the dataset

df= pd.read\_csv("dataset\_phishing.csv")

df.head()

df.describe()

df.shape

df.isnull().sum()

plt.figure(figsize=(5, 5)) # Adjust the figure size if needed

status\_counts = df['status'].value\_counts()

colors = ['skyblue', 'lightcoral']

explode = (0.5, 0)

plt.pie(status\_counts, labels=status\_counts.index, autopct='%1.1f%%', startangle=90, wedgeprops=dict(width=0.4), explode=explode, colors=colors)

plt.title("Distribution of Status")

# Add legend

plt.legend(status\_counts.index, loc="best")

plt.show()

# checking unique values and counts from the collected object features

df['status'].value\_counts()

tokenizer = RegexpTokenizer(r'[A-Za-z]+')

tokenizer.tokenize(df.url[0]) # this will fetch all the words from the first URL

# Tokenizing all the rows

print('Getting words tokenized ...')

t0= time.perf\_counter()

df['text\_tokenized'] = df.url.map(lambda t: tokenizer.tokenize(t))

t1 = time.perf\_counter() - t0

print('Time taken',t1 ,'sec')

stemmer = SnowballStemmer("english") # choose a language

# Getting all the stemmed words

print('Getting words stemmed ...')

t0= time.perf\_counter()

df['text\_stemmed'] = df['text\_tokenized'].map(lambda l: [stemmer.stem(word) for word in l])

t1= time.perf\_counter() - t0

print('Time taken',t1 ,'sec')

# Joining all the stemmmed words.

print('Get joiningwords ...')

t0= time.perf\_counter()

df['text\_sent'] = df['text\_stemmed'].map(lambda l: ' '.join(l))

t1= time.perf\_counter() - t0

print('Time taken',t1 ,'sec')

cv = CountVectorizer()

feature = cv.fit\_transform(df.text\_sent) #transform all text which we tokenize and stemed

feature[:5].toarray() # convert sparse matrix into array to print transformed features

joblib.dump(cv, 'count\_vectorizer.pkl')

trainX, testX, trainY, testY = train\_test\_split(feature, df.status)

from sklearn.ensemble import RandomForestClassifier

# Instantiate the RandomForestClassifier

rf\_classifier = RandomForestClassifier()

# Fit the model to the training data

rf\_classifier.fit(trainX, trainY)

# Make predictions on training and testing data

train\_preds = rf\_classifier.predict(trainX)

test\_preds = rf\_classifier.predict(testX)

train\_accuracy = accuracy\_score(trainY, train\_preds)

test\_accuracy = accuracy\_score(testY, test\_preds)

print("Training Accuracy:", train\_accuracy)

print("Testing Accuracy:", test\_accuracy)

# Generate confusion matrix and classification report

print("\nConfusion Matrix:")

print(confusion\_matrix(testY, test\_preds))

print("\nClassification Report:")

print(classification\_report(testY, test\_preds))

import joblib

# Save the trained model to a file

joblib.dump(rf\_classifier, 'random\_forest\_model.pkl')

# Load the trained model

lr\_model = joblib.load('random\_forest\_model.pkl')

# Function to preprocess the input URL

def preprocess\_url(url):

tokenizer = RegexpTokenizer(r'[A-Za-z]+')

words\_tokenized = tokenizer.tokenize(url)

stemmer = SnowballStemmer("english")

words\_stemmed = [stemmer.stem(word) for word in words\_tokenized]

return ' '.join(words\_stemmed)

# Function to make predictions

def predict\_url\_status(url):

preprocessed\_url = preprocess\_url(url)

feature = cv.transform([preprocessed\_url])

prediction = lr\_model.predict(feature)[0]

return prediction

url\_to\_predict = "http://vamoaestudiarmedicina.blogspot.com/"

prediction\_result = predict\_url\_status(url\_to\_predict)

if prediction\_result == 'phishing':

print(f"The URL '{url\_to\_predict}' is predicted to be a phishing site.")

else:

print(f"The URL '{url\_to\_predict}' is predicted to be a legitimate site.")

from sklearn.preprocessing import LabelEncoder

# Convert status labels to numerical values

label\_encoder = LabelEncoder()

df['status\_encoded'] = label\_encoder.fit\_transform(df['status'])

# Split the dataset

trainX, testX, trainY, testY = train\_test\_split(feature, df['status\_encoded'], test\_size=0.2, random\_state=42)

from sklearn.pipeline import Pipeline

from sklearn.neural\_network import BernoulliRBM

from sklearn.linear\_model import LogisticRegression

# Define the RBM model with 20 iterations

rbm = BernoulliRBM(n\_iter=20, random\_state=0, verbose=True)

# Define the Logistic Regression model

logistic = LogisticRegression(max\_iter=10000)

# Create a pipeline combining RBM and Logistic Regression

dbn\_classifier = Pipeline(steps=[('rbm', rbm), ('logistic', logistic)])

# Training the DBN model

print("Training Deep Belief Network...")

t0 = time.time()

dbn\_classifier.fit(trainX, trainY)

print("Training time:", time.time() - t0)

# Evaluate the DBN model

print('Training Accuracy:', dbn\_classifier.score(trainX.toarray(), trainY))

print('Testing Accuracy:', dbn\_classifier.score(testX.toarray(), testY))

# Obtain accuracy score for DBN classifier

dbn\_accuracy = dbn\_classifier.score(testX, testY)

print("DBN Accuracy", dbn\_accuracy)

from sklearn.metrics import confusion\_matrix

# Predictions on the test set

predictions = dbn\_classifier.predict(testX)

# Generate confusion matrix

conf\_matrix = confusion\_matrix(testY, predictions)

# Plot confusion matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap="YlGnBu",

xticklabels=label\_encoder.classes\_,

yticklabels=label\_encoder.classes\_)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix')

plt.show()

from sklearn.neural\_network import BernoulliRBM

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

# Feature scaling

scaler = StandardScaler(with\_mean=False)

scaled\_trainX = scaler.fit\_transform(trainX)

scaled\_testX = scaler.transform(testX)

# Create RBM model

rbm = BernoulliRBM(n\_components=256, learning\_rate=0.01, random\_state=0, verbose=True, n\_iter=20)

# Create logistic regression model for classification

logistic\_classifier = LogisticRegression()

# Create a pipeline to combine RBM and Logistic Regression

rbm\_classifier = Pipeline(steps=[('rbm', rbm), ('logistic', logistic\_classifier)])

# Training the model

rbm\_classifier.fit(scaled\_trainX, trainY)

# Testing the model

rbm\_accuracy = rbm\_classifier.score(scaled\_testX, testY)

Scores\_ml['RBM'] = np.round(rbm\_accuracy, 2)

print(f'RBM Testing Accuracy: {rbm\_accuracy}')

# Classification Report and Confusion Matrix

print('\nCLASSIFICATION REPORT\n')

print(classification\_report(rbm\_classifier.predict(scaled\_testX), testY, target\_names=['Bad', 'Good']))

con\_mat\_rbm = pd.DataFrame(confusion\_matrix(rbm\_classifier.predict(scaled\_testX), testY),

columns=['Predicted:Bad', 'Predicted:Good'],

index=['Actual:Bad', 'Actual:Good'])

print('\nCONFUSION MATRIX')

plt.figure(figsize=(6, 4))

sns.heatmap(con\_mat\_rbm, annot=True, fmt='d', cmap="YlGnBu")

# Define the models and their corresponding accuracy scores

models = ['Random Forest', 'DBN', 'RBM']

accuracy\_scores = [test\_accuracy, dbn\_accuracy, rbm\_accuracy]

# Check for missing or invalid accuracy scores

valid\_accuracy\_scores = [score if score is not None and not np.isnan(score) else 0 for score in accuracy\_scores]

# Plotting the bar chart

plt.figure(figsize=(10, 6))

bars = plt.bar(models, valid\_accuracy\_scores, color=['skyblue', 'orange', 'lightgreen'])

# Add labels to the bars

for bar, score in zip(bars, valid\_accuracy\_scores):

plt.text(bar.get\_x() + bar.get\_width() / 2, bar.get\_height() - 0.05, f'{score:.2f}', ha='center', color='black', fontsize=10)

plt.xlabel('Models')

plt.ylabel('Testing Accuracy')

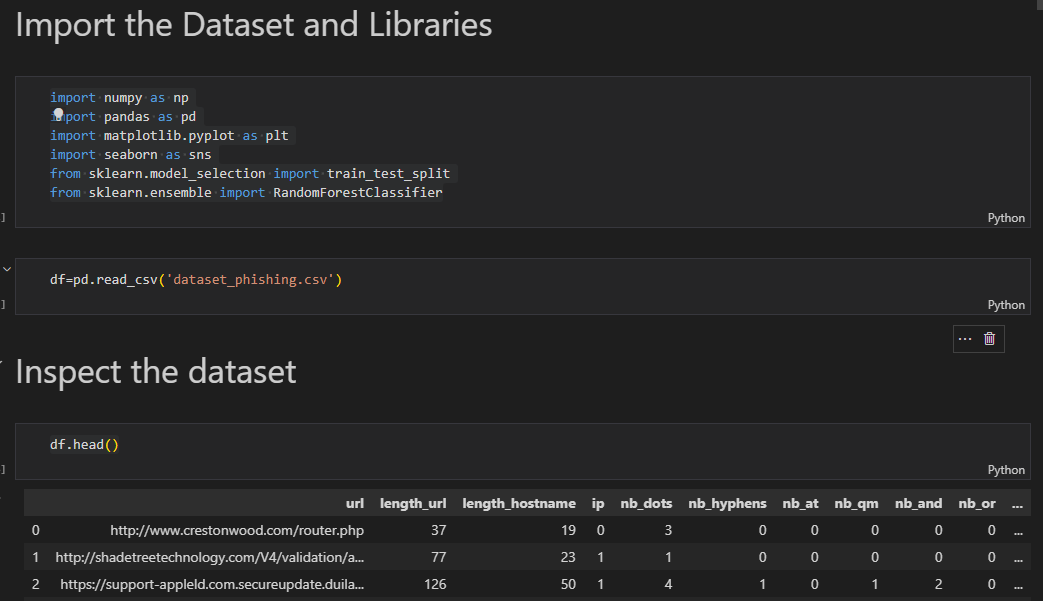
plt.title('Comparison of Model Testing Accuracy')

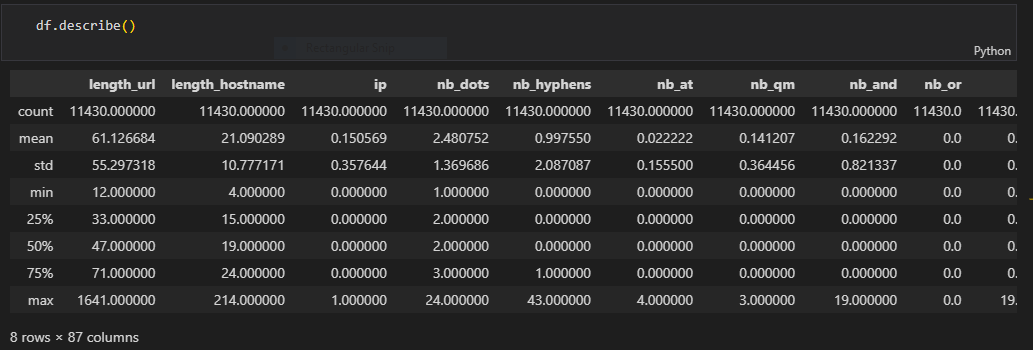
plt.ylim(0, 1) # Set the y-axis limit from 0 to 1

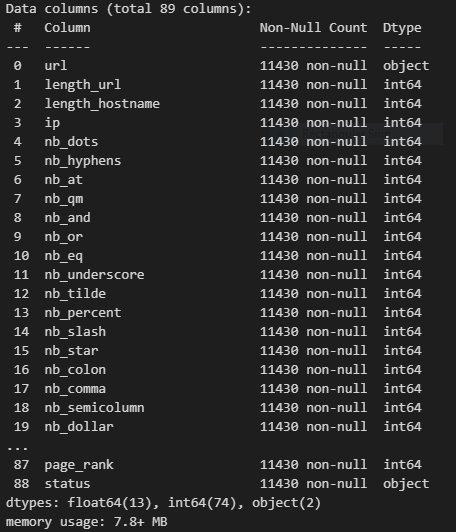
plt.grid(axis='y', linestyle='--', alpha=0.7)

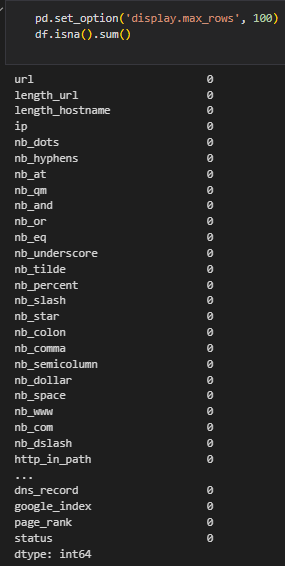
plt.show()

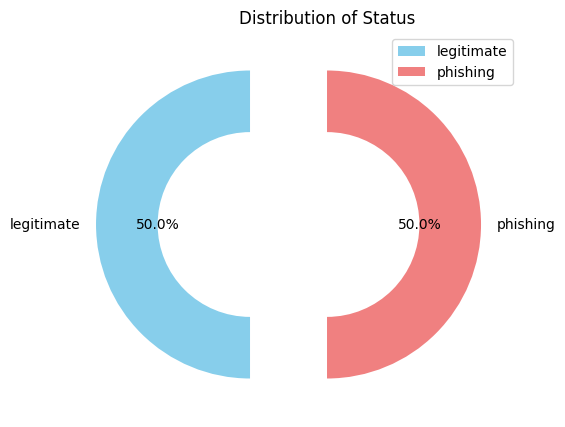
**Screenshots**

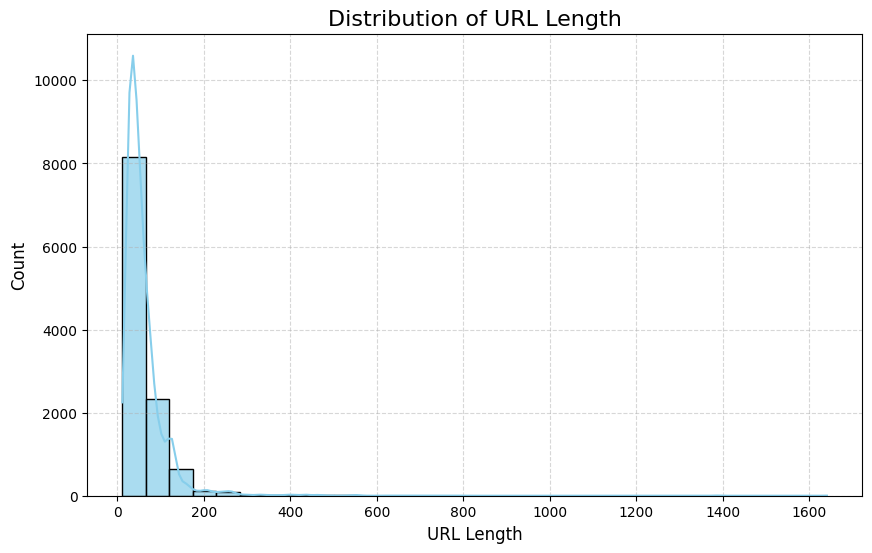


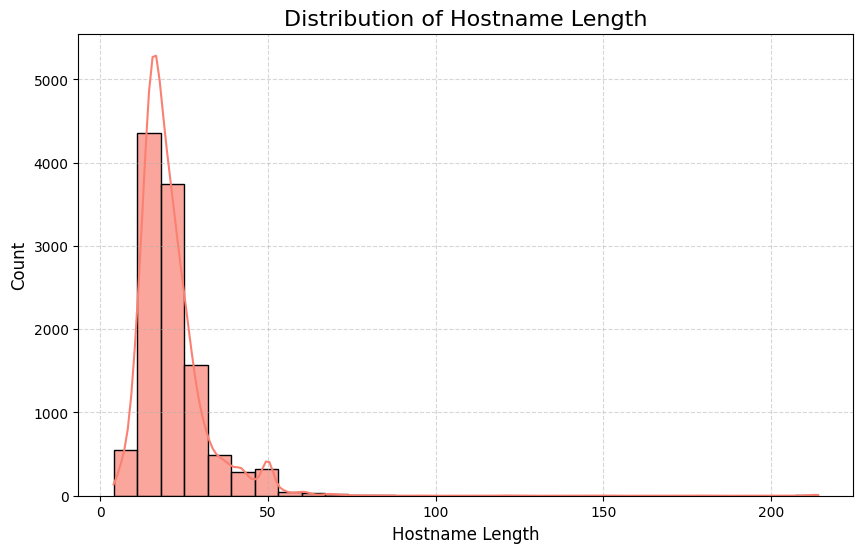


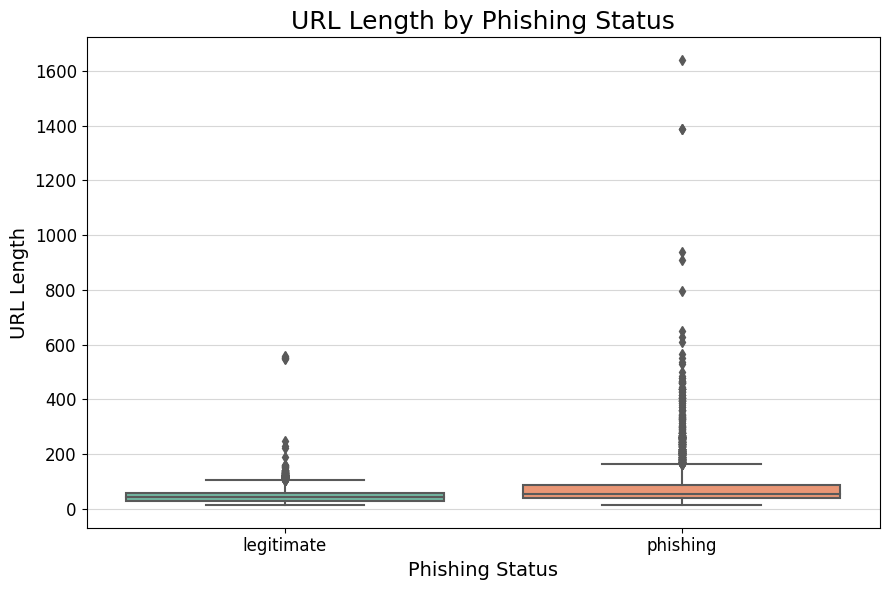


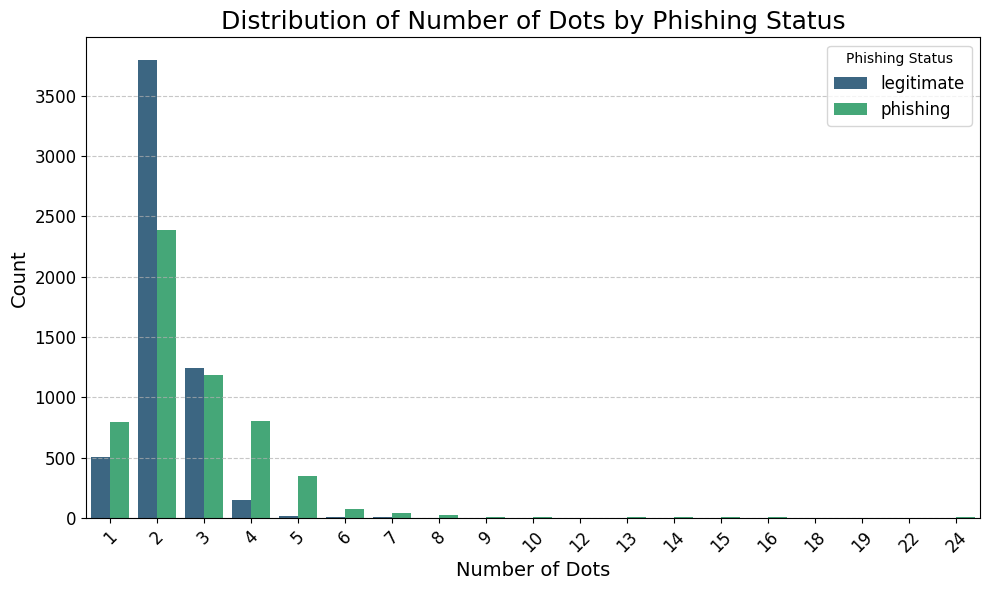


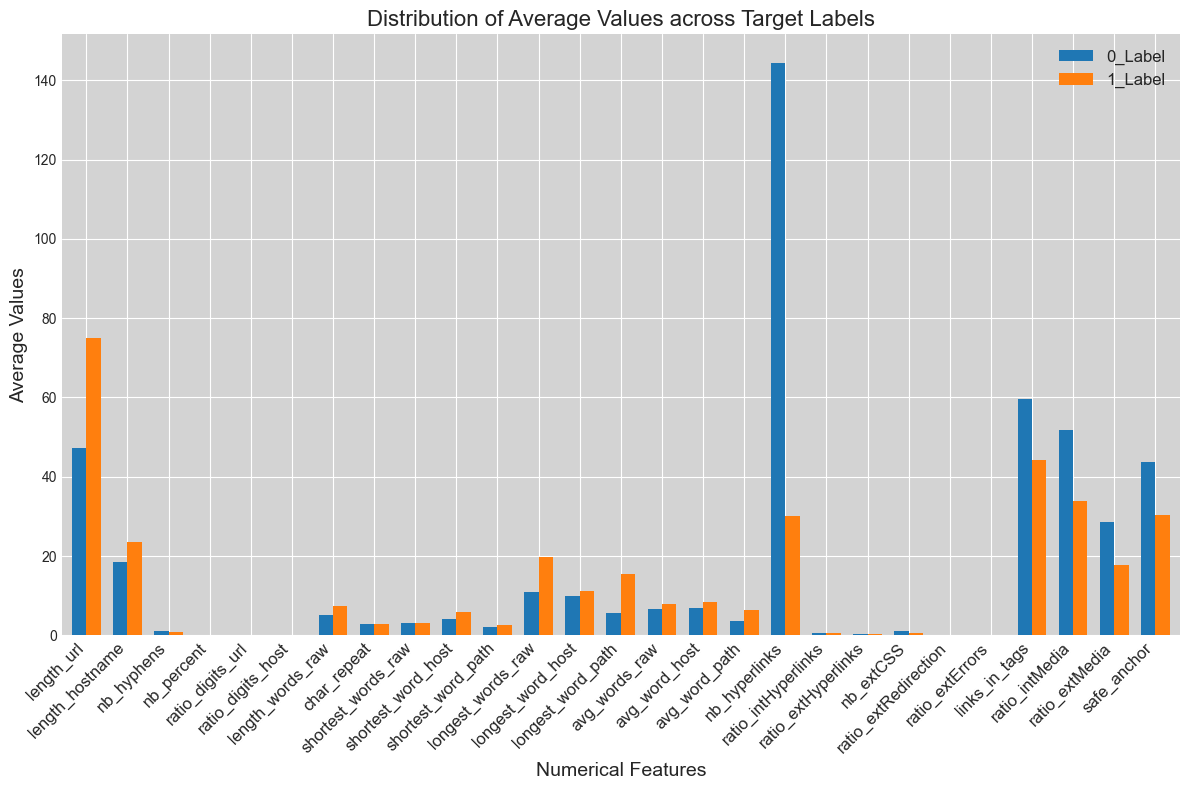


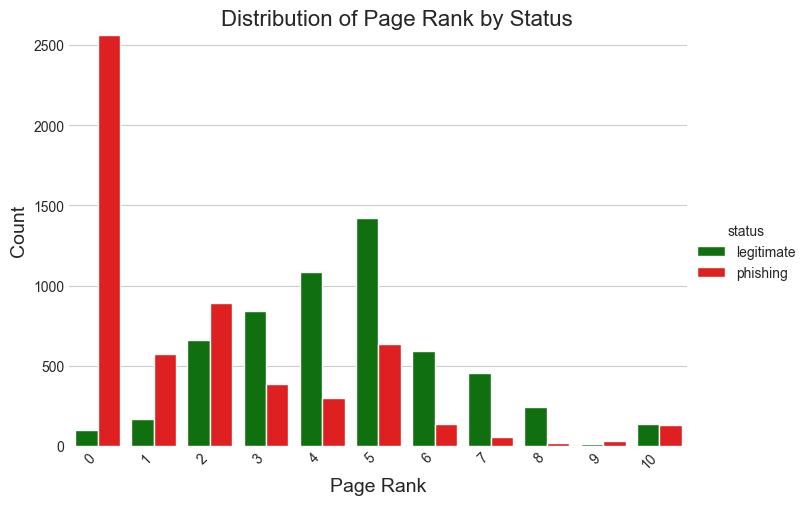


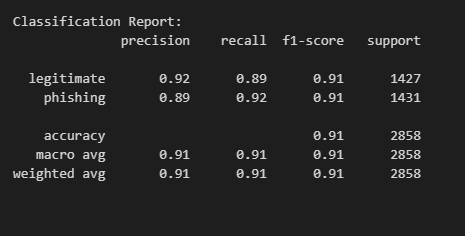


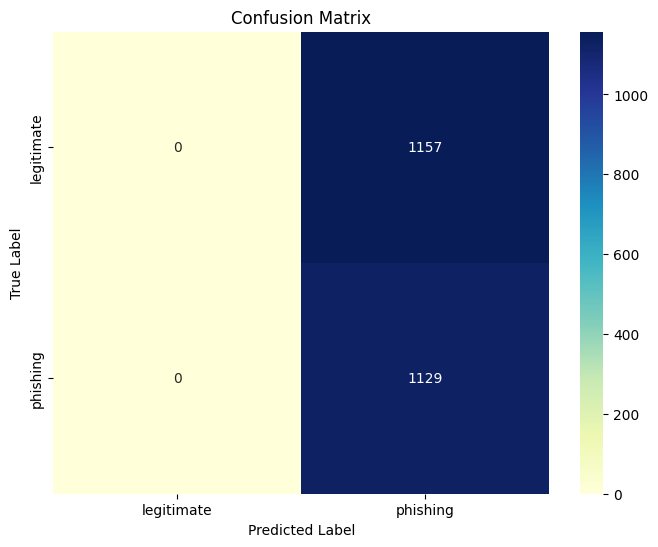


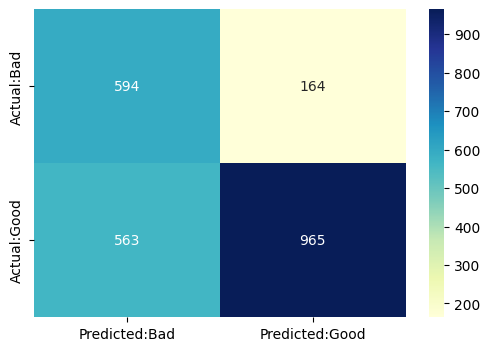


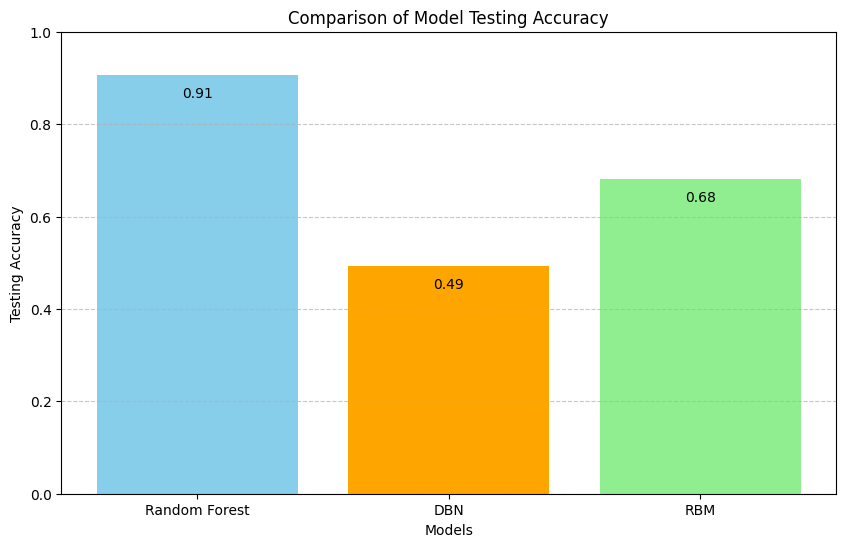












**Module 1: Data Exploration and Preprocessing**

This module focuses on loading and exploring the dataset, understanding its structure, and preprocessing steps. Key steps include:

Loading Data: Importing the dataset using pandas (pd.read\_csv) and displaying the first few rows to understand the data structure.

Data Summary: Utilizing df.describe() and df.info() to obtain summary statistics and information about the dataset.

Handling Missing Values: Identifying and handling missing values using df.isna().sum().

Data Transformation: Transforming categorical values into numerical representation, e.g., converting 'status' labels into binary values.

**Module 2: Exploratory Data Analysis (EDA)**

EDA performing Exploratory Data Analysis to understand relationships and patterns within the dataset.

**Correlation Analysis**: Computing and visualizing the correlation matrix using sns.heatmap to identify relationships between features.

**Feature Selection**: Implementing a function (feature\_selector\_correlation) to select relevant features based on a correlation threshold.

**Module 3: Model Training and Evaluation**

Models covers training machine learning models (Random Forest Classifier, Logistic Regression, SVC, Decision Tree) and evaluating their performance:

**Data Splitting**: Splitting the dataset into training and testing sets using train\_test\_split.

**Model Training**: Utilizing scikit-learn to train Random Forest (96), Logistic Regression(84), Support Vector Classifier (SVC) (70), and Decision Tree (93) models.

**Model Evaluation**: Creating a custom function (custom\_accuracy\_set) to evaluate the models on training and testing sets, including confusion matrix and classification report.

**Module 4: Model Comparison and Visualization**

**Model Comparison**: Using bar charts to compare training, testing, and cross-validation accuracies for each model.

**Visualization**: Visualizing feature importances using a bar chart for the Random Forest model.

**Module 5: Cross-Validation and Test Size Sensitivity**

**Test Size Sensitivity**: Examining how varying test sizes (e.g., 20%, 30%, 40%, 50%) affects model accuracy.

**Cross-Validation Scores**: Calculating cross-validation scores for each model and presenting the results.