FINAL PROJECT REPORT – GROUP 7

DATA ANALYSIS TOOLS ANALYTICS

Chintakunta, Nikhileswar - 100941492

Nimmala, Maneesha - 100939520

Sandeep Eeda - 100938744

Mandeep Kaur - 100973967

Harleen Kaur - 100952121

DATA-1202-04

Table of Contents

August 15, 2024

DURHAM COLLEGE
SUCCESS MATTERS

DATA-1202-04

Final Project Report: Data Analytics and Machine Learning for Cybersecurity

Introduction

In today's digital landscape, cybersecurity is of paramount importance, particularly with the growing threat of phishing attacks. Phishing URLs are a common tactic used by cybercriminals to deceive users into providing sensitive information, such as login credentials and financial details. This project focuses on utilizing machine learning techniques to analyze a dataset containing phishing URLs. Our goal is to preprocess the data, train multiple classifiers, and evaluate their performance to identify the most effective model for detecting phishing threats.

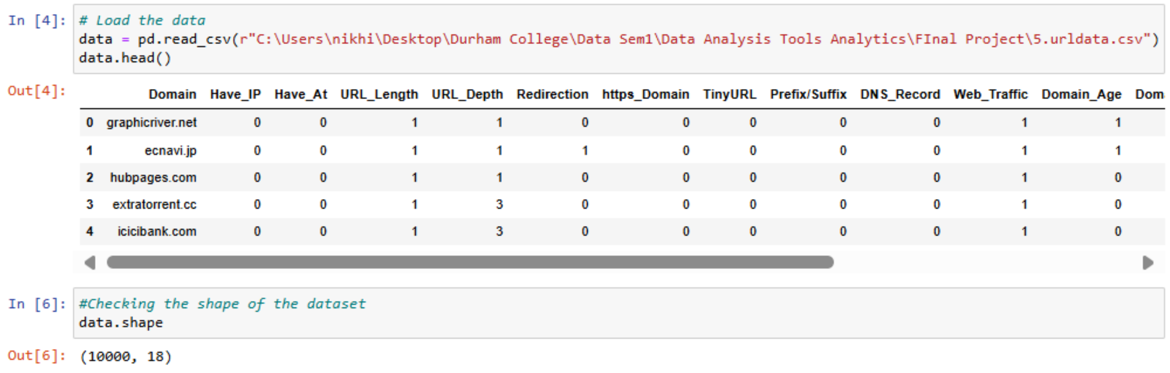
Dataset and Preprocessing

Dataset Description

The dataset used in this project was sourced from the provided 5.urldata.csv file. It contains various features related to URLs, including indicators that may suggest whether a URL is malicious (phishing) or benign. The dataset includes multiple columns, each representing different characteristics of the URLs. Understanding and processing this data is critical for building effective machine learning models.

Data Extraction

We began by loading the dataset using pandas. The dataset's structure was examined using the .info() and .shape() methods, revealing its size and the types of features included.



```
In [4]: # Load the data
        data = pd.read_csv(r"C:\Users\nikhi\Desktop\Durham College\Data Sem1\Data Analysis Tools Analytics\FInal Project\5.urldata.csv")
        data.head()
```

Out[4]:

	Domain	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Dom
0	graphicriver.net	0	0	1	1	0	0	0	0	0	1	1	
1	ecnavi.jp	0	0	1	1	1	0	0	0	0	1	1	
2	hubpages.com	0	0	1	1	0	0	0	0	0	1	0	
3	extratorrent.cc	0	0	1	3	0	0	0	0	0	1	0	
4	icicibank.com	0	0	1	3	0	0	0	0	0	1	0	

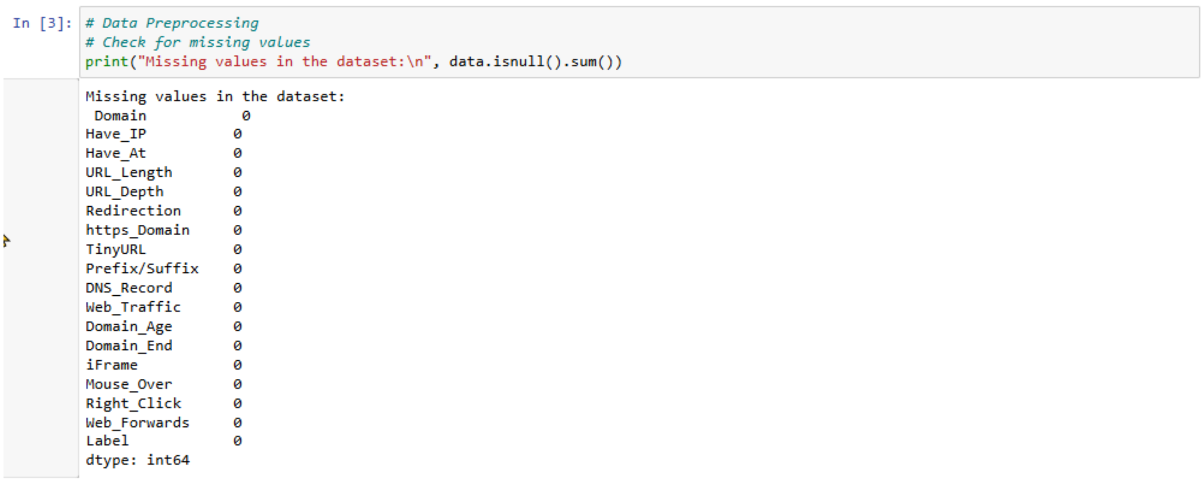
```
In [6]: #Checking the shape of the dataset
        data.shape

Out[6]: (10000, 18)
```

Handling Missing Values

To ensure data integrity, we first checked for any missing values across the dataset. Fortunately, there were no missing values, which allowed us to proceed without the need for imputation or removal of data points.

DURHAM COLLEGE
SUCCESS MATTERS

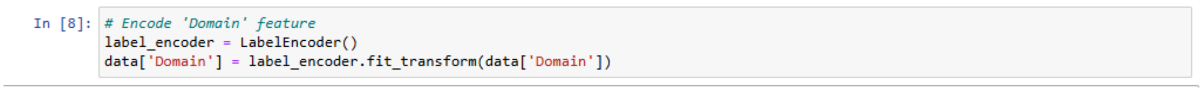
DATA-1202-04



```
In [3]: # Data Preprocessing
        # Check for missing values
        print("Missing values in the dataset:\n", data.isnull().sum())

Missing values in the dataset:
 Domain          0
Have_IP         0
Have_At         0
URL_Length      0
URL_Depth       0
Redirection     0
https_Domain    0
TinyURL         0
Prefix/Suffix   0
DNS_Record      0
Web_Traffic     0
Domain_Age      0
Domain_End      0
iFrame          0
Mouse_Over      0
Right_Click     0
Web_Forwards    0
Label           0
dtype: int64
```

Label Encoding

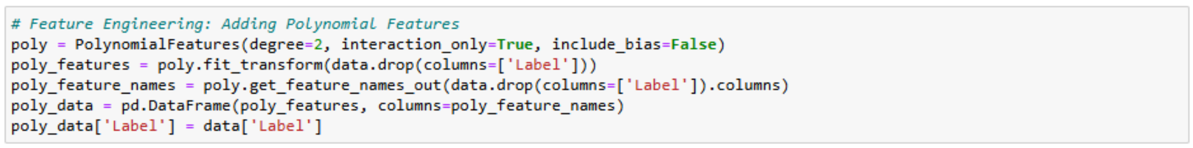
Given that the dataset contained categorical data in the 'Domain' feature, we applied label encoding using Label Encoder to convert this categorical data into a numerical format suitable for machine learning algorithms.



```
In [8]: # Encode 'Domain' feature
        label_encoder = LabelEncoder()
        data['Domain'] = label_encoder.fit_transform(data['Domain'])
```

Feature Engineering

To enhance the predictive power of our models, we implemented polynomial feature engineering. This involved creating interaction terms between the existing features using PolynomialFeatures from the sklearn library. This step added complexity to our models, potentially improving their ability to capture non-linear relationships in the data.



```
# Feature Engineering: Adding Polynomial Features
poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)
poly_features = poly.fit_transform(data.drop(columns=['Label']))
poly_feature_names = poly.get_feature_names_out(data.drop(columns=['Label']).columns)
poly_data = pd.DataFrame(poly_features, columns=poly_feature_names)
poly_data['Label'] = data['Label']
```

Data Splitting and Feature Scaling

We split the dataset into training and testing sets using the train_test_split function, with 70% of the data allocated for training and 30% for testing. This approach ensured that our models were trained on a substantial portion of the data while still leaving enough data for unbiased testing. To standardize the features, we applied StandardScaler. This scaling was necessary because many machine learning models perform better when the input data is normalized. The scaler was fitted to the training data and then applied to both the training and testing sets.

DATA-1202-04

```
# Splitting the data into training and testing sets
X = poly_data.drop(columns=['Label'])
y = poly_data['Label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

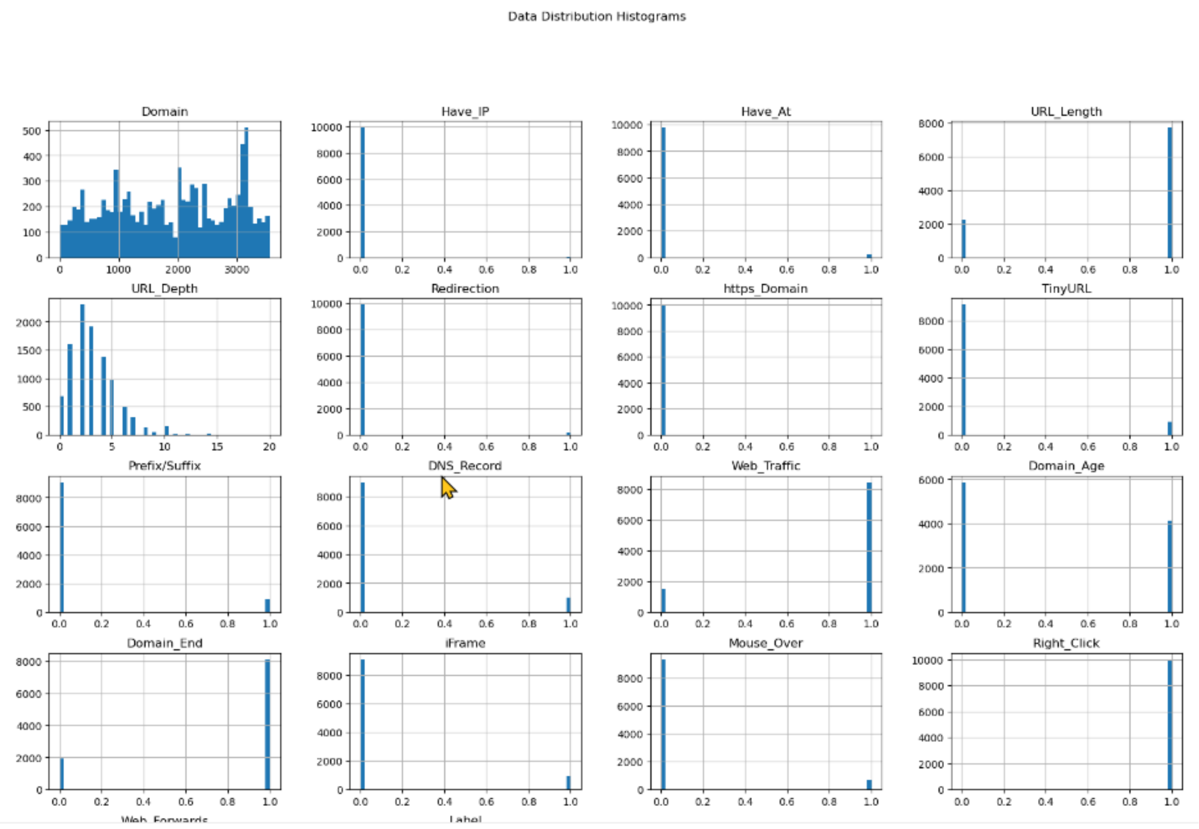
Exploratory Data Analysis (EDA)

Data Visualization

To gain insights into the dataset, we conducted exploratory data analysis (EDA) using various visualization techniques:

- **Histograms**: We plotted histograms for each feature to understand the distribution of the data. This step helped us identify any potential skewness or outliers that could affect model performance.

```
# Exploratory Data Analysis (EDA)
# Data distribution histograms
data.hist(bins=50, figsize=(20, 15))
plt.suptitle('Data Distribution Histograms')
plt.show()
```

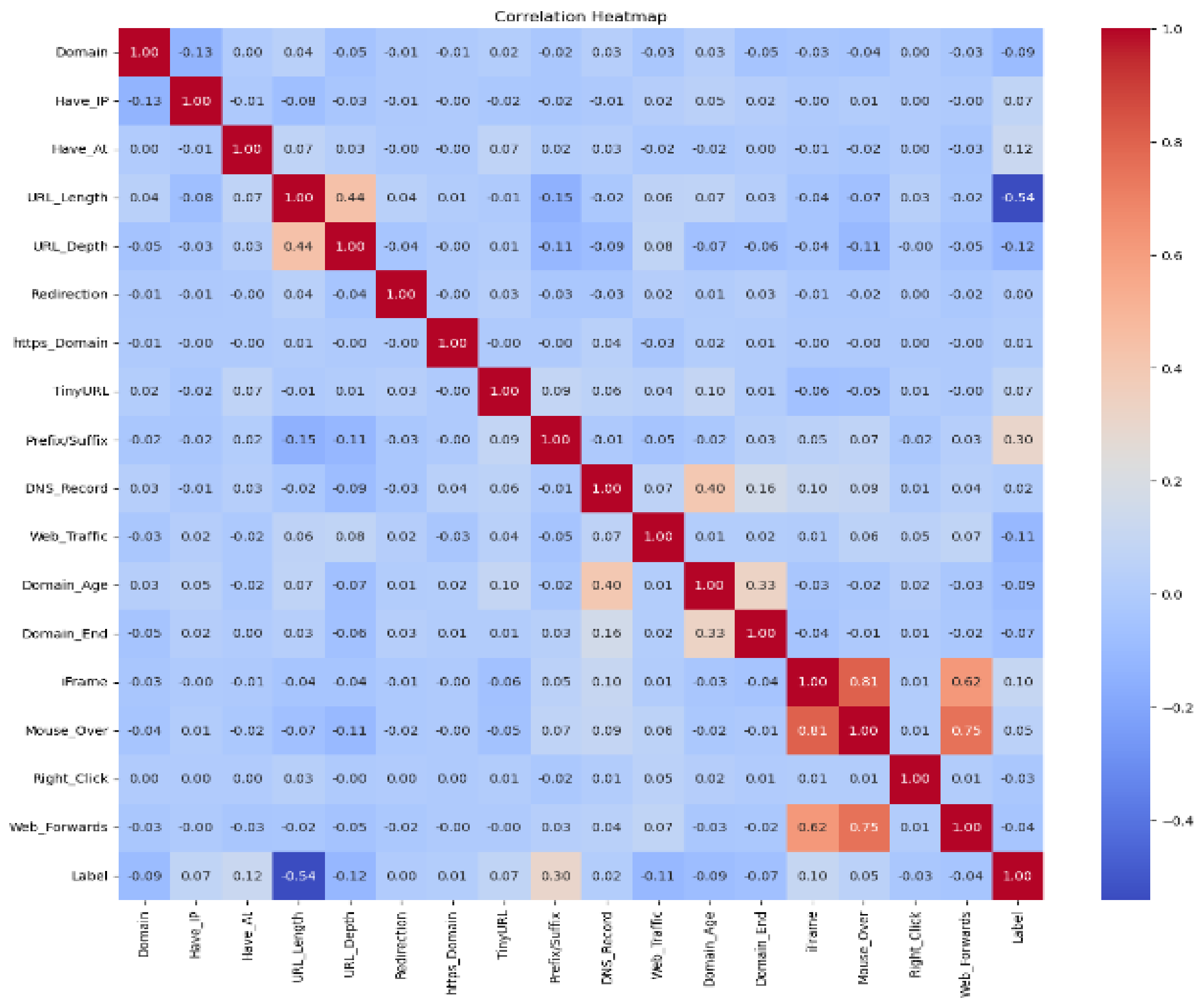


DURHAM COLLEGE
SUCCESS MATTERS

DATA-1202-04

- **Correlation Heatmap**: A correlation heatmap was generated to analyze the relationships between features. This visualization highlighted any strong correlations that could inform our feature selection and engineering processes.

```
# Correlation heatmap
plt.figure(figsize=(15,13))
sns.heatmap(data.corr(), annot=True, fmt='.2f', cmap='coolwarm', cbar=True)
plt.title('Correlation Heatmap')
plt.show()
```



DATA-1202-04

Model Selection and Training

Classifier Selection

Given the nature of our data and the problem at hand—phishing URL detection—we selected a diverse set of classifiers to evaluate:

- **Gaussian Naive Bayes (GaussianNB)**: A probabilistic classifier often used for high-dimensional data.
- **Logistic Regression**: A linear model suitable for binary classification tasks.
- **Multilayer Perceptron (MLPClassifier)**: A neural network model capable of capturing complex patterns in the data.
- **Random Forest Classifier**: An ensemble learning method that combines multiple decision trees to improve predictive accuracy.

Model Training

Each of these models was trained on the preprocessed dataset. During training, we employed cross-validation and grid search techniques to optimize the hyperparameters of the models. This approach ensured that our models were not only trained effectively but also fine-tuned for optimal performance.

Model Testing and Evaluation

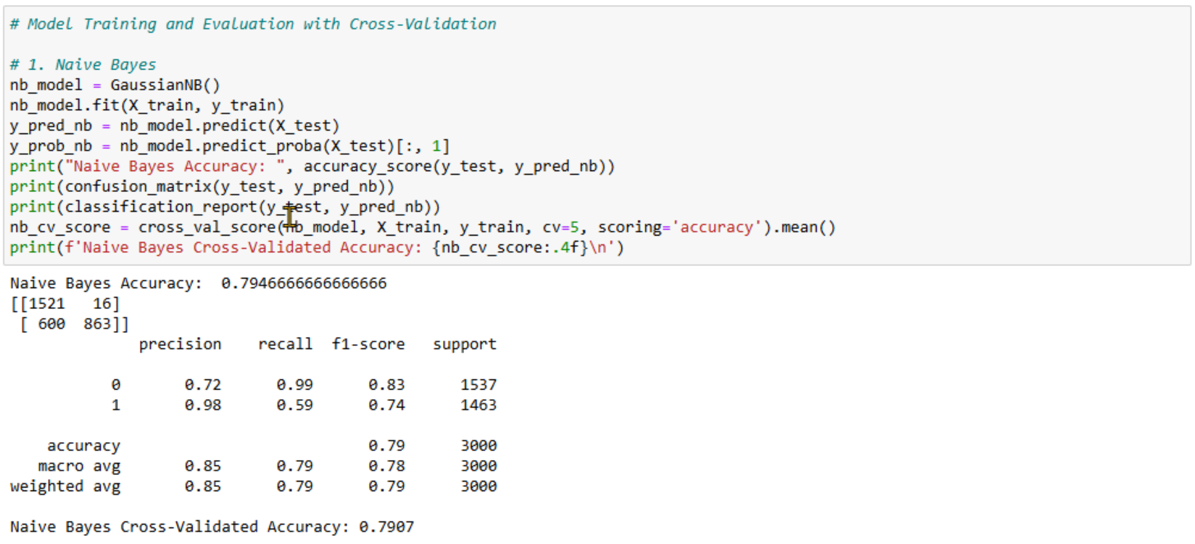
Performance Metrics

After training the models, we evaluated their performance using several metrics:

- **Accuracy**: The percentage of correct predictions made by the model.
- **Precision, Recall, F1 Score**: These metrics provided insights into the model's ability to correctly identify true positives while minimizing false positives and false negatives.

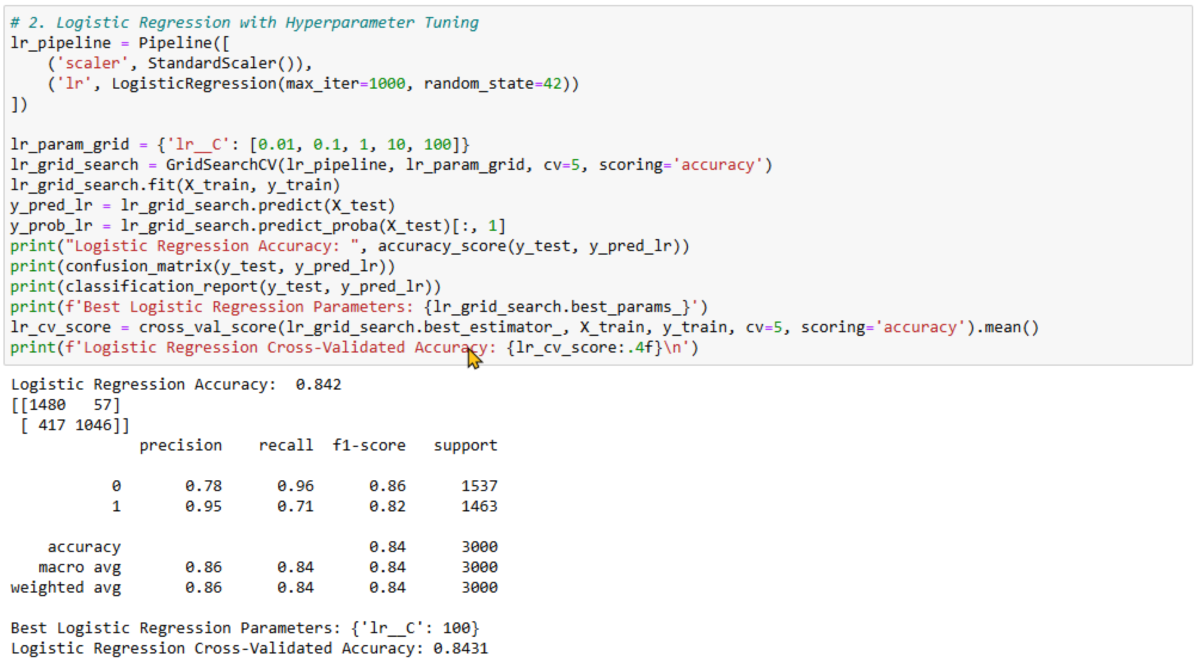
DURHAM COLLEGE
SUCCESS MATTERS

DATA-1202-04



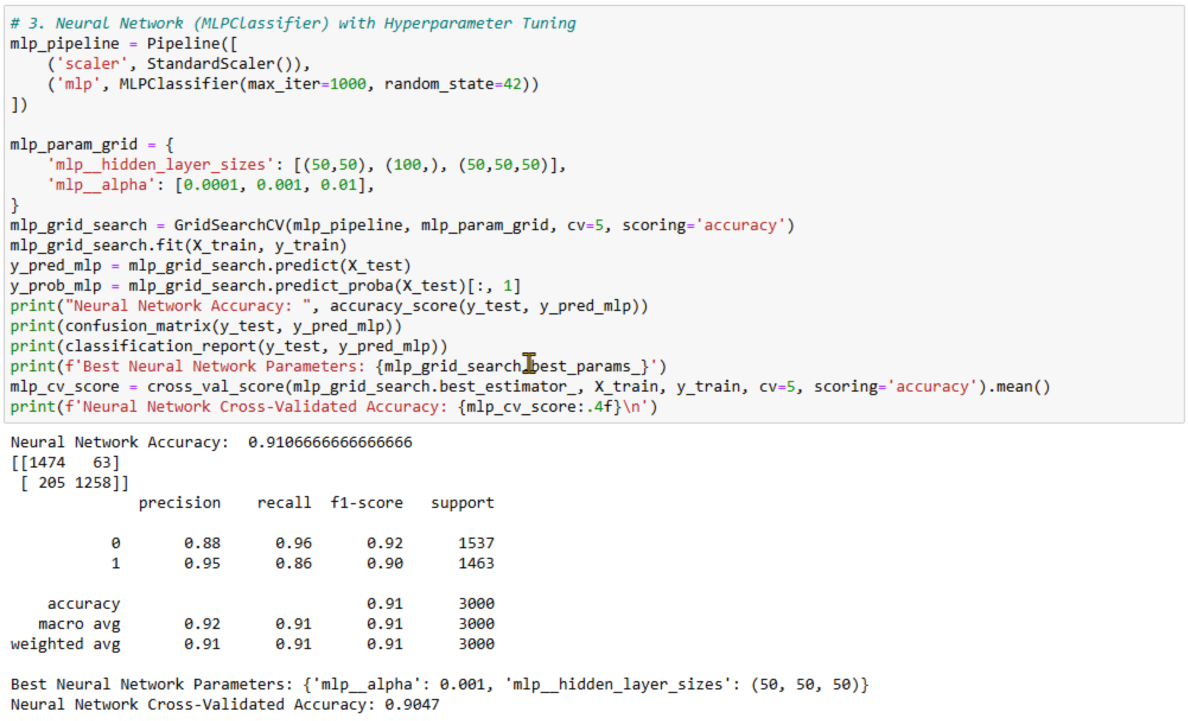
```
# Model Training and Evaluation with Cross-Validation

# 1. Naive Bayes
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
y_pred_nb = nb_model.predict(X_test)
y_prob_nb = nb_model.predict_proba(X_test)[:, 1]
print("Naive Bayes Accuracy: ", accuracy_score(y_test, y_pred_nb))
print(confusion_matrix(y_test, y_pred_nb))
print(classification_report(y_test, y_pred_nb))
nb_cv_score = cross_val_score(nb_model, X_train, y_train, cv=5, scoring='accuracy').mean()
print(f'Naive Bayes Cross-Validated Accuracy: {nb_cv_score:.4f}\n')
```

```
Naive Bayes Accuracy:  0.7946666666666666
[[1521   16]
 [ 600  863]]
              precision    recall  f1-score   support

           0       0.72      0.99      0.83      1537
           1       0.98      0.59      0.74      1463

    accuracy                           0.79      3000
   macro avg       0.85      0.79      0.78      3000
weighted avg       0.85      0.79      0.79      3000

Naive Bayes Cross-Validated Accuracy: 0.7907
```



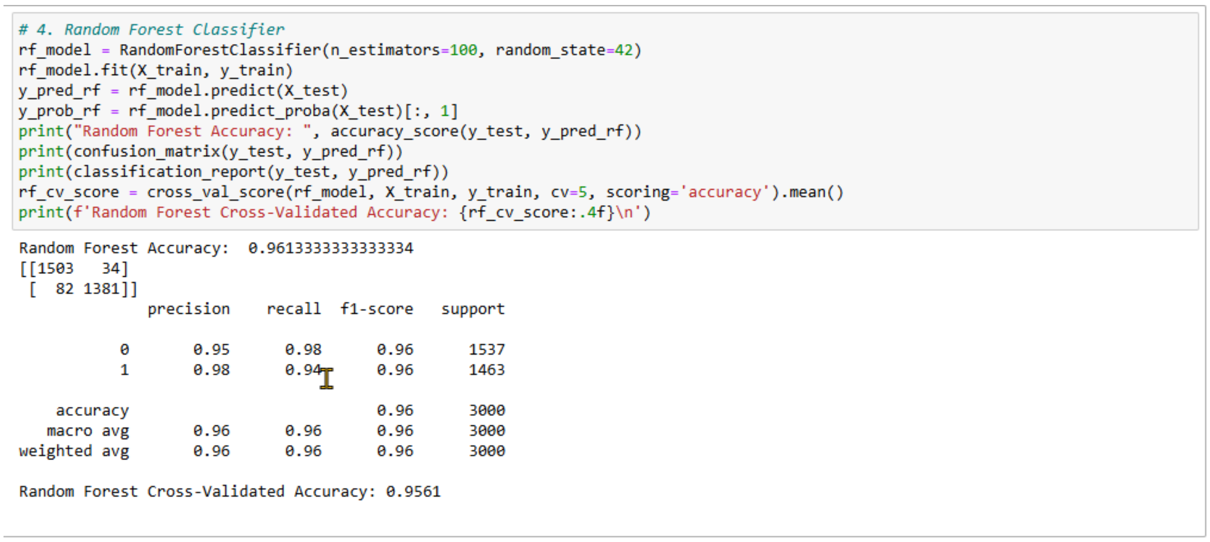
```
# 2. Logistic Regression with Hyperparameter Tuning
lr_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('lr', LogisticRegression(max_iter=1000, random_state=42))
])

lr_param_grid = {'lr__C': [0.01, 0.1, 1, 10, 100]}
lr_grid_search = GridSearchCV(lr_pipeline, lr_param_grid, cv=5, scoring='accuracy')
lr_grid_search.fit(X_train, y_train)
y_pred_lr = lr_grid_search.predict(X_test)
y_prob_lr = lr_grid_search.predict_proba(X_test)[:, 1]
print("Logistic Regression Accuracy: ", accuracy_score(y_test, y_pred_lr))
print(confusion_matrix(y_test, y_pred_lr))
print(classification_report(y_test, y_pred_lr))
print(f'Best Logistic Regression Parameters: {lr_grid_search.best_params_}')
lr_cv_score = cross_val_score(lr_grid_search.best_estimator_, X_train, y_train, cv=5, scoring='accuracy').mean()
print(f'Logistic Regression Cross-Validated Accuracy: {lr_cv_score:.4f}\n')
```

```
Logistic Regression Accuracy:  0.842
[[1480   57]
 [ 417 1046]]
              precision    recall  f1-score   support

           0       0.78      0.96      0.86      1537
           1       0.95      0.71      0.82      1463

    accuracy                           0.84      3000
   macro avg       0.86      0.84      0.84      3000
weighted avg       0.86      0.84      0.84      3000

Best Logistic Regression Parameters: {'lr__C': 100}
Logistic Regression Cross-Validated Accuracy: 0.8431
```



```
# 3. Neural Network (MLPClassifier) with Hyperparameter Tuning
mlp_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('mlp', MLPClassifier(max_iter=1000, random_state=42))
])

mlp_param_grid = {
    'mlp__hidden_layer_sizes': [(50,50), (100,), (50,50,50)],
    'mlp__alpha': [0.0001, 0.001, 0.01],
}
mlp_grid_search = GridSearchCV(mlp_pipeline, mlp_param_grid, cv=5, scoring='accuracy')
mlp_grid_search.fit(X_train, y_train)
y_pred_mlp = mlp_grid_search.predict(X_test)
y_prob_mlp = mlp_grid_search.predict_proba(X_test)[:, 1]
print("Neural Network Accuracy: ", accuracy_score(y_test, y_pred_mlp))
print(confusion_matrix(y_test, y_pred_mlp))
print(classification_report(y_test, y_pred_mlp))
print(f'Best Neural Network Parameters: {mlp_grid_search.best_params_}')
mlp_cv_score = cross_val_score(mlp_grid_search.best_estimator_, X_train, y_train, cv=5, scoring='accuracy').mean()
print(f'Neural Network Cross-Validated Accuracy: {mlp_cv_score:.4f}\n')
```

```
Neural Network Accuracy:  0.9106666666666666
[[1474   63]
 [ 205 1258]]
              precision    recall  f1-score   support

           0       0.88      0.96      0.92      1537
           1       0.95      0.86      0.90      1463

    accuracy                           0.91      3000
   macro avg       0.92      0.91      0.91      3000
weighted avg       0.91      0.91      0.91      3000

Best Neural Network Parameters: {'mlp__alpha': 0.001, 'mlp__hidden_layer_sizes': (50, 50, 50)}
Neural Network Cross-Validated Accuracy: 0.9047
```



```
# 4. Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
y_prob_rf = rf_model.predict_proba(X_test)[:, 1]
print("Random Forest Accuracy: ", accuracy_score(y_test, y_pred_rf))
print(confusion_matrix(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
rf_cv_score = cross_val_score(rf_model, X_train, y_train, cv=5, scoring='accuracy').mean()
print(f'Random Forest Cross-Validated Accuracy: {rf_cv_score:.4f}\n')
```

```
Random Forest Accuracy:  0.9613333333333334
[[1503   34]
 [  82 1381]]
              precision    recall  f1-score   support

           0       0.95      0.98      0.96      1537
           1       0.98      0.94      0.96      1463

    accuracy                           0.96      3000
   macro avg       0.96      0.96      0.96      3000
weighted avg       0.96      0.96      0.96      3000

Random Forest Cross-Validated Accuracy: 0.9561
```

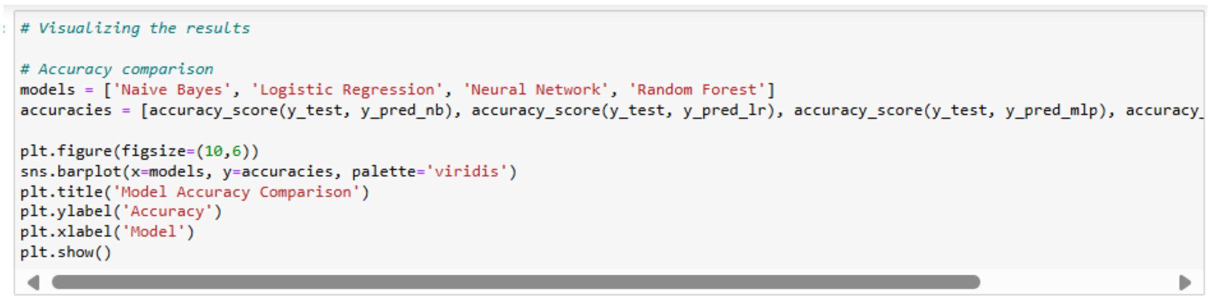
- **ROC AUC Score**: A measure of the model's ability to distinguish between the positive and negative classes.

Visualization and Comparison

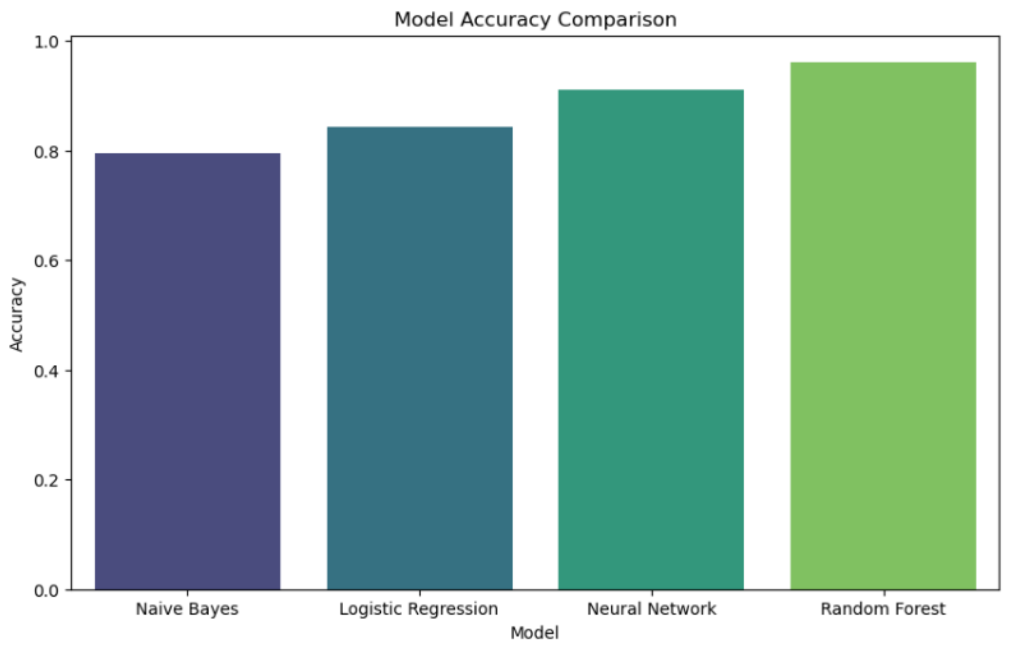
To further analyze the results, we plotted confusion matrices and ROC curves for each model. These visualizations helped us to compare the performance of the classifiers in a more intuitive way.

DURHAM COLLEGE
SUCCESS MATTERS

DATA-1202-04

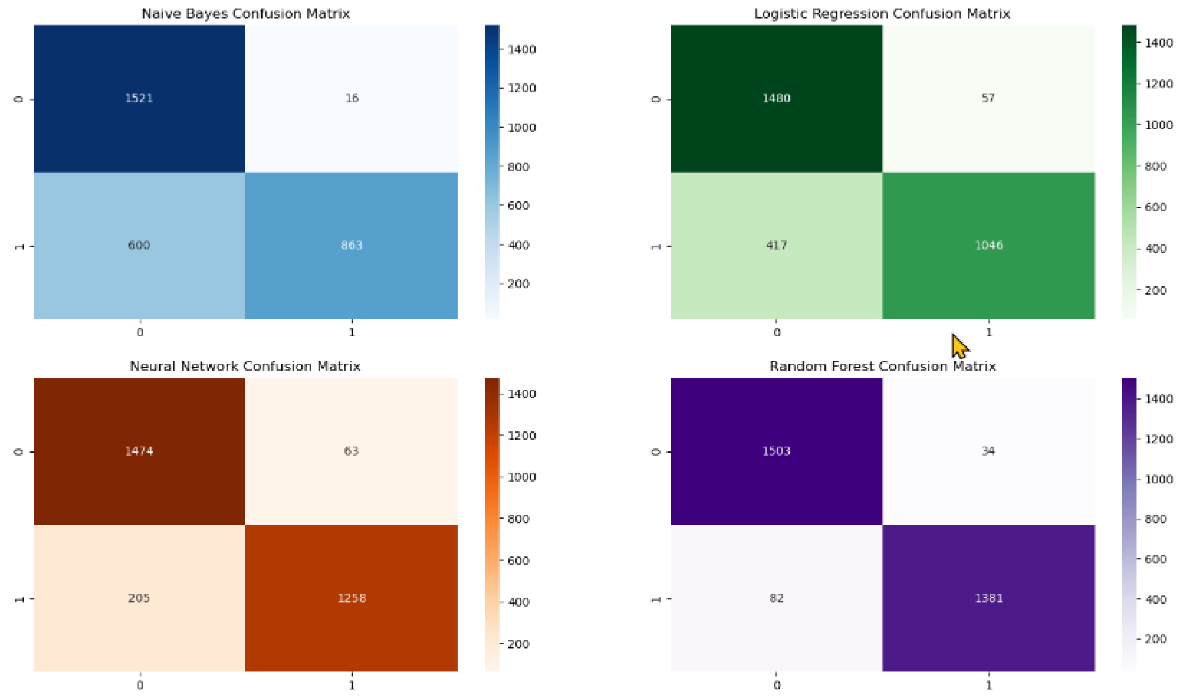


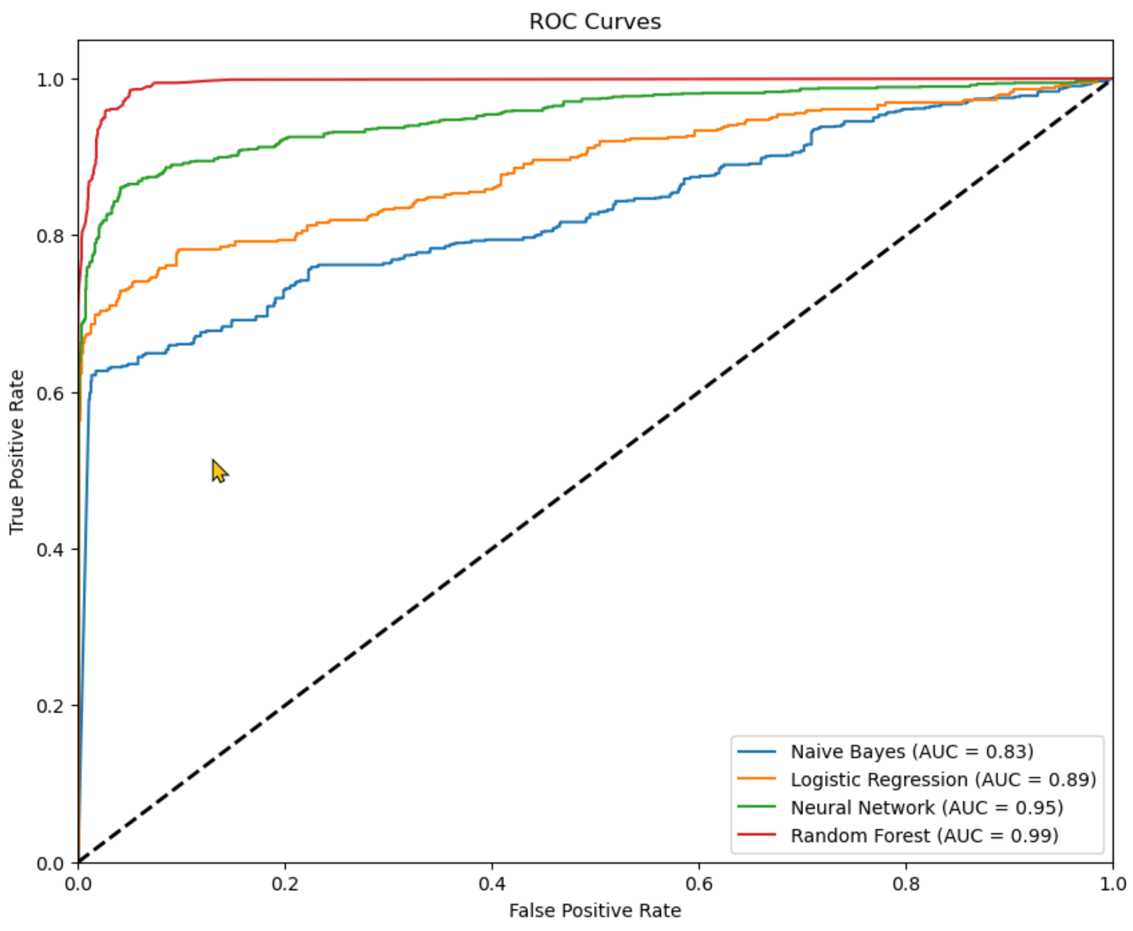
```
# Visualizing the results

# Accuracy comparison
models = ['Naive Bayes', 'Logistic Regression', 'Neural Network', 'Random Forest']
accuracies = [accuracy_score(y_test, y_pred_nb), accuracy_score(y_test, y_pred_lr), accuracy_score(y_test, y_pred_mlp), accuracy_

plt.figure(figsize=(10,6))
sns.barplot(x=models, y=accuracies, palette='viridis')
plt.title('Model Accuracy Comparison')
plt.ylabel('Accuracy')
plt.xlabel('Model')
plt.show()
```



DATA-1202-04

```
# Confusion matrices
fig, axes = plt.subplots(2, 2, figsize=(18, 10))

sns.heatmap(confusion_matrix(y_test, y_pred_nb), annot=True, fmt='d', cmap='Blues', ax=axes[0,0])
axes[0,0].set_title('Naive Bayes Confusion Matrix')

sns.heatmap(confusion_matrix(y_test, y_pred_lr), annot=True, fmt='d', cmap='Greens', ax=axes[0,1])
axes[0,1].set_title('Logistic Regression Confusion Matrix')

sns.heatmap(confusion_matrix(y_test, y_pred_mlp), annot=True, fmt='d', cmap='Oranges', ax=axes[1,0])
axes[1,0].set_title('Neural Network Confusion Matrix')

sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, fmt='d', cmap='Purples', ax=axes[1,1])
axes[1,1].set_title('Random Forest Confusion Matrix')

plt.show()
```





Upon evaluating the models, we found that the **Random Forest Classifier** outperformed the other models in terms of accuracy and overall predictive performance for detecting phishing URLs. The ensemble nature of the Random Forest allowed it to capture complex relationships in the data, making it the most effective model for this cybersecurity task.

The **Multilayer Perceptron** also showed promising results, particularly in capturing non-linear patterns in the data, thanks to its neural network structure. However, it required more computational resources and time to train compared to the Random Forest.

Challenges and Limitations

One of the challenges we faced was ensuring that the dataset was sufficiently balanced to avoid bias in the models. While we applied feature engineering techniques to enhance model performance, some features still presented multicollinearity, which could affect the models' interpretability.

DATA-1202-04

Conclusion

In this project, we successfully applied machine learning techniques to analyze a dataset of phishing URLs. By preprocessing the data, selecting and tuning multiple classifiers, and evaluating their performance, we identified the Random Forest Classifier as the most effective model for detecting phishing threats.