# Machine Learning Project Documentation

## Model Refinement

### 1. Overview

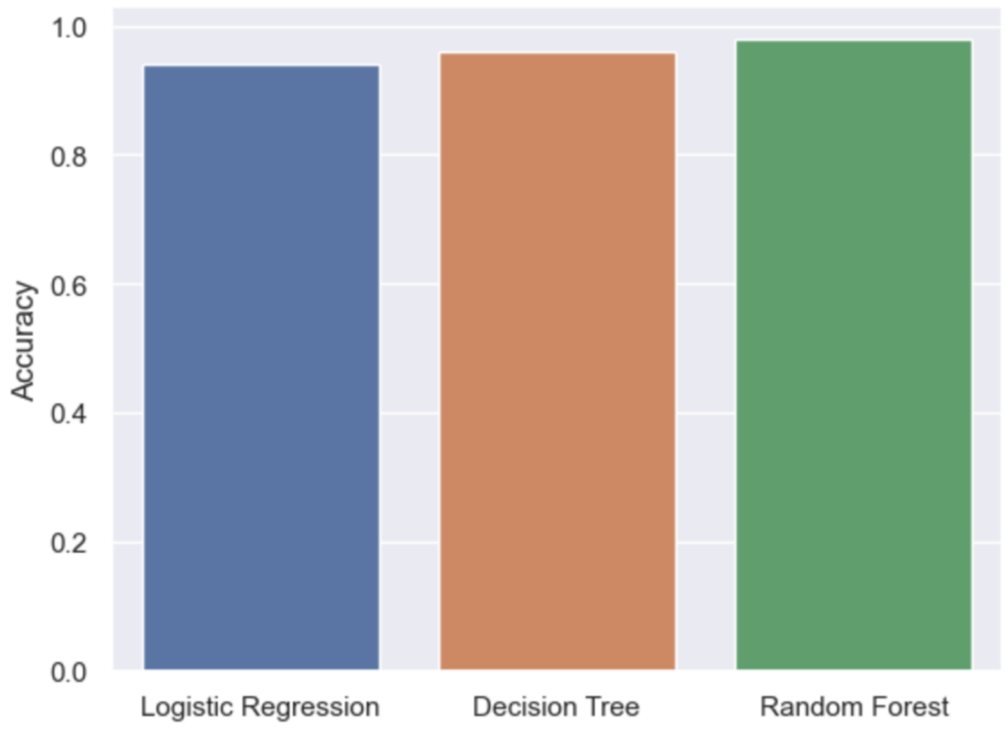
The model refinement phase in machine learning is a critical step where the initial model, created during the baseline or prototype phase, is iteratively improved upon. This phase is essential for enhancing the model's performance to ensure it meets the project's objectives effectively. Here's an overview of the model refinement phase:

- **Hyperparameter Tuning**
- **Feature Engineering**
- **Cross-Validation**
- **Model Selection**
- **Ensemble Methods**
- **Performance Metrics Evaluation**
- **Error Analysis**

Through these steps, the model refinement phase aims to incrementally build a more accurate, robust, and reliable machine learning model that performs well when deployed in a real-world setting, such as in identifying phishing websites in my project.

### 2. Model Evaluation

After conducting various model tests, the highlight model was Random Forest with 98.2% of prediction.



### 3. Hyperparameter Tuning

Hyperparameter tuning is an essential process in the refinement phase to improve a model's ability to predict accurately. Here's an overview of additional hyperparameter tuning steps that could be taken, along with insights that might be gained:

- For decision trees, hyperparameters like max_depth, min_samples_split can control the complexity of the trees.
- For random forest, hyperparameters like n_estimators characterized the number of trees in the forest and max_depth is the maximum depth of each tree.
- For logistic regression, hyperparameters like max_iter is Maximum number of iterations taken for the solvers to converge.

### 4. Cross-Validation

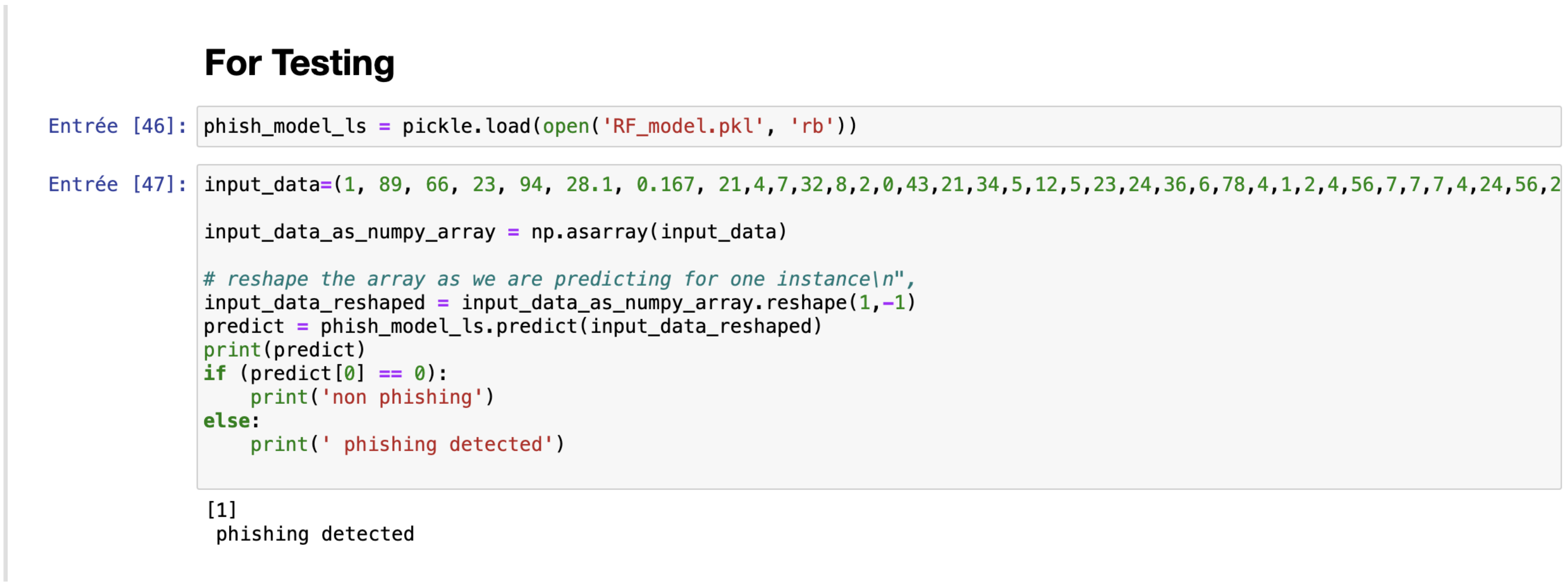
Cross-validation is a statistical method used to estimate the skill of machine learning models. It is commonly used to compare and select a model for a given predictive modeling problem because it ensures that every observation from the original dataset has the chance of appearing in training and test set.
The result of cross-validation provides a more robust and unbiased estimate of your model's performance on unseen data compared to a single train-test split.

## Test Submission

### Overview

The test submission phase is a critical point in the lifecycle of a machine learning model, where the model is evaluated on a test dataset to assess its performance. This phase serves as the final checkpoint before deployment or as a part of the model evaluation process.

### Data Preparation for Testing



#### For Testing

```
Entrée [46]: phish_model_ls = pickle.load(open('RF_model.pkl', 'rb'))
```

```
Entrée [47]: input_data=(1, 89, 66, 23, 94, 28.1, 0.167, 21,4,7,32,8,2,0,43,21,34,5,12,5,23,24,36,6,78,4,1,2,4,56,7,7,7,4,24,56,2

             input_data_as_numpy_array = np.asarray(input_data)

             # reshape the array as we are predicting for one instance\n",
             input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)
             predict = phish_model_ls.predict(input_data_reshaped)
             print(predict)
             if (predict[0] == 0):
                 print('non phishing')
             else:
                 print(' phishing detected')
```

```
[1]
 phishing detected
```

## Model Application

Utilizing the Python pickle module, we serialize our optimal machine learning model, effectively capturing its current state for future predictions. This serialization facilitates efficient model deployment or transfer across environments, enabling swift continuation of analysis without retraining overhead. By compressing the model into a byte stream, pickle ensures compact storage and rapid loading, enhancing the utility of our selected model for consistent performance in varying contexts. Furthermore, pickle supports parallel task execution, leveraging multicore processors to expedite computational workflows, a boon for complex machine learning operations.

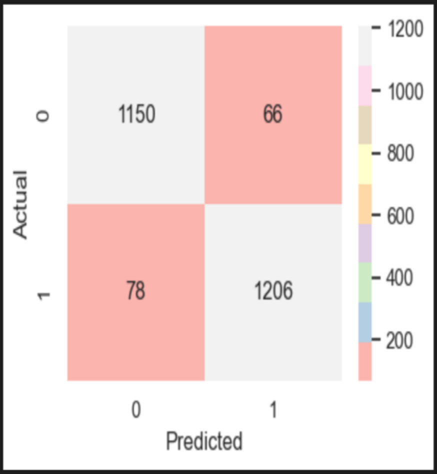
```
Entrée [24]: pickle.dump(logreg, open("logreg.pkl", "wb"))
```

```
Entrée [30]: pickle.dump(dect_model, open("dect_model.pkl", "wb"))
```

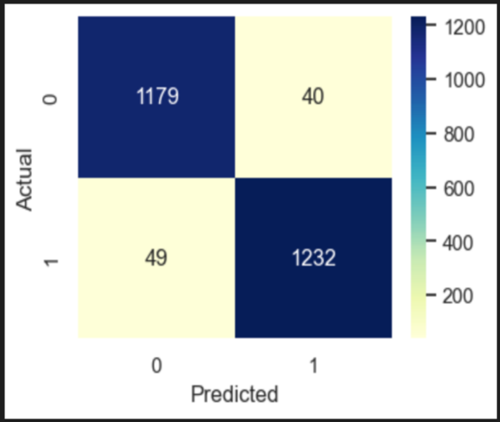
```
Entrée [36]: pickle.dump(RF_model, open("RF_model.pkl", "wb"))
```

## Test Metrics

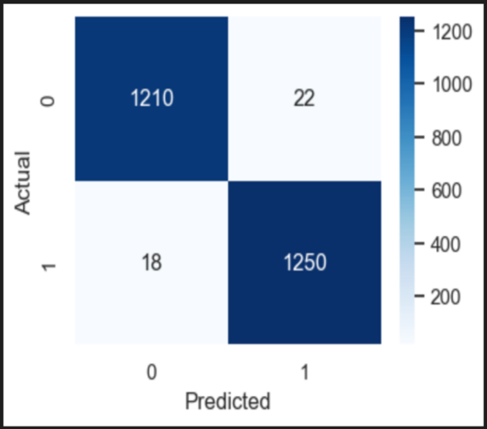
```
Training Accuracy : 0.9434666666666667
Testing Accuracy : 0.9424

 Logistic Regression CLASSIFICATION REPORT

Accuracy: 0.9424
              precision    recall  f1-score   support

  legitimate       0.95      0.94      0.94      1228
   malicious       0.94      0.95      0.94      1272

    accuracy                           0.94      2500
   macro avg       0.94      0.94      0.94      2500
weighted avg       0.94      0.94      0.94      2500

CONFUSION MATRIX
```



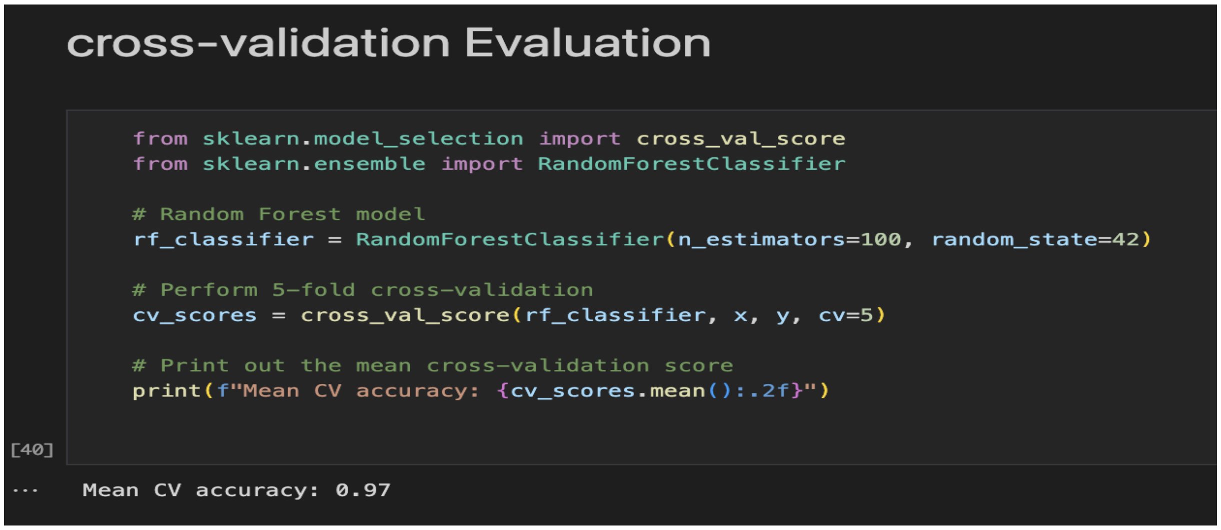
```
Training Accuracy : 1.0
Testing Accuracy : 0.9644

 Decision Tree CLASSIFICATION REPORT

Accuracy: 0.9644

              precision    recall  f1-score   support

  legitimate       0.97      0.96      0.96      1228
   malicious       0.96      0.97      0.97      1272

    accuracy                           0.96      2500
   macro avg       0.96      0.96      0.96      2500
weighted avg       0.96      0.96      0.96      2500

CONFUSION MATRIX
```



```
Training Accuracy : 1.0
Testing Accuracy : 0.984

 Random Forest CLASSIFICATION REPORT

Accuracy: 0.984

              precision    recall  f1-score   support

  legitimate       0.98      0.99      0.98      1228
   malicious       0.99      0.98      0.98      1272

    accuracy                           0.98      2500
   macro avg       0.98      0.98      0.98      2500
weighted avg       0.98      0.98      0.98      2500

CONFUSION MATRIX
```



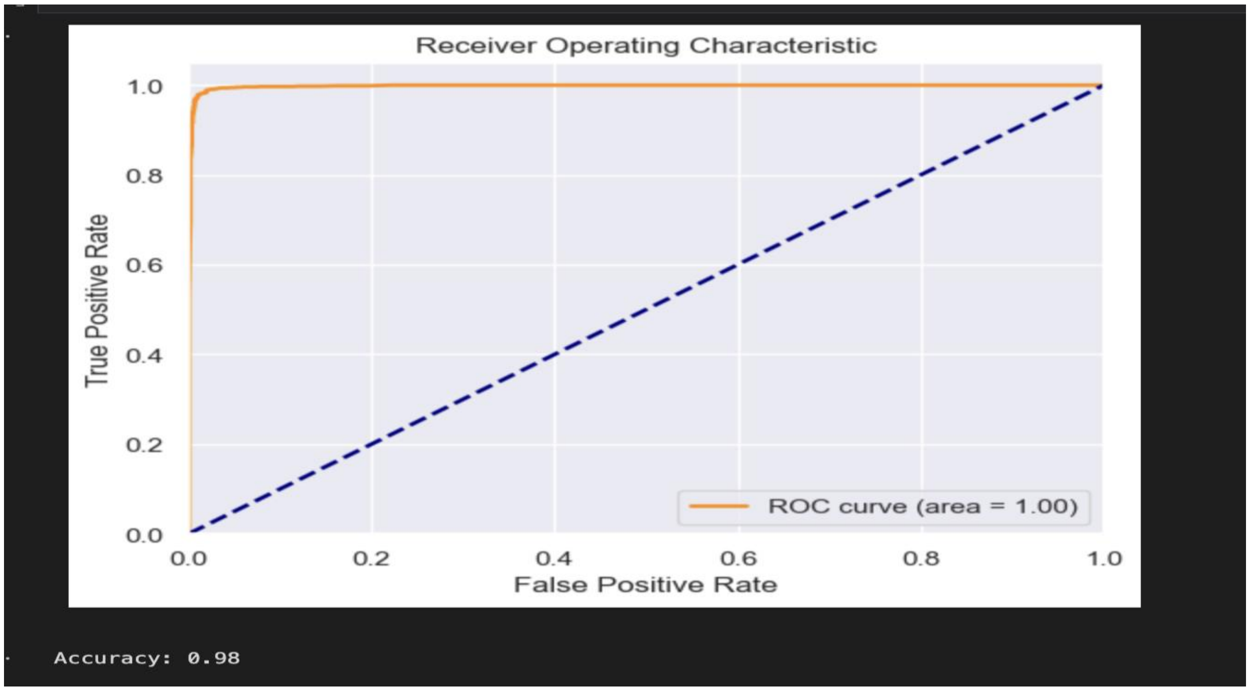
## Code Implementation



cross-validation Evaluation

```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

# Random Forest model
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Perform 5-fold cross-validation
cv_scores = cross_val_score(rf_classifier, x, y, cv=5)

# Print out the mean cross-validation score
print(f"Mean CV accuracy: {cv_scores.mean():.2f}")
```

```
[40]
...   Mean CV accuracy: 0.97
```



```
Accuracy: 0.98
```

## Conclusion

In the test submission phase, the refined model was subjected to a final evaluation using a reserved test dataset. Key performance metrics such as accuracy, precision, recall, and F1 score were computed, offering a comprehensive picture of the model's efficacy. The confusion matrix provided further insights into the model's specific strengths and weaknesses, such as its ability to minimize false positives and false negatives.