Quality Prediction and Webpage Phishing Detection Using Machine Learning

An Internship Report

Submitted in partial fulfilment of the requirements

for the award of the Degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION

BY

PRIYESH PUSHP

(BTECH/15229/20)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

BIRLA INSTITUTE OF TECHNOLOGY MESRA, OFF-CAMPUS PATNA-800014

2024

APPROVAL OF THE GUIDE

Recommended that the thesis entitled **"Quality Prediction and Webpage Phishing Detection Using Machine Learning"** presented by **Priyesh Pushp** under my supervision and guidance be accepted as fulfilling this part of the requirements for the award of Degree of **Bachelor of Technology.** To the best of my knowledge, the content of this thesis did not form a basis for the award of any previous degree to anyone else.

Date:

Dr. Priyadarshi Suraj
Asst. Professor
Dept. of ECE
Birla Institute of Technology
Mesra, Patna

DECLARATION CERTIFICATE

I certify that

a) The work contained in the thesis is original and has been done by myself under the general supervision of my supervisor.

b) The work has not been submitted to any other Institute for any other degree or diploma.

c) I have followed the guidelines provided by the Institute in writing the thesis.

d) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

e) Whenever I have used materials (data, theoretical analysis, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.

f) Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Priyesh Pushp
(BTECH/15229/20)

CERTIFICATE OF APPROVAL

This is to certify that the work embodied in this thesis entitled **“Quality Prediction and Webpage Phishing Detection Using Machine Learning”**, is carried out by **Priyesh Pushp (BTECH/15229/20)** has been approved for the degree of Bachelor of Technology of Birla Institute of Technology, Mesra, Patna.

Date:

Place:

Internal Examiner **External Examiner**

(Chairman)

In-Charge of Department

ABSTRACT

This presentation report explores the application of machine learning techniques in two distinct domains: Apple product quality prediction and web page phishing detection. In the realm of product quality prediction, we employ machine learning algorithms to analyse historical data related to Apple products, including manufacturing defects, customer complaints, and warranty claims. By leveraging this data, our model aims to forecast potential quality issues and enable proactive measures to improve product reliability and customer satisfaction.

In parallel, we delve into the domain of web page phishing detection, where machine learning algorithms are utilized to identify fraudulent web pages designed to deceive users into divulging sensitive information such as login credentials and financial details. Our approach involves feature extraction from web page content and URL characteristics, followed by training machine learning models to distinguish between legitimate and phishing websites. Through this endeavour, we seek to enhance cybersecurity measures and protect users from falling victim to online phishing scams.

By presenting these two case studies, we demonstrate the versatility and efficacy of machine learning in addressing diverse challenges across different domains, from product quality assurance to cybersecurity. Our findings underscore the potential of machine learning as a powerful tool for predictive analytics and risk mitigation in various real-world applications.

ACKNOWLEDGEMENT

I would like to express my profound gratitude to my project guide, **Dr. Priyadarshi Suraj** for his guidance and support during my thesis work. I benefited greatly by working under his guidance. It was his effort for which I am able to develop a detailed insight on this subject and special interest to study further. His encouragement motivation and support has been invaluable throughout my studies at BIT, Mesra, Off Campus Patna.

I convey my sincere gratitude to **Dr. Ritesh Kumar Badhai**, In-Charge, Dept. of ECE, BIT, Mesra, Off Campus Patna, for providing me various facilities needed to complete my project work. I would also like to thank all the faculty members of ECE department who have directly or indirectly helped during the course of the study. I would also like to thank all the staff (technical and non-technical) and my friends at BIT, Mesra, Off Campus Patna who have helped me greatly during the course.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout the years of my study. This accomplishment would not have been possible without them.

My apologies and heartful gratitude to all who have assisted me yet have not been acknowledged by name.

Thank you.

DATE: Priyesh Pushp

(BTECH/15229/20)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Quality Prediction Using Machine Learning

1.1 Introduction

Embarking on a journey to understand machine learning (ML) can be as thrilling as it is challenging. The project “Quality Prediction Using Machine Learning” serves not only as a practical application of ML techniques but also as a rich learning experience. This project is a testament to the transformative power of ML in deciphering complex patterns and making informed predictions. Embarking on a journey to understand machine learning (ML) can be as thrilling as it is challenging. The project “Quality Prediction Using Machine Learning” serves not only as a practical application of ML techniques but also as a rich learning experience. This project is a testament to the transformative power of ML in deciphering complex patterns and making informed predictions.

The essence of learning ML lies in its application. As we delve into the project, we encounter the core components of ML: data preprocessing, feature engineering, model selection, and performance evaluation. Each step is an opportunity to grasp the intricacies of algorithms and the importance of data quality. By predicting the quality of a tangible product, we witness firsthand the impact of ML in real-world scenarios.

Data is the bedrock of ML. Through this project, we learn to curate datasets, handle missing values, and understand the significance of data diversity. We explore how data representation influences the learning process and the outcomes of predictive models.

Algorithms are the tools that carve insights from data. This project introduces us to a variety of algorithms, from simple linear regressions to complex neural networks. We learn to match the problem at hand with the most suitable algorithm, considering factors like accuracy, interpretability, and computational efficiency.

Evaluation metrics are the measures of success. They guide us in refining models and in understanding the trade-offs between different types of errors. Through this project, we learn that a model's performance extends beyond accuracy; it encompasses precision, recall, and the balance between them.

The ultimate goal of ML is to create a positive impact. This project highlights the potential of ML to enhance quality control processes, reduce waste, and improve consumer satisfaction. It underscores the role of ML in driving innovation and efficiency across industries.

In conclusion, "Quality Prediction Using Machine Learning" is more than a project; it's a comprehensive learning module. It equips us with the knowledge and skills to tackle complex problems and paves the way for future explorations in the field of ML. As we progress, we not only predict quality but also embody the quality of learning that ML fosters.

1.2 Methodology

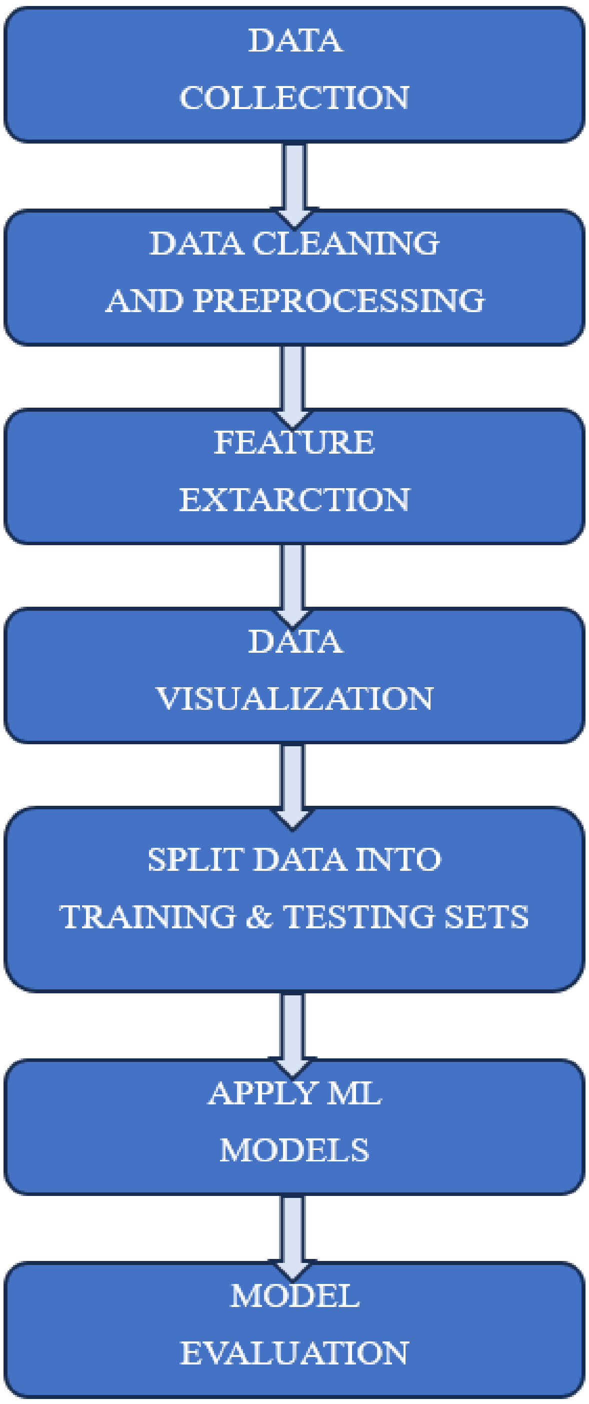
The methodology for Quality Prediction using Machine Learning requires following steps:

1. Exploratory Data Analysis:
    - Familiarizing with the Dataset.
    - Identifying and understanding all the features available.
    - Checking for missing values, outliers, and anomalies in the data.
2. Data Preprocessing:
    - Cleaning the data by handing missing values, outliers, and anomalies appropriately.
    - Exploring distribution of the target variable (quality) to understand class distribution.
3. Feature Engineering:
    - Accessing the relevance of existing features for predicting apple quality.
    - Creating new features that might enhance the model's predictive capabilities.
4. Data Visualisation:
    - Generating visualisations to gain insights into the relationships between different and quality.
    - Creating visualisations that help communicate your findings effectively.
5. Model Selection and training:
    - Choosing appropriate machine learning algorithms for classification tasks.
    - Splitting the dataset into training and testing sets.

- Training chosen models on the training data.

6. Model Evaluation:
    - Evaluating the performance of models using appropriate metrics (accuracy, precision, recall, etc.).
    - Adjusting hyperparameters to optimize model’s performance.
7. Model Interpretation:
    - Interpreting model's predictions.
    - Identifying the key features influencing the quality predictions.

1.3 Proposed Algorithm



1.4 Literature Review

1. "Prediction of Apple Internal Quality Using Spectral Absorption and Scattering Properties" by J. Qin, R. Lu, Y. Peng [1]: The objective of this research was to measure the absorption and reduced scattering coefficients (a and 's, respectively) of apples using a newly developed spatially resolved hyperspectral imaging technique and relate them to fruit firmness and soluble solids content (SSC).

2. "Application of Hyperspectral Imaging and Acoustic Emission Techniques for Apple Quality Prediction" by Nader Ekramirad, Ahmed Rady, Akinbode A. Adedeji, Reza Alimardani [2]: This study results showed that hyperspectral imaging (HSI) could accurately predict all the attributes except TSS, while the AE method was capable of predicting fruit firmness, b* colour index, and TSS.

3. "Apple quality assessment by fusion three sensors" by Zou Xiaobo and Zhao Jiewen [3]: In this research paper, the three sensors are combined to evaluate the quality of apples through a decision tree, and only 6 apples in set A, 1 apple in set B were misclassified.

4. "A method for prediction by combining data analysis and neural networks: Application to prediction of apple quality using near infra-red spectraag" by L. Bochereau, P. Bourgine, B. Palagos [4]: The results of this experiment were used to derive a model for predicting apple quality from near infra-red spectra. The percentage variation explained (R^2) of the linear model was equal to 0·82 and the remaining error was reduced by 5% using multilayer neural networks.

5. "Non-destructive prediction of quality of intact apple using near infrared spectroscopy" by S.N. Jha and Ruchi Garg [5]: The objective of this study was to investigate the potential of NIRS for prediction of TSS, titratable acidity and acidity/TSS ratio for commonly used 5 cultivars of apple to minimize the effect of variety during prediction.

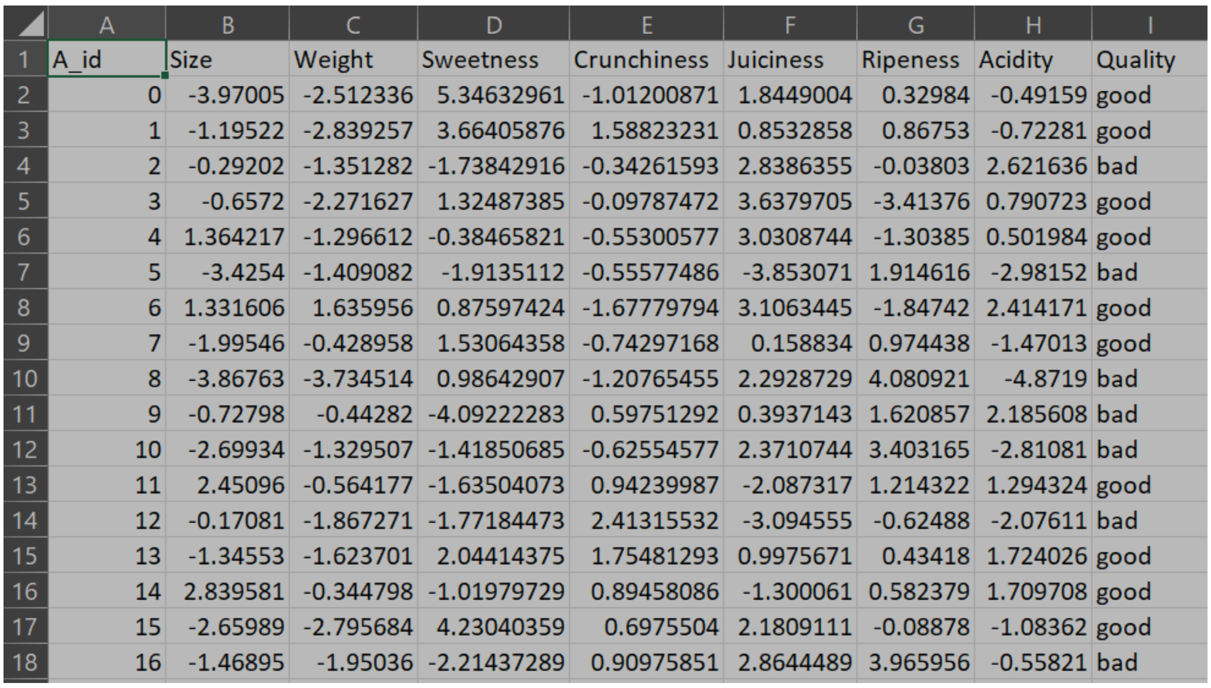
6. "Prediction of Apple Fruit Quality using Preharvest Mineral Nutrients" by E. Fallahi, B. Fallahi, J.B. Retamales, C. Valdés, S.J. Tabatabaei [6]: In this research , increase in

fruit N was always negatively associated with fruit yellow or red colour and increase in fruit Ca was negatively associated with bitter pit but was positively associated with fruit firmness. Apple fruits with greater N concentrations had greater ethylene concentrations and respiration rates.

1.5 Data Collection and Analysis

In this research, I employed the dataset obtained from Kaggle.

- This meticulously curated dataset serves as a detailed compendium of fruit attributes, offering a granular view of the various factors that define the essence and quality of fruits. It is an extensive collection that includes a unique identifier for each fruit, known as the fruit ID, which ensures precise tracking and differentiation among the diverse fruit varieties. The dataset captures critical dimensions such as the size and weight, providing a quantitative measure of the fruits' physical characteristics.



	A	B	C	D	E	F	G	H	I
1	A_id	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	Acidity	Quality
2	0	-3.97005	-2.512336	5.34632961	-1.01200871	1.8449004	0.32984	-0.49159	good
3	1	-1.19522	-2.839257	3.66405876	1.58823231	0.8532858	0.86753	-0.72281	good
4	2	-0.29202	-1.351282	-1.73842916	-0.34261593	2.8386355	-0.03803	2.621636	bad
5	3	-0.6572	-2.271627	1.32487385	-0.09787472	3.6379705	-3.41376	0.790723	good
6	4	1.364217	-1.296612	-0.38465821	-0.55300577	3.0308744	-1.30385	0.501984	good
7	5	-3.4254	-1.409082	-1.9135112	-0.55577486	-3.853071	1.914616	-2.98152	bad
8	6	1.331606	1.635956	0.87597424	-1.67779794	3.1063445	-1.84742	2.414171	good
9	7	-1.99546	-0.428958	1.53064358	-0.74297168	0.158834	0.974438	-1.47013	good
10	8	-3.86763	-3.734514	0.98642907	-1.20765455	2.2928729	4.080921	-4.8719	bad
11	9	-0.72798	-0.44282	-4.09222283	0.59751292	0.3937143	1.620857	2.185608	bad
12	10	-2.69934	-1.329507	-1.41850685	-0.62554577	2.3710744	3.403165	-2.81081	bad
13	11	2.45096	-0.564177	-1.63504073	0.94239987	-2.087317	1.214322	1.294324	good
14	12	-0.17081	-1.867271	-1.77184473	2.41315532	-3.094555	-0.62488	-2.07611	bad
15	13	-1.34553	-1.623701	2.04414375	1.75481293	0.9975671	0.43418	1.724026	good
16	14	2.839581	-0.344798	-1.01979729	0.89458086	-1.300061	0.582379	1.709708	good
17	15	-2.65989	-2.795684	4.23040359	0.6975504	2.1809111	-0.08878	-1.08362	good
18	16	-1.46895	-1.95036	-2.21437289	0.90975851	2.8644489	3.965956	-0.55821	bad

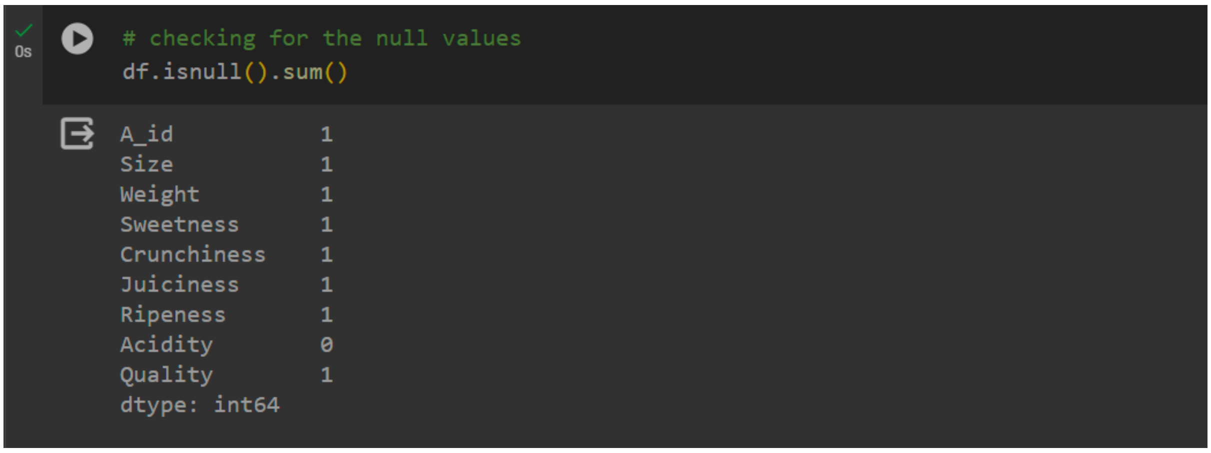
Fig 1.1: Dataset of Apple Quality Prediction

- Moreover, it delves into the sensory aspects, encompassing sweetness and crunchiness, which are indicative of taste and texture, respectively. Juiciness and ripeness are also recorded, reflecting the fruits' freshness and stage of maturity. Acidity levels are included to complete the flavour profile. The culmination of these attributes is the quality rating, which synthesizes the data into an overall assessment of the fruit's condition. This dataset is instrumental for stakeholders in the agricultural and food industries, enabling informed decisions and quality control measures.

1.6 Data Preprocessing and Cleaning

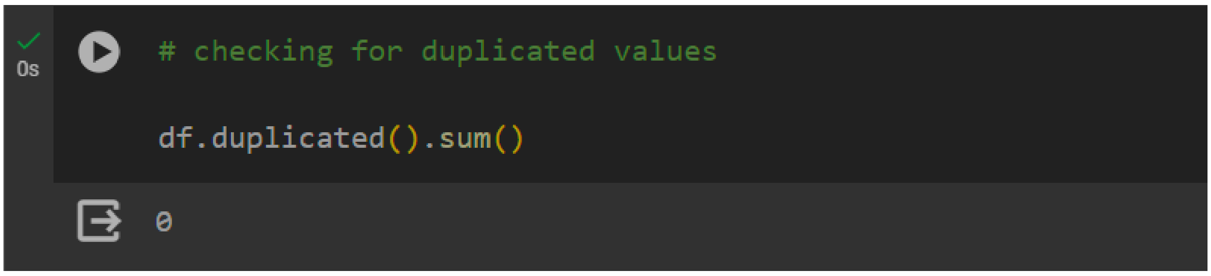
Data preprocessing and cleaning are vital steps in the classification process, particularly when dealing with news data from diverse sources. These techniques aim to eliminate noise and errors, preparing the data for training machine learning models effectively. During this phase, we remove punctuation marks, numbers, special characters, and empty tokens as they do not contribute to prediction.

- In data cleaning we will first check for null values, duplicated values, and outliers.



```
# checking for the null values
df.isnull().sum()
```

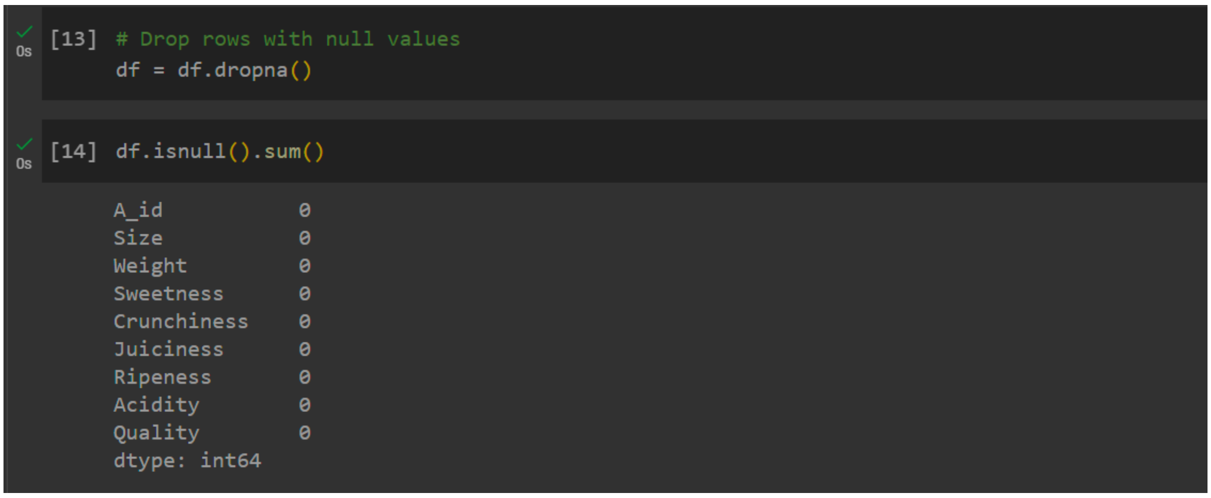
```
A_id           1
Size           1
Weight         1
Sweetness      1
Crunchiness    1
Juiciness      1
Ripeness       1
Acidity        0
Quality        1
dtype: int64
```



```
# checking for duplicated values

df.duplicated().sum()
```

```
0
```

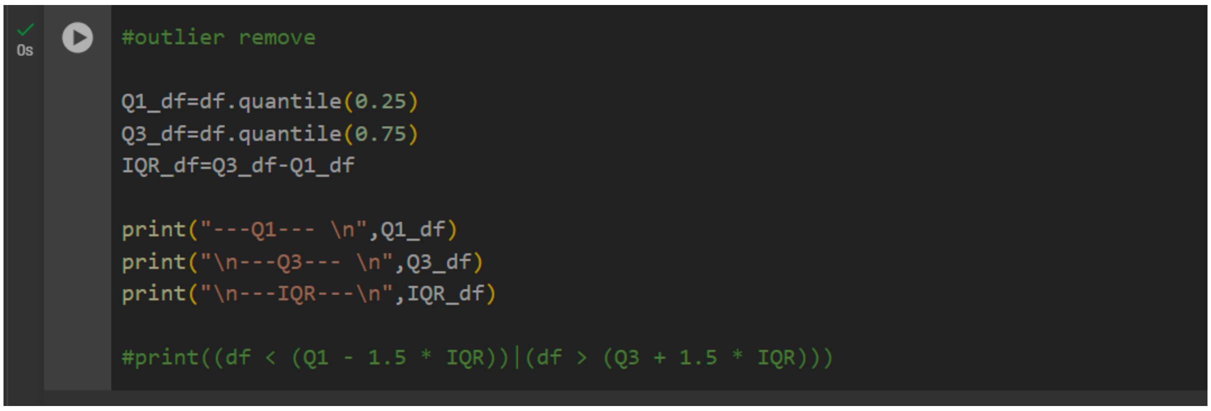
- After that we will remove rows of null, and duplicated values, and outliers if any.



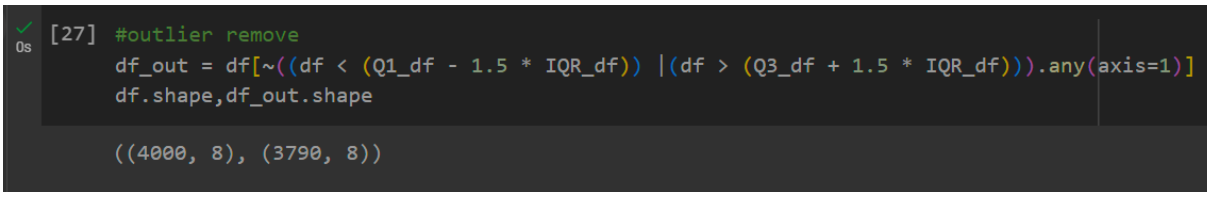
```
[13] # Drop rows with null values
     df = df.dropna()
```

```
[14] df.isnull().sum()
```

```
A_id           0
Size           0
Weight         0
Sweetness      0
Crunchiness    0
Juiciness      0
Ripeness       0
Acidity        0
Quality        0
dtype: int64
```



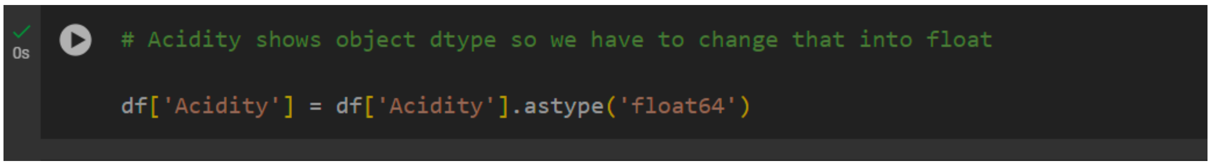
```
#outlier remove

Q1_df=df.quantile(0.25)
Q3_df=df.quantile(0.75)
IQR_df=Q3_df-Q1_df

print("---Q1--- \n",Q1_df)
print("\n---Q3--- \n",Q3_df)
print("\n---IQR---\n",IQR_df)

#print((df < (Q1 - 1.5 * IQR))|(df > (Q3 + 1.5 * IQR)))
```



```
[27] #outlier remove
     df_out = df[~((df < (Q1_df - 1.5 * IQR_df)) |(df > (Q3_df + 1.5 * IQR_df))).any(axis=1)]
     df.shape,df_out.shape
```

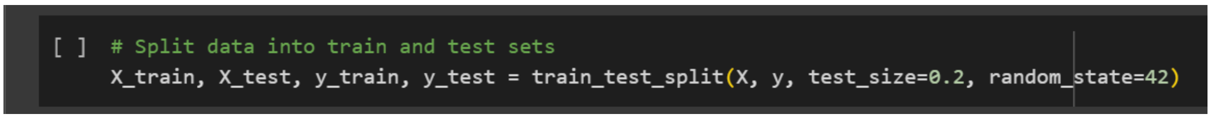
```
((4000, 8), (3790, 8))
```

- We will also convert data types into suitable data types to get better prediction, for example: object data type to float type.



```
# Acidity shows object dtype so we have to change that into float

df['Acidity'] = df['Acidity'].astype('float64')
```

- Splitting the Data**:** To ensure a reliable evaluation of the model's performance, we partitioned the dataset into training and testing subsets using the train_test_split function provided by Scikit-learn. By allocating 80% of the data to the training set, we effectively trained the model on a substantial portion of the dataset. The remaining 20% of the data constituted the testing set, enabling us to thoroughly assess the model's performance and its ability to generalize to unseen data.



```
[ ] # Split data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

1.7 Feature Extraction using TF-IDF

The feature extraction method utilized in this study is TF-IDF (Term Frequency-Inverse Document Frequency), a well-established approach in text analysis (Korkmaz et al., 2021). TF-IDF leverages two important metrics: term frequency (TF) and inverse document frequency (IDF). TF measures the frequency of each unique word within a document, while IDF assigns higher weights to words that are rare across all documents. This combination effectively adjusts the weights of common words downward and amplifies the importance of less frequent words in the feature vector. As a result, TF-IDF enhances the informativeness of the features, allowing the model to focus on the distinguishing characteristics of the text data. TF-

IDF is calculated using the formula provided by Anoop et al. (2019).

$$tf - idf(t,d) = tf(t,d) * idf(t) \quad \textbf{(1)}$$

where IDF (Inverse Document Frequency) is calculated using the formula:

$$idf = log\ (N / DF) \quad \textbf{(2)}$$

where N is the total number of documents in the corpus and DF is the number of documents that contain the specific word.

$$idf(t) = log\left[\frac{n}{df(t)}\right] + 1 \quad \textbf{(3)}$$

To incorporate both structural and syntactic information from the texts, we employed a diverse set of n-grams (Wynne & Wint, 2019), encompassing character in-grams and word in-grams. This approach enabled our model to capture valuable patterns and insights from the data.

TF-IDF Vectorizer (Term Frequency * Inverse Document Frequency)

a. Term Frequency: Term frequency (TF) is a metric that quantifies the frequency of a word's occurrence in a specific document, expressed as the number of times the word appears divided by the total number of words in that document.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}} \quad (4)$$

b. Inverse Document Frequency: Inverse Document Frequency (IDF) is computed by taking the logarithm of the ratio between the total number of documents in the corpus and the number of documents that contain a specific word "w." This measure assesses the significance of rare words present in the entire collection of

$$idf(w) = log(\frac{N}{df_t}) \quad (5)$$

documents.

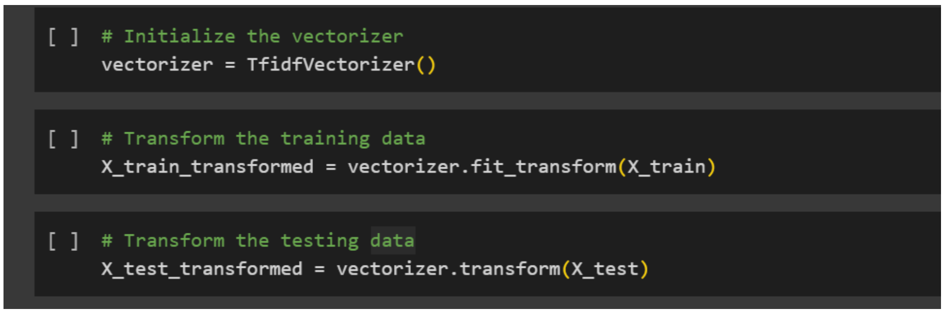
c. Now, TF-IDF Vectorizer:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right) \quad (6)$$

This vectorizer is formerly predefined in the scikit-learn library, thus we can import it using the following code:

```python
from sklearn.feature_extraction.text import TfidfVectorizer
```

Once we have imported the ‘TF-IDF Vectorizer’, we can use this vectorizer on our training dataset and transform its values on both the training and testing datasets. Now, the resulting feature vectors are represented as a sparse matrix, which can be converted into arrays for use with machine learning algorithms. This allows us to build an accurate and efficient fake news detection model leveraging the power of TF-IDF and n-gram features.

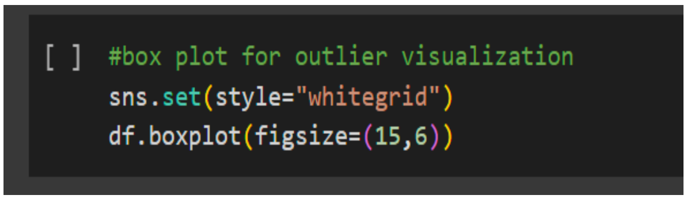


```python
[ ]  # Initialize the vectorizer
     vectorizer = TfidfVectorizer()

[ ]  # Transform the training data
     X_train_transformed = vectorizer.fit_transform(X_train)

[ ]  # Transform the testing data
     X_test_transformed = vectorizer.transform(X_test)
```

1.8 Data Visualization

Data visualization is the art and science of transforming data into graphical representations, such as charts, graphs, and maps. It's a powerful way to communicate complex information clearly and effectively, enabling viewers to grasp difficult concepts or identify new patterns quickly. With the aid of visualization tools, one can highlight trends, outliers, and correlations in data that might go unnoticed in text-based data. It's an essential skill in data science and analytics, helping to make data-driven decisions more accessible and understandable to a broader audience.

- **Boxplot:** In apple quality prediction, a boxplot is significant because it visually summarizes the distribution of data points, such as the size or sweetness of apples. It shows the median, quartiles, and outliers, which helps in understanding the variability and identifying any unusual observations.



```
[ ]  #box plot for outlier visualization
     sns.set(style="whitegrid")
     df.boxplot(figsize=(15,6))
```

This can be crucial for deciding quality grades and for spotting any factors that might affect the apple's quality. For instance, a boxplot can reveal if most apples have a similar size or if there are some significantly larger or smaller, which could indicate quality issues.

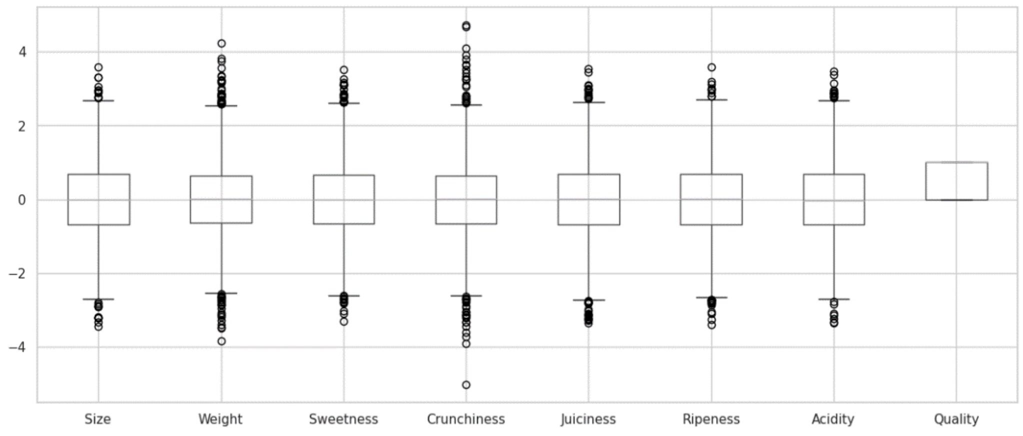
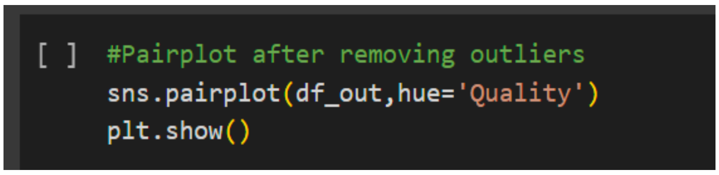


Fig 1.2: Box Plot of Apple Quality

- **Pairplot:** In apple quality prediction, a pairplot is significant as it visually represents the relationships between multiple variables at once. For example, it can show how the sweetness of an apple correlates with its size or how the weight might relate to its crunchiness. By displaying scatterplots for each variable pairing and histograms for individual variables, pairplots help in identifying patterns, trends, and outliers.



```
[ ]  #Pairplot after removing outliers
     sns.pairplot(df_out,hue='Quality')
     plt.show()
```

This is crucial for understanding which factors most significantly impact apple quality and for selecting the most relevant features for building predictive models. Pairplots thus play a vital role in exploratory data analysis, leading to more informed decisions in the quality assessment of apples.

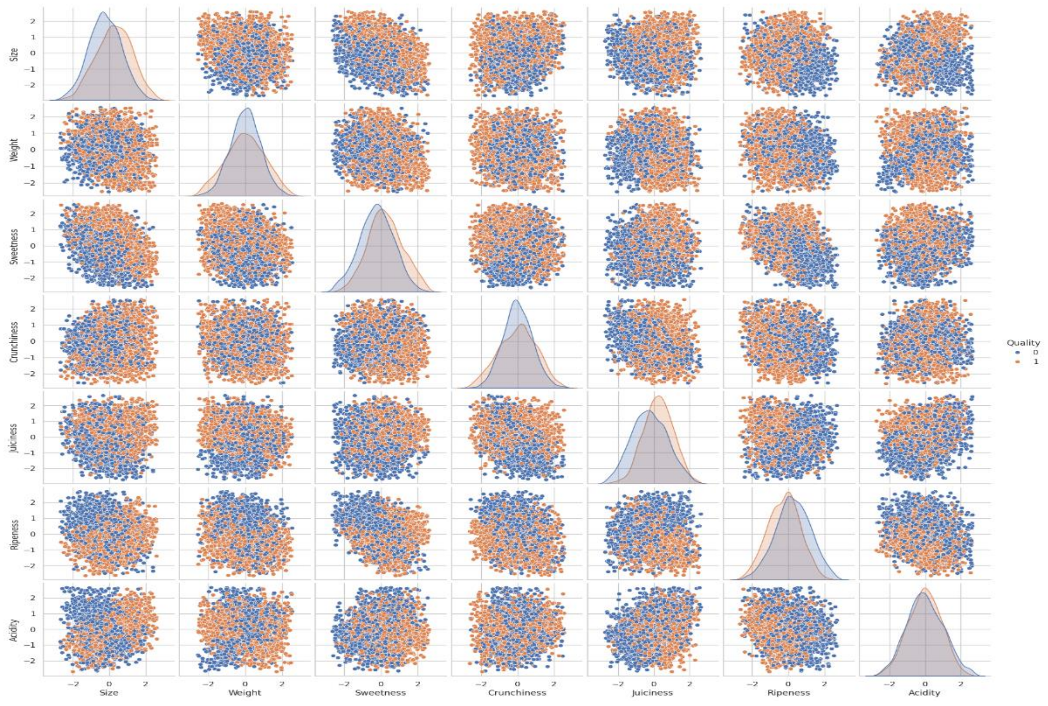
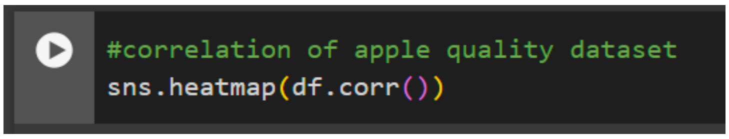


Fig 1.3: Pair Plot of Apple Quality Prediction

- **Heatmap:** In apple quality prediction, a heatmap is significant as it visually represents the strength of relationships between different quality attributes. For instance, it can illustrate how factors like sweetness, acidity, and crunchiness correlate with each other and with the overall quality rating.



```
#correlation of apple quality dataset
sns.heatmap(df.corr())
```

By using colour intensities, a heatmap can quickly reveal which attributes are most closely linked to high-quality apples, guiding growers and researchers in focusing on the traits that matter most for improving apple quality. This tool is invaluable for identifying key quality determinants and optimizing breeding and cultivation practices.

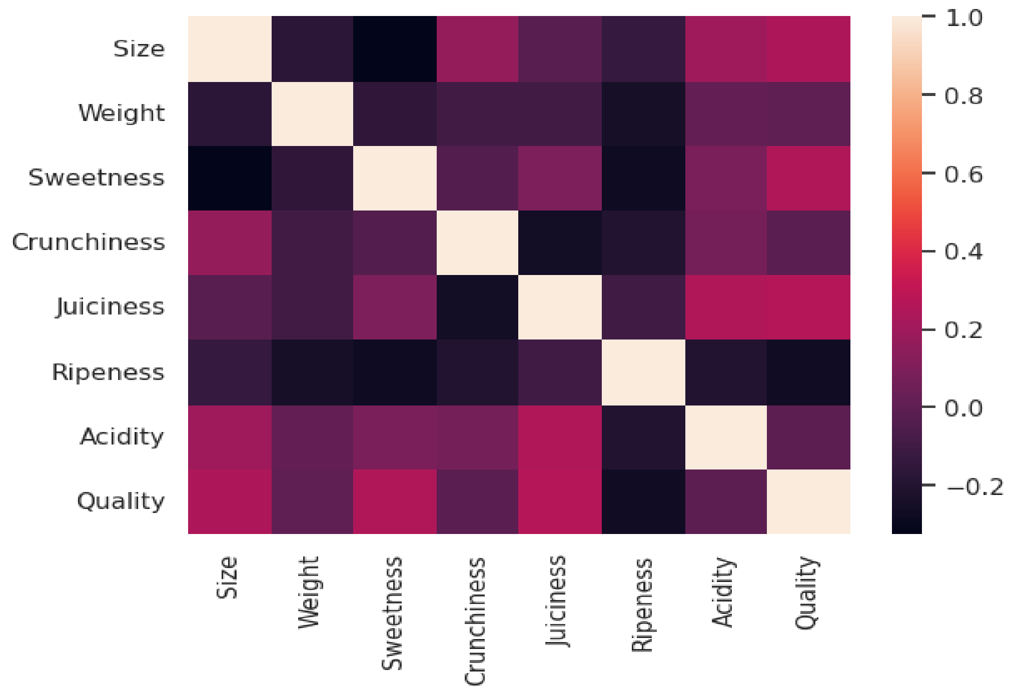
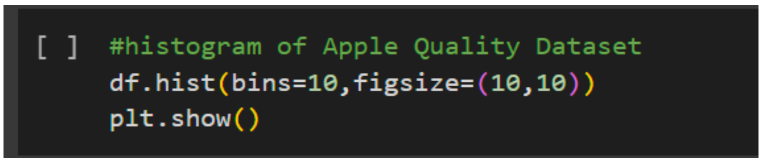


Fig 1.4: Heatmap of Apple Quality Prediction

- **Histogram:** In machine learning, a histogram is a visual chart that displays frequency distribution. It helps identify patterns and outliers by showing how often values occur in a dataset.



```
[ ] #histogram of Apple Quality Dataset
    df.hist(bins=10,figsize=(10,10))
    plt.show()
```

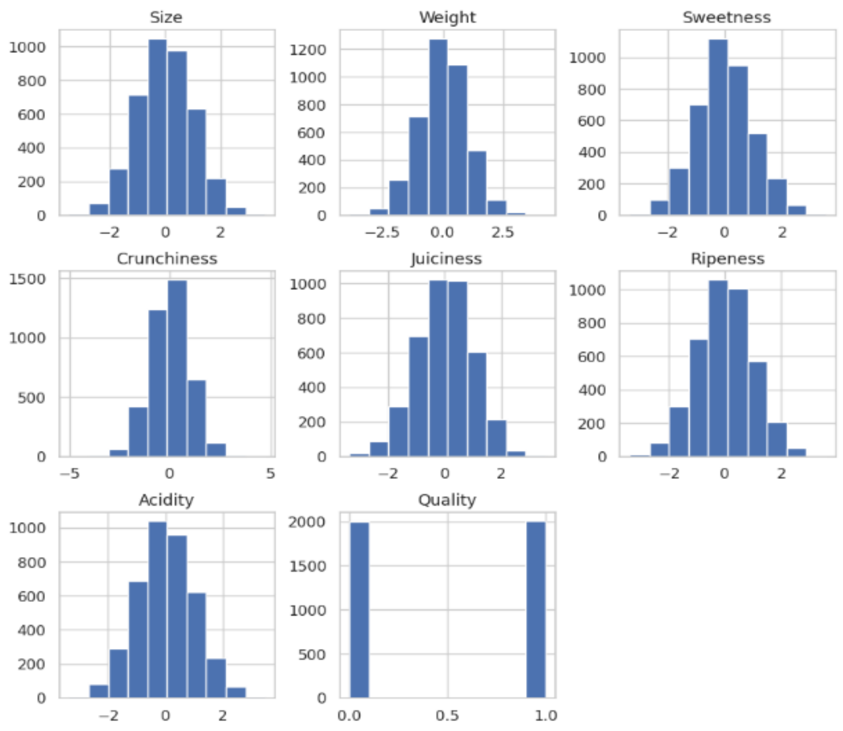
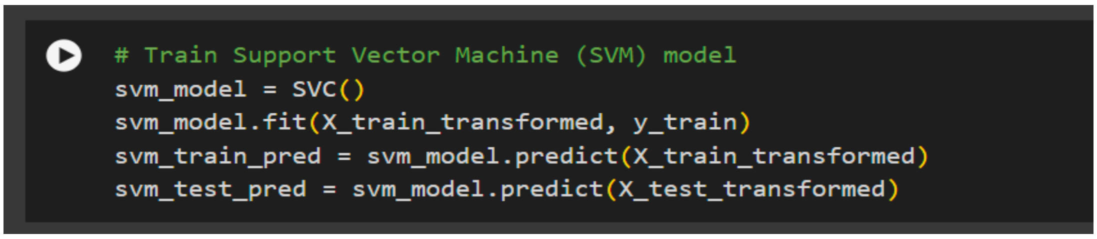


Fig 1.5: Histogram of Apple Quality

1.9 Applying Different Models

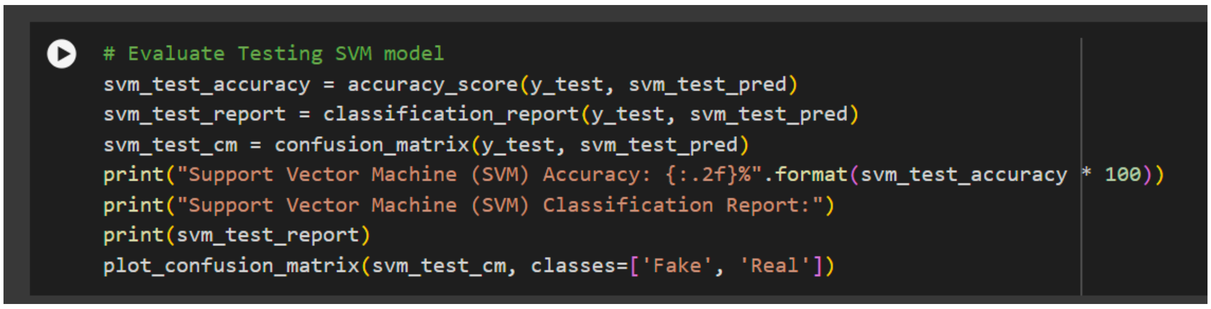
- **Support Vector Machine Model**

Support Vector Machine (SVM) is a powerful machine learning model used for classification and regression tasks. In the context of apple quality prediction, SVM can be particularly effective due to its ability to handle high-dimensional data and find the optimal boundary between different quality classes of apples.

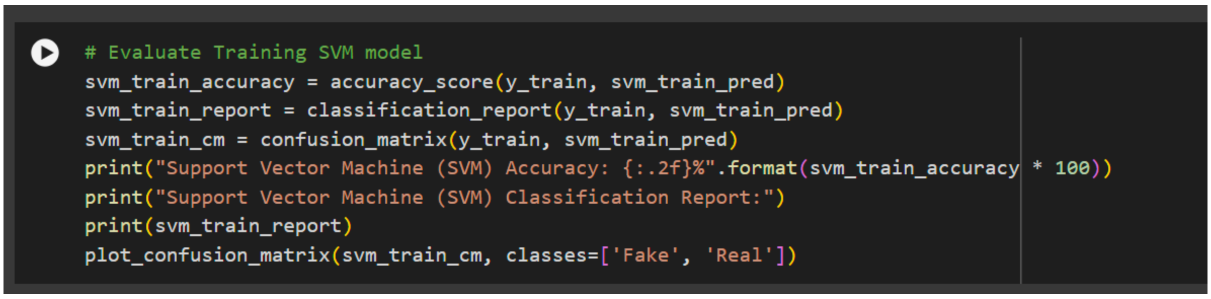


```
# Train Support Vector Machine (SVM) model
svm_model = SVC()
svm_model.fit(X_train_transformed, y_train)
svm_train_pred = svm_model.predict(X_train_transformed)
svm_test_pred = svm_model.predict(X_test_transformed)
```

By using features such as size, weight, colour, texture, and sugar content, SVM can learn from training data to classify apples into categories like ‘good’, ‘average’, or ‘poor’ quality. It does this by finding the hyperplane that best separates the different classes in the feature space. The model then uses this hyperplane to predict the quality of new, unseen apples based on their features. SVM is favoured in quality prediction tasks for its accuracy and robustness, especially when the data is not linearly separable.



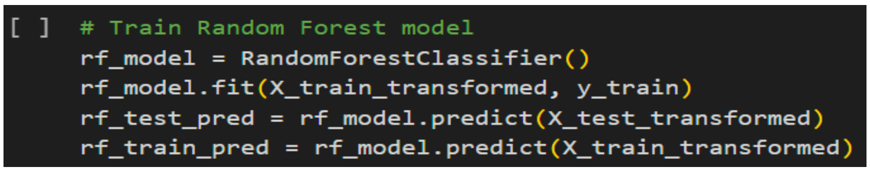
```python
# Evaluate Testing SVM model
svm_test_accuracy = accuracy_score(y_test, svm_test_pred)
svm_test_report = classification_report(y_test, svm_test_pred)
svm_test_cm = confusion_matrix(y_test, svm_test_pred)
print("Support Vector Machine (SVM) Accuracy: {:.2f}%".format(svm_test_accuracy * 100))
print("Support Vector Machine (SVM) Classification Report:")
print(svm_test_report)
plot_confusion_matrix(svm_test_cm, classes=['Fake', 'Real'])
```



```python
# Evaluate Training SVM model
svm_train_accuracy = accuracy_score(y_train, svm_train_pred)
svm_train_report = classification_report(y_train, svm_train_pred)
svm_train_cm = confusion_matrix(y_train, svm_train_pred)
print("Support Vector Machine (SVM) Accuracy: {:.2f}%".format(svm_train_accuracy * 100))
print("Support Vector Machine (SVM) Classification Report:")
print(svm_train_report)
plot_confusion_matrix(svm_train_cm, classes=['Fake', 'Real'])
```

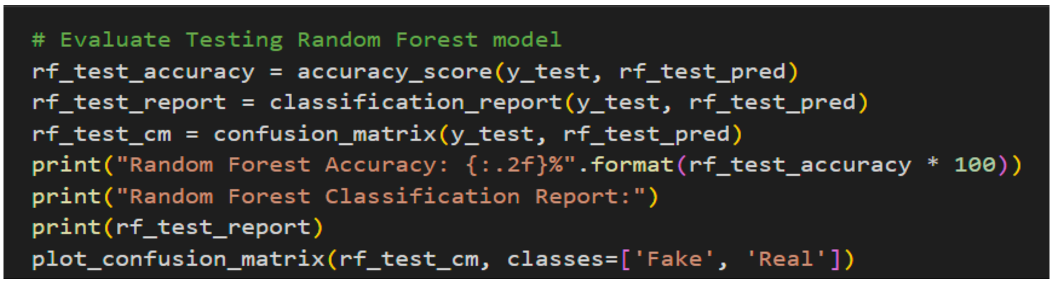
• Random Forest Model

Random Forest is an ensemble learning technique used for classification, regression, and other tasks.

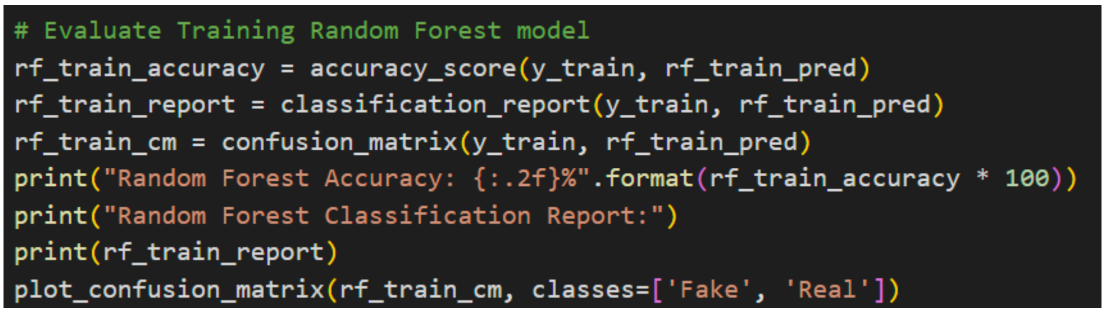


```python
# Train Random Forest model
rf_model = RandomForestClassifier()
rf_model.fit(X_train_transformed, y_train)
rf_test_pred = rf_model.predict(X_test_transformed)
rf_train_pred = rf_model.predict(X_train_transformed)
```

It builds multiple decision trees during training and combines their predictions to make the final prediction. Each tree is trained on a random subset of the data and features, reducing overfitting, making it ideal for classifying apples based on features like colour, size, and texture.



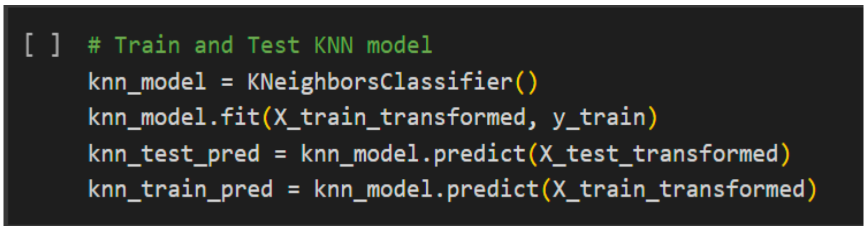
```python
# Evaluate Testing Random Forest model
rf_test_accuracy = accuracy_score(y_test, rf_test_pred)
rf_test_report = classification_report(y_test, rf_test_pred)
rf_test_cm = confusion_matrix(y_test, rf_test_pred)
print("Random Forest Accuracy: {:.2f}%".format(rf_test_accuracy * 100))
print("Random Forest Classification Report:")
print(rf_test_report)
plot_confusion_matrix(rf_test_cm, classes=['Fake', 'Real'])
```



```
# Evaluate Training Random Forest model
rf_train_accuracy = accuracy_score(y_train, rf_train_pred)
rf_train_report = classification_report(y_train, rf_train_pred)
rf_train_cm = confusion_matrix(y_train, rf_train_pred)
print("Random Forest Accuracy: {:.2f}%".format(rf_train_accuracy * 100))
print("Random Forest Classification Report:")
print(rf_train_report)
plot_confusion_matrix(rf_train_cm, classes=['Fake', 'Real'])
```

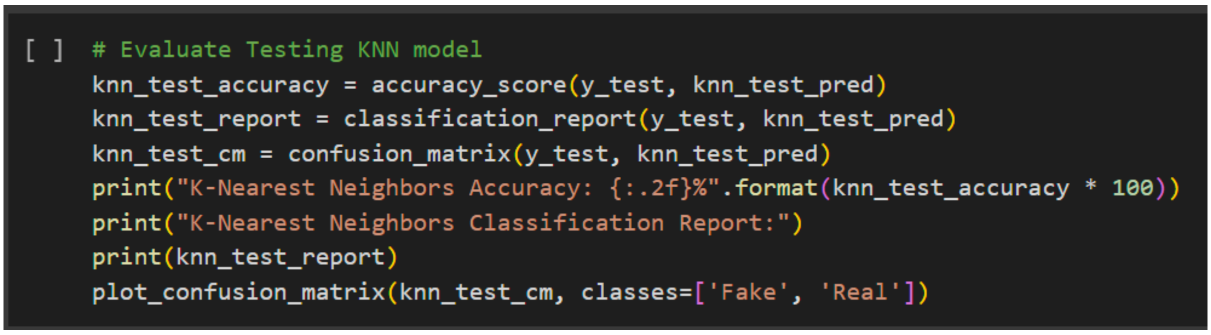
- **KNN Model**

K-Nearest Neighbor (KNN) is a machine learning algorithm used for classification and regression tasks. In the context of apple quality prediction, KNN can be employed to assess the quality of apples based on features like colour, size, and texture.



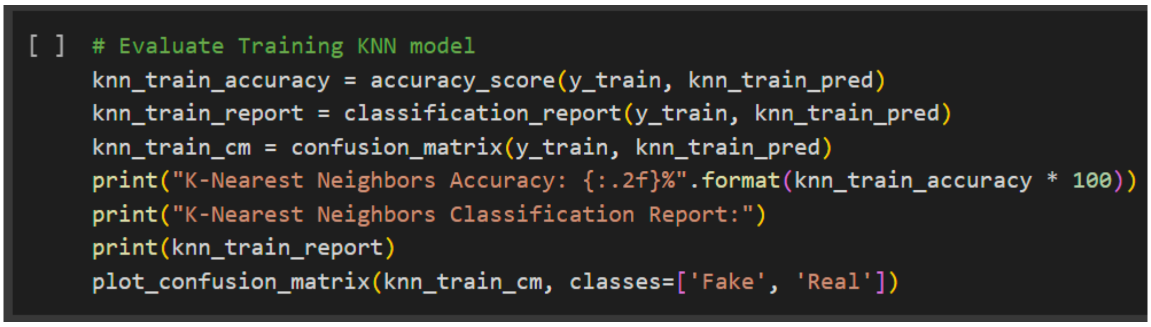
```
[ ]  # Train and Test KNN model
     knn_model = KNeighborsClassifier()
     knn_model.fit(X_train_transformed, y_train)
     knn_test_pred = knn_model.predict(X_test_transformed)
     knn_train_pred = knn_model.predict(X_train_transformed)
```

The algorithm learns from a labeled dataset of apples with known quality attributes. It calculates the distance between each apple’s features and its neighbors.



```
[ ]  # Evaluate Testing KNN model
     knn_test_accuracy = accuracy_score(y_test, knn_test_pred)
     knn_test_report = classification_report(y_test, knn_test_pred)
     knn_test_cm = confusion_matrix(y_test, knn_test_pred)
     print("K-Nearest Neighbors Accuracy: {:.2f}%".format(knn_test_accuracy * 100))
     print("K-Nearest Neighbors Classification Report:")
     print(knn_test_report)
     plot_confusion_matrix(knn_test_cm, classes=['Fake', 'Real'])
```

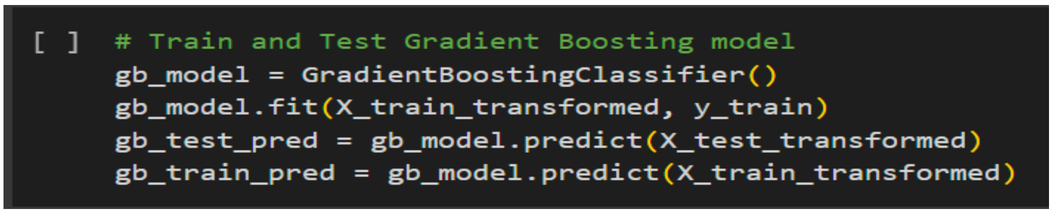
Given a new apple, KNN identifies its k-nearest neighbors (where k is a user-defined parameter). The majority class (for classification) or the average value (for regression) of these neighbors determines the predicted quality.



```
[ ]  # Evaluate Training KNN model
     knn_train_accuracy = accuracy_score(y_train, knn_train_pred)
     knn_train_report = classification_report(y_train, knn_train_pred)
     knn_train_cm = confusion_matrix(y_train, knn_train_pred)
     print("K-Nearest Neighbors Accuracy: {:.2f}%".format(knn_train_accuracy * 100))
     print("K-Nearest Neighbors Classification Report:")
     print(knn_train_report)
     plot_confusion_matrix(knn_train_cm, classes=['Fake', 'Real'])
```

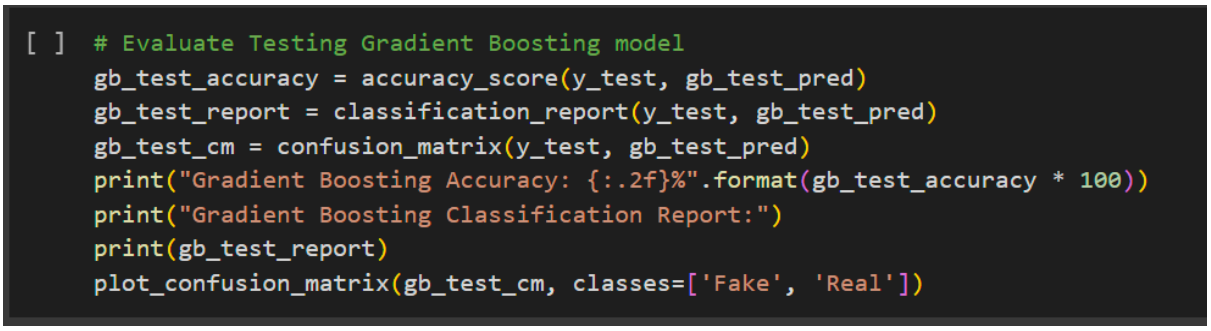
• Gradient Boosting Model

Gradient Boosting is an ensemble learning method that leverages the collective strength of multiple weak learners, usually decision trees, to construct a robust predictive model. The technique constructs the model in a sequential manner, with each new learner focused on rectifying the errors made by its predecessors.

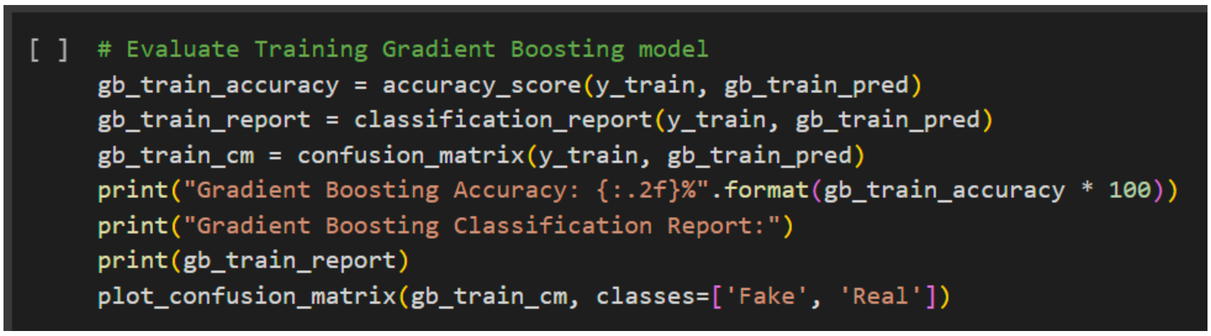


```
[ ]  # Train and Test Gradient Boosting model
     gb_model = GradientBoostingClassifier()
     gb_model.fit(X_train_transformed, y_train)
     gb_test_pred = gb_model.predict(X_test_transformed)
     gb_train_pred = gb_model.predict(X_train_transformed)
```

Utilizing gradient descent, Gradient Boosting efficiently minimizes the loss function, leading to improved model performance. Its versatility makes it suitable for handling complex data and finding applications in various machine learning tasks, including regression and classification.



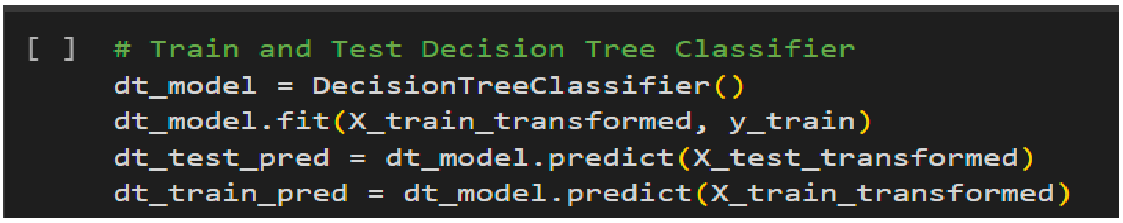
```
[ ]  # Evaluate Testing Gradient Boosting model
     gb_test_accuracy = accuracy_score(y_test, gb_test_pred)
     gb_test_report = classification_report(y_test, gb_test_pred)
     gb_test_cm = confusion_matrix(y_test, gb_test_pred)
     print("Gradient Boosting Accuracy: {:.2f}%".format(gb_test_accuracy * 100))
     print("Gradient Boosting Classification Report:")
     print(gb_test_report)
     plot_confusion_matrix(gb_test_cm, classes=['Fake', 'Real'])
```



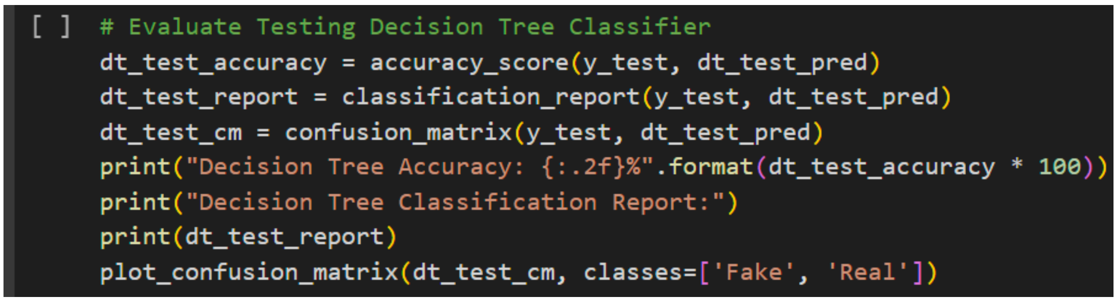
```
[ ]  # Evaluate Training Gradient Boosting model
     gb_train_accuracy = accuracy_score(y_train, gb_train_pred)
     gb_train_report = classification_report(y_train, gb_train_pred)
     gb_train_cm = confusion_matrix(y_train, gb_train_pred)
     print("Gradient Boosting Accuracy: {:.2f}%".format(gb_train_accuracy * 100))
     print("Gradient Boosting Classification Report:")
     print(gb_train_report)
     plot_confusion_matrix(gb_train_cm, classes=['Fake', 'Real'])
```

- **Decision Tree Classifier**

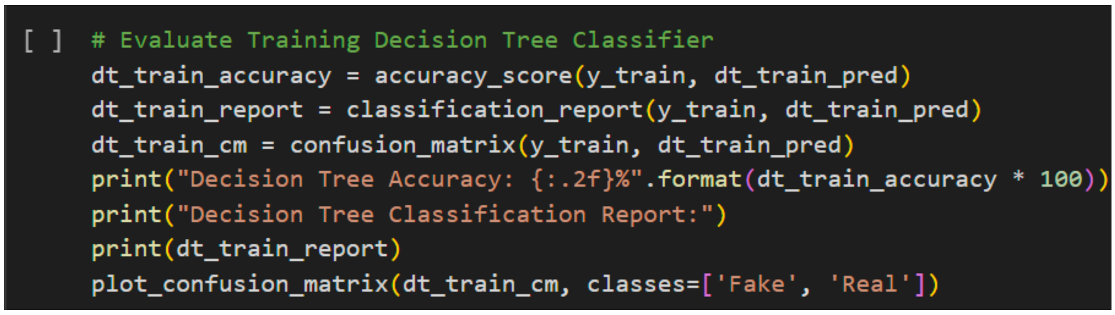
Decision tree is a popular machine learning algorithm for classification and regression tasks. It recursively splits the data based on features to create a tree-like structure, making decisions at each node. It's interpretable, handles non-linear relationships, and can handle both numerical and categorical data.



```
[ ]  # Train and Test Decision Tree Classifier
     dt_model = DecisionTreeClassifier()
     dt_model.fit(X_train_transformed, y_train)
     dt_test_pred = dt_model.predict(X_test_transformed)
     dt_train_pred = dt_model.predict(X_train_transformed)
```



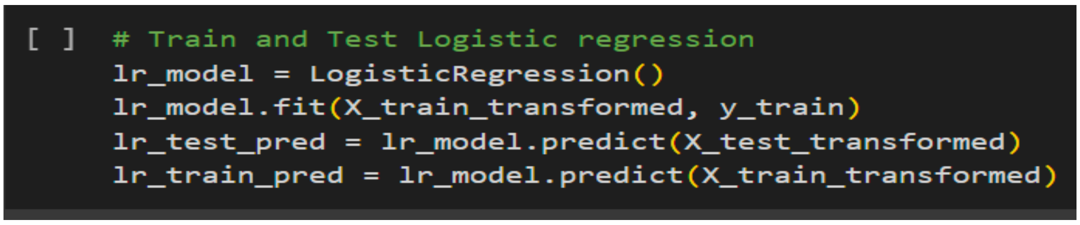
```
[ ]  # Evaluate Testing Decision Tree Classifier
     dt_test_accuracy = accuracy_score(y_test, dt_test_pred)
     dt_test_report = classification_report(y_test, dt_test_pred)
     dt_test_cm = confusion_matrix(y_test, dt_test_pred)
     print("Decision Tree Accuracy: {:.2f}%".format(dt_test_accuracy * 100))
     print("Decision Tree Classification Report:")
     print(dt_test_report)
     plot_confusion_matrix(dt_test_cm, classes=['Fake', 'Real'])
```



```
[ ]  # Evaluate Training Decision Tree Classifier
     dt_train_accuracy = accuracy_score(y_train, dt_train_pred)
     dt_train_report = classification_report(y_train, dt_train_pred)
     dt_train_cm = confusion_matrix(y_train, dt_train_pred)
     print("Decision Tree Accuracy: {:.2f}%".format(dt_train_accuracy * 100))
     print("Decision Tree Classification Report:")
     print(dt_train_report)
     plot_confusion_matrix(dt_train_cm, classes=['Fake', 'Real'])
```

- **Logistic Regression**

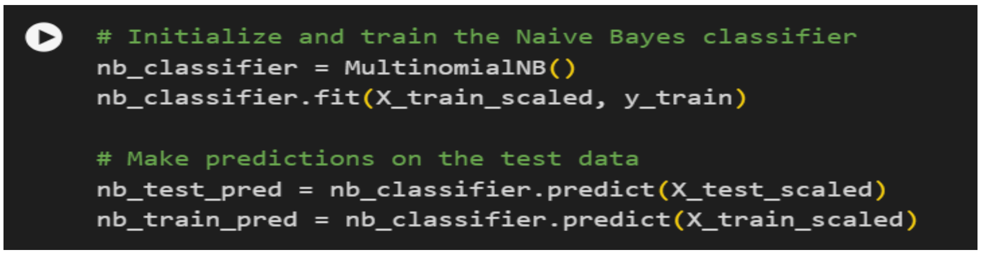
Logistic regression is a statistical and machine learning algorithm used for binary classification tasks. It establishes the connection between a dependent binary variable (target) and one or more independent variables (features) using the logistic function. The output is a probability score that predicts the likelihood of the target belonging to a specific class.



```
[ ]  # Train and Test Logistic regression
     lr_model = LogisticRegression()
     lr_model.fit(X_train_transformed, y_train)
     lr_test_pred = lr_model.predict(X_test_transformed)
     lr_train_pred = lr_model.predict(X_train_transformed)
```

- **Naive Bayes Classifier**

Naive Bayes is a probabilistic classifier based on Bayes' theorem. It assumes independence between features, making computations efficient.

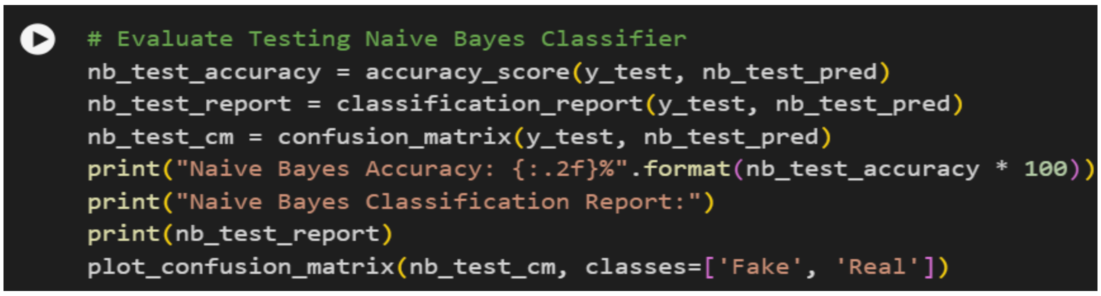


```
# Initialize and train the Naive Bayes classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_scaled, y_train)

# Make predictions on the test data
nb_test_pred = nb_classifier.predict(X_test_scaled)
nb_train_pred = nb_classifier.predict(X_train_scaled)
```

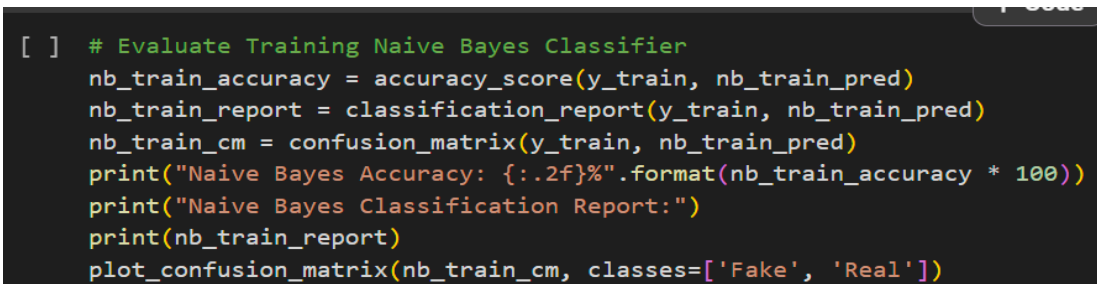
Bayes' theorem offers a method to calculate the posterior probability, denoted as P(c|x), by utilizing the prior probability P(c), the probability of the input data P(x), and the conditional probability P(x|c). In the context of the Naive Bayes classifier, it assumes that the influence of a predictor's value (x) on a particular class (c) is independent of the values of other predictors. This assumption is known as class conditional independence, which simplifies the modelling process and facilitates efficient computation in various machine learning tasks.

$$\underset{\text{Posterior Probability}}{P(c\mid x)} = \frac{\overset{\text{Likelihood}}{P(x\mid c)}\,\overset{\text{Class Prior Probability}}{P(c)}}{\underset{\text{Predictor Prior Probability}}{P(x)}} \qquad (7)$$

$$P(c\mid X) = P(x_1\mid c)\times P(x_2\mid c)\times\cdots\times P(x_n\mid c)\times P(c)$$



```
# Evaluate Testing Naive Bayes Classifier
nb_test_accuracy = accuracy_score(y_test, nb_test_pred)
nb_test_report = classification_report(y_test, nb_test_pred)
nb_test_cm = confusion_matrix(y_test, nb_test_pred)
print("Naive Bayes Accuracy: {:.2f}%".format(nb_test_accuracy * 100))
print("Naive Bayes Classification Report:")
print(nb_test_report)
plot_confusion_matrix(nb_test_cm, classes=['Fake', 'Real'])
```



```
[ ] # Evaluate Training Naive Bayes Classifier
nb_train_accuracy = accuracy_score(y_train, nb_train_pred)
nb_train_report = classification_report(y_train, nb_train_pred)
nb_train_cm = confusion_matrix(y_train, nb_train_pred)
print("Naive Bayes Accuracy: {:.2f}%".format(nb_train_accuracy * 100))
print("Naive Bayes Classification Report:")
print(nb_train_report)
plot_confusion_matrix(nb_train_cm, classes=['Fake', 'Real'])
```

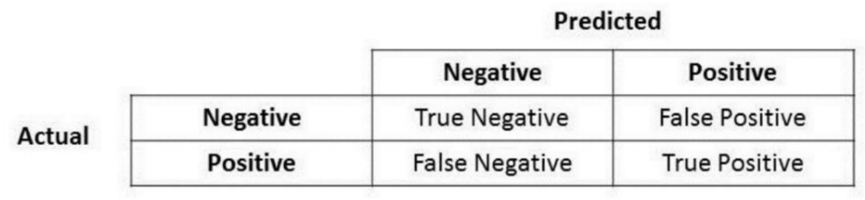
1.10 Assessing Classification Metrices

To evaluate your model's performance and measure its accuracy, you can leverage several classification metrices provided by scikit-learn. Commonly used metrics includes:

1. Confusion Matrix
2. Accuracy Score
3. Precision
4. Recall
5. F1-Score

These metrics provide valuable insights into how well your model is making predictions and its ability to correctly identify positive and negative instances in the data.

- **Confusion Matrix:** These metrics assess the performance of the model by examining both the correct and incorrect predictions it makes. They provide a comprehensive view of how well the model classifies instances and helps to identify true positives, true negatives, false positives, and false negatives.



		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

- **Accuracy Score:** It is the ratio of correct predictions to the total number of predictions made by the model.

$$\text{Accuracy} = (TP + TN) \,/\, (TP + TN + FP + FN) \tag{8}$$

- **Precision:** In situations where the exactness of the model's positive predictions is crucial, Precision is the preferred metric. Precision reveals the proportion of correctly labelled positive instances compared to the total instances that the classifier predicted as positive. It helps us assess the accuracy of positive predictions and is valuable in scenarios where minimizing false positives is essential.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive} = \frac{True\ Positive}{Total\ Predicted\ Positive} \quad \text{(9)}$$

- **Recall:** Recall is a metric that evaluates the model's ability to correctly identify instances belonging to the positive class. In other words, it measures the proportion of actual positive labels that the model correctly identified as positive.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative} = \frac{True\ Positive}{Total\ Actual\ Positive} \quad \text{(10)}$$

- **F1-Score:** Accuracy serves as a prominent evaluation metric in machine learning, offering a comprehensive measure of a model's overall performance. By considering both precision and recall scores, accuracy assesses the number of correct predictions made by the model across the entire dataset.

$$\text{F1} = 2 \times \frac{Precision * Recall}{Precision + Recall} \quad \text{(11)}$$

1.11 Classification Metrics After Feature Extraction

After feature extraction, we present the classification matrices for Apple Quality Prediction datasets. These matrices provide a comprehensive overview of the model's performance in each dataset, including true positive, true negative, false positive, and false negative counts.

a) Classification Metrics of Testing Data

Classification Metrics With TF-IDF Vectorizer					
		APPLE QUALITY PREDICTION			
S. No	ML Models	Accuracy	Precision	Recall	F1-Score
1.	**Support Vector Machine (SVM)**	**88.39**	**0.86**	**0.91**	**0.89**
2.	**Random Forest**	**87.86**	**0.86**	**0.90**	**0.88**
3.	**K-Nearest Neighbors (KNN)**	**88.52**	**0.86**	**0.92**	**0.89**
4.	**Gradient Boosting**	**84.04**	**0.82**	**0.86**	**0.84**
5.	**Decision Tree**	**78.63**	**0.78**	**0.79**	**0.78**
6.	**Logistic Regression**	**73.09**	**0.72**	**0.75**	**0.73**
7.	**Naive Bayes**	**65.44**	**0.75**	**0.45**	**0.56**

Table 1.1: Classification report of Testing Data of Apple Quality Prediction

In Testing data of Apple quality Prediction Dataset, when employing the TF-IDF vectorizer for feature extraction, the K-Nearest Neighbors model achieved an impressive accuracy of 88.52%. Moreover, it demonstrated recall, precision, and F1-score, with a value of 0.86, 0.92 and 0.89 respectively, indicating excellent performance in correctly classifying positive instances. Subsequently, the Support Vector Machine (SVM) also showed promising results with accuracy of 88.39%, making it the second-best performing model in the evaluation.

b) Classification Metrics of Training Data

Classification Metrics With TF-IDF Vectorizer					
APPLE QUALITY PREDICTION					
S. No	ML Models	Accuracy	Precision	Recall	F1-Score
1.	**Support Vector Machine (SVM)**	**90.60**	**0.90**	**0.91**	**0.91**
2.	**Random Forest**	**100**	**1**	**1**	**1**
3.	**K-Nearest Neighbors (KNN)**	**93.24**	**0.93**	**0.94**	**0.93**
4.	**Gradient Boosting**	**90.50**	**0.89**	**0.92**	**0.90**
5.	**Decision Tree**	**100**	**1**	**1**	**1**
6.	**Logistic Regression**	**74.44**	**0.74**	**0.75**	**0.74**
7.	**Naive Bayes**	**65.11**	**0.75**	**0.44**	**0.55**

Table 1.2: Classification report of Training Data of Apple Quality Prediction

In Training data of Apple quality Prediction Dataset, when employing the TF-IDF vectorizer for feature extraction, the Random Forest model and Decision Tree Classifier achieved an impressive accuracy of 100%. Moreover, it demonstrated perfect recall, precision, and F1-score, all with a value of 1, indicating excellent performance in correctly classifying positive instances. Subsequently, the K-Nearest Neighbors (KNN) also showed promising results with accuracy of 93.24%, making it the second-best performing model in the evaluation.

1.12 Accuracy Bar Graph of Apple Quality Prediction

In machine learning projects, a Bar Graph is a visual tool used to compare different groups or categories. It represents data with rectangular bars, where the length of each bar is proportional to the value it signifies. Bar graphs are particularly effective for showcasing the distribution of categorical data and highlighting significant patterns.

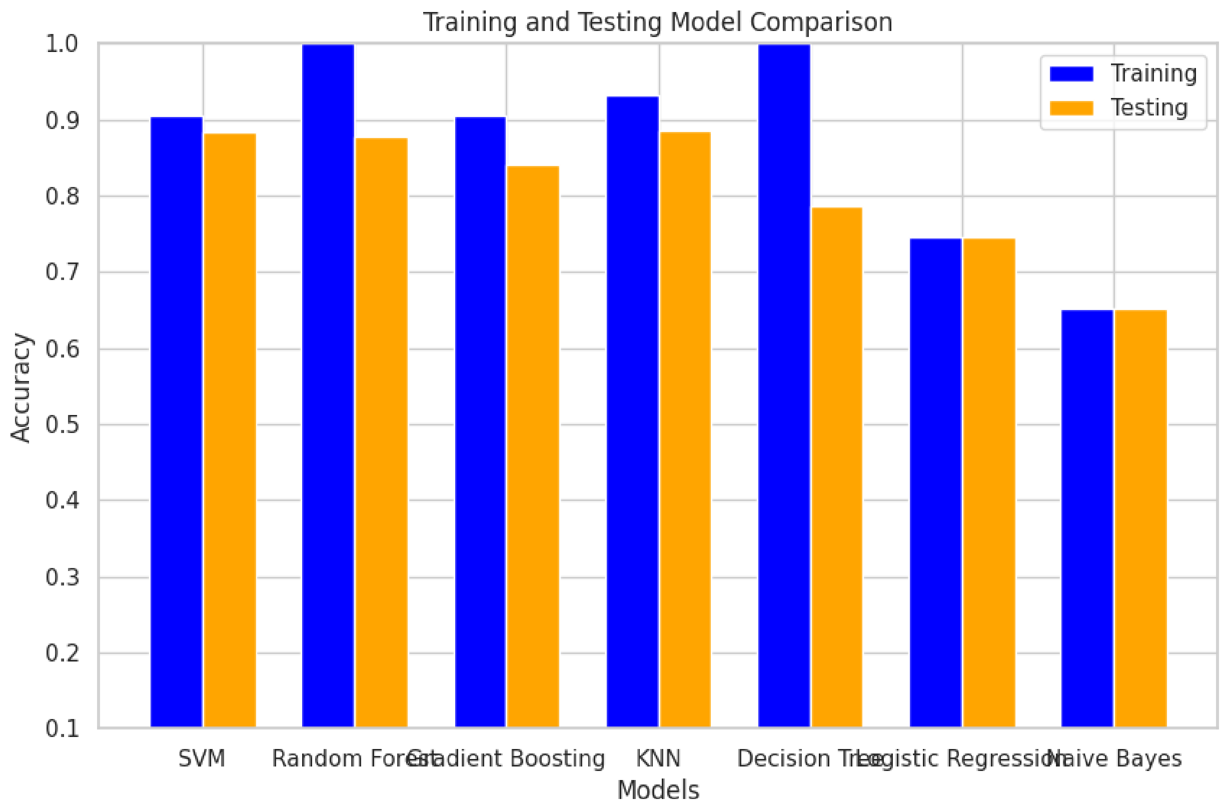
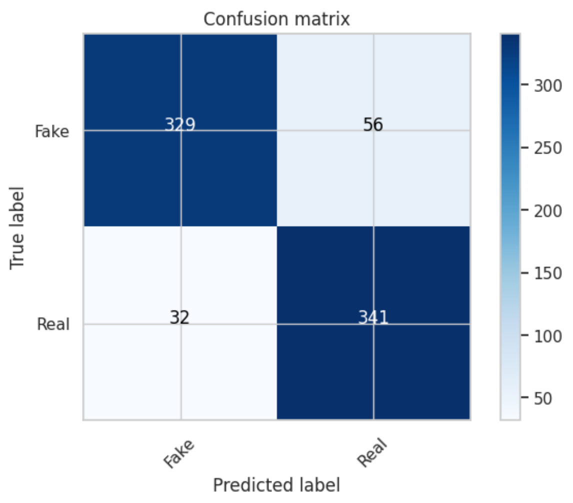


Fig 1.6: Accuracy Comparison of Testing and Training Data of Apple Quality

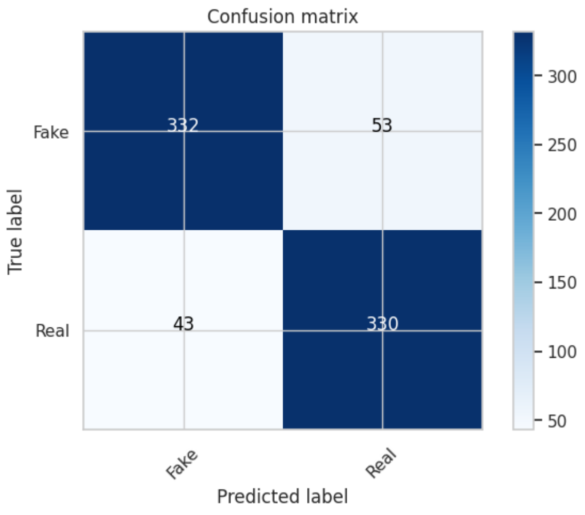
Here, a single bar graph effectively displays the accuracy of both the testing data and training data. This visual comparison aids in assessing the model's performance across different datasets.

1.13 Confusion Matrix for Each Model

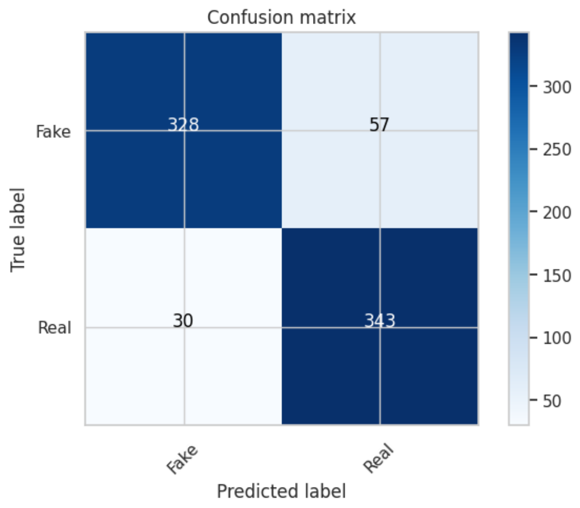
a) For Testing Models



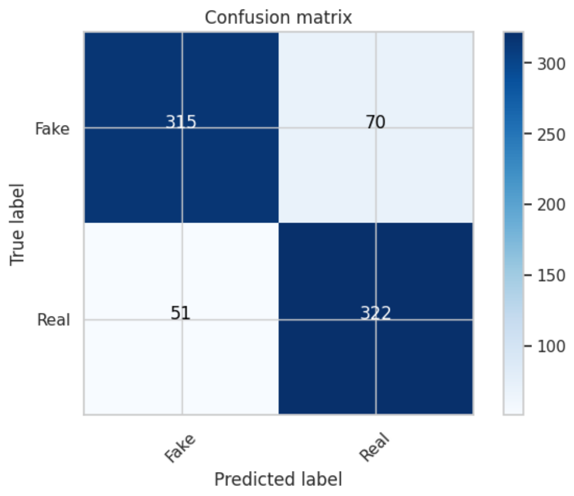
Model: SVM



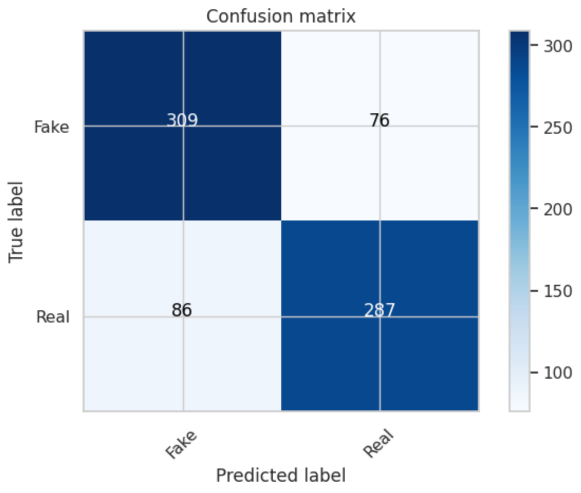
Model: Random Forest



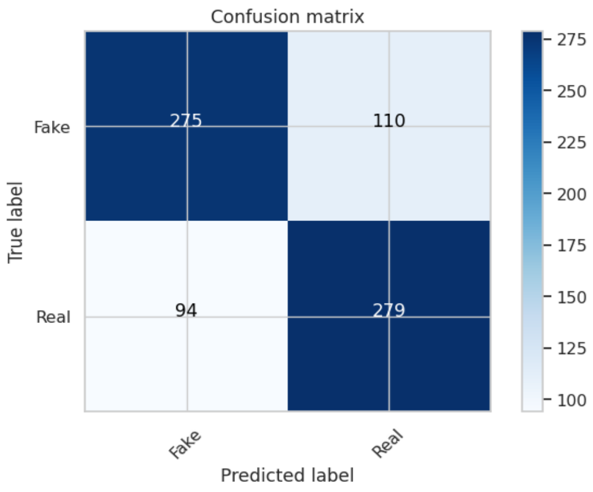
Model: KNN



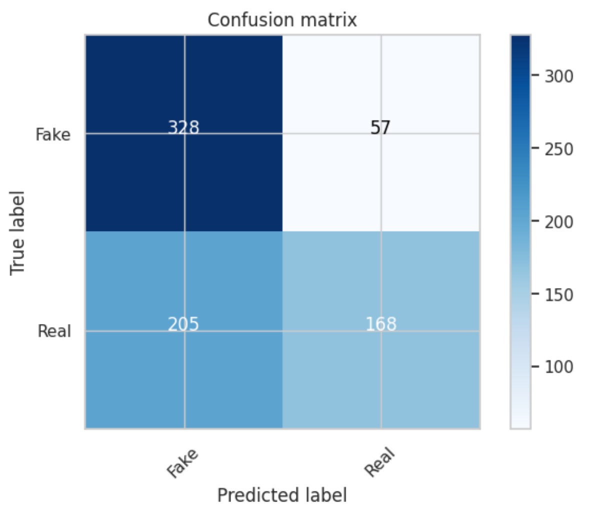
Model: Gradient Boosting



Model: Decision Tree



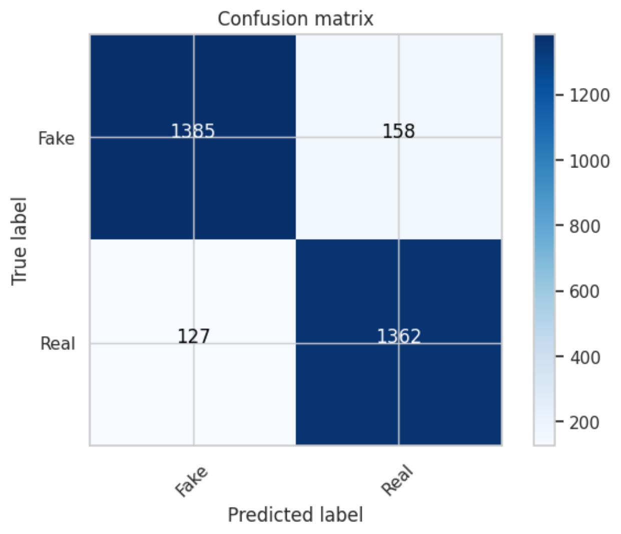
Model: Logistic Regression



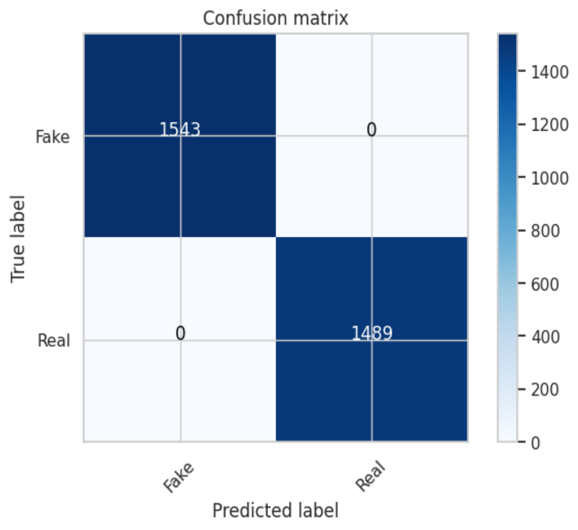
Model: Naïve Bayes

Fig 1.7: Confusion Matrix of Testing Data of Apple Quality Prediction

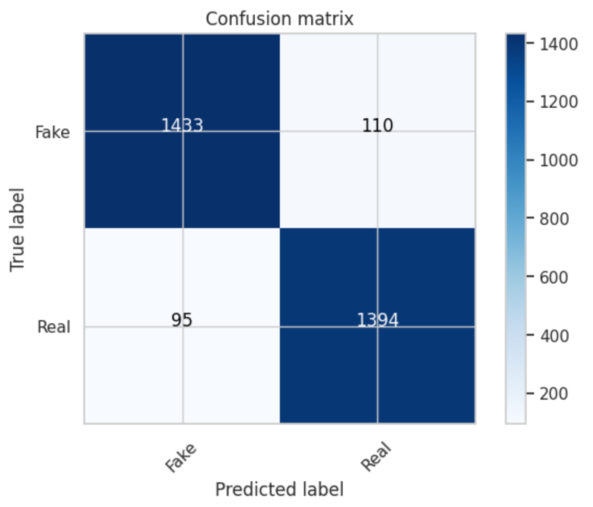
b) For Training Models



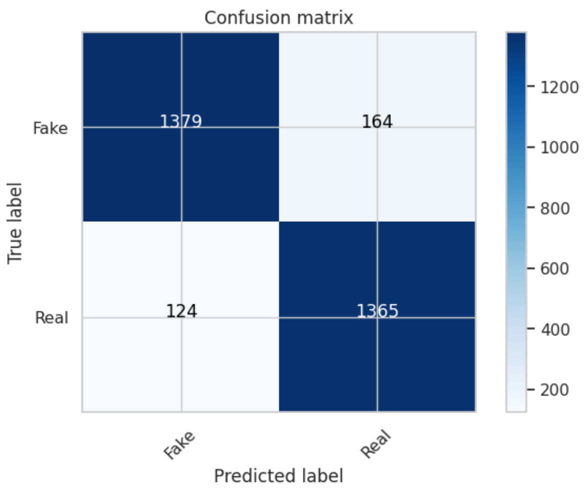
Model: SVM



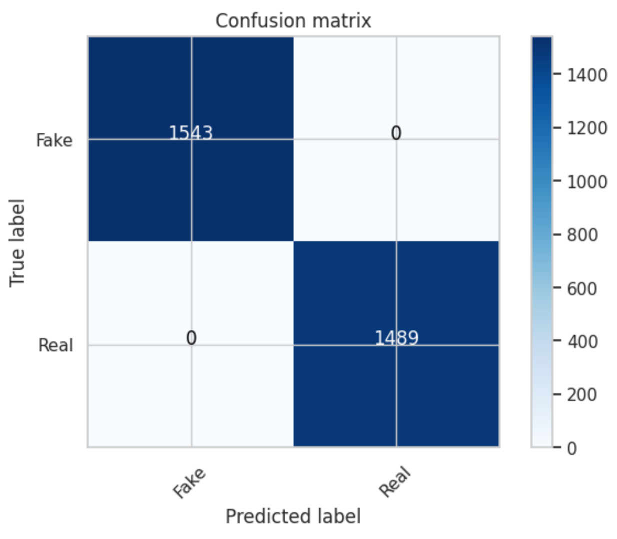
Model: Random Forest



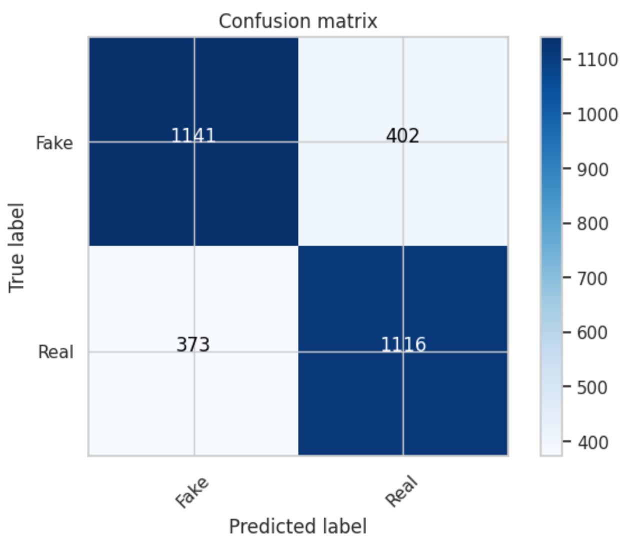
Model: KNN



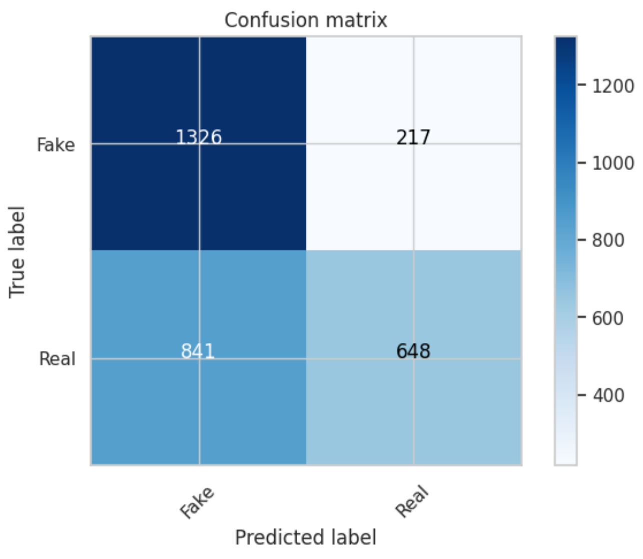
Model: Gradient Boosting



Model: Decision Tree



Model: Logistic regression



Model: Naïve Bayes

Fig 1.8: Confusion Matrix of Training Data of Apple Quality Prediction

1.14 AUC-ROC & F1-Score Curve

- **AUC-ROC CURVE**

The AUC-ROC curve is significant in machine learning for apple quality prediction as it measures a model’s ability to distinguish between good and bad quality apples. A higher AUC indicates better model performance.

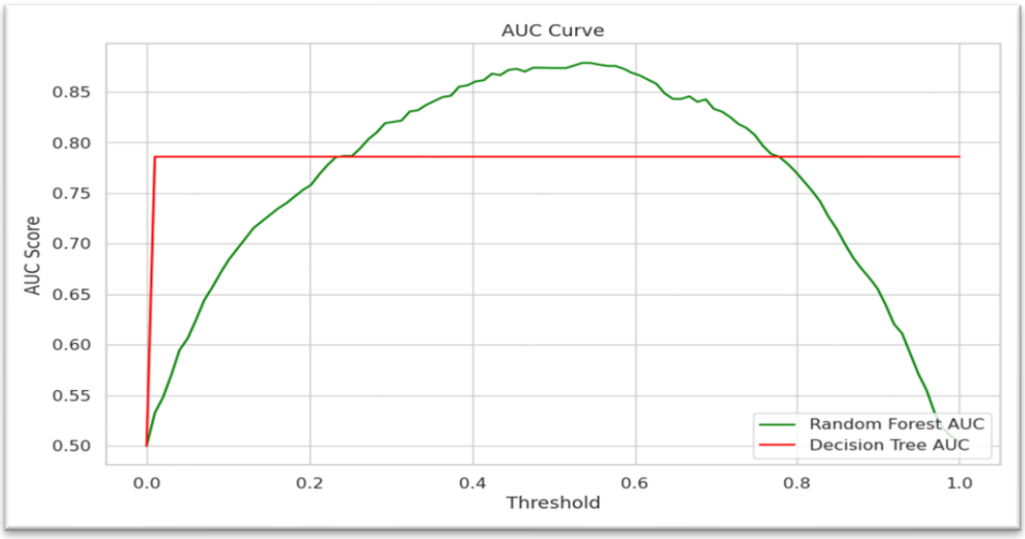


Fig 1.9: AUC-ROC Curve of Apple Quality Prediction

- **F1-SCORE CURVE**

The F1-Score is a balanced measure of a model's precision and recall, crucial for evaluating apple quality predictions, particularly in imbalanced datasets.

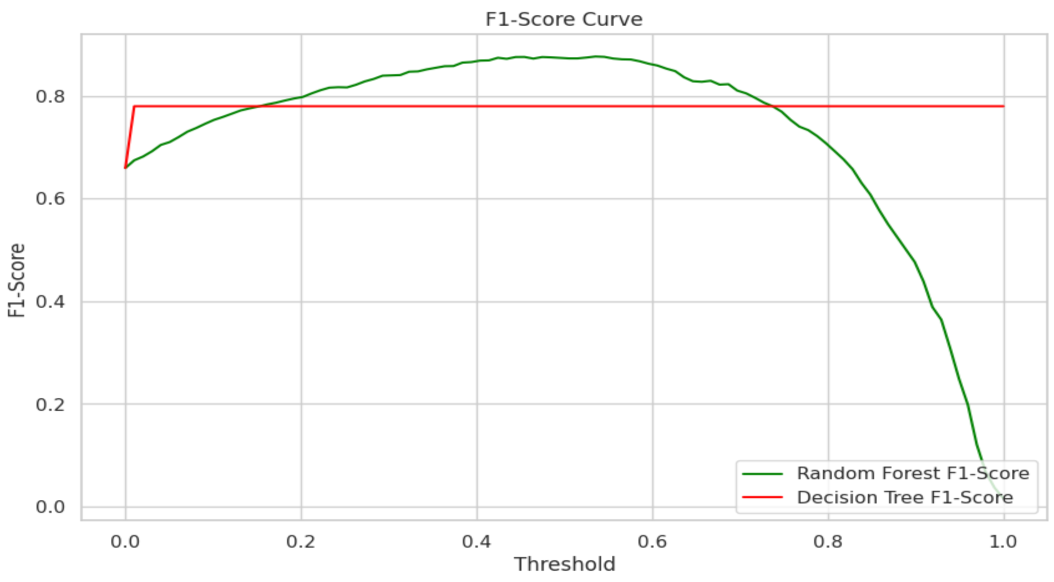


Fig 1.10: F1-Score Curve of Apple Quality Prediction

1.15 Conclusion

In our college major project, we explored the application of machine learning algorithms for predicting the quality of apples. The project's cornerstone was the comparative analysis of various models to determine the most effective approach for this task. The K-Nearest Neighbors (KNN) algorithm emerged as the top performer on the testing data, achieving an impressive accuracy of 88.52%. This result underscores KNN's capability to generalize well to new, unseen data, making it a reliable choice for practical applications.

On the training data, both the Random Forest and Decision Tree models achieved perfect accuracy scores of 100%. These results highlight the models' ability to capture the nuances of the dataset fully. However, the true test of a model's utility lies in its performance on unseen data, which is where KNN excelled.

The project also included an in-depth analysis of each model's performance through confusion matrices. These matrices provided valuable insights into the true positives, false positives, true negatives, and false negatives, allowing us to fine-tune our models further.

Additionally, we derived ROC-AUC and F1-Score curves for the Decision Tree and Random Forest models. The ROC-AUC curve served as a testament to the models' discriminative power, while the F1-Score curve offered a harmonized view of precision and recall, ensuring that our models not only identified high-quality apples accurately but also minimized the misclassification of poor-quality ones.

Overall, this project has demonstrated the potential of machine learning in revolutionizing the apple quality assessment process. By leveraging these algorithms, we can aim for a future where quality control is not only automated but also more accurate and efficient, benefiting both producers and consumers in the agricultural industry.

1.16 Future Work

In the realm of apple quality prediction, there are numerous avenues for future exploration and enhancement. Firstly, expanding the dataset with a wider variety of apple samples from different regions, seasons, and cultivation practices can enhance model generalization and robustness. This could involve collaboration with agricultural organizations or orchards to collect diverse and representative samples.

Additionally, integrating advanced sensing technologies such as spectroscopy or hyperspectral imaging can provide more detailed and comprehensive information about apple quality attributes. These technologies can capture subtle variations in color, texture, and chemical composition, enabling more accurate quality assessments.

Furthermore, exploring advanced machine learning techniques such as deep learning models or ensemble methods can improve predictive accuracy and reliability. These approaches can effectively leverage the complex relationships between various quality attributes and enhance the predictive capabilities of the models.

Moreover, incorporating real-time environmental data such as weather conditions, soil moisture levels, and pest infestation rates can enhance the models' ability to adapt to changing growing conditions and predict quality fluctuations accurately.

Lastly, deploying the developed models in agricultural settings and conducting extensive field testing and validation is essential to assess their effectiveness and practicality. Continuous monitoring and refinement based on feedback from growers and agricultural experts can ensure the models meet the needs of the industry and provide valuable insights for optimizing apple quality and production processes.

1.17 References

1. Prediction of Apple Internal Quality Using Spectral Absorption and Scattering Properties, J. Qin, R. Lu, Y. Peng, 2009, Journal of the ASABE, 52, 2, 499, 486, 10.13031/2013.26807, https://app.dimensions.ai/details/publication/pub.1064894540, 2024/04/27

2. Application of Hyperspectral Imaging and Acoustic Emission Techniques for Apple Quality Prediction, Ekramirad, Nader, Rady, Ahmed, Adedeji, Akinbode A., Alimardani, Reza, 2017, Journal of the ASABE, 60, 4, 1391, 1401, 10.13031/trans.12184, https://app.dimensions.ai/details/publication/pub.1091452475, https://uknowledge.uky.edu/cgi/viewcontent.cgi?article=1212&context=bae_facpub, 2024/04/27

3. Zou Xiaobo and Zhao Jiewen, "Apple quality assessment by fusion three sensors," SENSORS, 2005 IEEE, Irvine, CA, USA, 2005, pp. 4 pp.-, doi: 10.1109/ICSENS.2005.1597717.

4. L. Bochereau, P. Bourgine, B. Palagos, A method for prediction by combining data analysis and neural networks: Application to prediction of apple quality using near infra-red spectraag, Journal of Agricultural Engineering Research, Volume 51, 1992, Pages 207-216, ISSN 0021-8634, https://doi.org/10.1016/0021-8634(92)80038-T.

5. Jha, S.N., Garg, R. Non-destructive prediction of quality of intact apple using near infrared spectroscopy. J Food Sci Technol 47, 207–213 (2010). https://doi.org/10.1007/s13197-010-0033-1

6. Fallahi, E., Fallahi, B., Retamales, J.B., Valdés, C. and Tabatabaei, S.J. (2006). PREDICTION OF APPLE FRUIT QUALITY USING PREHARVEST MINERAL NUTRIENTS. Acta Hortic. 721, 259-264, DOI: 10.17660/ActaHortic.2006.721.35, https://doi.org/10.17660/ActaHortic.2006.721.35

Chapter 2

Web Page Phishing Detection Using Machine Learning

2.1 Introduction

Phishing is a significant cybersecurity threat where attackers create fake websites to steal sensitive information. Machine learning offers a powerful solution to detect and prevent phishing attacks. By analysing patterns in URL structures, website content, and other metadata, machine learning algorithms can identify phishing sites with high accuracy. This project aims to develop a machine learning-based system to automatically detect phishing websites in real-time. The approach involves training classification models on a dataset containing features of both legitimate and phishing websites. Features such as URL length, use of HTTPS, domain age, and website traffic are used to train models like Random Forests, Support Vector Machines, and Neural Networks. These models learn to distinguish between benign and malicious sites based on their features.

To evaluate the effectiveness of the machine learning models, we use metrics like accuracy, precision, recall, and the F1-score. Additionally, the ROC-AUC curve provides insights into the trade-offs between true positive rates and false positive rates at different thresholds, helping to fine-tune the model's performance.

The real-world application of this project is vast. Integrating the system into web browsers and email clients can provide users with immediate warnings about potential phishing threats. It can also be used by organizations to protect their networks and sensitive data from phishing attacks. The system's ability to adapt to new phishing techniques through continuous learning makes it a robust tool in the fight against cybercrime.

In conclusion, this project not only contributes to the field of cybersecurity by providing an advanced tool for phishing detection but also demonstrates the versatility of machine learning in addressing complex problems. The ongoing development and refinement of the system will ensure it remains effective against the ever-evolving landscape of phishing threats, making the online environment safer for everyone.

2.2 Methodology

The methodology for Web Page Phishing Detection using Machine Learning requires following steps:

1. **Data Exploration and Preprocessing:**
    - Load the dataset and perform initial exploratory data analysis (EDA).
    - Identify any missing values or outliers and handle them appropriately.
    - Visualize the distribution of the target classes to understand the balance.

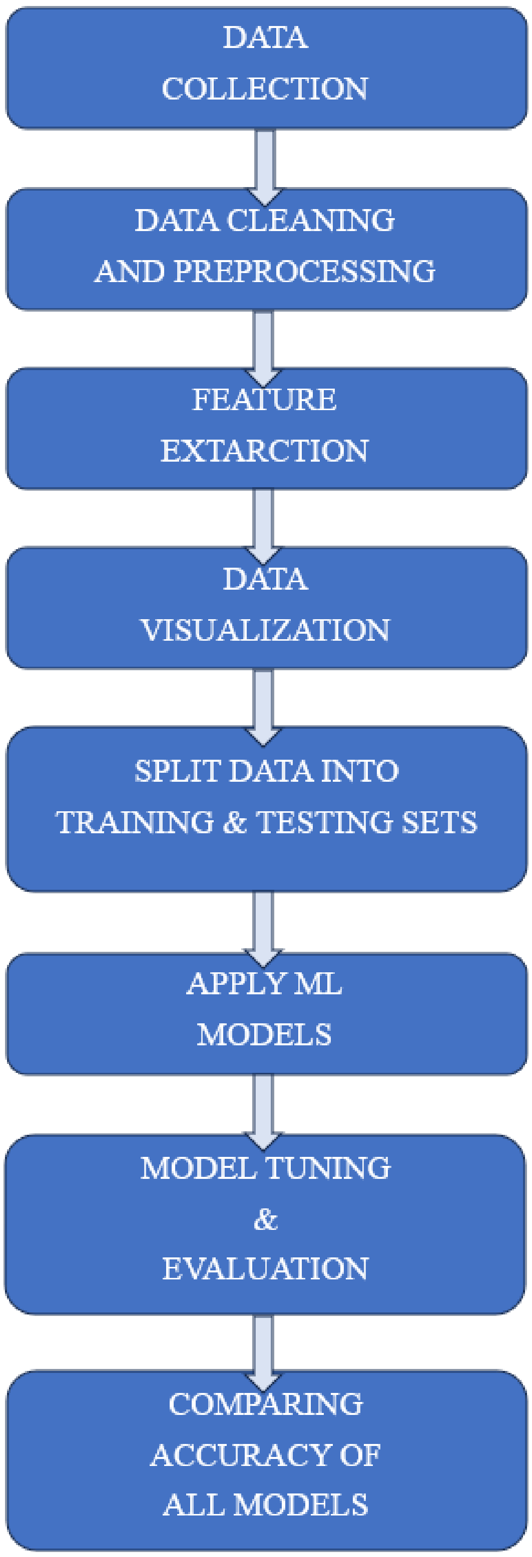
2. **Feature Engineering:**
    - Analyse the existing features and consider creating new relevant features that might enhance the model's performance.
    - Investigate the correlation between features and remove any highly correlated ones.

3. **Model Selection:**
    - Choose appropriate machine learning algorithms for phishing detection (e.g., Decision Trees, Random Forest, Logistic Regression, etc.).
    - Implement a baseline model and evaluate its performance using metrics such as accuracy, precision, recall, and F1-score.

4. **Model Tuning and Evaluation:**
    - Fine-tune the hyperparameters of the chosen model to optimize its performance.
    - Split the dataset into training and testing sets for model evaluation.
    - Evaluate the model on the testing set and analyse the results.

5. **Interpretability and Explainability:**
    - Employ techniques to interpret and explain the model's predictions.
    - Understand the features contributing most to the classification.

2.3 Proposed Algorithm


DATA COLLECTION
DATA CLEANING AND PREPROCESSING
FEATURE EXTARCTION
DATA VISUALIZATION
SPLIT DATA INTO TRAINING & TESTING SETS
APPLY ML MODELS
MODEL TUNING & EVALUATION
COMPARING ACCURACY OF ALL MODELS

2.4 Literature Review

1. “Phishing Detection: Analysis of Visual Similarity Based Approaches” by Ankit Kumar Jain and B. B. Gupta [1]: This study offers an extensive examination of phishing attacks, including their tactics and recent advancements in visual similarity-based methods for detection. It also conducts a comparative analysis to assess their effectiveness.

2. “Detecting visually similar Web pages: Application to phishing detection” by Teh-Chung Chen, Scott Dick and James Miller [2]: This research introduces a novel method for detecting visual similarity between web pages, inspired by Gestalt theory. By treating web pages as indivisible entities and comparing their super signals using algorithmic complexity theory, our approach effectively identifies phishing scams.

3. “Phishing-Alarm: Robust and Efficient Phishing Detection via Page Component Similarity” by Jian Mao, Wenqian Tian, Pei Li, Tao Wei and Zhenkai Liang [3]: This paper introduces Phishing-Alarm, a novel solution for detecting phishing attacks. It utilizes CSS to quantify visual similarity between web pages, employing weighted page-component similarity for accuracy. Implemented as a prototype in Google Chrome, it demonstrates effectiveness with low performance overhead in real-world evaluations.

4. “Machine learning based phishing detection from URLs” by Ozgur Koray Sahingoz, Ebubekir Buber, Onder Demir and Banu Diri [4]: This research paper introduces a real-time anti-phishing system utilizing seven classification algorithms and NLP-based features. Notable features include language independence, large dataset usage, real-time execution, new website detection, third-party independence, and feature-rich classifiers. Experimental results, using a new dataset, show Random Forest with NLP features achieves 97.98% accuracy in phishing URL detection.

5. “Anomaly Based Web Phishing Page Detection” by Ying Pan and Xuhua Ding [5]: This paper introduces a novel, phishing detection method that operates independently of specific phishing implementations. It scrutinizes anomalies within web pages, focusing on discrepancies between a site's identity and its structural elements and HTTP transactions. This approach requires no user expertise or prior knowledge of the site and imposes a high evasion cost on adversaries. Experimental results demonstrate its effectiveness, boasting low miss and false-positive rates.

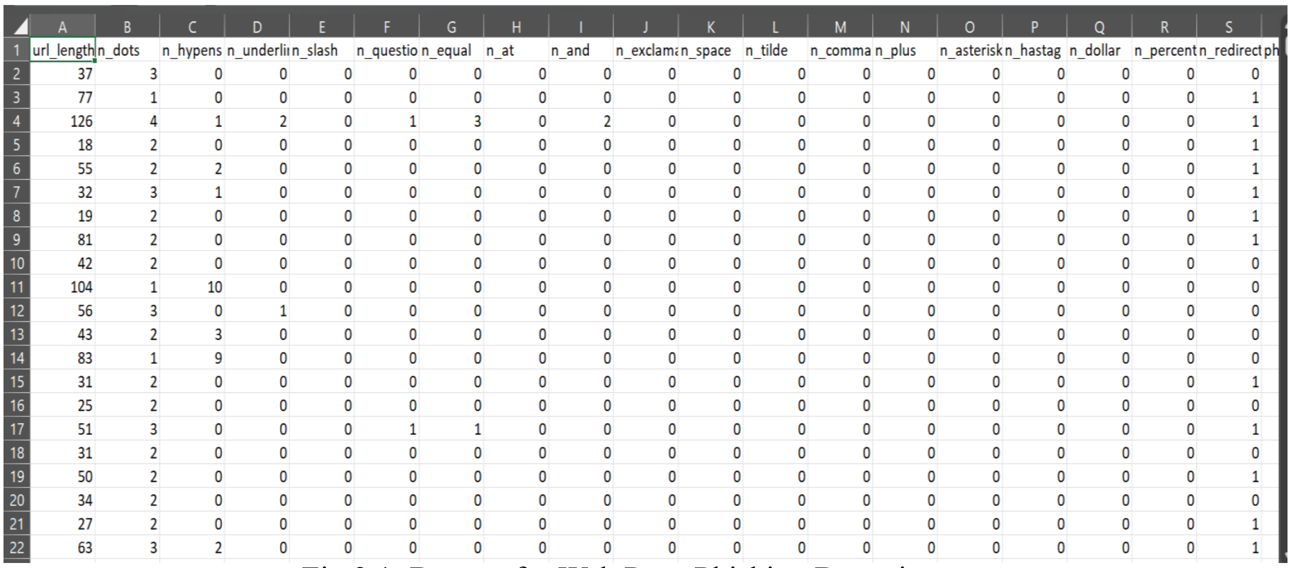
6. “Phishing Web Page Detection” by Liu Wenyin, Guanglin Huang, Liu Xiaoyue, Xiaotie Deng and Zhang Min [6]: This research proposed approach detects phishing web pages based on visual similarity, suitable for enterprise anti-phishing solutions. It enables legitimate page owners to identify suspicious pages resembling their own. By decomposing pages into salient block regions and assessing block, layout, and style similarities, it effectively identifies phishing suspects. Preliminary experiments demonstrate its accuracy with minimal false alarms, suitable for real-time online application.

7. “Web Phishing Detection using a Deep Learning Framework” by Ping Yi, Yuxiang Guan, Futai Zou, Yao Yao, Wei Wang and Ting Zhu [7]: This paper explores using deep learning to combat web phishing, a significant threat to internet services. Two feature types, original and interaction, are designed for phishing detection, with a Deep Belief Network (DBN) model proposed. Testing with real ISP data demonstrates around 90% true positive and 0.6% false positive rates for the DBN-based detection model.

2.5 Data Collection

In this research, I employed the dataset obtained from Kaggle.

The dataset was created by combining two datasets that share the same features. To ensure relevance and eliminate duplication, only key features were preserved in the merged dataset. This curated selection offers a detailed perspective on the common attributes of the original datasets, while keeping the feature set concise and targeted.



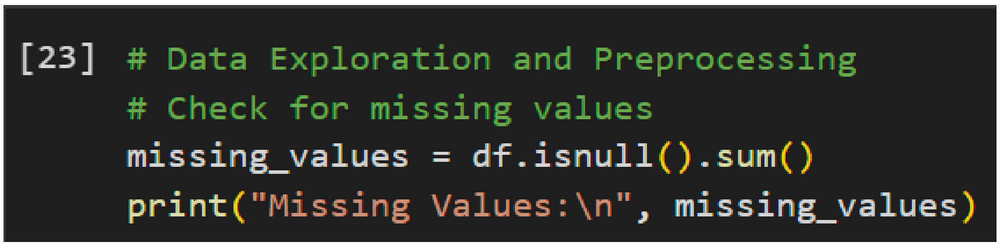
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	
1	url_length	n_dots	n_hypens	n_underli	n_slash	n_questio	n_equal	n_at	n_and	n_exclama	n_space	n_tilde	n_comma	n_plus	n_asterisk	n_hastag	n_dollar	n_percent	n_redirect	ph
2	37	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	77	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
4	126	4	1	2	0	1	3	0	2	0	0	0	0	0	0	0	0	0	1	
5	18	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
6	55	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
7	32	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
8	19	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
9	81	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
10	42	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
11	104	1	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
12	56	3	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
13	43	2	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
14	83	1	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
15	31	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
16	25	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
17	51	3	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	
18	31	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
19	50	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
20	34	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
21	27	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
22	63	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	

Fig 2.1: Dataset for Web Page Phishing Detection

2.6 Data Preprocessing and Cleaning

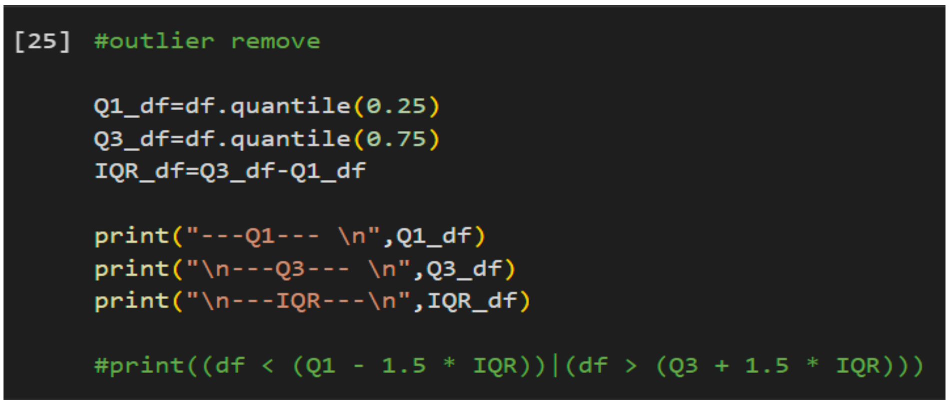
Data preprocessing and cleaning are essential in the classification workflow, especially with news data from various origins. These methods aim to remove noise and inaccuracies, readying the data for effective machine learning model training. In this stage, we discard punctuation, numerals, special symbols, and blank tokens, as they hold no predictive value.

- In Data Cleaning, we will first check for missing values.

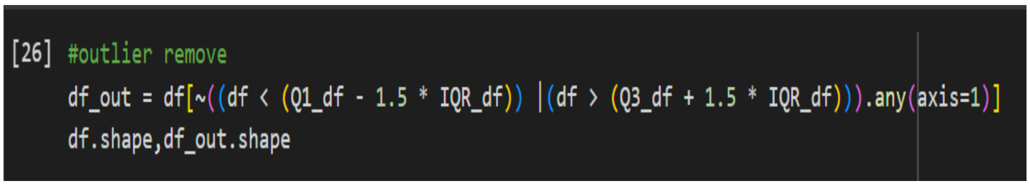


```
[23] # Data Exploration and Preprocessing
     # Check for missing values
     missing_values = df.isnull().sum()
     print("Missing Values:\n", missing_values)
```

- After that, we will remove Outliers.

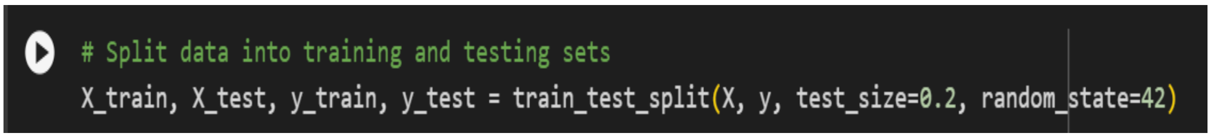


```
[25] #outlier remove

     Q1_df=df.quantile(0.25)
     Q3_df=df.quantile(0.75)
     IQR_df=Q3_df-Q1_df

     print("---Q1--- \n",Q1_df)
     print("\n---Q3--- \n",Q3_df)
     print("\n---IQR---\n",IQR_df)

     #print((df < (Q1 - 1.5 * IQR))|(df > (Q3 + 1.5 * IQR)))
```



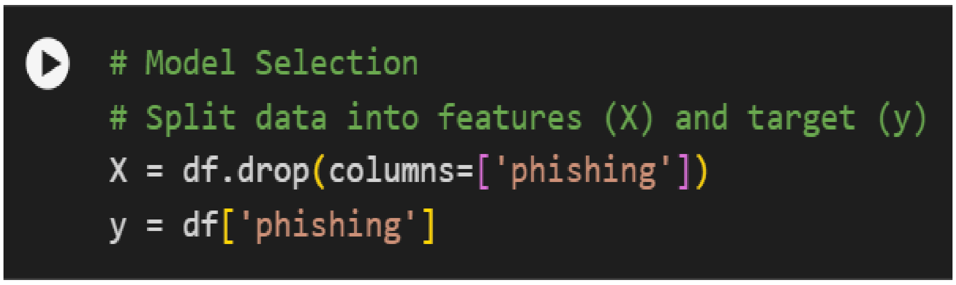
```
[26] #outlier remove
     df_out = df[~((df < (Q1_df - 1.5 * IQR_df)) |(df > (Q3_df + 1.5 * IQR_df))).any(axis=1)]
     df.shape,df_out.shape
```

- Splitting the Data: Splitting data in preprocessing is a critical step for machine learning, where the dataset is divided into training and testing sets. This separation allows models to learn from one subset (training) and be evaluated on another (testing), ensuring they can generalize well to new data. Typically, a 70-30 or 80-20 split is used, but variations may apply based on specific project needs.



```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

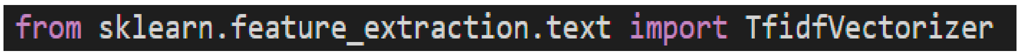
- In machine learning, data is split into features (X) and target (Y) to distinguish between the input variables (features) and the output variable (target). The features are the data points that the model uses to make predictions, while the target is the outcome we're trying to predict. This separation allows the model to learn the relationship between the features and the target during training and to accurately predict the target for new, unseen data.



```
# Model Selection
# Split data into features (X) and target (y)
X = df.drop(columns=['phishing'])
y = df['phishing']
```

2.7 Feature Extraction Using TF-IDF Vectorizer

In web page phishing detection, TF-IDF Vectorizer is used for feature extraction to transform text data into numerical values. It calculates the importance of words in documents relative to a collection, helping to identify key terms indicative of phishing. This numerical representation allows machine learning models to process and classify web pages as legitimate or fraudulent based on textual content, enhancing the detection of phishing activities.



```
from sklearn.feature_extraction.text import TfidfVectorizer
```

1. Term Frequency (TF): This measures how frequently a term occurs in a document. The TF for a term "t" in a document "d" is calculated as:

$$TF(t, d) = \frac{\text{(Number of times term t appears in document d)}}{\text{(Total number of terms in document d)}} \quad \textbf{(12)}$$

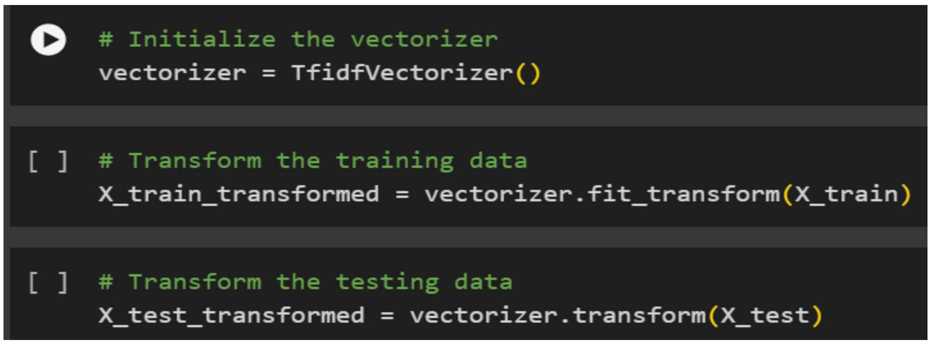
2. Inverse Document Frequency (IDF): This measures the importance of a term across a set of documents. The IDF for a term "t" is calculated as:

$$IDF(t) = \log \text{(Total number of documents)} / \text{(Number of documents with term t)} \quad \textbf{(13)}$$

3. TF-IDF: The TF-IDF score is the product of TF and IDF, representing the relative importance of a term in a document within a corpus. The formula is:

$$\boldsymbol{TF\text{-}IDF(t,d) = TF(t,d) \times IDF(t)} \quad \textbf{(14)}$$

These formulas help to evaluate how important a word is to a document in a collection or corpus. The higher the TF-IDF score, the more relevant the term is in that particular document.

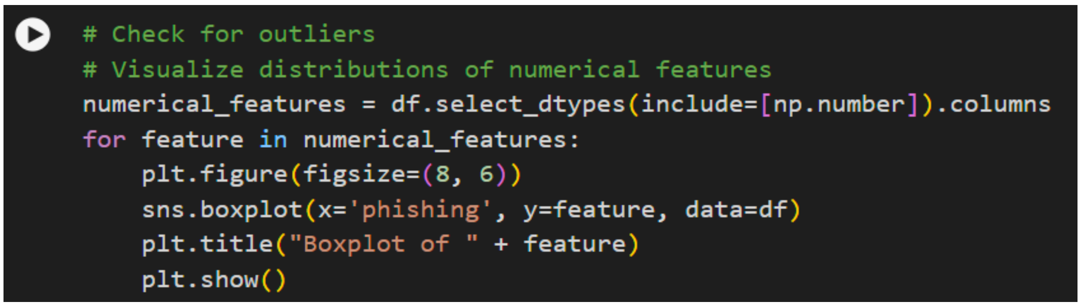


```
# Initialize the vectorizer
vectorizer = TfidfVectorizer()

[ ] # Transform the training data
X_train_transformed = vectorizer.fit_transform(X_train)

[ ] # Transform the testing data
X_test_transformed = vectorizer.transform(X_test)
```

2.8 Data Visualization

Data visualization transforms complex data into graphical formats like charts, graphs, and maps, effectively communicating intricate details and revealing hidden patterns. It's a vital skill in data science, aiding in the interpretation of trends, outliers, and correlations that text data might obscure, thus facilitating clearer, data-driven decisions.

- **Boxplot:** In web page phishing detection, a boxplot can summarize the distribution of feature values, such as URL length or domain age.



```
# Check for outliers
# Visualize distributions of numerical features
numerical_features = df.select_dtypes(include=[np.number]).columns
for feature in numerical_features:
    plt.figure(figsize=(8, 6))
    sns.boxplot(x='phishing', y=feature, data=df)
    plt.title("Boxplot of " + feature)
    plt.show()
```

It highlights the median, quartiles, and outliers, aiding in the detection of anomalous data points that could signify phishing attempts.

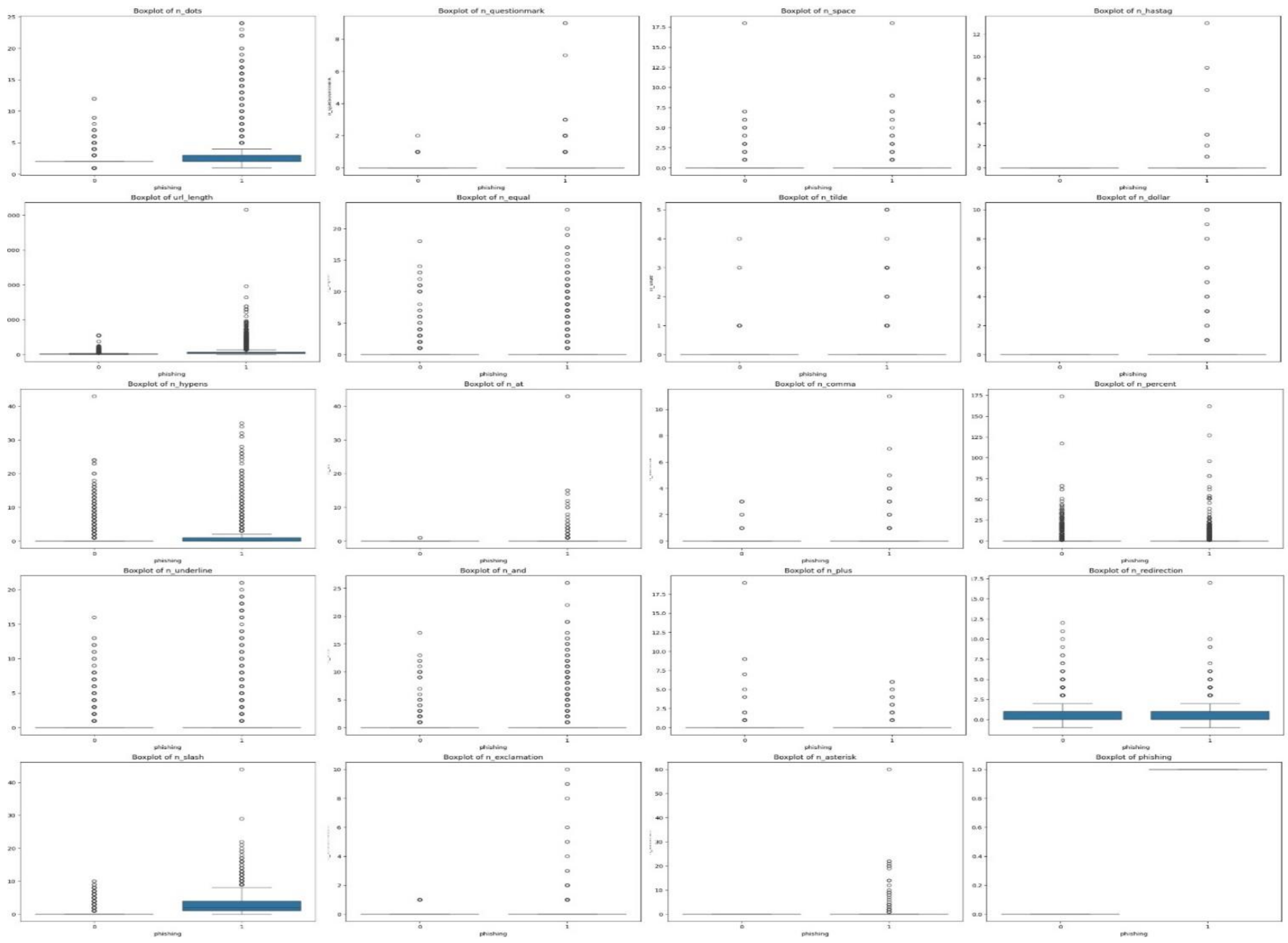
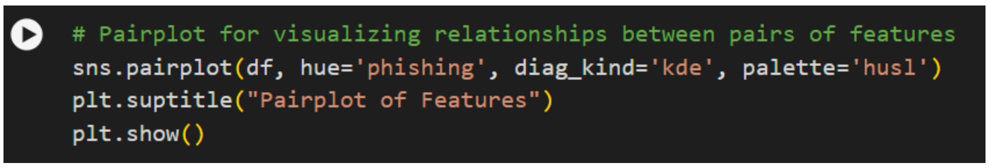


Fig 2.2: Box Plot of Web Page Phishing Detection

- **Pairplot:** A pairplot is useful for visualizing the relationships between various features, such as keyword frequency and special character count in URLs.



```
# Pairplot for visualizing relationships between pairs of features
sns.pairplot(df, hue='phishing', diag_kind='kde', palette='husl')
plt.suptitle("Pairplot of Features")
plt.show()
```

It can display scatterplots for each feature pair and histograms for individual features, helping to pinpoint the most predictive attributes for phishing classification.

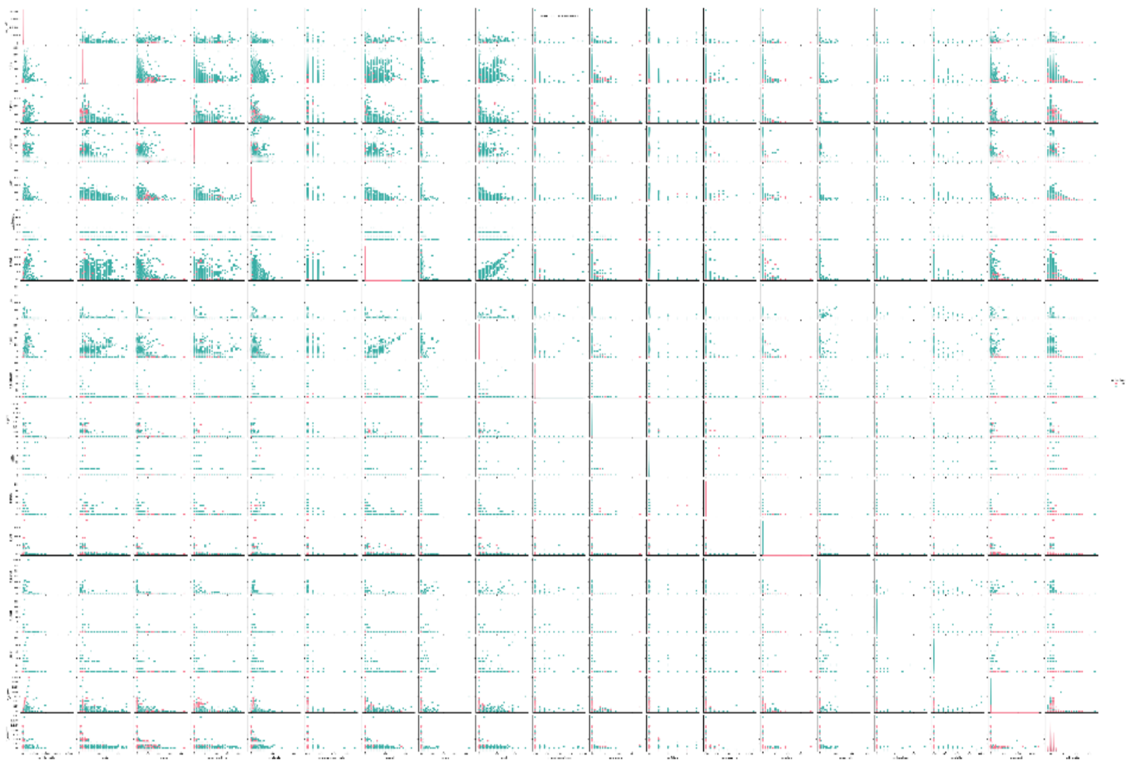
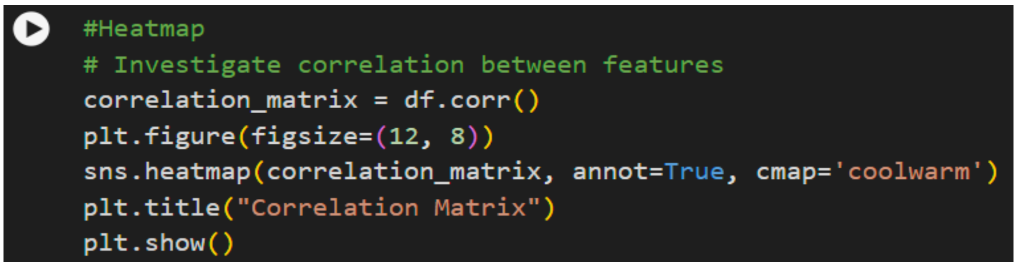


Fig 2.3: Pair Plot of Web Page Phishing Detection

- **Heatmap:** Heatmaps are significant for illustrating the correlation between different features used in phishing detection.



```
#Heatmap
# Investigate correlation between features
correlation_matrix = df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```

They can show how various URL and website characteristics relate to each other, using colour intensities to indicate the strength of these relationships, which is crucial for identifying the most influential factors in phishing detection.

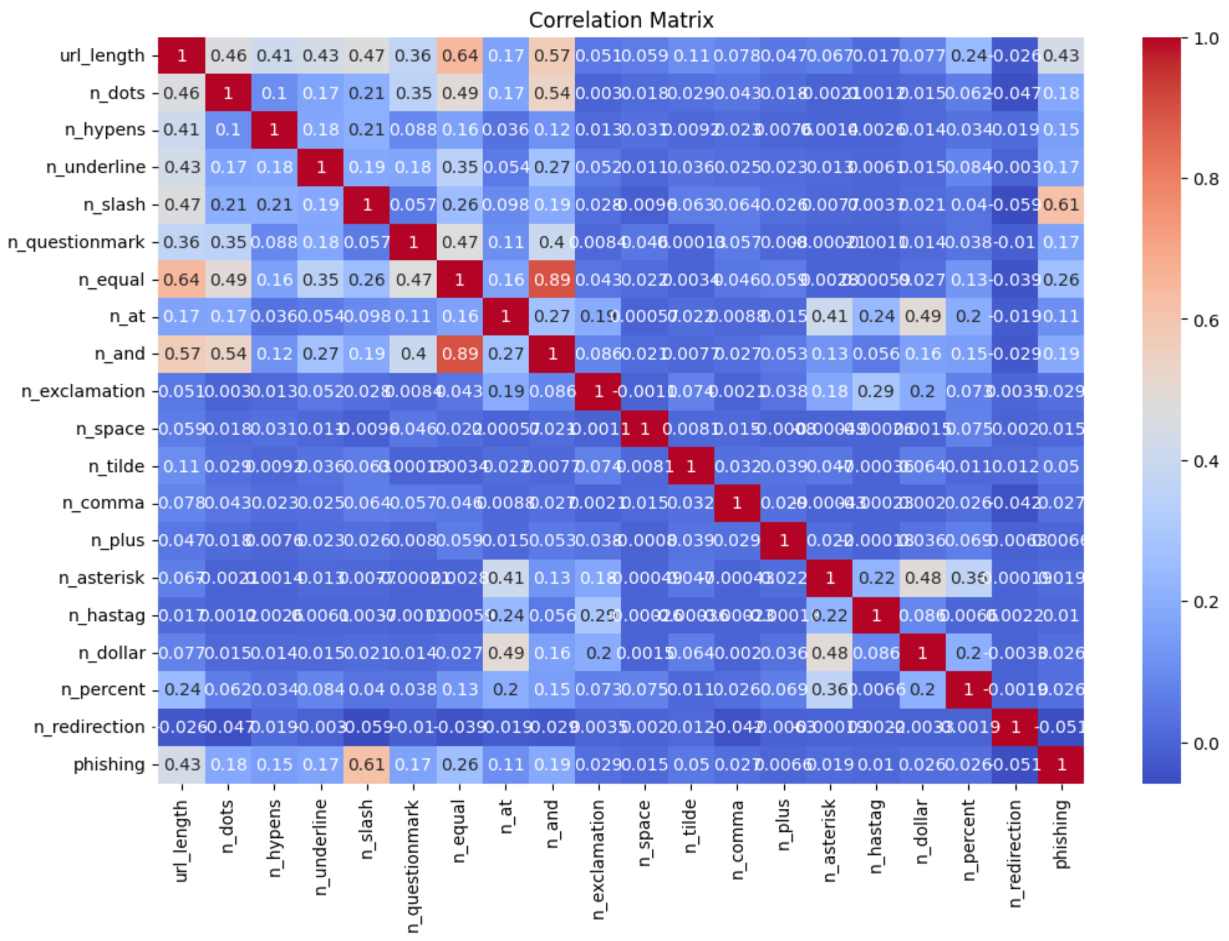
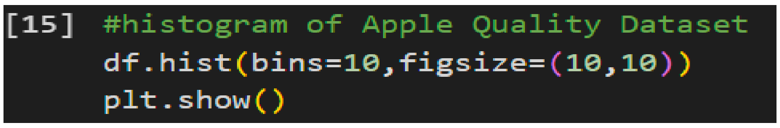


Fig 2.4: Heatmap of Web Page Phishing Detection

- **Histogram:** Histograms in machine learning display the frequency distribution of data points, such as the number of times specific keywords appear in URLs.



```
[15] #histogram of Apple Quality Dataset
     df.hist(bins=10,figsize=(10,10))
     plt.show()
```

This visualization helps in recognizing common patterns and outliers within the dataset.

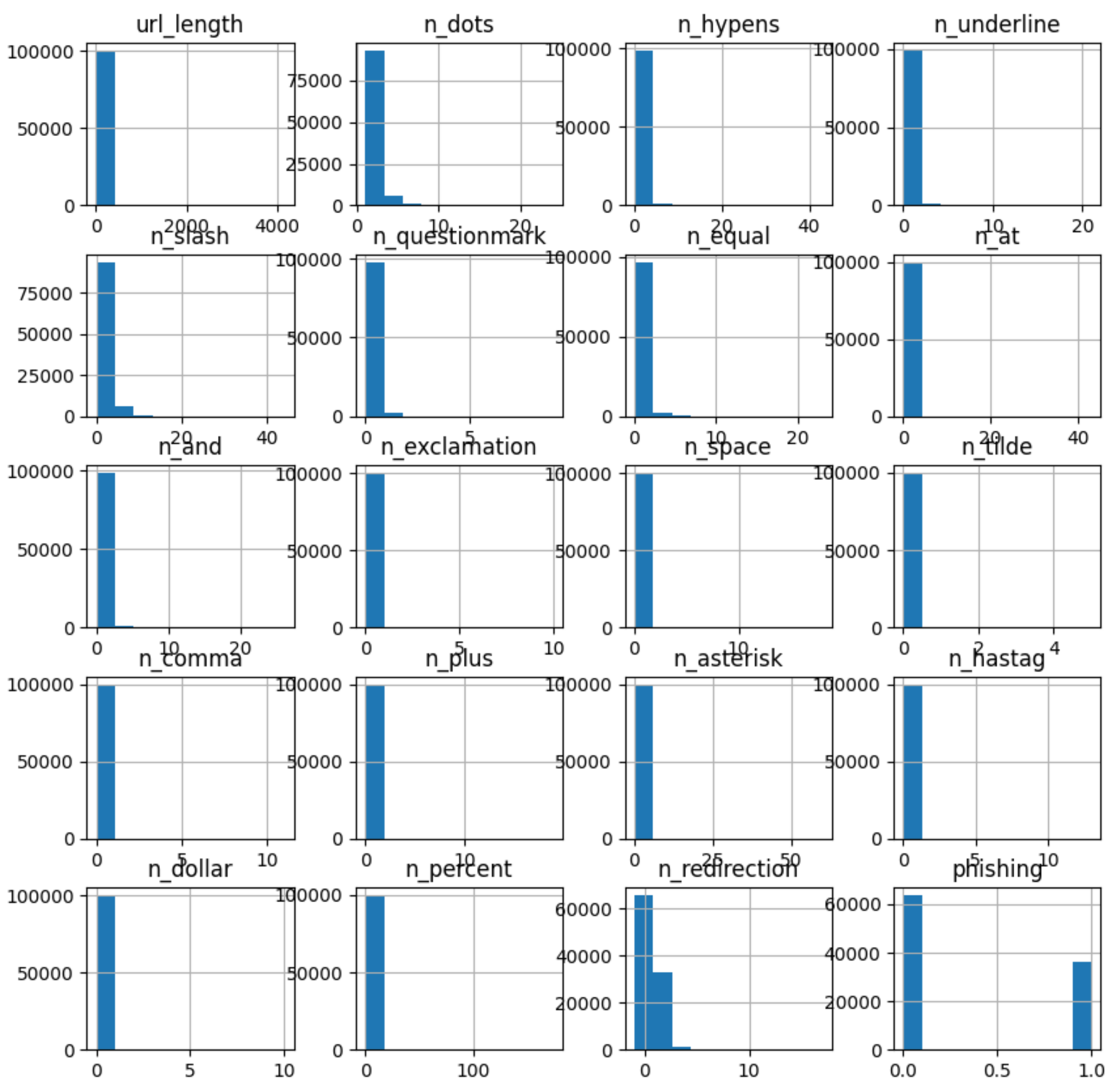
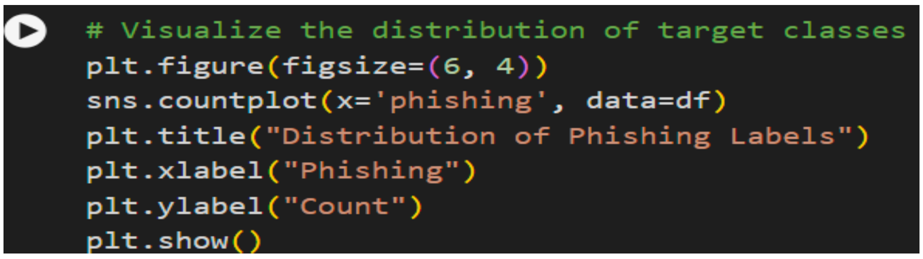


Fig 2.5: Histogram of Web Page Phishing Detection

- **Target Class Visualization:** Visualizing the target class in phishing detection involves graphically representing the distribution of legitimate versus phishing websites.



```
# Visualize the distribution of target classes
plt.figure(figsize=(6, 4))
sns.countplot(x='phishing', data=df)
plt.title("Distribution of Phishing Labels")
plt.xlabel("Phishing")
plt.ylabel("Count")
plt.show()
```

This can be done through bar charts or pie charts, providing a clear view of the class balance in the dataset, which is essential for understanding model performance and addressing any class imbalance issues.

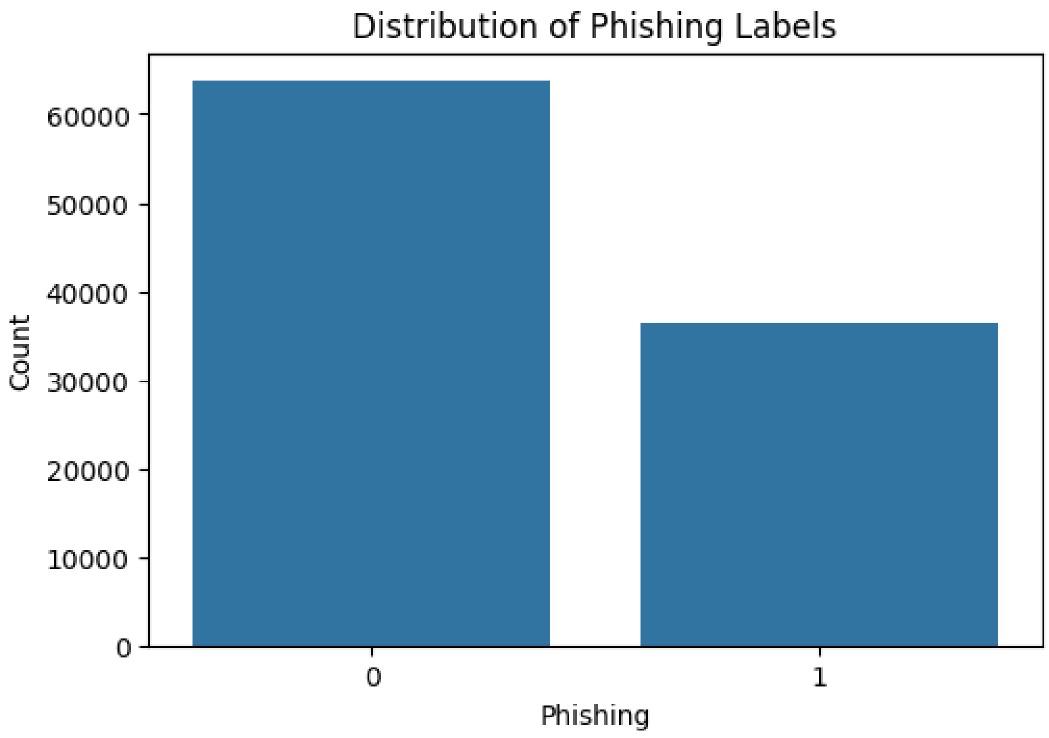
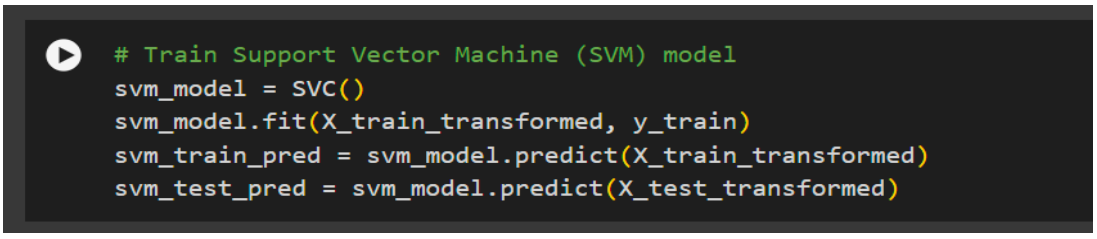


Fig 2.6: Target Class Visualization of Web Page Phishing Detection

2.9 Applying Different Models

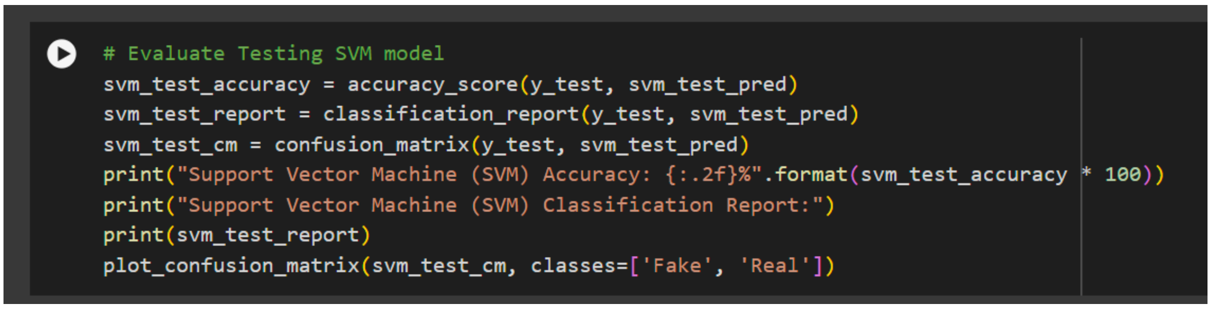
- ### Support Vector Machine Model

Support Vector Machine (SVM) is a robust machine learning algorithm used in web page phishing detection to classify websites as legitimate or fraudulent. It works by finding the optimal hyperplane that separates phishing sites from genuine ones in a high-dimensional space.

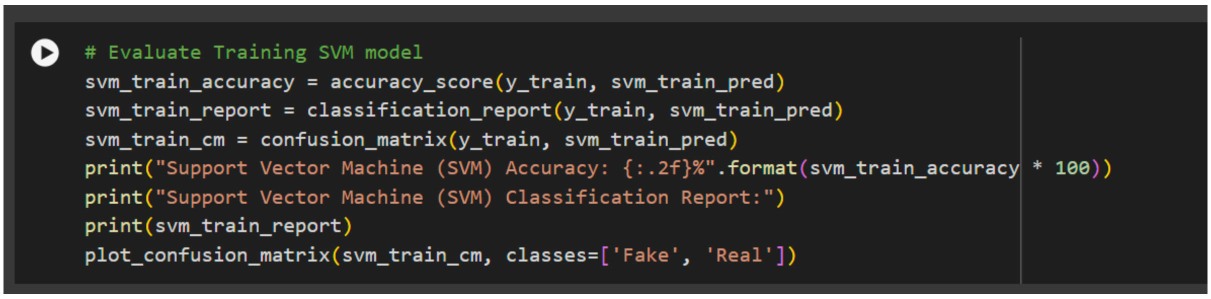


```
# Train Support Vector Machine (SVM) model
svm_model = SVC()
svm_model.fit(X_train_transformed, y_train)
svm_train_pred = svm_model.predict(X_train_transformed)
svm_test_pred = svm_model.predict(X_test_transformed)
```

Features like URL length, HTTPS usage, and abnormal site behaviour are fed into the SVM, which processes and identifies patterns indicative of phishing. The strength of SVM lies in its ability to handle non-linear boundaries through kernel functions, making it highly effective in distinguishing complex phishing characteristics. By leveraging SVM, cybersecurity systems can accurately pinpoint phishing threats, thereby protecting users from potential cyberattacks.



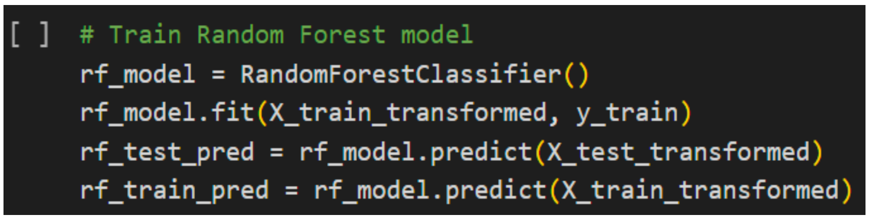
```python
# Evaluate Testing SVM model
svm_test_accuracy = accuracy_score(y_test, svm_test_pred)
svm_test_report = classification_report(y_test, svm_test_pred)
svm_test_cm = confusion_matrix(y_test, svm_test_pred)
print("Support Vector Machine (SVM) Accuracy: {:.2f}%".format(svm_test_accuracy * 100))
print("Support Vector Machine (SVM) Classification Report:")
print(svm_test_report)
plot_confusion_matrix(svm_test_cm, classes=['Fake', 'Real'])
```



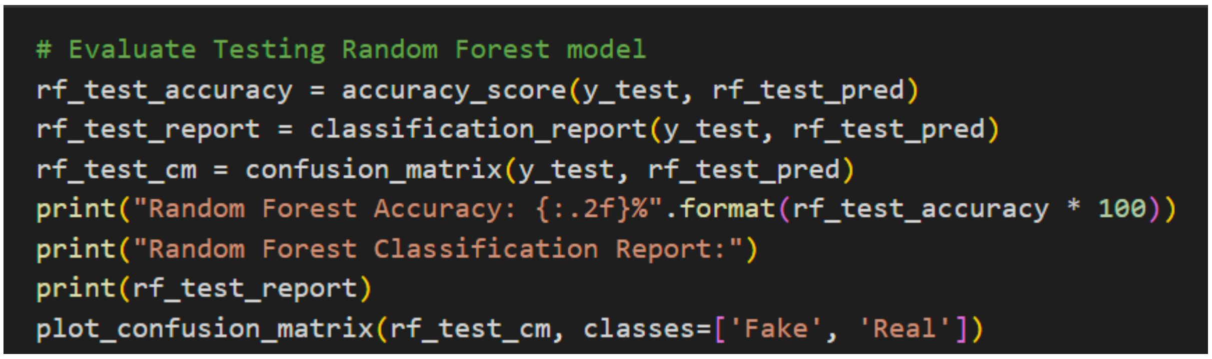
```python
# Evaluate Training SVM model
svm_train_accuracy = accuracy_score(y_train, svm_train_pred)
svm_train_report = classification_report(y_train, svm_train_pred)
svm_train_cm = confusion_matrix(y_train, svm_train_pred)
print("Support Vector Machine (SVM) Accuracy: {:.2f}%".format(svm_train_accuracy * 100))
print("Support Vector Machine (SVM) Classification Report:")
print(svm_train_report)
plot_confusion_matrix(svm_train_cm, classes=['Fake', 'Real'])
```

- **Random Forest Model**

Random Forest is an ensemble learning method that combines multiple decision trees during training. Each tree is constructed using a random subset of the data, and the final output is determined by the majority class predicted by individual trees.

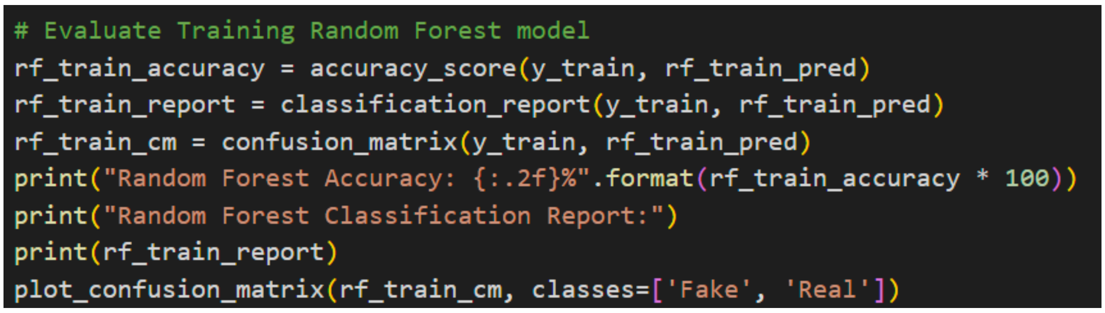


```python
[ ]  # Train Random Forest model
     rf_model = RandomForestClassifier()
     rf_model.fit(X_train_transformed, y_train)
     rf_test_pred = rf_model.predict(X_test_transformed)
     rf_train_pred = rf_model.predict(X_train_transformed)
```



```python
# Evaluate Testing Random Forest model
rf_test_accuracy = accuracy_score(y_test, rf_test_pred)
rf_test_report = classification_report(y_test, rf_test_pred)
rf_test_cm = confusion_matrix(y_test, rf_test_pred)
print("Random Forest Accuracy: {:.2f}%".format(rf_test_accuracy * 100))
print("Random Forest Classification Report:")
print(rf_test_report)
plot_confusion_matrix(rf_test_cm, classes=['Fake', 'Real'])
```

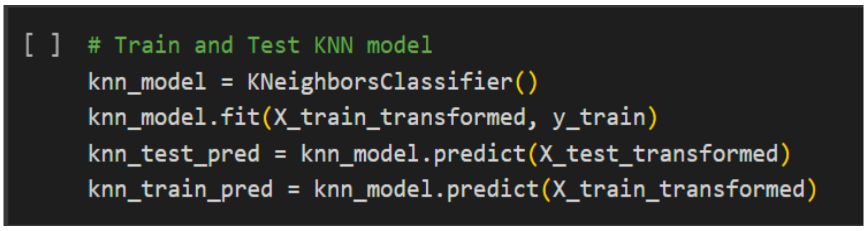
Random Forest is highly effective for phishing detection due to its robustness, ability to handle large datasets, and accurate classification of websites as legitimate or fraudulent.



```
# Evaluate Training Random Forest model
rf_train_accuracy = accuracy_score(y_train, rf_train_pred)
rf_train_report = classification_report(y_train, rf_train_pred)
rf_train_cm = confusion_matrix(y_train, rf_train_pred)
print("Random Forest Accuracy: {:.2f}%".format(rf_train_accuracy * 100))
print("Random Forest Classification Report:")
print(rf_train_report)
plot_confusion_matrix(rf_train_cm, classes=['Fake', 'Real'])
```

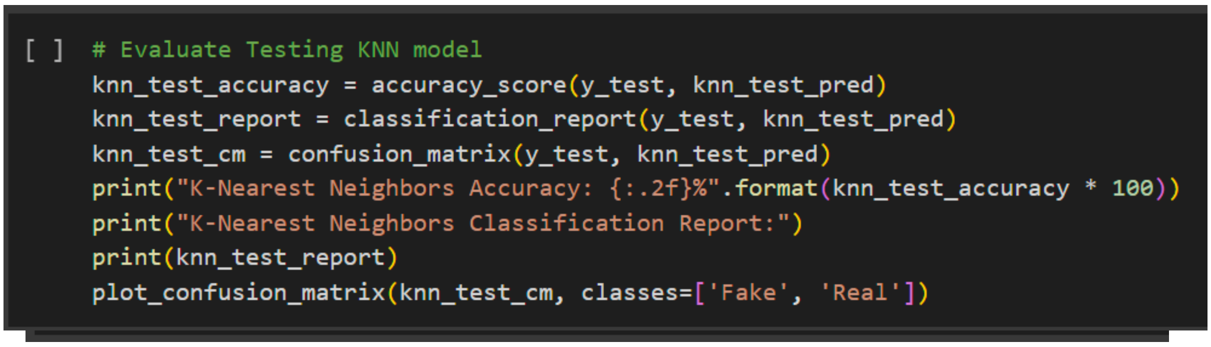
- **K-Nearest Neighbors (KNN) Model**

K-Nearest Neighbors (KNN) is a versatile algorithm that operates on the principle of proximity, classifying websites by examining the characteristics of their nearest neighbors in the dataset.



```
[ ]  # Train and Test KNN model
     knn_model = KNeighborsClassifier()
     knn_model.fit(X_train_transformed, y_train)
     knn_test_pred = knn_model.predict(X_test_transformed)
     knn_train_pred = knn_model.predict(X_train_transformed)
```

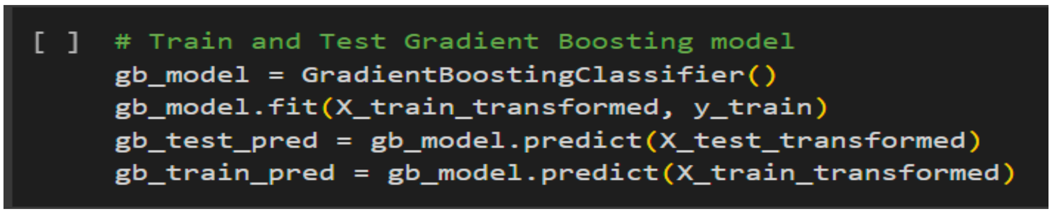
In the context of phishing detection, KNN considers the most similar websites—based on features like URL structure, keyword frequency, and domain information—to determine if a new website is likely to be legitimate or a phishing attempt. This method is particularly effective when there's a clear distinction between the attributes of phishing and non-phishing sites, allowing for accurate and quick classification.



```
[ ]  # Evaluate Testing KNN model
     knn_test_accuracy = accuracy_score(y_test, knn_test_pred)
     knn_test_report = classification_report(y_test, knn_test_pred)
     knn_test_cm = confusion_matrix(y_test, knn_test_pred)
     print("K-Nearest Neighbors Accuracy: {:.2f}%".format(knn_test_accuracy * 100))
     print("K-Nearest Neighbors Classification Report:")
     print(knn_test_report)
     plot_confusion_matrix(knn_test_cm, classes=['Fake', 'Real'])
```

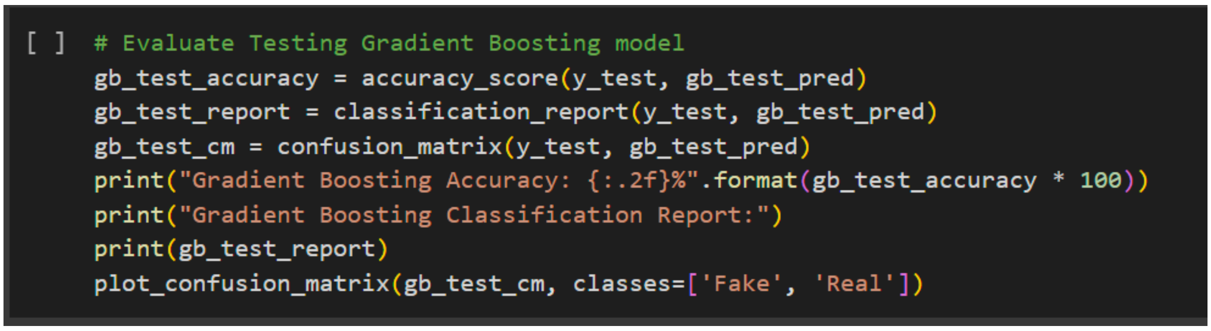
- **Gradient Boosting Model**

Gradient Boosting is a powerful machine learning technique that incrementally builds an ensemble of weak prediction models, typically decision trees, to create a strong overall model.

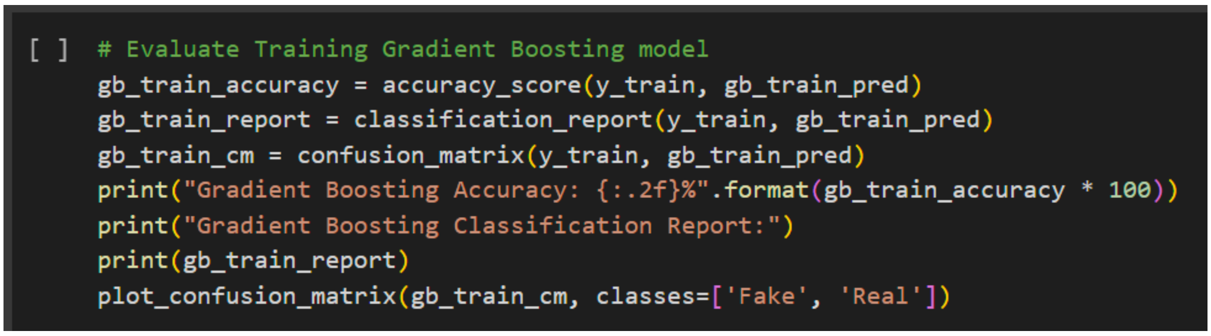


```
[ ]  # Train and Test Gradient Boosting model
     gb_model = GradientBoostingClassifier()
     gb_model.fit(X_train_transformed, y_train)
     gb_test_pred = gb_model.predict(X_test_transformed)
     gb_train_pred = gb_model.predict(X_train_transformed)
```

It's adept at optimizing a loss function, making it highly effective for complex tasks like phishing detection. By sequentially correcting errors from previous models, Gradient Boosting excels in accuracy and can manage diverse data types, making it a valuable tool for identifying phishing websites with precision.



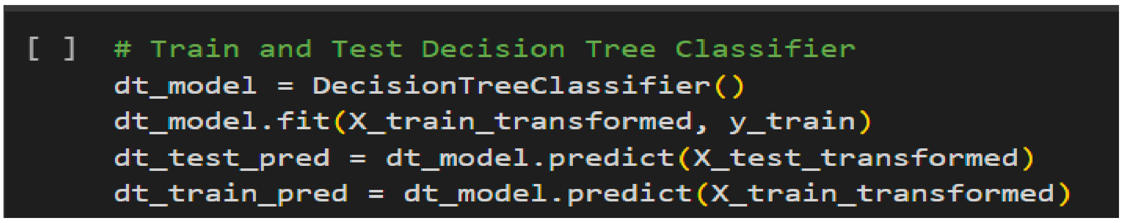
```
[ ]  # Evaluate Testing Gradient Boosting model
     gb_test_accuracy = accuracy_score(y_test, gb_test_pred)
     gb_test_report = classification_report(y_test, gb_test_pred)
     gb_test_cm = confusion_matrix(y_test, gb_test_pred)
     print("Gradient Boosting Accuracy: {:.2f}%".format(gb_test_accuracy * 100))
     print("Gradient Boosting Classification Report:")
     print(gb_test_report)
     plot_confusion_matrix(gb_test_cm, classes=['Fake', 'Real'])
```



```
[ ]  # Evaluate Training Gradient Boosting model
     gb_train_accuracy = accuracy_score(y_train, gb_train_pred)
     gb_train_report = classification_report(y_train, gb_train_pred)
     gb_train_cm = confusion_matrix(y_train, gb_train_pred)
     print("Gradient Boosting Accuracy: {:.2f}%".format(gb_train_accuracy * 100))
     print("Gradient Boosting Classification Report:")
     print(gb_train_report)
     plot_confusion_matrix(gb_train_cm, classes=['Fake', 'Real'])
```

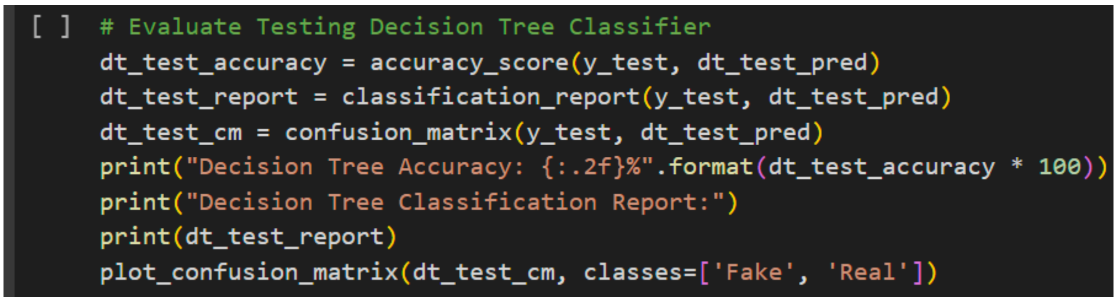
- **Decision Tree Classifier**

The Decision Tree is a decision support algorithm that uses a tree-like structure to model decisions and their potential outcomes. It’s particularly user-friendly, offering a clear visualization of the decision-making process.

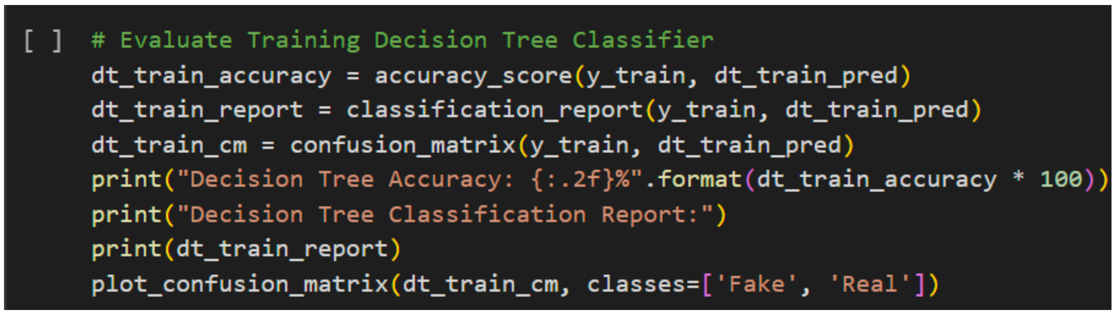


```
[ ]  # Train and Test Decision Tree Classifier
     dt_model = DecisionTreeClassifier()
     dt_model.fit(X_train_transformed, y_train)
     dt_test_pred = dt_model.predict(X_test_transformed)
     dt_train_pred = dt_model.predict(X_train_transformed)
```

In phishing detection, a Decision Tree can effectively classify websites by learning from features such as URL characteristics and content keywords.



```
[ ]  # Evaluate Testing Decision Tree Classifier
     dt_test_accuracy = accuracy_score(y_test, dt_test_pred)
     dt_test_report = classification_report(y_test, dt_test_pred)
     dt_test_cm = confusion_matrix(y_test, dt_test_pred)
     print("Decision Tree Accuracy: {:.2f}%".format(dt_test_accuracy * 100))
     print("Decision Tree Classification Report:")
     print(dt_test_report)
     plot_confusion_matrix(dt_test_cm, classes=['Fake', 'Real'])
```

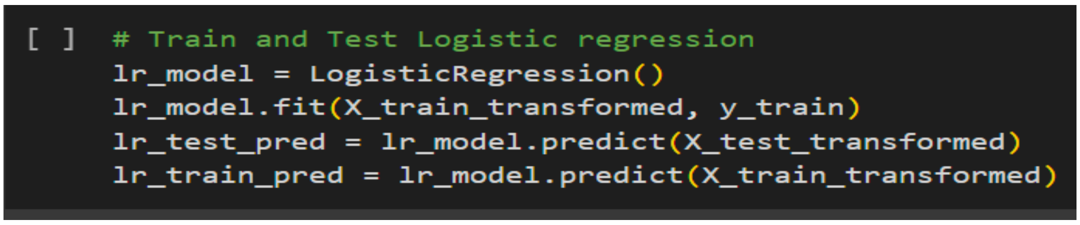


```
[ ]  # Evaluate Training Decision Tree Classifier
     dt_train_accuracy = accuracy_score(y_train, dt_train_pred)
     dt_train_report = classification_report(y_train, dt_train_pred)
     dt_train_cm = confusion_matrix(y_train, dt_train_pred)
     print("Decision Tree Accuracy: {:.2f}%".format(dt_train_accuracy * 100))
     print("Decision Tree Classification Report:")
     print(dt_train_report)
     plot_confusion_matrix(dt_train_cm, classes=['Fake', 'Real'])
```

It splits the data into branches, leading to leaves that represent the classification outcome—phishing or legitimate. This simplicity, coupled with the ability to handle both categorical and numerical data, makes Decision Trees a valuable tool for identifying phishing websites.

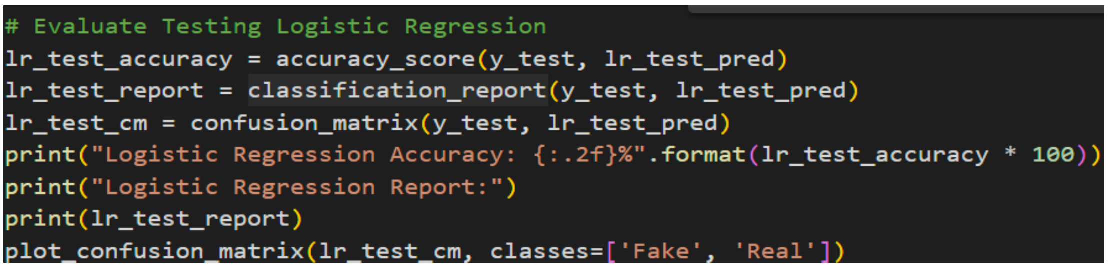
- **Logistic Regression**

Logistic Regression is a statistical model that excels in binary classification tasks, such as phishing detection.



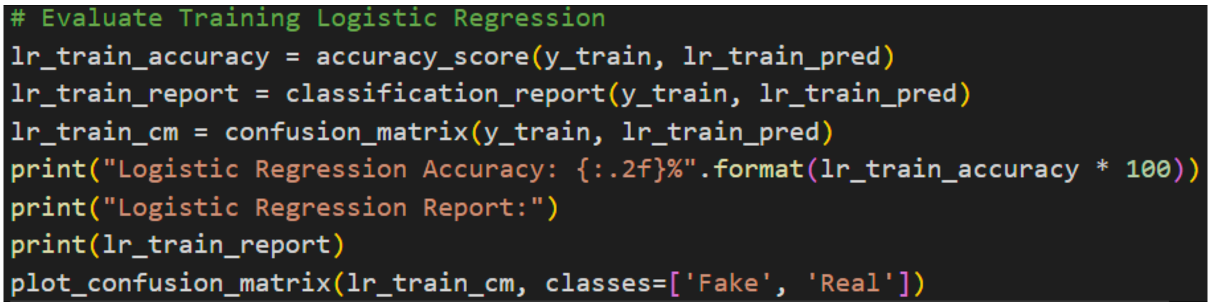
```
[ ]  # Train and Test Logistic regression
     lr_model = LogisticRegression()
     lr_model.fit(X_train_transformed, y_train)
     lr_test_pred = lr_model.predict(X_test_transformed)
     lr_train_pred = lr_model.predict(X_train_transformed)
```

It estimates the probability that a given website is phishing by applying a logistic function to a linear combination of features. These features might include the number of external links, the presence of an SSL certificate, or keyword analysis within the site's content.



```
# Evaluate Testing Logistic Regression
lr_test_accuracy = accuracy_score(y_test, lr_test_pred)
lr_test_report = classification_report(y_test, lr_test_pred)
lr_test_cm = confusion_matrix(y_test, lr_test_pred)
print("Logistic Regression Accuracy: {:.2f}%".format(lr_test_accuracy * 100))
print("Logistic Regression Report:")
print(lr_test_report)
plot_confusion_matrix(lr_test_cm, classes=['Fake', 'Real'])
```

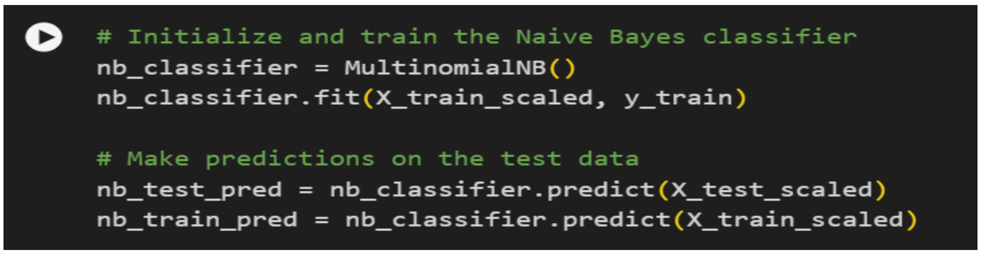
By quantifying the odds of a site being malicious, Logistic Regression provides a clear, probabilistic framework for making predictions, which is invaluable for maintaining online security.



```
# Evaluate Training Logistic Regression
lr_train_accuracy = accuracy_score(y_train, lr_train_pred)
lr_train_report = classification_report(y_train, lr_train_pred)
lr_train_cm = confusion_matrix(y_train, lr_train_pred)
print("Logistic Regression Accuracy: {:.2f}%".format(lr_train_accuracy * 100))
print("Logistic Regression Report:")
print(lr_train_report)
plot_confusion_matrix(lr_train_cm, classes=['Fake', 'Real'])
```

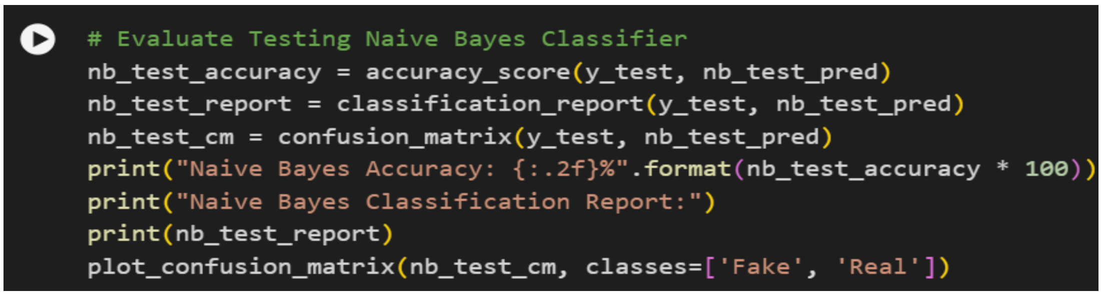
- **Naive Bayes**

Naive Bayes is a straightforward yet effective classifier that applies Bayes' theorem, assuming strong independence between features. It's a workhorse for phishing detection, valued for its computational efficiency and solid performance, even on extensive datasets.

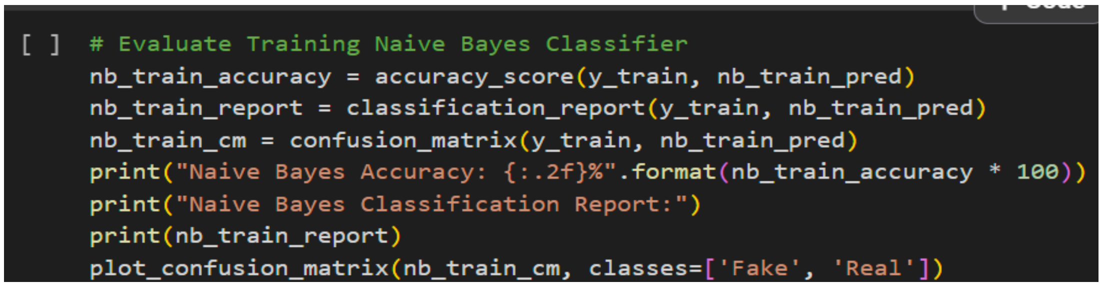


```
# Initialize and train the Naive Bayes classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_scaled, y_train)

# Make predictions on the test data
nb_test_pred = nb_classifier.predict(X_test_scaled)
nb_train_pred = nb_classifier.predict(X_train_scaled)
```

By calculating the probability of a website being phishing, it quickly filters out potential threats, making it a go-to algorithm for cybersecurity professionals.



```
# Evaluate Testing Naive Bayes Classifier
nb_test_accuracy = accuracy_score(y_test, nb_test_pred)
nb_test_report = classification_report(y_test, nb_test_pred)
nb_test_cm = confusion_matrix(y_test, nb_test_pred)
print("Naive Bayes Accuracy: {:.2f}%".format(nb_test_accuracy * 100))
print("Naive Bayes Classification Report:")
print(nb_test_report)
plot_confusion_matrix(nb_test_cm, classes=['Fake', 'Real'])
```

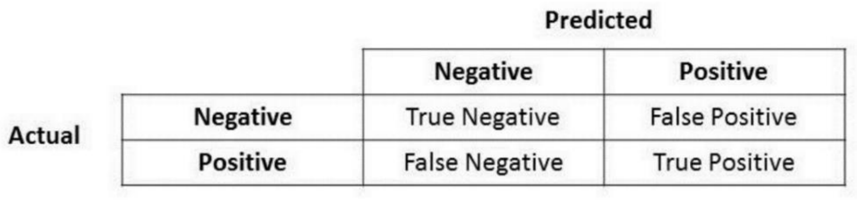


```
[ ] # Evaluate Training Naive Bayes Classifier
    nb_train_accuracy = accuracy_score(y_train, nb_train_pred)
    nb_train_report = classification_report(y_train, nb_train_pred)
    nb_train_cm = confusion_matrix(y_train, nb_train_pred)
    print("Naive Bayes Accuracy: {:.2f}%".format(nb_train_accuracy * 100))
    print("Naive Bayes Classification Report:")
    print(nb_train_report)
    plot_confusion_matrix(nb_train_cm, classes=['Fake', 'Real'])
```

2.10 Assessing Classification Metrices

To gauge the effectiveness of the model and quantify its precision, scikit-learn offers a suite of classification metrics. Key metrics include:

1. **Confusion Matrix**: A table that clarifies the accuracy of predictions by contrasting them with actual classifications, highlighting true positives, true negatives, false positives, and false negatives.



		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

2. **Accuracy Score**: The proportion of total correct predictions, providing a quick snapshot of overall model performance.

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN) \qquad (15)$$

3. **Precision**: The ratio of true positives to all positive predictions, indicating the model's exactness.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive} \qquad (16)$$

$$= \frac{True\ Positive}{Total\ Predicted\ Positive}$$

4. **Recall**: The fraction of actual positives correctly identified, reflecting the model's completeness.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative} \qquad (17)$$

$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

5. **F1-Score**: A harmonic mean of precision and recall, offering a balance between the two for a comprehensive performance measure.

$$F1 = 2 \times \frac{Precision * Recall}{Precision + Recall} \quad (18)$$

These metrics collectively offer deep insights into your model's predictive capabilities and its success in accurately classifying both positive and negative cases within the data.

2.11 Classification Metrics After Feature Extraction

After feature extraction, we present the classification matrices for Web Page Phishing datasets. These matrices provide a comprehensive overview of the model's performance in each dataset, including true positive, true negative, false positive, and false negative counts.

a) Classification Metrics of Testing Data

Classification Metrics With TF-IDF Vectorizer					
WEB PAGE PHISHING DETECTION					
S. No	ML Models	Accuracy	Precision	Recall	F1-Score
1.	**Support Vector Machine (SVM)**	**88.36**	**0.86**	**0.81**	**0.84**
2.	**Random Forest**	**88.93**	**0.86**	**0.83**	**0.85**
3.	**K-Nearest Neighbors (KNN)**	**88.31**	**0.85**	**0.82**	**0.84**
4.	**Gradient Boosting**	**88.19**	**0.84**	**0.83**	**0.84**
5.	**Decision Tree**	**87.95**	**0.86**	**0.80**	**0.83**
6.	**Logistic Regression**	**85.71**	**0.87**	**0.72**	**0.79**
7.	**Naive Bayes**	**66.65**	**0.97**	**0.09**	**0.17**

Table 2.1: Classification Report of Testing Data of Web Page Phishing Detection

In the evaluation of the Testing data from the Web Page Phishing Dataset, the utilization of the TF-IDF vectorizer for feature extraction resulted in notable performance metrics for the Random Forest model. Specifically, it achieved an accuracy of 88.93%, showcasing its effectiveness in classification tasks. Furthermore, the model exhibited commendable recall, precision, and F1-score values of 0.86, 0.83, and 0.85 respectively, indicating its capability to accurately identify positive instances. Following this, the Support Vector Machine (SVM) also demonstrated promising results, securing an accuracy of 88.36% and positioning itself as the second-best performing model in the assessment.

b) Classification Metrics of Training Data

Classification Metrics With TF-IDF Vectorizer					
WEB PAGE PHISHING DETECTION					
S. No	ML Models	Accuracy	Precision	Recall	F1-Score
1.	**Support Vector Machine (SVM)**	**88.31**	**0.86**	**0.81**	**0.83**
2.	**Random Forest**	**92.45**	**0.91**	**0.88**	**0.89**
3.	**K-Nearest Neighbors (KNN)**	**90.18**	**0.88**	**0.85**	**0.86**
4.	**Gradient Boosting**	**88.37**	**0.84**	**0.84**	**0.84**
5.	**Decision Tree**	**92.45**	**0.92**	**0.87**	**0.89**
6.	**Logistic Regression**	**85.82**	**0.87**	**0.72**	**0.79**
7.	**Naive Bayes**	**66.86**	**0.97**	**0.09**	**0.16**

Table 2.2: Classification Report of Training Data of Web Page Phishing Detection

In the evaluation of the Training data from the Web Page Phishing Dataset, both the Random Forest and Decision Tree models achieved an impressive accuracy of 92.45%. They demonstrated commendable recall, precision, and F1-score values, with Random Forest at 0.91, 0.88, and 0.89 respectively, and Decision Tree at 0.92, 0.87, and 0.89 respectively. Additionally, the K-Nearest Neighbors (KNN) model showed promising results with an accuracy of 90.18%, positioning itself as the second-best performing model in the assessment.

2.12 Accuracy Bar Graph of Web Page Phishing Detection

In machine learning projects, a Bar Graph is a visual tool used to compare different groups or categories. It represents data with rectangular bars, where the length of each bar is proportional to the value it signifies.

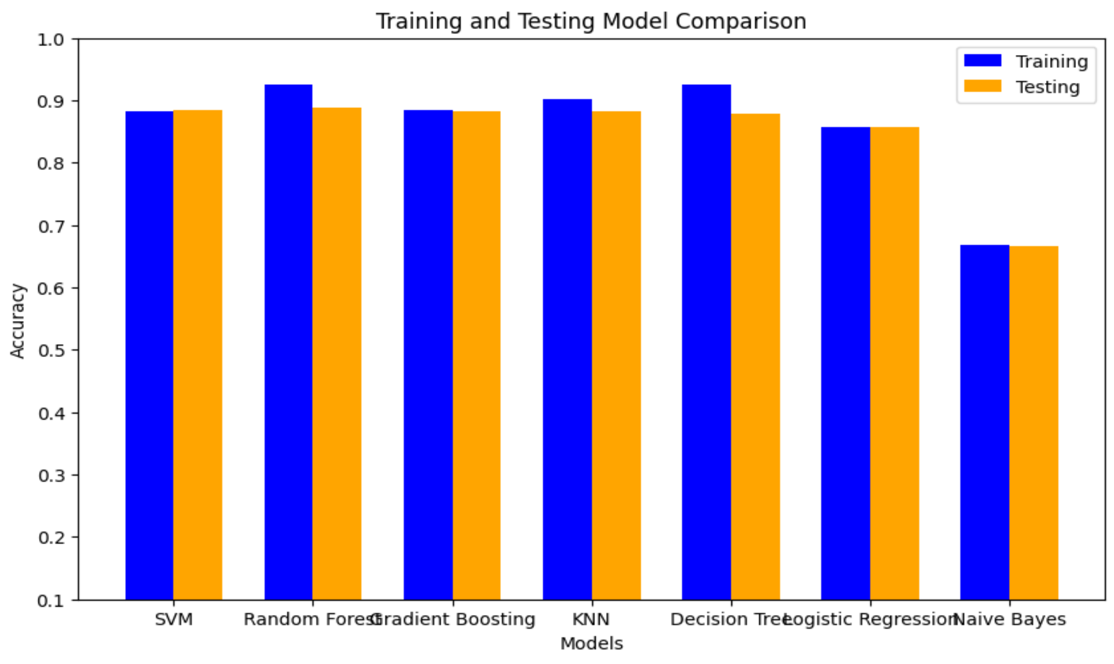
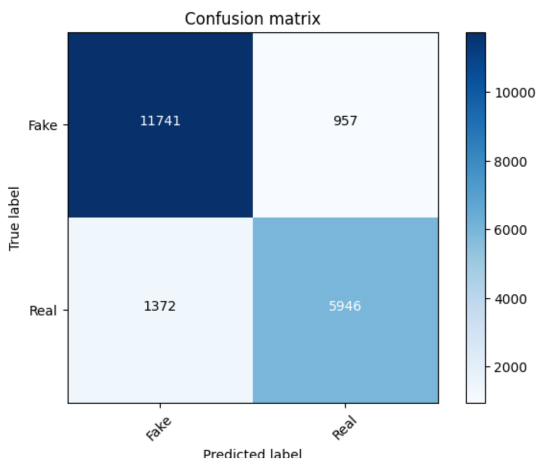


Fig 2.7: Accuracy Comparison of Testing and Training Data of Web Page Phishing Detection

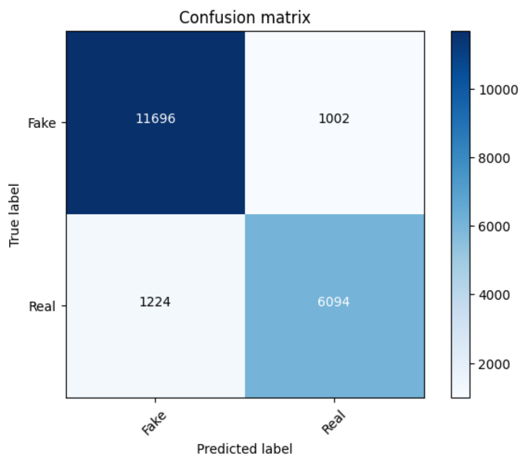
Here, a single bar graph effectively displays the accuracy of both the testing data and training data. This visual comparison aids in assessing the model's performance across different datasets.

2.13 Confusion Matrix for Each Model

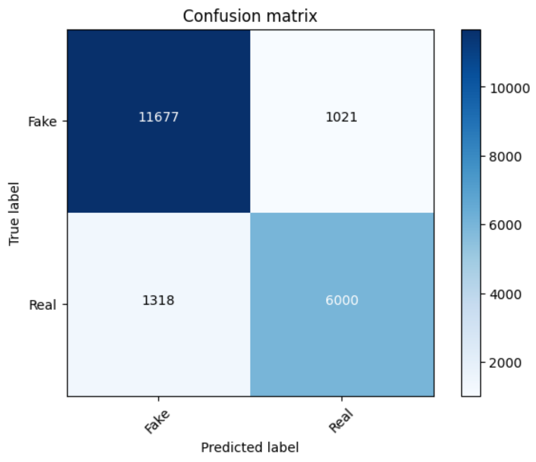
- Testing Data



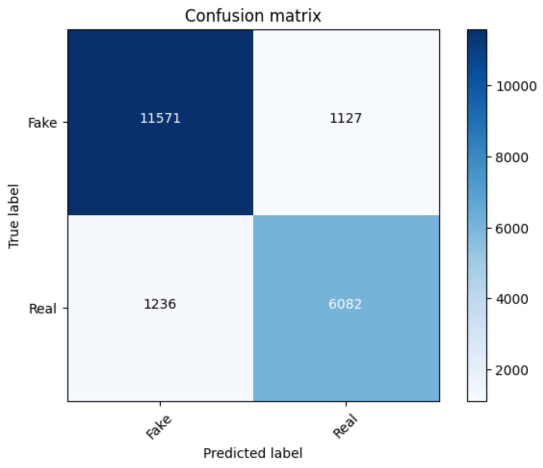
Model: SVM



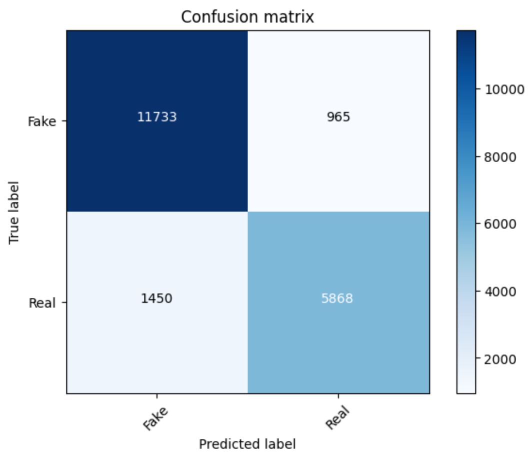
Model: Random Forest



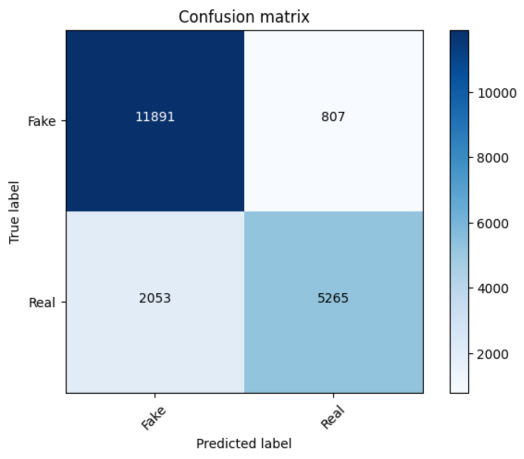
Model: KNN



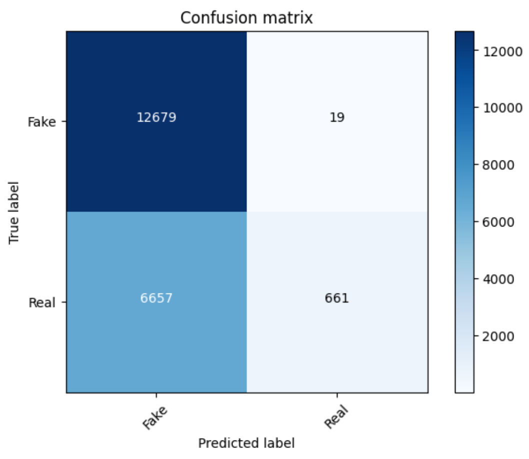
Model: Gradient Boosting



Model: Decision Tree



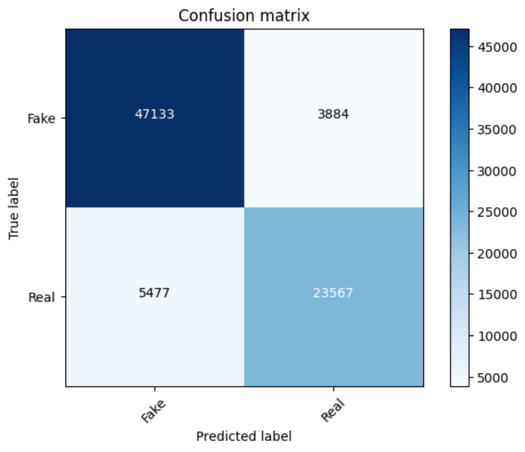
Model: Logistic Regression



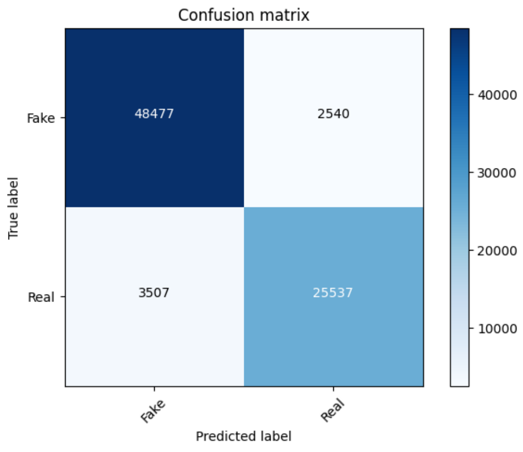
Model: Naïve Bayes

Fig 2.8: Confusion Matrix of Testing Data of Web Page Phishing Detection

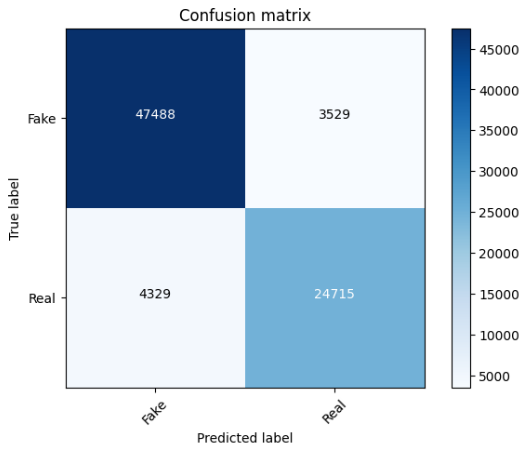
- Training Data



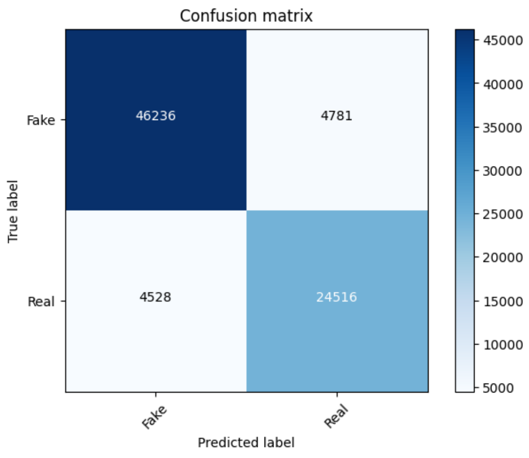
Model: SVM



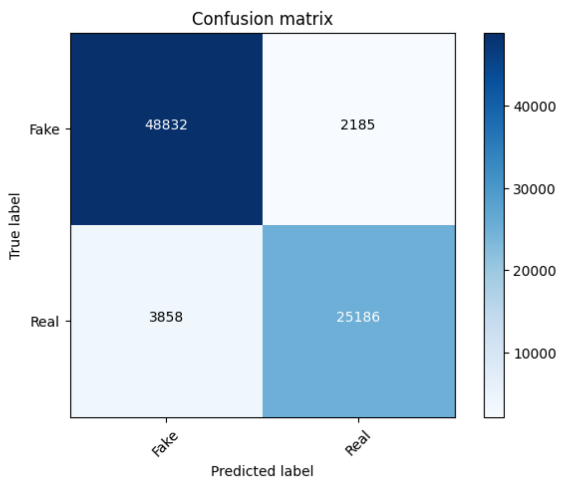
Model: Random Forest



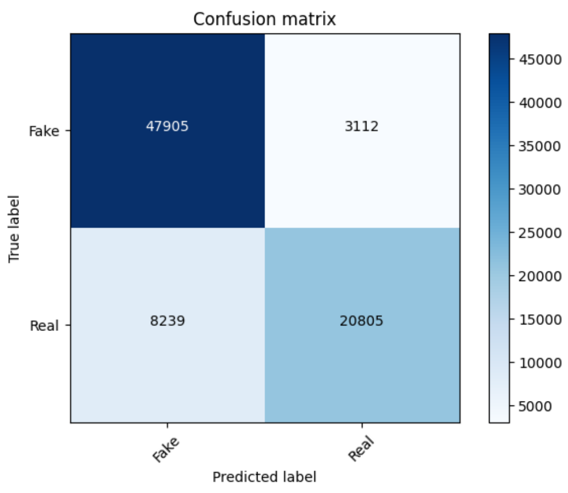
Model: KNN



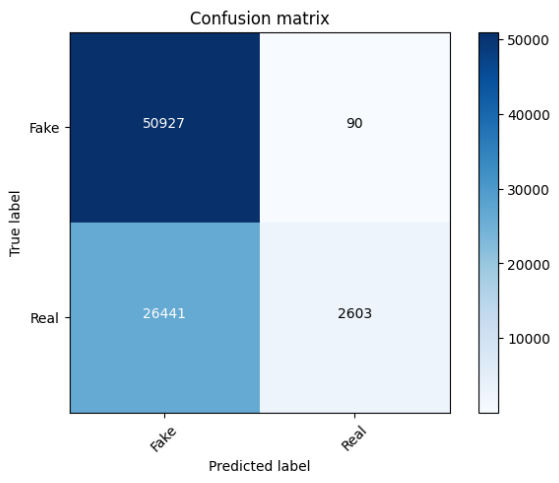
Model: Gradient Boosting



Model: Decision Tree



Model: Logistic Regression



Model: Naïve Bayes

Fig 2.9: Confusion Matrix of Training Data of Web Page Phishing Detection

2.14 AUC-ROC & F1-Score Curve

- **AUC-ROC CURVE**

The AUC-ROC curve is significant in machine learning for web page phishing detection as it measures a model's ability. A higher AUC indicates better model performance.

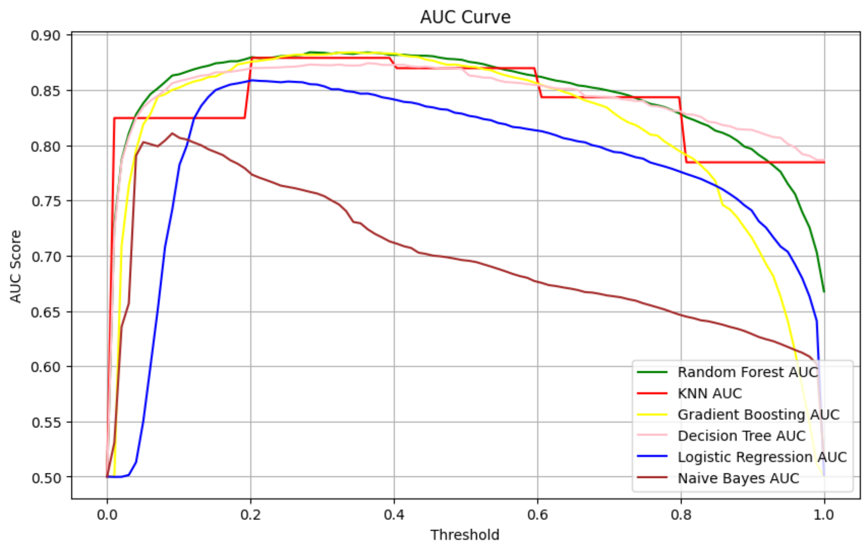


Fig 2.10: AUC-ROC Curve of Web Page Phishing Detection

- **F1-SCORE CURVE**

The F1-Score curve visually illustrates the trade-off between precision and recall as a model's decision threshold varies. It helps identify the optimal balance for robust detection.

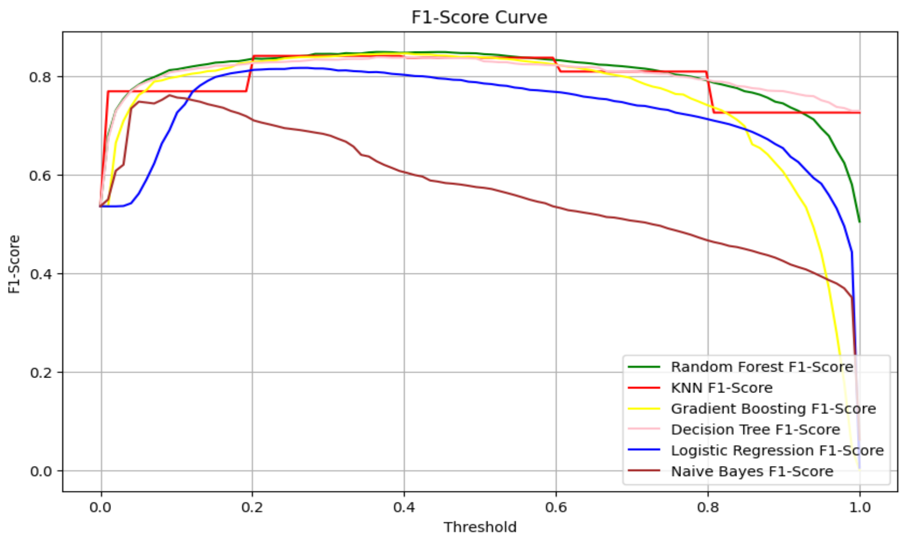


Fig 2.11: F1-Score Curve of Web Page Phishing Detection

2.15 Conclusion

In this web page phishing detection project, we embarked on a comprehensive journey from data collection to model evaluation. Initially, we obtained the dataset from Kaggle and embarked on data cleaning and preprocessing steps, ensuring the dataset was devoid of any missing values. Following this, we delved into exploratory data analysis, extracting various insights through data visualization techniques. These visualizations provided us with a deeper understanding of the dataset's characteristics and potential patterns.

Subsequently, we proceeded with feature extraction, aiming to distil the most relevant information from the dataset for our modeling endeavours. Once the data was adequately prepared, we split it into training and testing subsets, laying the foundation for our model training and evaluation.

We employed a diverse set of machines learning models, including Support Vector Machine (SVM), Random Forest (RF), K-Nearest Neighbors (KNN), Gradient Boosting (GB), Decision Tree (DT), Logistic Regression (LR), and Naive Bayes (NB). Through rigorous evaluation on the testing data, we observed that Random Forest consistently outperformed other models in terms of accuracy. Moreover, Decision Tree also demonstrated commendable performance, achieving the highest accuracy in the training data.

To delve deeper into model performance, we examined the confusion matrices and classification reports for Random Forest and Decision Tree, elucidating their predictive capabilities in classifying phishing and non-phishing web pages. Furthermore, we explored the models' performance using additional metrics such as ROC-AUC curves and F1-Score curves, providing insights into their trade-offs between true positive rate and false positive rate, as well as precision and recall.

Finally, we consolidated the training and testing data accuracies into a single bar graph, offering a holistic view of model performance across different datasets. Through this comprehensive analysis and evaluation, we gained valuable insights into the effectiveness of various machine learning models for web page phishing detection, paving the way for future enhancements and applications in cybersecurity.

2.16 Future Work

In the realm of web page phishing detection, there are several avenues for future exploration and refinement. Firstly, enhancing the dataset with more diverse and representative samples can improve model generalization and robustness. Additionally, integrating more advanced feature engineering techniques, such as deep learning-based methods or natural language processing approaches, can extract deeper insights from web page content and structure.

Furthermore, exploring ensemble methods to combine the strengths of multiple models could lead to further performance improvements. Techniques like model stacking or boosting can leverage the diversity of individual models to enhance overall predictive accuracy.

Moreover, incorporating real-time data streams and dynamic feature extraction mechanisms can enable proactive detection of emerging phishing threats. Techniques like online learning or incremental model updates can facilitate the continuous adaptation of models to evolving phishing tactics.

Lastly, deploying the developed models into production environments and conducting extensive real-world testing and validation is crucial for assessing their effectiveness and scalability. Continuous monitoring and refinement based on feedback from real-world usage can ensure the models remain effective in combating evolving phishing threats.

2.17 References

1. Ankit Kumar Jain, B. B. Gupta, "Phishing Detection: Analysis of Visual Similarity Based Approaches", Security and Communication Networks, vol. 2017, Article ID 5421046, 20 pages, 2017. https://doi.org/10.1155/2017/5421046

2. ACM Transactions on Internet Technology Volume 10, Issue 2, Article No.: 5, pp 1–38, https://doi.org/10.1145/1754393.1754394

3. J. Mao, W. Tian, P. Li, T. Wei and Z. Liang, "Phishing-Alarm: Robust and Efficient Phishing Detection via Page Component Similarity," in IEEE Access, vol. 5, pp. 17020-17030, 2017, doi: 10.1109/ACCESS.2017.2743528.

4. Ozgur Koray Sahingoz, Ebubekir Buber, Onder Demir, Banu Diri, Machine learning based phishing detection from URLs, Expert Systems with Applications, Volume 117, 2019, Pages 345-357, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2018.09.029.

5. Y. Pan and X. Ding, "Anomaly Based Web Phishing Page Detection," 2006 22nd Annual Computer Security Applications Conference (ACSAC'06), Miami Beach, FL, USA, 2006, pp. 381-392, doi: 10.1109/ACSAC.2006.13.

6. Liu Wenyin, Guanglin Huang, Liu Xiaoyue, Xiaotie Deng and Zhang Min, "Phishing Web page detection," Eighth International Conference on Document Analysis and Recognition (ICDAR'05), Seoul, South Korea, 2005, pp. 560-564 Vol. 2, doi: 10.1109/ICDAR.2005.190.

7. Ping Yi, Yuxiang Guan, Futai Zou, Yao Yao, Wei Wang, Ting Zhu, "Web Phishing Detection Using a Deep Learning Framework", Wireless Communications and Mobile Computing, vol. 2018, Article ID 4678746, 9 pages, 2018. https://doi.org/10.1155/2018/4678746