A Project Report Entitled

Banknote Authentication Using ML Algorithms

Project Report submitted to

RAYAT SHIKSHAN SANSTHA'S
SADGURU GADAGE MAHARAJ COLLEGE, KARAD
(An Autonomous College)
DEPARTMENT OF STATISTICS

AS PARTIAL FULFILLMENT FOR AWARD OF THE DEGREE
MASTER OF SCIENCE
IN
STATISTICS

SUBMITTED BY,
Miss. Prajakta Jaywant Bille .
M.Sc II (Statistics)

Under the guidance of
Mrs. A. S. Patil
2023-2024

CERTIFICATE

This is to certify that the project report entitled ‘‘ Banknote Authentication Using Machine Learning Algorithms.**”** being Submitted by **Miss. Prajakta Jaywant Bille** as a partial fulfillment for the award of degree of M.Sc. II (Statistics) is a record of work carried out by him under my supervision and guidance. To the best of my knowledge the matter presented in the survey has not been submitted earlier.

Place: Karad
Date :

Teacher in charge Examiner PG Co-ordinator Head
Department of Statistics

Index

Introduction

Counterfeit banknotes are fake replicas of the real one which seems to be perfect banknote until watched and verified carefully. The most prominent threat to banknotes is advancement of technology in the field of digitization, scanning and printing techniques. Preserving genuinely of any currency is one of the critical issues all over the world. The currencies are printed in various denominations in every country and it is a most important issue among all. Craving to hegemonies financial market and knock down particular country's economy, some of the culprits actuate forged banknotes into the market. It is very difficult to identify from naked eye which banknote is genuine and fake, chiefly when banknotes are in a bulk and all notes have apparently homogeneous features. Hence there is a need of a system which can accurately classify the fake note from bunch of genuine notes.

Despite a decrease in the use of currency due to the recent growth in the use of electronic transactions, cash transactions remain very important in the global market. Banknotes are used to carry out financial activities. To continue with smooth cash transactions, entry of forged banknotes in circulation should be preserved. There has been a drastic increase in the rate of fake notes in the market. Fake money is an imitation of the genuine notes and is created illegally for various motives. It is difficult for human-eye to recognize a fake note because they are created with great accuracy to look alike a genuine note. Security aspects of banknotes have to be considered and security features are to be introduced to mitigate fake currency. Hence, there is a dire need in banks and ATM machines to implement a system that classifies a note as genuine or fake.

The digitization of financial transactions has led to an increased demand for secure methods of banknote authentication. In response to this need, this project proposes a novel approach to banknote authentication utilizing statistical features extracted from banknote images. By analyzing key statistical properties such as variance, skewness, kurtosis, and entropy, this project aims to develop a robust model capable of accurately distinguishing between authentic banknotes

and counterfeit ones. Machine learning techniques offer a promising solution by automating the authentication process and providing rapid results with high accuracy. Through the implementation of machine learning algorithms and statistical analysis, this project seeks to contribute to the advancement of automated banknote authentication systems. The proposed model will not only enhance the efficiency of financial transactions but also help combat counterfeit activities, thereby fostering trust and security in the financial sector.

Furthermore, the deployment of the authentication model through a Flask-based API, integrated with Swagger for streamlined interaction, will enable seamless integration into existing financial systems, making it accessible and easy to use for various applications.

Objectives

1) To develope a predictive model for banknote authentication dataset.

2) To evaluate various machine learning algorithms.

3) To evaluate the effectiveness of the classification model by assessing metrics such as accuracy, precision, recall, and F1 score, ensuring its reliability in identifying and categorizing banknotes.

4) To form API on the basis of best fitted classification model for banknote authentication.

Methodology

1)Data Collection: Acquired a dataset containing statistical features such as variance, skewness, kurtosis, and entropy of banknotes. Data were extracted from images that were taken for the evaluation of an authentication procedure for banknotes.

2)Data Preprocessing: Conducted preprocessing steps specific to the statistical features dataset, such as ensuring data consistency, handling missing values, deal with duplicates, and normalizing or scaling the features to a common range to facilitate model training.

3)Exploratory data Analysis (EDA): Performed exploratory data analysis to understand the distribution, range, and statistical properties of the features. This involved visualizations, summary statistics, and correlation analysis to uncover any patterns or anomalies within the dataset.

4)Model Development: Developed classification models utilizing machine learning algorithms suitable for the dataset, such as Logistic regression, Naïve bayes, Decision trees, or ensemble methods like Random Forests. The models were trained to predict the authenticity of banknotes based on the selected statistical features.

4) Model Training and evaluation: Split the dataset into training and testing sets to train the models on a subset of data and evaluate their performance on unseen data. Used performance metrics like accuracy, precision, recall, and F1-score to assess model performance.

5) Validation and Cross Validation: Utilized techniques such as k-fold cross-validation to validate model performance across multiple subsets of the data, ensuring robustness and reliability.

6) Model Interpretation: The process of understanding and explaining the behavior and decisions made by a machine learning model.

7) API : Integrating the trained model into a real-world application or system where it can be used for banknote authentication tasks.

Tools and Techniques Used

- **Techniques:**

  Machine learning classification models:

  1) Naïve Bayes
  2) Logistic Regression
  3) Decision Tree Classifier
  4) Random Forest

- **Tools:**

  Jupyter Notebook

Data Description

Data were extracted from images that were taken from genuine and forged banknote-like specimens. For digitization, an industrial camera usually used for print inspection was used. The final images have 400 x 400 pixels. Due to the object lens and distance to the investigated object gray-scale pictures with a resolution of about 660 dpi were gained. Wavelet Transform tool were used to extract features from images.

The Dataset consist 1372 rows and 5 parameters.
The dataset is taken from UCI repository:
(https://archive.ics.uci.edu/dataset/267/banknote+authentication)

The columns are as follows:

Attribute Name	Value Type	Description
Variance of Wavelet Transformed Image	Continuous	Variance is the measure of how a pixel varies from its neighboring pixels
Skewness of Wavelet Transformed Image	Continuous	Skewness measures how asymmetrical the image is
Kurtosis of Wavelet Transformed Image	Continuous	Kurtosis is a measure of whether the data is heavy tailed or light-tailed relative to a normal distribution
Entropy of Image	Continuous	The entropy or average information of an image is a measure of the degree of randomness in the image

Class	Integer	Class 0 represents genuine notes whereas class 1 represents counterfeit notes

The following table shows the first five rows from the dataset.

:

	variance	skewness	curtosis	entropy	class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

There are 1372 rows and 5 columns in the above dataset.

SUMMARY STATISTICS:

After collecting the data, we further process to understand the data. The following table shows us the information about the dataset.

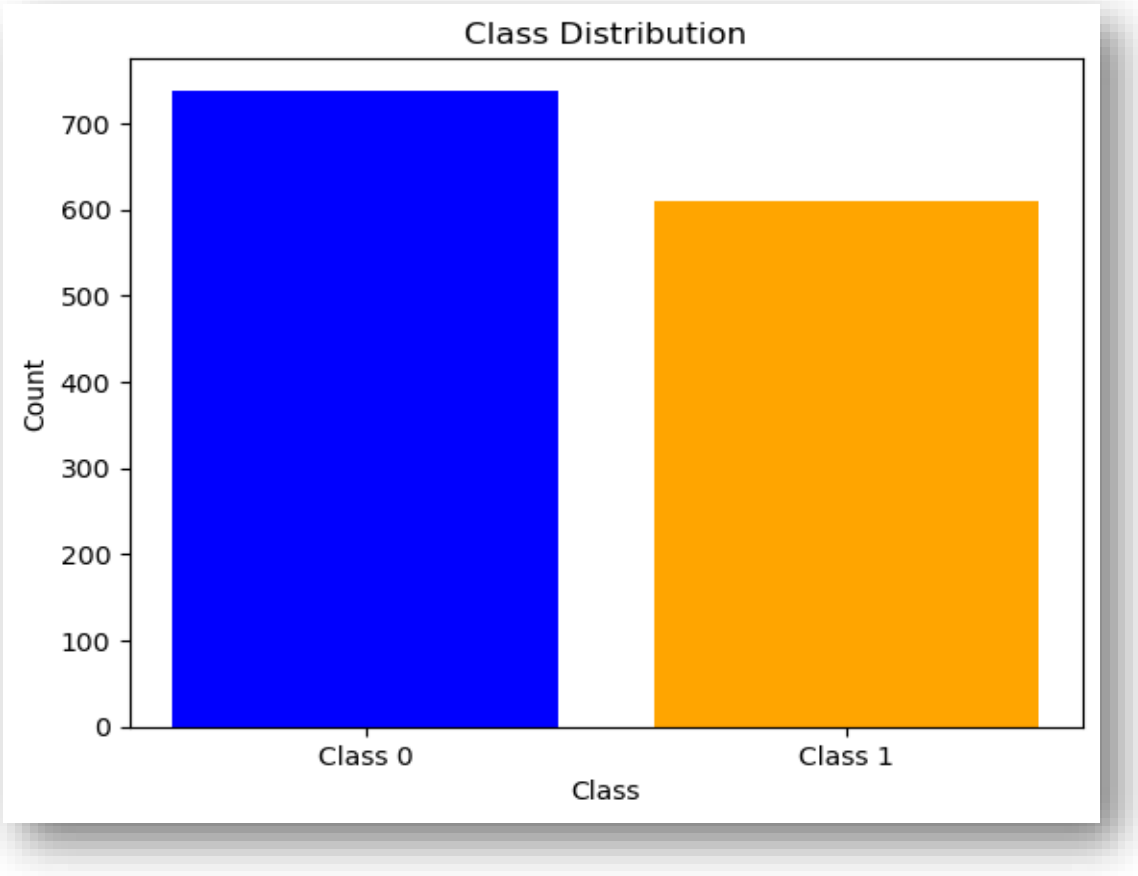
	variance	skewness	curtosis	entropy	class
count	1372.000000	1372.000000	1372.000000	1372.000000	1372.000000
mean	0.433735	1.922353	1.397627	-1.191657	0.444606
std	2.842763	5.869047	4.310030	2.101013	0.497103
min	-7.042100	-13.773100	-5.286100	-8.548200	0.000000
25%	-1.773000	-1.708200	-1.574975	-2.413450	0.000000
50%	0.496180	2.319650	0.616630	-0.586650	0.000000
75%	2.821475	6.814625	3.179250	0.394810	1.000000
max	6.824800	12.951600	17.927400	2.449500	1.000000

We can see that values vary with different means and standard deviations, some normalization or standardization would be required prior to modeling.

Exploratory Data Analysis

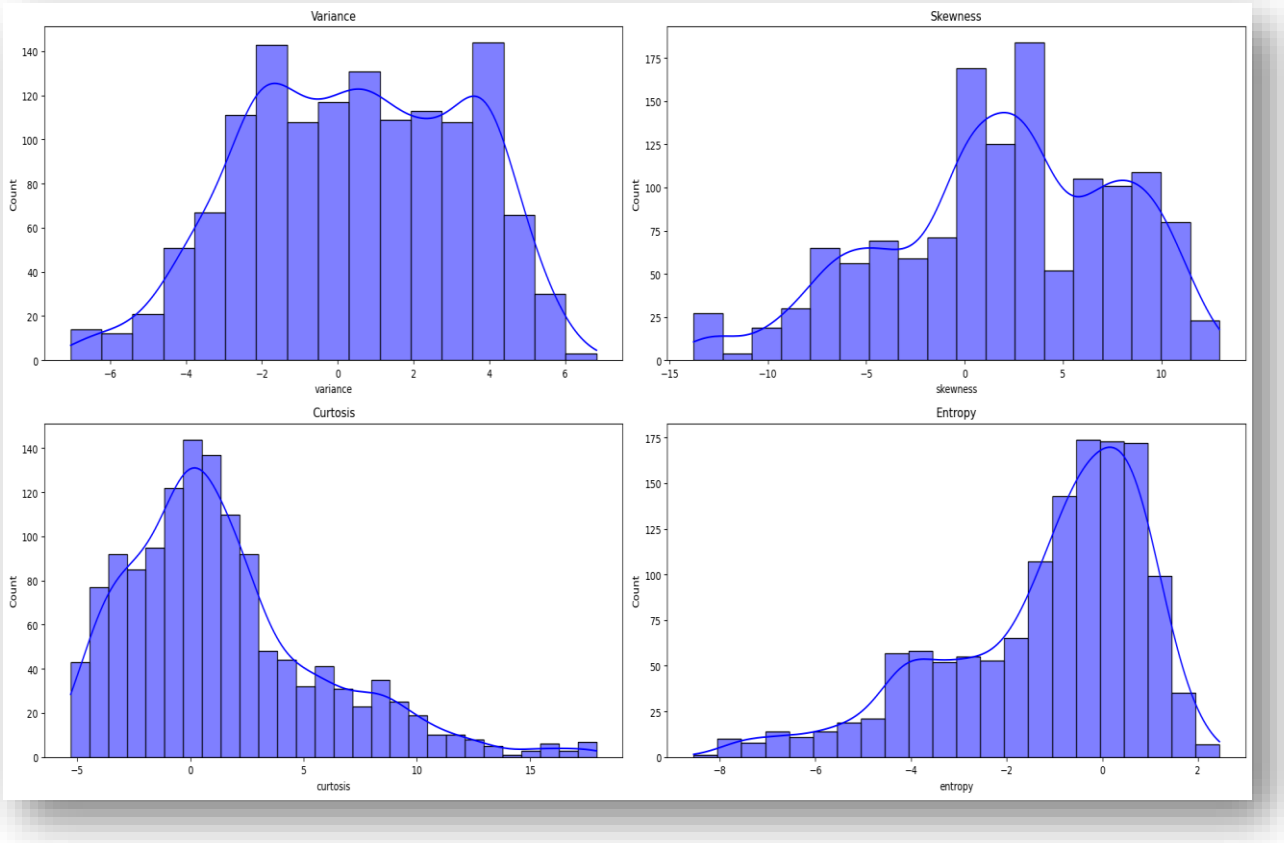
Exploratory Data Analysis (EDA) is a crucial initial step in the data analysis process that aims to understand the underlying structure, patterns, and relationships within a dataset. It involves summarizing the main characteristics of the data, often using statistical and visual methods, to gain insights that inform subsequent analysis and modeling decisions. By visually exploring and summarizing the data, EDA enables a deeper understanding of its characteristics, helps identify potential issues such as outliers or missing values, and informs subsequent preprocessing and modeling decisions. Moreover, EDA fosters hypothesis generation and guides the selection of appropriate analytical techniques, ultimately leading to more robust and interpretable results. In essence, conducting EDA is essential to unlock the full potential of the data and drive informed decision-making processes.

1) **Bar Plot to study the target variable:**



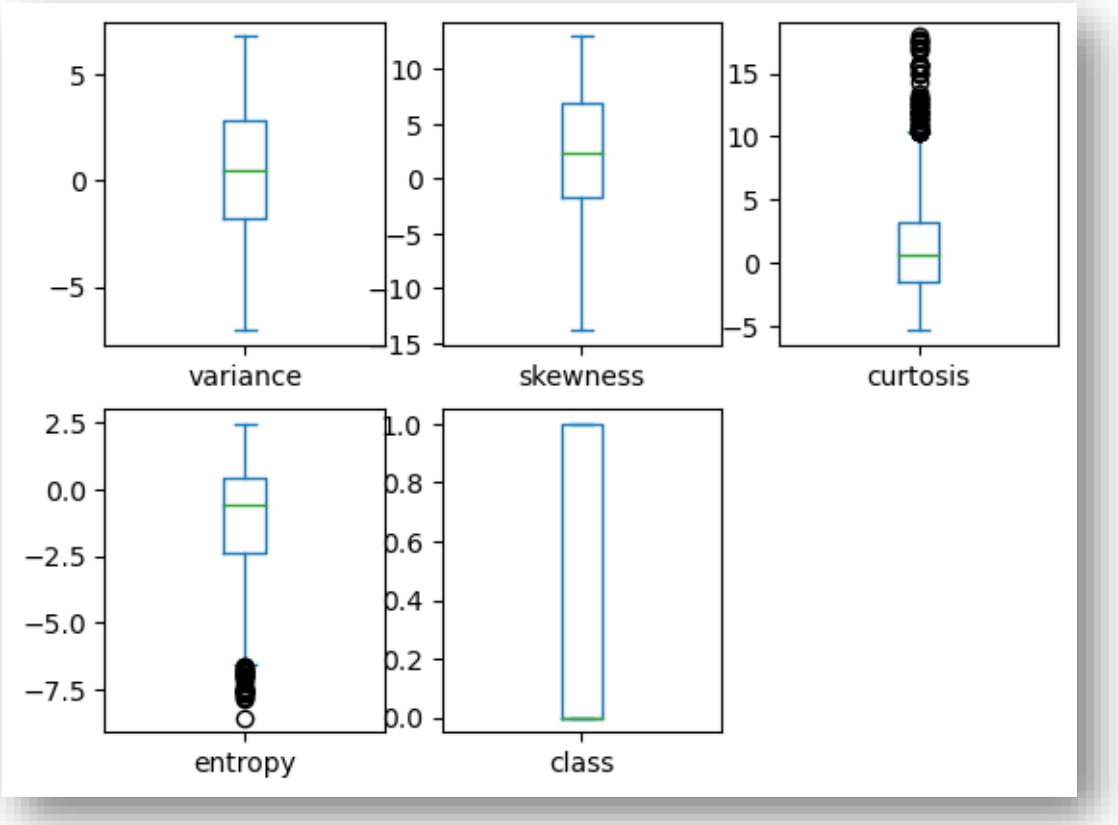
From the above bar plot we can clearly see the data is balanced data.

2) Histplot for features in the data:



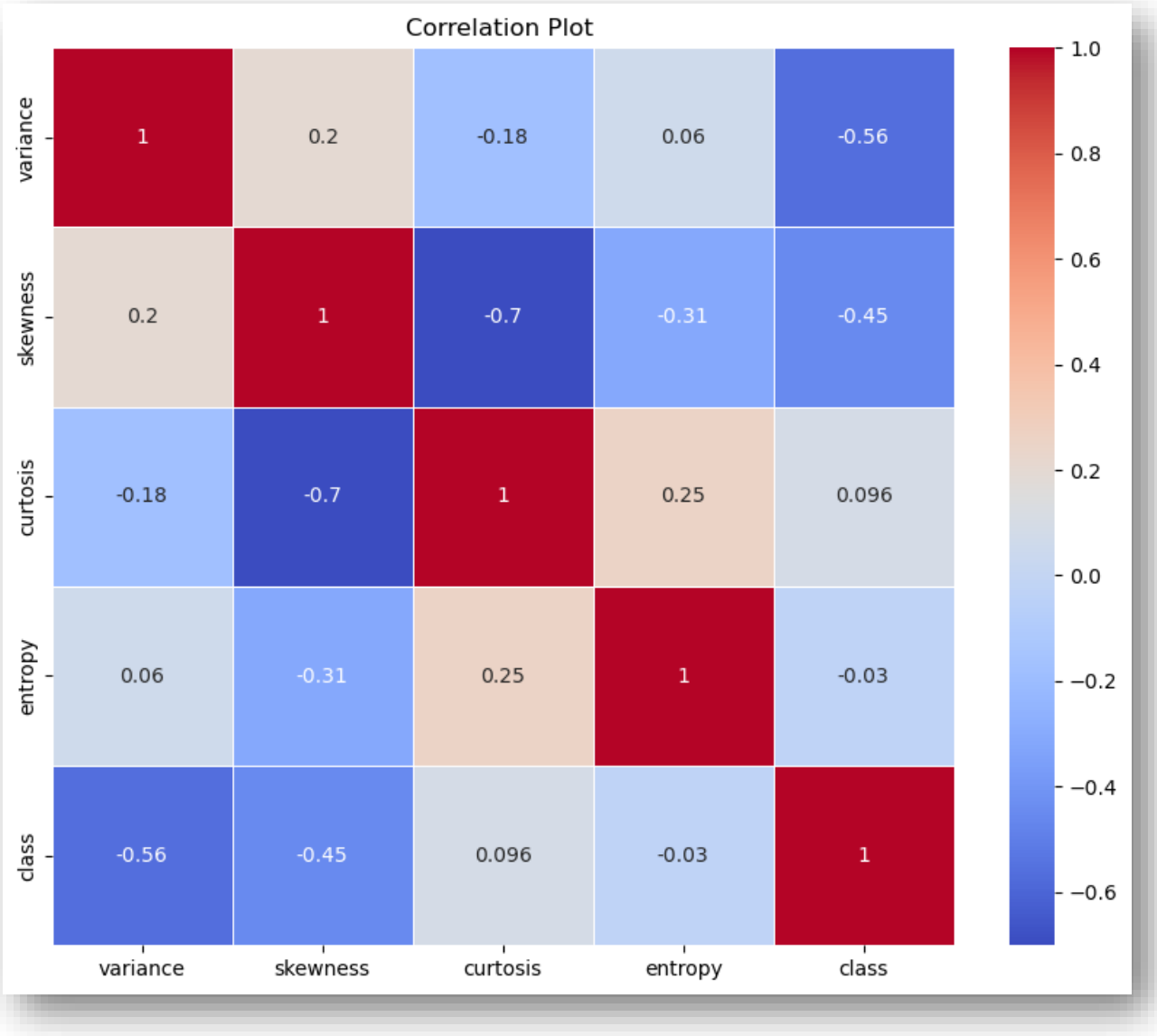
We can see that the first two variables have a Gaussian-like distribution and the next two input variables may have a skewed Gaussian distribution or an exponential distribution. They may be outliers in entropy and curtosis variable and data is not normalized. To be sure, we're going to see at using the box plot.

3)Box-Plot:



The entropy and kurtosis variable have an outlier. This should be handled or cleaned in the data preparation step. The examining the proportion of target variable (class variable) is balance.

4)Correlation Plot :



The above plot shows the correlation between the dataset of every column with every other column.

Data Preprocessing

- **Removal of Duplicate rows :**
  There are 24 duplicate rows with identical feature values. These can be safely removed using the drop duplicates () function in pandas.

- **Handling the outliers:**
  As we have clearly seen in the above box plot that there are so many outliers present in curtosis and entropy. Now we have to handle outliers in order to make the data clean.

1) Calculating Z-Scores:

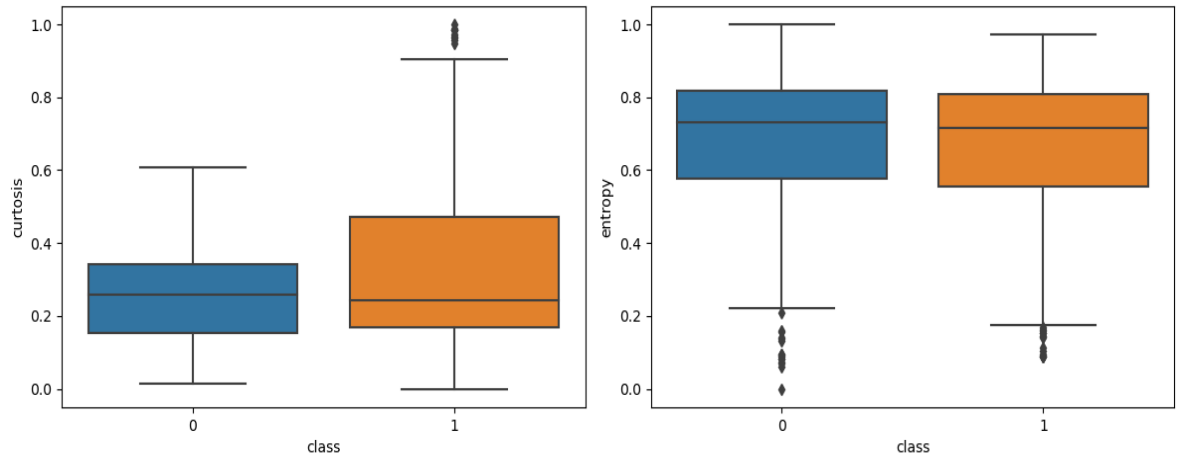
- The first step is to calculate the z-scores for a specific numerical feature in the dataset. For example you're calculating z-scores for the 'variance' column.
- Z-scores measure how many standard deviations an observation is from the mean of the feature. It indicates how unusual or extreme a particular observation is relative to the rest of the dataset.
- The formula for calculating the z-score of an observation x in a feature is : z= $\frac{x-mean(x)}{std(x)}$
- Using this formula calculate the z-score for each feature in the data.

2) Replacing Outliers with Median:

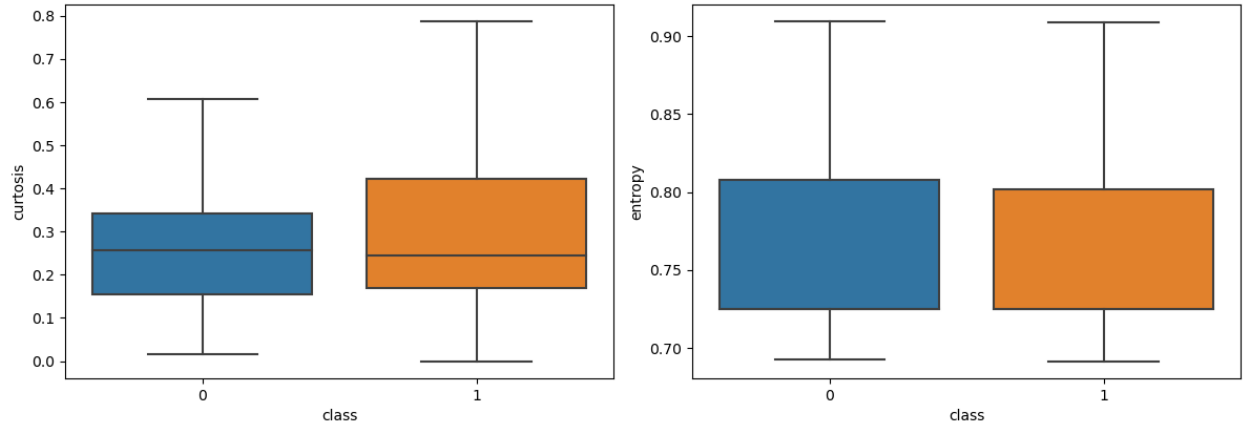
- The next step is to identify outliers based on the calculated z-scores.
- Once outliers are identified, they are replaced with the median value of the feature. Replacing with the median is a common technique for handling extreme values without significantly affecting the overall distribution of the data.

Overall, this technique involves calculating z-scores for a numerical feature, identifying outliers and replacing outliers with the median value.

Before Handling outliers:



After Handling outliers:



Here we can clearly see that the outliers present in the dataset were removed and now the data is cleaned.

<u>Feature Scailing</u>: Ensuring that all features have the same scale to prevent certain features from dominating others. The technique we have used here is StandardScalar. The StandardScalar is a preprocessing technique used to standardize features by removing the mean and scailing to unit variance. It transforms the data such that it has a mean of 0 and a standard deviation of 1.

Model Building

Model building involves training and evaluating predictive models to extract insights or make predictions from data. It encompasses data preprocessing, feature engineering, algorithm selection, training, evaluation, and deployment. Models are trained using various machine learning algorithms. Evaluation metrics like accuracy and precision assess model performance. Fine-tuning involves adjusting hyperparameters for optimal results. Finally, selected models are deployed into production for real-world applications. Continuous monitoring and updating ensure models remain effective over time. Model building is a pivotal phase in any data-driven project, where theoretical constructs meet practical application. It involves the creation and refinement of mathematical or computational representations that capture patterns and relationships within the data. This process transforms raw data into actionable insights, guiding decision-making and predicting future outcomes. Through meticulous experimentation and validation, models evolve, adapting to the intricacies of the data and the nuances of the problem at hand. From simple linear regressions to complex deep learning architectures, each model is crafted with precision, aiming to uncover hidden insights and empower organizations to make informed decisions. In this journey from data to knowledge, model building serves as a beacon of intelligence, illuminating the path towards understanding and discovery.

1)Naïve Bayes:

Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem with the assumption of independence between features. For banknote authentication, Naive Bayes calculates the probability that a banknote is authentic or counterfeit given its features like variance, skewness, kurtosis, and entropy. Despite its simplicity and assumption of feature independence, Naive Bayes can perform well on datasets with categorical or continuous features. It's computationally efficient and can handle large datasets. However, its performance may degrade if the independence assumption is violated or if features are highly correlated.

One of the key advantages of Naive Bayes is its interpretability; it provides clear probabilistic explanations for its predictions. Furthermore, Naive Bayes is sensitive to the quality of the training data, particularly in the presence of missing values or outliers.

Overall, Naive Bayes is a powerful and efficient algorithm for banknote authentication, offering a balance between simplicity, speed, and interpretability. With careful consideration of its assumptions and limitations, Naive Bayes can be an effective choice for classification tasks in various domains.

2)Logistic Regression:

Logistic Regression is a fundamental and widely-used classification algorithm, well-suited for binary classification tasks such as banknote authentication. Unlike linear regression, which predicts continuous values, logistic regression models the probability that a given observation belongs to a particular class. In the context of banknote authentication, logistic regression estimates the probability that a banknote is authentic based on its features like variance, skewness, kurtosis, and entropy. One of the main advantages of logistic regression is its simplicity and interpretability. The model provides coefficients for each feature, indicating their impact on the predicted probability of authenticity. Additionally, logistic regression is computationally efficient, making it suitable for large datasets. It can handle both numerical and categorical features, allowing for flexibility in model specification. Logistic regression is also sensitive to outliers and multicollinearity, which can affect model performance.

Overall, logistic regression is a versatile and powerful algorithm for banknote authentication, offering a balance between simplicity, interpretability, and predictive performance. With appropriate feature selection and model tuning, logistic regression can effectively classify banknotes as authentic or counterfeit, providing valuable insights for fraud detection and risk management.

3)Decision Tree:

Decision Trees are powerful non-parametric supervised learning

algorithms used for classification and regression tasks. In the context of banknote authentication, Decision Trees recursively partition the feature space into subsets based on feature values, making sequential decisions that lead to the prediction of banknote authenticity. Each internal node of the tree represents a decision based on a feature, while each leaf node represents the predicted class (authentic or counterfeit).

One of the main advantages of Decision Trees is their interpretability. The resulting tree structure provides clear insights into the decision-making process, allowing users to understand the criteria used for classification. Additionally, Decision Trees can handle both numerical and categorical data, making them versatile for various types of datasets. They are robust to outliers and can capture non-linear relationships between features and the target variable.

However, Decision Trees are prone to overfitting, especially with deep trees or complex datasets. Overfitting occurs when the model learns the training data too well and fails to generalize to unseen data. Techniques like pruning, limiting tree depth, or using ensemble methods like Random Forests can mitigate overfitting and improve model performance.

Overall, Decision Trees are effective models for banknote authentication, offering a balance between interpretability and predictive performance. With appropriate tuning and regularization, Decision Trees can accurately classify banknotes as authentic or counterfeit, aiding in fraud detection and risk management.

Top of Form

4)Random Forest:

Random Forest is a powerful ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) or the mean prediction (regression) of the individual trees. In the context of banknote authentication, Random Forest combines the predictions of multiple decision trees to improve accuracy and reduce overfitting compared to a single decision tree.

One of the main advantages of Random Forest is its ability to handle high-dimensional feature spaces efficiently. By aggregating the predictions of multiple trees, Random Forest reduces variance and

increases model robustness, resulting in improved generalization performance. Additionally, Random Forest is less prone to overfitting compared to individual decision trees, making it suitable for complex datasets with noisy or unbalanced classes.

Random Forest can handle both numerical and categorical features and is robust to outliers and missing values. It automatically selects feature subsets at each split, further improving model diversity and predictive performance. However, Random Forest models are less interpretable compared to individual decision trees due to the ensemble nature of the algorithm.

Overall, Random Forest is a versatile and effective algorithm for banknote authentication, offering a balance between accuracy, robustness, and scalability. With proper parameter tuning and cross-validation, Random Forest can accurately classify banknotes as authentic or counterfeit, providing valuable insights for fraud detection and risk management.

Overall Summary of Classification Models:

The classification report provides a comprehensive evaluation of the performance of a classification model. Here's how to interpret the various metrics:

- **Precision:**
  Precision measures the accuracy of positive predictions made by the model. It is calculated as the ratio of true positive predictions to the total number of positive predictions .

  $$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives+False Positives}}$$

- **Recall:**
  Recall measures the ability of the model to correctly identify positive instances from all actual positive instances.

  $$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives+False Negatives}}$$

- **Specificity:**
  Specificity measures the ability of the model to correctly identify negative instances from all actual negative instances.

  $$\text{Specificity} = \frac{\text{True Negatives}}{\text{True Negatives+True Positives}}$$

- **Sensitivity:**
  Measures the proportion of actual positive instances that are correctly identified as positive by the model.

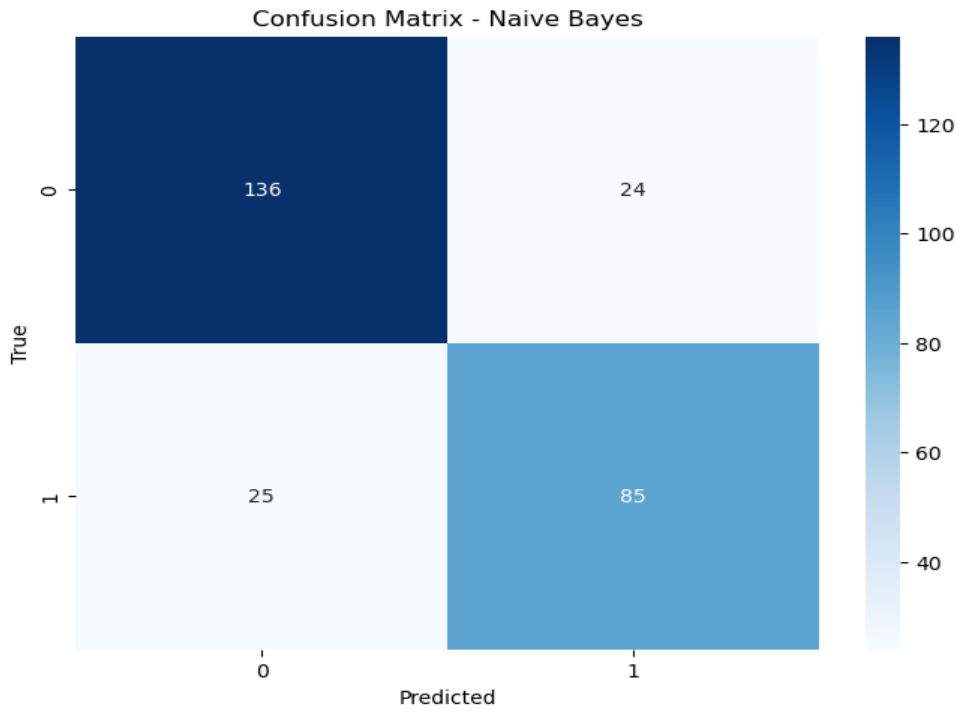
  $$\text{Sensitivity} = \frac{\text{True Positives}}{\text{True Positives+False Negatives}}$$

- **Accuracy:**
  Accuracy measures the overall correctness of the model's predictions. It is calculated as the ratio of correctly predicted instances to the total number of instances.

  $$\text{Accuracy} = \frac{\text{True Positives+True Negatives}}{\text{True Positives+False Positives+True Negatives+False Negatives}}$$

1)Naïve Bayes:

```
Accuracy: 0.8185185185185185
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.85      0.85       160
           1       0.78      0.77      0.78       110

    accuracy                           0.82       270
   macro avg       0.81      0.81      0.81       270
weighted avg       0.82      0.82      0.82       270
```

Interpretation:

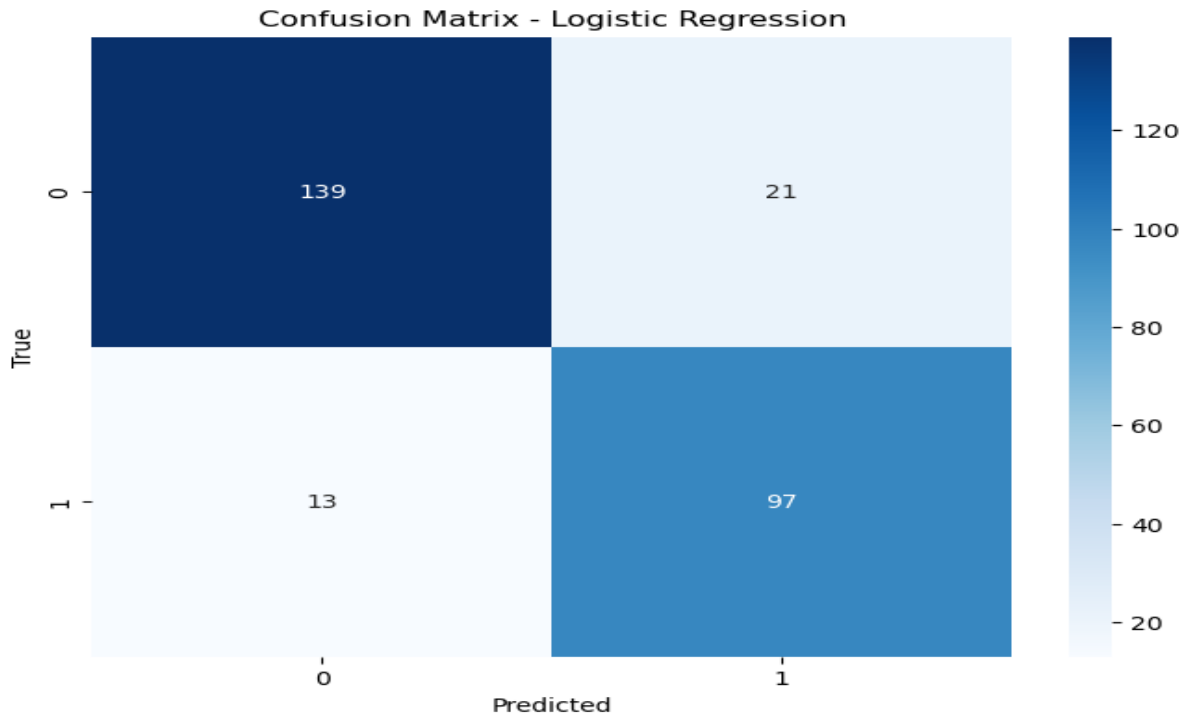
1)In this report, for class 0, the precision is 0.84, meaning that out of all the banknotes predicted to be genuine (class 0), 84% were actually genuine. For class 1, out of all the banknotes predicted to be fake (class 1) 78% were actually fake.

2) For class 0, the recall is 0.85, indicating that the model correctly identified 85% of all actual genuine banknotes. For class 1 recall is 77%.

3) The overall accuracy of the model is 0.82 or 82%, indicating that the model correctly classified 82% of all banknotes.

2)Logistic Regression:

```
Accuracy: 0.8740740740740741
```



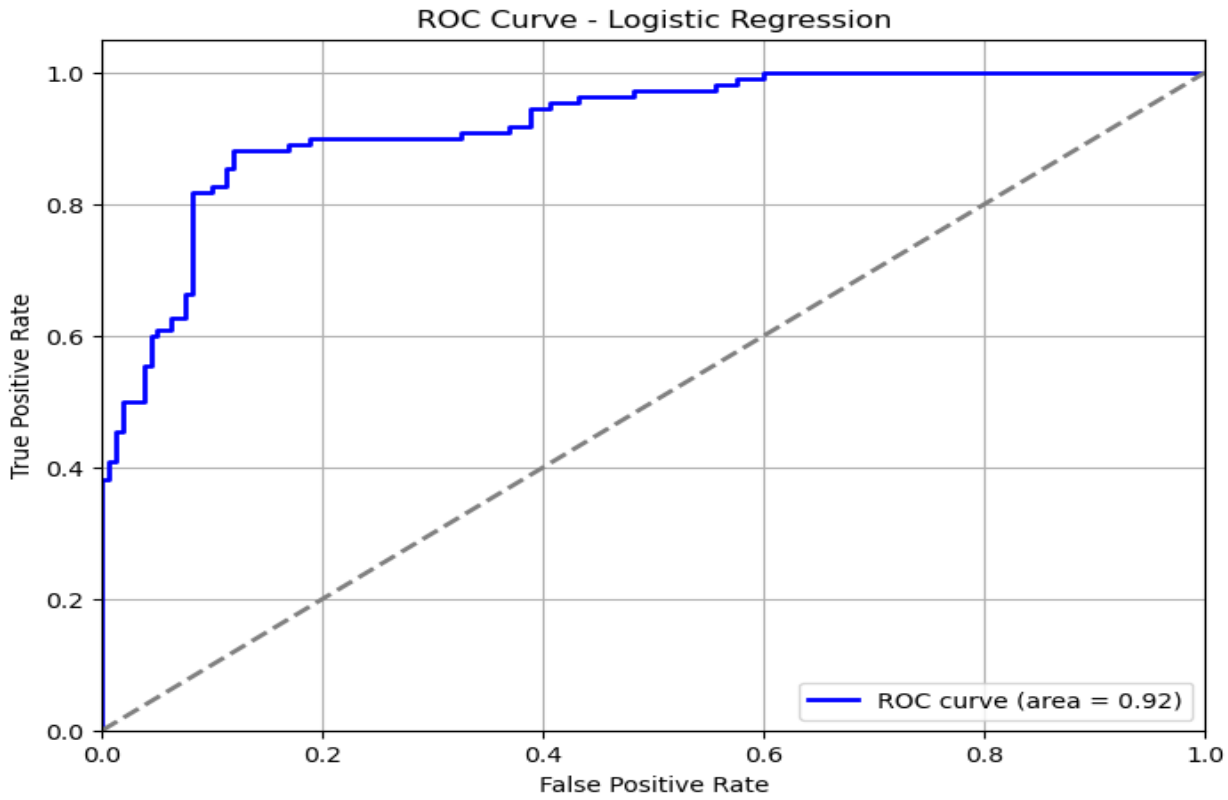

```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.87      0.89       160
           1       0.82      0.88      0.85       110

    accuracy                           0.87       270
   macro avg       0.87      0.88      0.87       270
weighted avg       0.88      0.87      0.87       270
```

Interpretation:

1)In this report, for class 0, the precision is 0.91, meaning that out of all the banknotes predicted to be genuine (class 0), 91% were actually genuine. For class 1, out of all the banknotes predicted to be fake (class 1) 82% were actually fake.

2) For class 0, the recall is 0.87, indicating that the model correctly identified 87% of all actual genuine banknotes. For class 1 recall is 88% indicating that the model correctly identified 88% of all actual fake banknotes.

3) The overall accuracy of the model is 0.87 or 87%, indicating that the model correctly classified 87% of all banknotes.

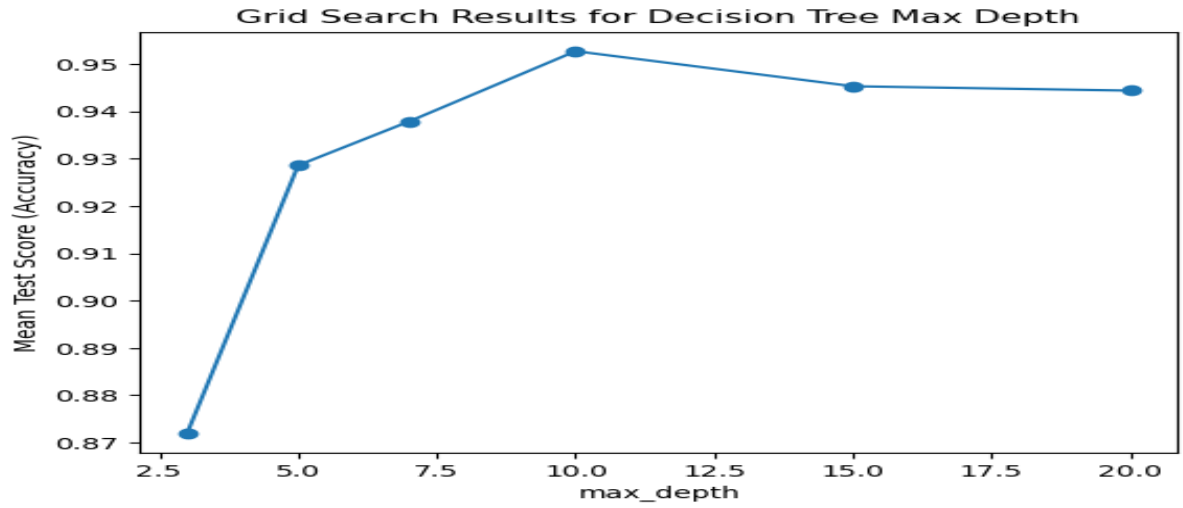


Interpretation:

An AUC of 0.92 indicates that the model has excellent discrimination ability in distinguishing between the positive and negative classes.

3)Decision Tree:

Grid Search:

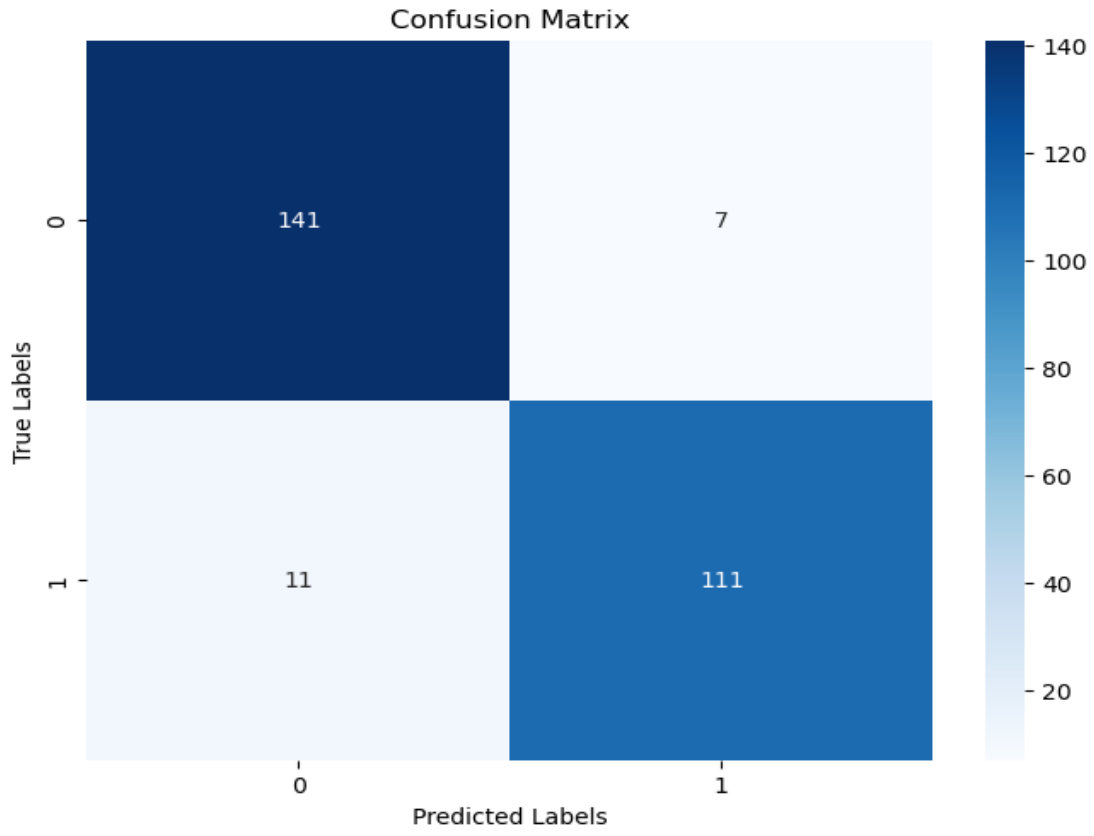


Best Parameters: {'max_depth': 10}

Hence we take max depth 10 for building the decision tree.

Accuracy: 0.93316108

Confusion Matrix:



Classification Report:

	precision	recall	f1-score	support
0	0.93	0.95	0.94	148
1	0.94	0.91	0.93	122
accuracy			0.93	270
macro avg	0.93	0.93	0.93	270
weighted avg	0.93	0.93	0.93	270

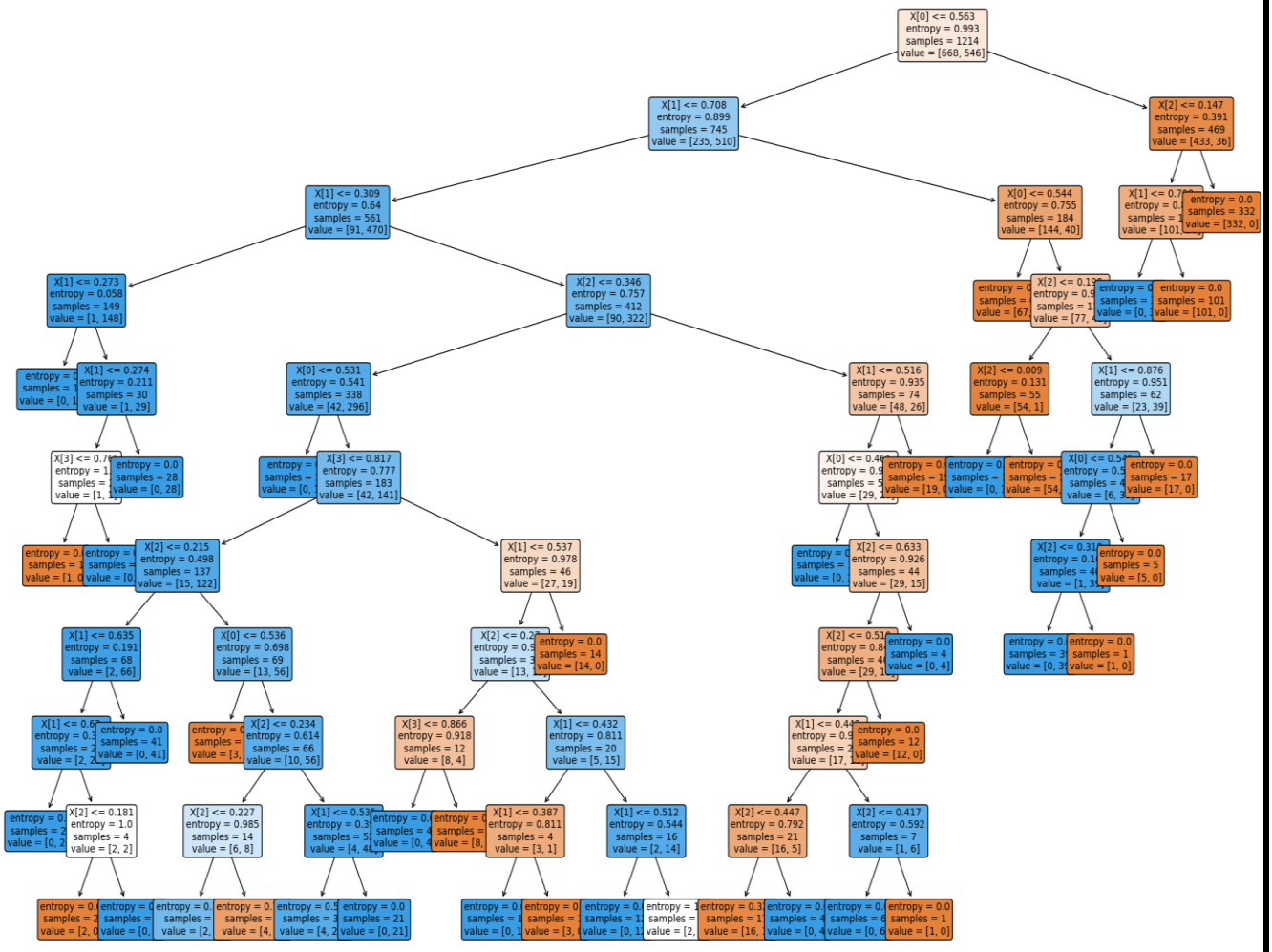
Interpretation:

1)In this report, for class 0, the precision is 0.93, meaning that out of all the banknotes predicted to be genuine (class 0), 93% were actually genuine. For class 1, out of all the banknotes predicted to be fake (class 1) 94% were actually fake.

2) For class 0, the recall is 0.95, indicating that the model correctly identified 95% of all actual genuine banknotes. For class 1 recall is 0.91

indicating that the model correctly identified 91% of all actual fake banknotes.
3) The overall accuracy of the model is 0.93 or 93%, indicating that the model correctly classified 93% of all banknotes.

- The decision tree we that we have build looks like this:



In the above tree there are some terms Entropy, Samples & Values which are defined or given as follows:

Entropy

- Entropy is a measure of impurity or randomness in a set of data. In the context of a decision tree, entropy is used as a criterion to determine the best split at each node.
- A split with lower entropy indicates that the resulting subsets are more homogeneous (i.e., contain mostly one class), making it a better split.
- Entropy values closer to zero indicate higher purity or homogeneity.

Samples

- This refers to the number of samples or data points at a particular node in the decision tree.
- It represents the size of the dataset subset that reaches that node during the tree-building process.

Values

- The values indicate how many samples belong to each class at the node.
- That is values of 0 and 1.

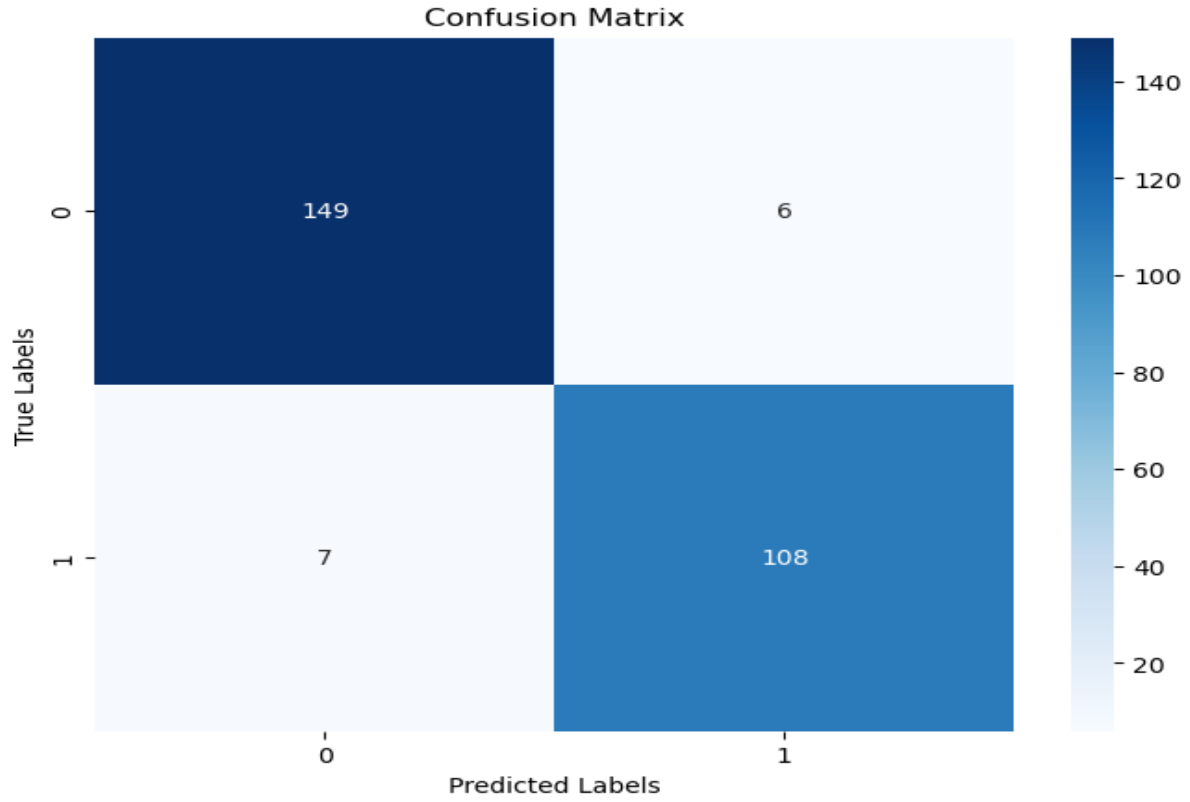
Interpretation:

1) The root node splits the dataset based on the feature at index 0 which is variance, with a threshold of 0.554. If the value of feature X[0] is less than or equal to 0.554, it goes to the left branch; otherwise, it goes to the right branch.
2) Gini impurity of the root node is 0.497, calculated based on the class distribution of the samples.
3) Internal nodes represent decision rules based on feature values.
4) Leaf nodes represent the final decision or prediction made by the classifier.

Random Forest:

```
Accuracy: 0.9543
```

Confusion Matrix:




```
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       155
           1       0.95      0.94      0.94       115

    accuracy                           0.95       270
   macro avg       0.95      0.95      0.95       270
weighted avg       0.95      0.95      0.95       270
```

Interpretation:

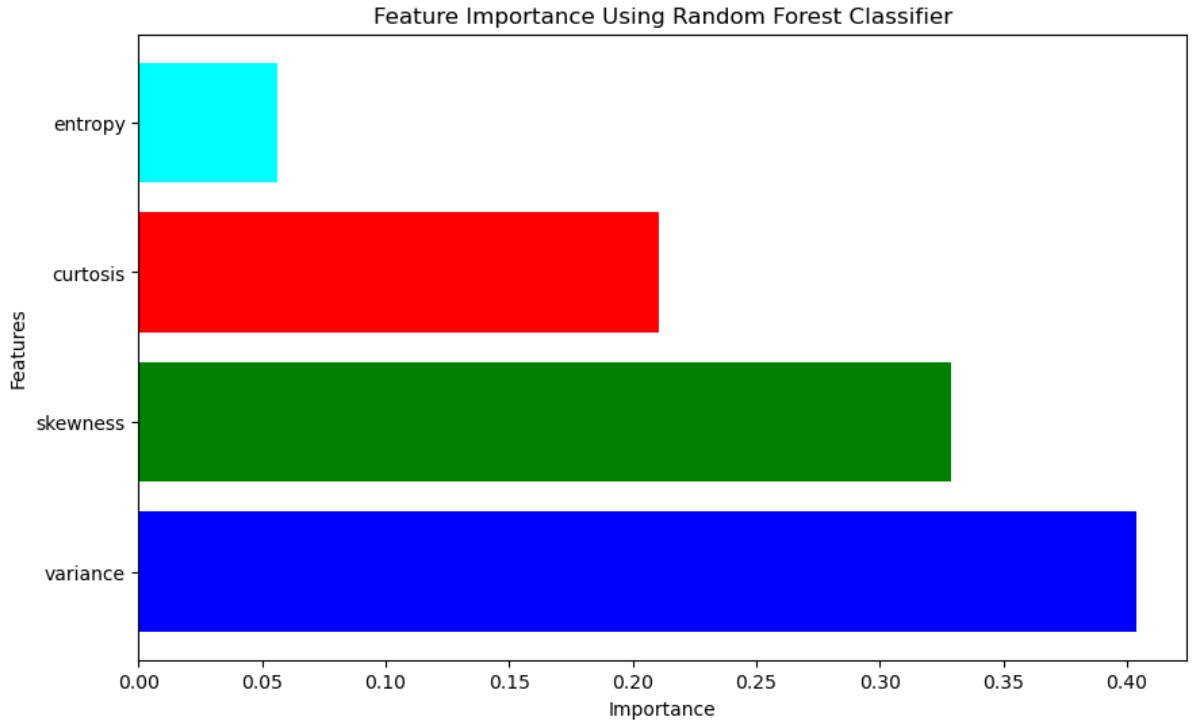
1)In this report, for class 0, the precision is 0.96, meaning that out of all the banknotes predicted to be genuine (class 0), 96% were actually genuine. For class 1, out of all the banknotes predicted to be fake (class 1) 95% were actually fake.

2) For class 0, the recall is 0.96, indicating that the model correctly identified 96% of all actual genuine banknotes. For class 1 recall is 94% indicating that the model correctly identified 94% of all actual fake

banknotes.
3) The overall accuracy of the model is 0.95 or 95%, indicating that the model correctly classified 95% of all banknotes.

Feature Importance Using Random Forest Classifier

Feature importance in machine learning refers to the measure of the contribution of each feature in predicting the target variable. It provides insights into which features have the most influence on the model's predictions and helps in understanding the underlying relationships in the data. By analyzing feature importance, stakeholders can prioritize features for further analysis, feature engineering, or model optimization, leading to improved model performance and interpretability. The most important predictor will be assigned the highest number and the least important will be assigned the lowest number.



The features are ranked from most important to least important based on their importance score. In this case, the most important feature is variance followed by, skewness, curtosis and then entropy.

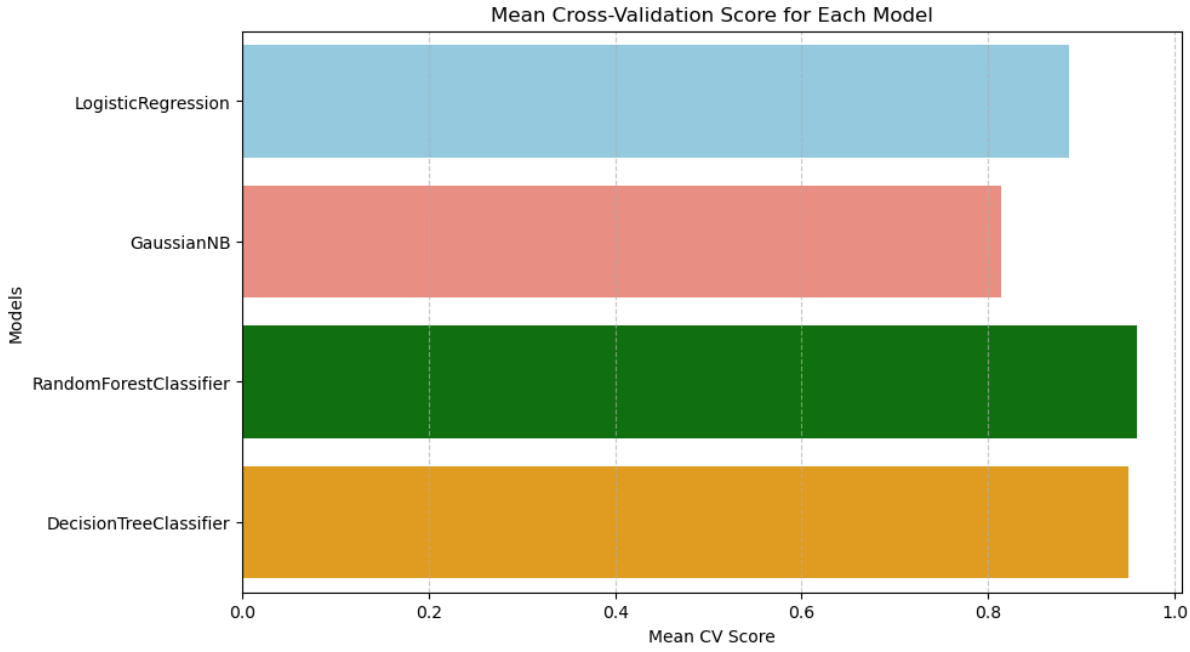
K-fold cross validation method:

K-fold cross-validation is a robust technique used to evaluate the performance of machine learning models while mitigating issues like overfitting and data variability. In this method, the dataset is divided into k equally sized subsets, or folds. The model is trained k times, each time using k-1 folds as training data and one fold as validation data. This process ensures that each data point is used for validation exactly once. The main advantage of k-fold cross-validation is that it provides a more reliable estimate of model performance compared to a single train-test split. By averaging the results of k iterations, we obtain a more stable evaluation metric, reducing the impact of data partitioning variability. Moreover, it allows us to utilize the entire dataset for both training and validation, maximizing data utilization and potentially improving model generalization.

Fold	GaussianNB	Logistic Regression	DecisionTree	Random Forest
1	0.82	0.87	0.95	0.97
2	0.81	0.87	0.96	0.98
3	0.83	0.9	0.96	0.97
4	0.85	0.91	0.94	0.97
5	0.78	0.89	0.95	0.95
6	0.82	0.92	0.94	0.95
7	0.79	0.87	0.94	0.95
8	0.78	0.87	0.94	0.95
9	0.83	0.89	0.96	0.96
10	0.83	0.88	0.94	0.95

Comparison of performance of different models using average of 10-fold average accuracy scores for each model.

Classifier	k fold avg acc	Standard deviation of CV scores
GaussianNB	0.814	0.038448
Logistic Regression	0.887	0.026625
DecisionTree	0.95	0.016622
Random Forest	0.96	0.014423



Interpretation:

- 10-fold avg. accuracy score for Random Forest Classifier is 96% which is higher than 10-fold avg. accuracy scores for all other models.
- So, Random Forest Classifier is the best fitted model for Classification of Banknote Authentication.

Major Findings

- Using Holdout method Naïve Bayes, Logistic Regression, Decision Tree, Random Forest has good testing accuracies.
- Among all these models Random Forest has highest testing accuracy.
- Random Forest has highest discriminating power hence it is best fitted model for prediction of banknote 'genuine' or 'fake' using Holdout method.
- Variance is the most important feature in the classifiacation of banknotes 'genuine' or 'fake'.
- Using 10-Fold Cross Validation, Random Forest Classifier provides highest 10-Fold average accuracy score and less standard deviation therefore, it is best fitted model for classification of banknotes.

Flask Api

A Flask API is a web application interface built using the Flask framework in Python. It defines routes that map URLs to functions, allowing communication between software components over the internet. Each route handles HTTP methods like GET and POST, processing requests and generating responses. Flask APIs are commonly used for building web services, RESTful APIs, microservices, web and mobile applications. It's known for its simplicity and ease of use, making it a popular choice for developing web APIs. A Flask API is a web application built using the Flask framework that exposes endpoints for clients to interact with, typically through HTTP requests. These endpoints define the functionality and data that clients can access or manipulate, allowing for the development of web-based services and applications.

1) **Flask API with Random Forest Model:**

- **Flask App Setup**: Ensure your Flask application is structured correctly with the necessary routes and functions to handle incoming requests.
- **Model Integration**: Integrate your trained Random Forest model into the Flask app. This involves loading the model and defining functions to preprocess input data and make predictions using the model.
- **API Endpoints**: Define API endpoints in your Flask app to handle different functionalities. For example:
  - **/predict** (GET): Accepts query parameters with banknote features and returns the prediction result.
  - **/predict_file** (POST): Accepts files of banknotes, extracts features, and returns the prediction result.

2) **Testing with Postman**:

- **Postman Setup**: Install Postman and create requests to test your Flask API endpoints.
- **Testing Process:**
  - For the /predict endpoint, send GET requests with query parameters containing banknote features to simulate predictions.
  - For the /predict_file endpoint, send POST requests with image

files containing banknotes to test image-based predictions.

3) Deployment with Flasgger:

- **Flasgger Integration**: Integrate Flasgger into your Flask app to generate API documentation using the OpenAPI Specification (OAS).
- **Documenting API Endpoints**: Use docstrings in your route functions to provide descriptions, parameters, and response details. Flasgger will parse these docstrings and generate the Swagger UI for interactive documentation.

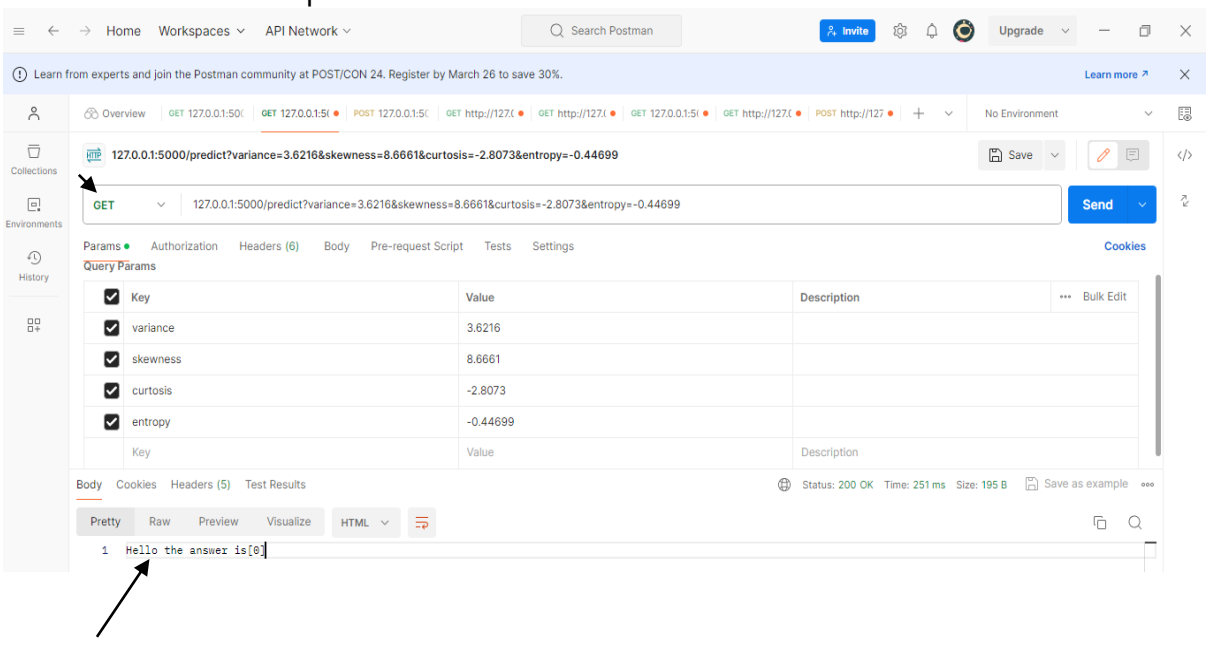
➢ **Testing with Postman**:

Here TestFile is:

variance	skewness	curtosis	entropy
3.6216	8.6661	-2.8073	-0.44699
4.5459	8.1674	-2.4586	-1.4621
3.866	-2.6383	1.9242	0.10645
3.4566	9.5228	-4.0112	-3.5944
-0.47465	-4.3496	1.9901	0.7517
1.0552	1.1857	-2.6411	0.11033
1.1644	3.8095	-4.9408	-4.0909
-4.4779	7.3708	-0.31218	-6.7754
-2.7338	0.45523	2.4391	0.21766

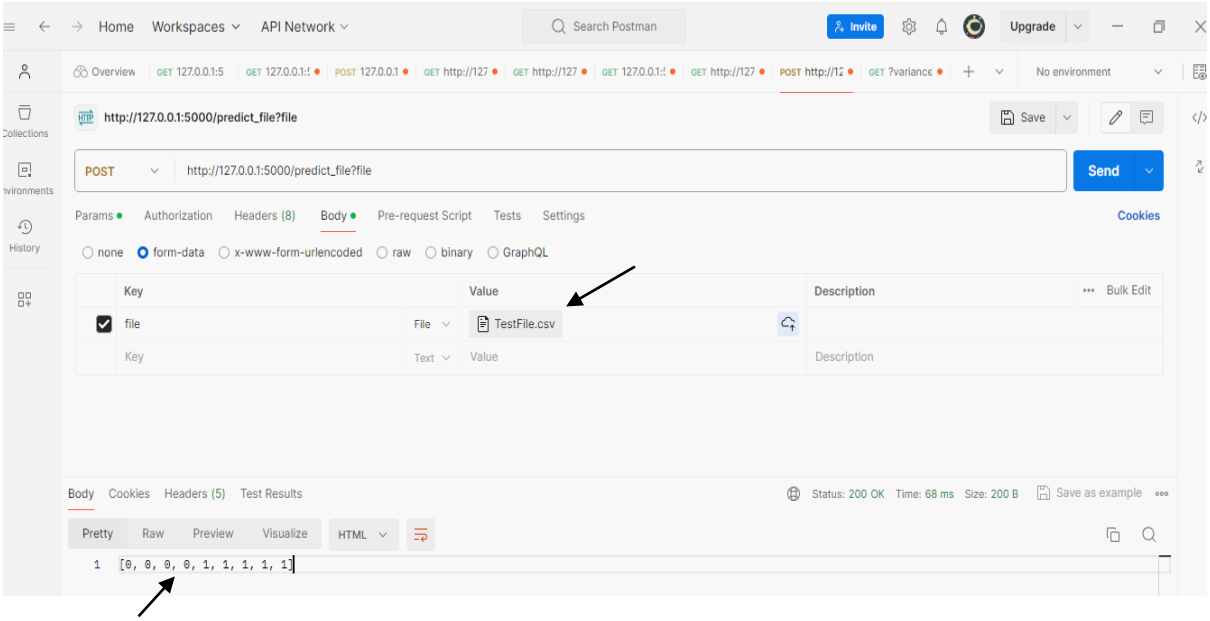
API Endpoints:

- **/predict** (GET): Accepts query parameters with banknote features and returns the prediction result.



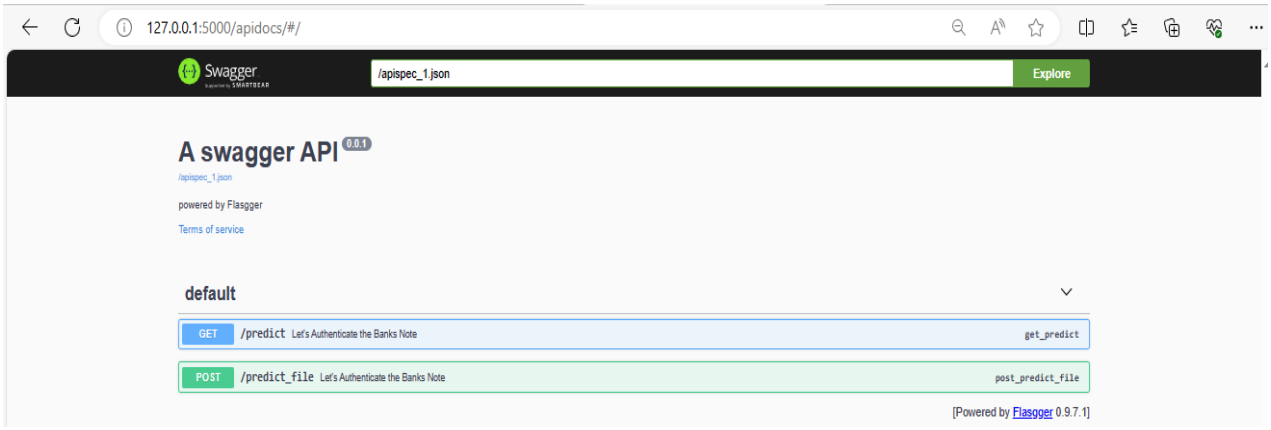
Here we done Testing through Postman and for corresponding variance, skewness , Kurtosis and entropy ,we get answer as class 0 and which is correct.

- **/predict_file** (POST): Accepts image or features files of banknotes, extracts features, and returns the prediction result.

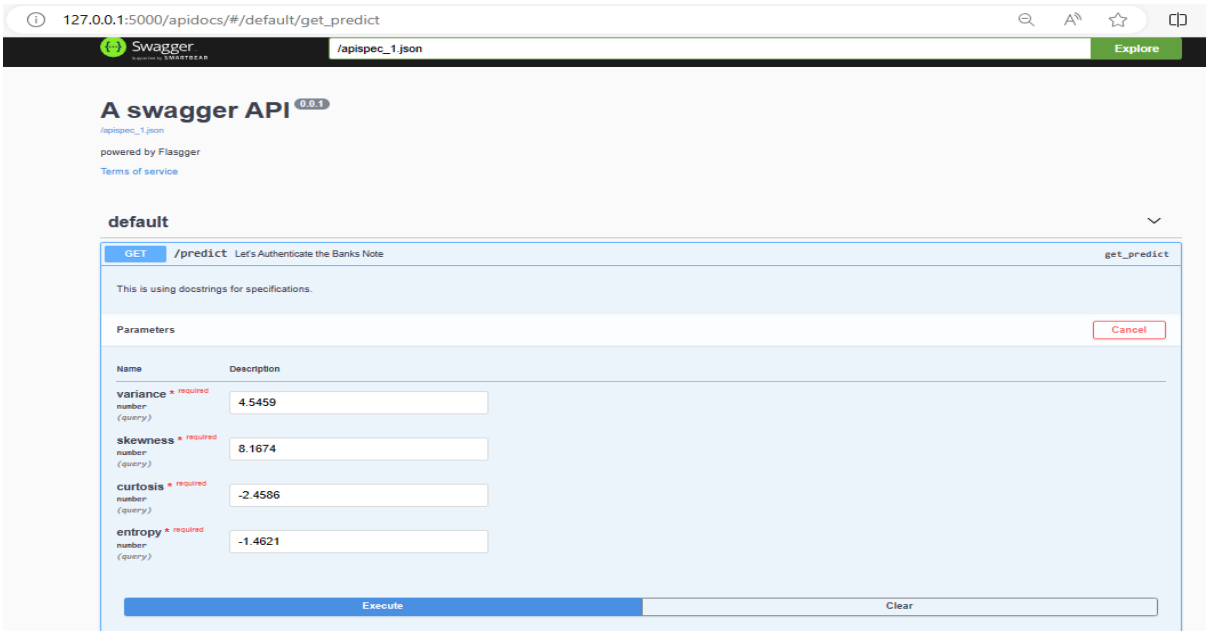


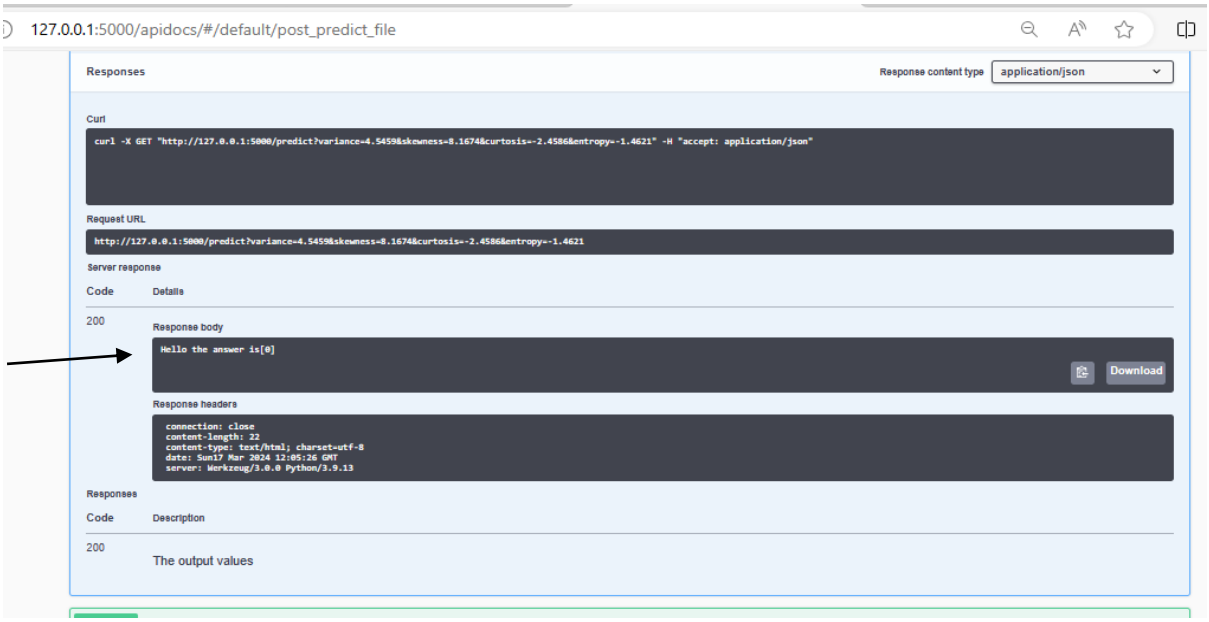
In the above file containing 9 observations and given answer is correct.

Flassger: In the context of Flask API development, Flasgger is a powerful tool that simplifies the documentation process and enhances the development workflow. It seamlessly integrates with Flask to automatically generate interactive API documentation based on the OpenAPI Specification (formerly known as Swagger). Flasgger provides a user-friendly interface, allowing developers to describe API endpoints, parameters, responses, and examples using YAML or JSON files. Additionally, it includes Swagger UI, a feature-rich interface for exploring and testing APIs directly in the browser.

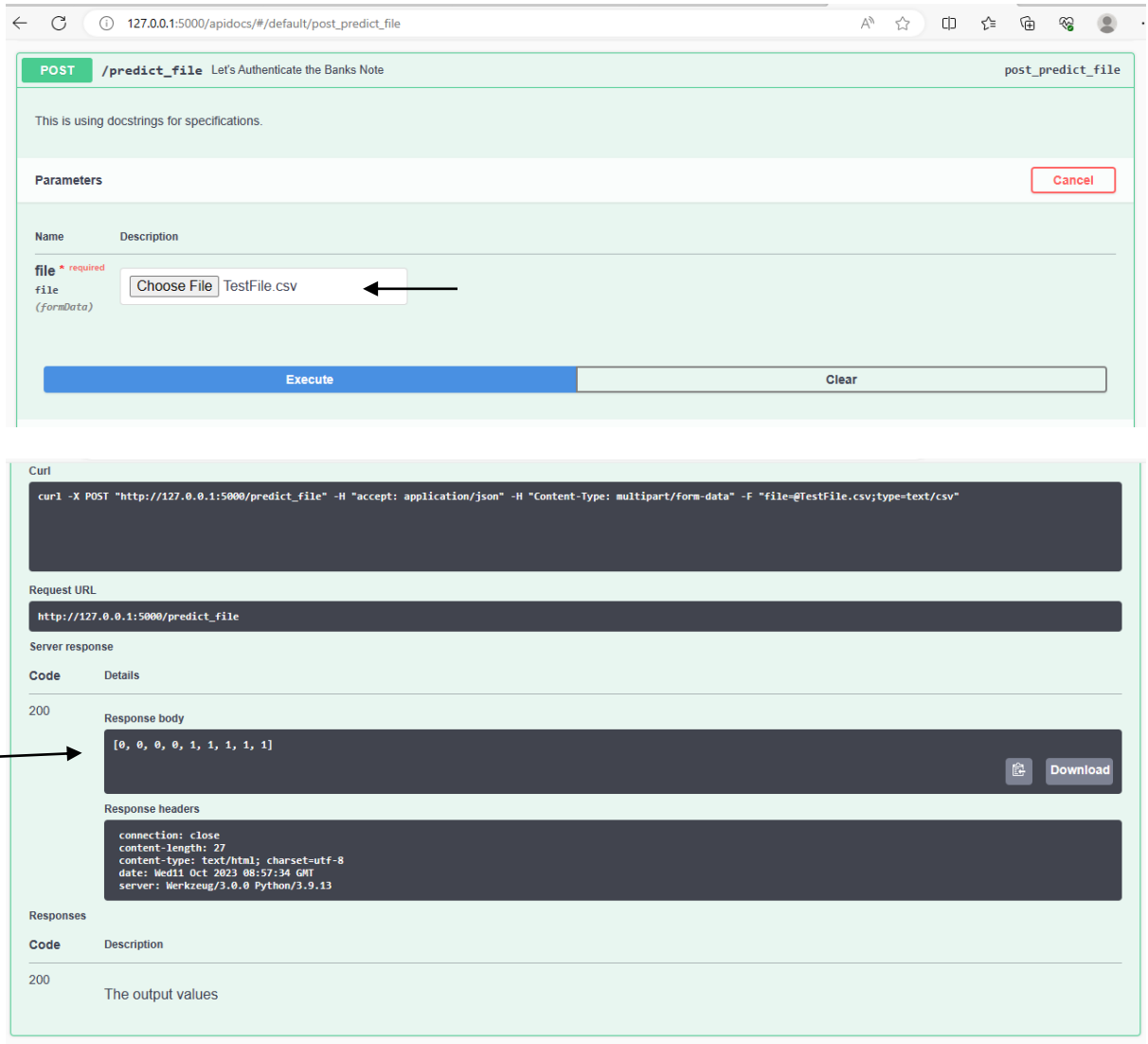


1) **/predict** (GET):





/predict_file (POST):



Interpretation:

Above is the interactive interface of API in which in get request accepts parameters and give result and in the post request accept file of features extracted from banknotes and give results.

Appendix

#Importing the necessary libraries

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
import pandas as pd
from pandas import read_excel
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")
```

#Load the dataset

```
df=pd.read_csv("C:\\Users\\dell\\OneDrive\\Documents\\BankNote
_Authentication.csv")
df.head()

df.info()

df.describe()

df.isnull().sum()

df[df.duplicated ()]

df.duplicated().sum ()

df = df.drop_duplicates ()
df

df.describe().T

df.corr()

df.cov()

df.skew()
```

```
df ['class'].unique ()

df.memory_usage()

import pandas as pd
# Assuming you have loaded your data into a DataFrame named df
class_proportions = df ['class'].value_counts (normalize=True)
# Display the proportions
print(class_proportions)

df['class'].value_counts ()

import pandas as pd
import matplotlib.pyplot as plt
# Assuming df is your DataFrame and 'class_column' is the name of the column representing the class
class_counts = df['class'].value_counts()
# Display the counts
print(class_counts)
# Plot the bar chart
plt.bar(class_counts.index, class_counts.values,
color=['blue', 'orange'])
plt.xlabel('Class')
plt.ylabel ('Count')
plt.title ('Class Distribution')
plt.xticks (class_counts.index, ['Class 0', 'Class 1'])
plt.show ()
```

#Correlation Plot

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Compute the correlation matrix
corr_matrix = df.corr()
# Set up the matplotlib figure
plt.figure (figsize =(10, 8))
# Plot the heatmap
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm',
linewidths=.5)
# Set plot title
plt.title('Correlation Plot ')
# Show plot
plt.show()
```

#Histplot:

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Plot for Variance
```
plt.figure(figsize=(20, 10))
plt.subplot(2, 2, 1)
sns.histplot (df ['variance'], kde=True, color="blue")
plt.title('Variance')
```

Plot for Skewness
```
plt.subplot (2, 2, 2)
sns.histplot (df ['skewness'], kde=True, color="blue")
plt.title ('Skewness')
```

Plot for Curtosis
```
plt.subplot(2, 2, 3)
sns.histplot(df ['curtosis'], kde=True, color="blue")
plt.title('Curtosis')
```

Plot for Entropy
```
plt.subplot(2, 2, 4)
sns.histplot (df ['entropy'], kde=True, color="blue")
plt.title ('Entropy')

plt.tight_layout ()
plt.show ()
```

#Box and the whisker plots
```
df.plot (kind='box', subplots=True, layout= (2, 3),
sharex=False, sharey=False)

import pandas as pd

# Define normalization function
def normalize(x):
    return (x - x.min ()) / (x.max() - x.min())

# Assuming df is your DataFrame with columns: variance,
skewness, curtosis, entropy, class

# Extract the "class" column
class_column = df ['class']

# Apply normalization function to the first four columns
df_normalized = df.iloc [:,:4].apply (normalize)

# Combine the normalized columns with the "class" column
df_normalized = pd.concat([df_normalized, class_column],
axis=1)
```

Display summary statistics for the normalized DataFrame
print (df_normalized. Describe())
df=df_normalized
df

#For outlier handling:

```
from scipy.stats import zscore
df [df.z_score_variance'] = zscore (df.variance)
df [(df.z_score_variance>2) | (df.z_score_variance<-2)]
import numpy as np
df ['variance'] = np.where ((df.z_score_variance > 2) |
(df.z_score_variance < -2), df ['variance'].median (),
df['variance'])

df ['z_score_variance'] = zscore (df.variance)
df [(df.z_score_variance>2)| (df.z_score_variance<-2)]
df ['variance'] = np.where ((df.z_score_variance > 2) |
(df.z_score_variance < -2), df ['variance'].median(),
df['variance'])

df ['z_score_variance'] = zscore (df.variance)
df [(df.z_score_variance>2)| (df.z_score_variance<-2)]
df ['variance'] = np.where ((df.z_score_variance > 2) |
(df.z_score_variance < -2), df['variance'].median(),
df['variance'])

df ['z_score_variance']=zscore(df.variance)
df [(df.z_score_variance>3)|(df.z_score_variance<-1)]
df ['variance'] = np.where((df.z_score_variance > 3) |
(df.z_score_variance < -1), df['variance'].median(),
df['variance'])

df ['z_score_variance']=zscore(df.variance)
df [(df.z_score_variance>2)|(df.z_score_variance<-2)]
df ['variance'] = np.where((df.z_score_variance > 2) |
(df.z_score_variance < -2), df['variance'].median(),
df['variance'])

df ['z_score_variance']=zscore(df.variance)
df [(df.z_score_variance>2)|(df.z_score_variance<-2)]
df ['variance'] = np.where((df.z_score_variance > 2) |
(df.z_score_variance < -2), df['variance'].median(),
df['variance'])

df ['z_score_variance']=zscore(df.variance)
df [(df.z_score_variance>2)|(df.z_score_variance<-2)]
df ['variance'] = np.where((df.z_score_variance > 2) |
(df.z_score_variance < -2), df['variance'].median(),
df['variance'])

df ['z_score_variance']=zscore(df.variance)
```

```
df[(df.z_score_variance>2)|(df.z_score_variance<-4)]
df['variance'] = np.where((df.z_score_variance > 2) |
(df.z_score_variance < -4), df['variance'].median(),
df['variance'])

import matplotlib.pyplot as plt
import seaborn as sns

# Assuming df is your DataFrame

plt.figure (figsize=(12, 8))

plt.subplot (2, 2, 3)
sns.boxplot(x='class', y='curtosis', data=df)

plt.subplot(2, 2, 4)
sns.boxplot(x='class', y='entropy', data=df)

plt.tight_layout()
plt.show ()

from scipy.stats import zscore
df['z_score_entropy']=zscore(df.entropy)
df [(df.z_score_entropy>2)|(df.z_score_entropy<-2)]
df['entropy'] = np.where ((df.z_score_entropy > 3) |
(df.z_score_entropy < -2), df['entropy'].median(),
df['entropy'])

df['z_score_entropy']=zscore(df.entropy)
df[(df.z_score_entropy>2)|(df.z_score_entropy<-2)]
df['entropy'] = np.where((df.z_score_entropy > 3) |
(df.z_score_entropy < -3), df['entropy'].median(),
df['entropy'])

df ['z_score_entropy']=zscore(df.entropy)
df[(df.z_score_entropy>3)|(df.z_score_entropy<-1)]
df ['entropy'] = np.where((df.z_score_entropy > 3) |
(df.z_score_entropy < -1), df['entropy'].median(),
df['entropy'])

df ['z_score_entropy']=zscore(df.entropy)
df[(df.z_score_entropy>2)|(df.z_score_entropy<-2)]
df['entropy'] = np.where((df.z_score_entropy > 2) |
(df.z_score_entropy < -2), df['entropy'].median(),
df['entropy'])

df ['z_score_entropy']=zscore(df.entropy)
df [(df.z_score_entropy>2)|(df.z_score_entropy<-2)]

df ['entropy'] = np.where((df.z_score_entropy > 3) |
(df.z_score_entropy < -2), df['entropy'].median(),
```

```
df['entropy'])

df['z_score_entropy']=zscore(df.entropy)
df[(df.z_score_entropy>3)|(df.z_score_entropy<-1)]
df ['entropy'] = np.where((df.z_score_entropy > 3) |
(df.z_score_entropy < -1), df['entropy'].median(),
df['entropy'])

import matplotlib.pyplot as plt
import seaborn as sns

# Assuming df is your DataFrame

plt.figure (figsize=(12, 8))


plt.subplot(2, 2, 4)
sns.boxplot(x='class', y='entropy', data=df)

plt.tight_layout ()
plt.show ()

# Outliers for curtosis
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming df is your DataFrame

plt.figure (figsize =(12, 8))

plt.subplot (2, 2, 3)
sns.boxplot(x='class', y='curtosis', data=df)


plt.tight_layout()
plt.show ()


from scipy.stats import zscore
df ['z_score_curtosis'] =zscore (df.curtosis)
df [(df.z_score_curtosis>3)| (df.z_score_curtosis<-2)]
import numpy as np
df ['curtosis'] = np.where ((df.z_score_curtosis > 3) |
(df.z_score_curtosis < -2), df['curtosis'].median(),
df['curtosis'])

from scipy.stats import zscore
df ['z_score_curtosis'] = zscore(df.curtosis)
df [(df.z_score_curtosis>3)|(df.z_score_curtosis<-2)]
import numpy as np
df ['curtosis'] = np.where((df.z_score_curtosis > 3) |
```

```
(df.z_score_curtosis < -2), df['curtosis'].median(),
df['curtosis'])

import matplotlib.pyplot as plt
import seaborn as sns

# Assuming df is your DataFrame

plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 3)
sns.boxplot(x='class', y='curtosis', data=df)


plt.tight_layout()
plt.show()

df

columns_to_drop = ['z_score_variance', 'z_score_entropy',
'z_score_curtosis']

# Drop the specified columns
df = df.drop(columns=columns_to_drop, axis=1)

# Print the updated DataFrame
df

#shape
df.shape
```

#Splitting the data into training and testing

```
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt

# Assuming df is your DataFrame
```

 # Separate features and target variable

```
X = df.drop('class', axis=1)
y = df['class']
```

 # Split data into train and test sets (holdout method)

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

 # Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

#Logistic Regression

```
# Train and test Logistic Regression
logistic_model = LogisticRegression()
logistic_model.fit(X_train_scaled, y_train)
y_pred_logistic = logistic_model.predict(X_test_scaled)

# Evaluate Logistic Regression
accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
print("Logistic Regression:")
print("Accuracy:", accuracy_logistic)
print("Classification Report:")
print(classification_report(y_test, y_pred_logistic))

# Calculate confusion matrix for Logistic Regression
conf_matrix_logistic = confusion_matrix(y_test,
y_pred_logistic)

# Define class labels
class_names = ['0', '1']

# Plot confusion matrix for Logistic Regression
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_logistic, annot=True, fmt="d",
cmap="Blues", xticklabels=class_names,
yticklabels=class_names)
plt.title('Confusion Matrix - Logistic Regression')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Train Logistic Regression model
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)

# Get predicted probabilities for the positive class (class 1)
y_prob_logistic = logistic_model.predict_proba(X_test)[:, 1]

# Compute ROC curve and ROC area for the Logistic Regression
classifier
fpr_logistic, tpr_logistic, _ = roc_curve(y_test,
y_prob_logistic)
```

```
roc_auc_logistic = auc(fpr_logistic, tpr_logistic)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_logistic, tpr_logistic, color='blue', lw=2,
label='ROC curve (area = %0.2f)' % roc_auc_logistic)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Logistic Regression')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```

#Naive Bayes:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,
classification_report

# Train and test Naive Bayes
nb_model = GaussianNB()
nb_model.fit(X_train_scaled, y_train)
y_pred_nb = nb_model.predict(X_test_scaled)

# Evaluate Naive Bayes
accuracy_nb = accuracy_score(y_test, y_pred_nb)
print("Naive Bayes:")
print("Accuracy:", accuracy_nb)
print("Classification Report:")
print(classification_report(y_test, y_pred_nb))

# Calculate confusion matrix for Naive Bayes
conf_matrix_nb = confusion_matrix(y_test, y_pred_nb)

# Define class labels
class_names = ['0', '1']

# Plot confusion matrix for Naive Bayes
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_nb, annot=True, fmt="d", cmap="Blues",
xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix - Naive Bayes')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

#Decision Tree Classifier:

```
df
```

```
features_cols=['variance','skewness','curtosis','entropy']

X=df[features_cols]
y = df['class']

X

Y

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
from sklearn.tree import DecisionTreeClassifier
classifier=DecisionTreeClassifier(criterion='gini')
classifier.fit(X_train,y_train)
classifier.predict(X_test)
X_test
classifier.score(X_test,y_test)



from sklearn.metrics import confusion_matrix, classification_report

# Assuming 'classifier' is your trained DecisionTreeClassifier and
'X_test', 'y_test' are your test data
# Replace 'classifier', 'X_test', and 'y_test' with your actual
variables

# Generate predictions on the test set
y_pred = classifier.predict(X_test)

# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Generate classification report
class_report = classification_report(y_test, y_pred)
print("\nClassification Report:")
print(class_report)

import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import numpy as np

# Assuming 'y_test' and 'y_pred' are your actual and predicted
labels, respectively

# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Set labels for confusion matrix
labels = np.unique(y_test)

# Create heatmap with seaborn
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g',
xticklabels=labels, yticklabels=labels)

# Add labels and title
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')

# Show plot
plt.show()

from sklearn import tree
tree.plot_tree(classifier,fontsize=6)

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
import matplotlib.pyplot as plt

# Assuming you have your features (X) and labels (y)
# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y,
test_size=0.2, random_state=42)

# Define the Decision Tree classifier
classifier = DecisionTreeClassifier()

# Define the parameter grid with different max_depth values
param_grid = {'max_depth': [3, 5, 7, 10, 15, 20]}

# Use GridSearchCV for cross-validated grid search
grid_search = GridSearchCV(classifier, param_grid, cv=5,
scoring='accuracy')
grid_search.fit(X_train, y_train)

# Get the results of the grid search
results = grid_search.cv_results_

# Plot the mean test scores across different max_depth values
plt.plot(param_grid['max_depth'], results['mean_test_score'],
marker='o')
plt.xlabel('max_depth')
plt.ylabel('Mean Test Score (Accuracy)')
plt.title('Grid Search Results for Decision Tree Max Depth')
plt.show()

from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

classifier = DecisionTreeClassifier(criterion='entropy',
max_depth=10, random_state=0)
classifier.fit(X_train, y_train)

plt.figure(figsize=(25, 17))  # Adjust the figsize according to your
preference
plot_tree(classifier, filled=True, feature_names=None,
```

```
class_names=None, rounded=True, fontsize=10)
plt.show()

#Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
import matplotlib.pyplot as plt

# Assuming you have your features (X) and labels (y)
# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y,
test_size=0.2, random_state=1)

# Define the Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=0)

# Define the parameter grid with different max_depth values
param_grid = {'max_depth': [3, 5, 7, 10, 15, 20]}

# Use GridSearchCV for cross-validated grid search
grid_search_rf = GridSearchCV(rf_classifier, param_grid, cv=5,
scoring='accuracy')
grid_search_rf.fit(X_train, y_train)

# Print the best parameters and corresponding accuracy
print("Best Parameters: ", grid_search_rf.best_params_)
print("Best Accuracy: ", grid_search_rf.best_score_)

# Visualize the performance over different max_depth values
results = grid_search_rf.cv_results_
plt.plot(param_grid['max_depth'], results['mean_test_score'],
marker='o')
plt.xlabel('max_depth')
plt.ylabel('Mean Test Score (Accuracy)')
plt.title('Random Forest Grid Search Results')
plt.show()

from sklearn.metrics import confusion_matrix, classification_report

# Get the best model from the grid search
best_rf_model = grid_search_rf.best_estimator_

# Make predictions on the validation set
y_pred_val = best_rf_model.predict(X_val)

# Generate confusion matrix
conf_matrix = confusion_matrix(y_val, y_pred_val)
print("Confusion Matrix:")
print(conf_matrix)

# Generate classification report
class_report = classification_report(y_val, y_pred_val)
print("\nClassification Report:")
print(class_report)

import seaborn as sns
```

```
import matplotlib.pyplot as plt

# Plot confusion matrix as heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import numpy as np

# Assuming you have your features (X) and labels (y)
# Train the Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=0)
rf_classifier.fit(X_train, y_train)

# Get feature importances
importances = rf_classifier.feature_importances_
features = X.columns

# Sort feature importances in descending order
indices = np.argsort(importances)[::-1]

# Plot
plt.figure(figsize=(10, 6))
plt.barh(range(X.shape[1]), importances[indices], color=['blue',
'green', 'red', 'cyan'])
plt.yticks(range(X.shape[1]), [features[i] for i in indices])
plt.xlabel('Importance')
plt.ylabel('Features')
plt.title('Feature Importance Using Random Forest Classifier')
plt.show()

#Cross Validation

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier  # Import
DecisionTreeClassifier
import pandas as pd

# Load your dataset as 'df'

# Define features and target variable
X = df.drop('class', axis=1)  # Features
y = df['class']  # Target variable
```

```
# Define models
models = {
    "LogisticRegression": LogisticRegression(max_iter=1000),
    "GaussianNB": GaussianNB(),
    "RandomForestClassifier": RandomForestClassifier(),
    "DecisionTreeClassifier": DecisionTreeClassifier()  # Replace
KNeighborsClassifier with DecisionTreeClassifier
}

# Initialize KFold with 10 folds
kf = KFold(n_splits=10, shuffle=True, random_state=1)

# Perform cross-validation for each model
results = {}
for model_name, model in models.items():
    cv_scores = []
    for train_index, test_index in kf.split(X, y):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        # Fit the model on the training data
        model.fit(X_train, y_train)

        # Evaluate the model on the test data
        score = model.score(X_test, y_test)
        cv_scores.append(score)

    results[model_name] = cv_scores

    # Print the cross-validation scores for each model
    print(model_name + " Cross-validation scores:", cv_scores)
    print("Mean CV score:", sum(cv_scores) / len(cv_scores))
    print("Standard deviation of CV scores:",
pd.Series(cv_scores).std())
    print("\n")

# Convert results to DataFrame
result_df = pd.DataFrame(results)

# Define colors for each model
colors = ['skyblue', 'salmon', 'green', 'orange', 'purple']

# Generate chart
plt.figure (figsize = (10, 6))
sns.barplot(x=result_df.mean(), y = result_df.columns,
palette=colors)
plt.xlabel ('Mean CV Score')
plt.ylabel ('Models')
plt.title ('Mean Cross-Validation Score for Each Model')
plt.grid (axis='x', linestyle='--', alpha=0.7)
plt.show()
```

#Random Forest Classifier

```
import pandas as pd
import numpy as np

df=pd.read_csv('C:\\Users\\dell\\Downloads\\BankNote_Authentic
ation (1).csv')
df

X=df.iloc[:,:-1]
y=df.iloc[:,-1]

X.head()
y.head()

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0
.3,random_state=0)

from sklearn.ensemble import RandomForestClassifier

# Assuming you have X_train and y_train defined
classifier = RandomForestClassifier()  # Create an instance of
RandomForestClassifier

# Fit the model with training data
classifier.fit(X_train, y_train)

y_pred=classifier.predict(X_test)
from sklearn.metrics import accuracy_score
score=accuracy_score(y_test,y_pred)

score

import pickle
pickle_out=open('classifier.pkl','wb')
pickle.dump(classifier,pickle_out)
pickle_out.close()

classifier.predict([[2,3,4,1]])
```

#Flask Api

```
"""Created on 12 Jan 2023

@auther: Prajakta Bille"""

from flask import Flask,request
import pandas as pd
import numpy as np
import pickle

app=Flask(__name__)
pickle_in=open('classifier.pkl','rb')
classifier=pickle.load(pickle_in)

@app.route('/')
def welcome():
    return "Welcome All"

@app.route('/predict')
def predict_note_authentication():
    variance=request.args.get('variance')
    skewness=request.args.get('skewness')
    curtosis=request.args.get('curtosis')
    entropy=request.args.get('entropy')

prediction=classifier.predict([[variance,skewness,curtosis,ent
ropy]])
    return "The predicted value is"+ str(prediction)

@app.route('/predict_file',methods=["POST"])
def predict_note_file():
    df_test=pd.read_csv(request.files.get("file"))
    df_test.fillna(df_test.mean(), inplace=True)
    prediction=classifier.predict(df_test)
    return "The predicted values for the csv is"+
str(list(prediction))



if __name__=='__main__':
    app.run()
```

#Flassger

```
"""Created on 12 Jan 2023

@auther: Prajakta Bille"""

from flask import Flask,request
import pandas as pd
import numpy as np
import pickle
import flasgger
from flasgger import Swagger

app=Flask(__name__)
Swagger(app)

pickle_in=open('classifier.pkl','rb')
classifier=pickle.load(pickle_in)

@app.route('/')
def welcome():
    return "Welcome All"

@app.route('/predict',methods=["Get"])
def predict_note_authentication():

"""Let's Authenticate the Banks Note
    This is using docstrings for specifications.
    ---
    parameters:
      - name: variance
        in: query
        type: number
        required: true
      - name: skewness
        in: query
        type: number
        required: true
      - name: curtosis
        in: query
        type: number
        required: true
      - name: entropy
        in: query
        type: number
        required: true
    responses:
        200:
            description: The output values
```

```
"""
    variance=request.args.get('variance')
    skewness=request.args.get('skewness')
    curtosis=request.args.get('curtosis')
    entropy=request.args.get('entropy')

prediction=classifier.predict([[variance,skewness,curtosis,ent
ropy]])
    print(prediction)
    return "Hello the answer is"+str(prediction)

@app.route('/predict_file',methods=["POST"])
def predict_note_file():

"""Let's Authenticate the Banks Note
    This is using docstrings for specifications.
    ---
    parameters:
      - name: file
        in: formData
        type: file
        required: true

    responses:
        200:
            description: The output values

"""
    df_test=pd.read_csv(request.files.get("file"))
    df_test.fillna(df_test.mean(), inplace=True)
    print(df_test.head())
    prediction=classifier.predict(df_test)

    return str(list(prediction))


if __name__=='__main__':
    app.run()
```

Reference

- Book : Machine Learning using Python
  Manaranjan Pradhan and U Dinesh Kumar

- Research Paper : Banknote Authentication Using Machine Learning Classification Algorithms by Michael Wiryaseputra
  https://www.researchgate.net/publication/364333508

- Reserch Paper:Analysis of Banknote Authentication System using Machine Learning Techniques by Sumeet Shahani and Priya R.L.
  https://www.researchgate.net/publication/323223299

- Book : Designing APIs with Swagger and OpenAPI
  Joshua S. Ponelat
  Lukas L. Rosenstock