Contents:

ABSTRACT

Counterfeit currency is an extremely common yet pertinent problem of presence of fake, inauthentic or copies of real currency in the market and economy that is faced by various nations across the globe. The increasing technological advancements have created the possibility for creating more counterfeit currency which is circulated in the market which reduces the overall economy of the country. Fake and inauthentic notes, therefore, continue to be a major source of problem for the economy.

Fake Currency Detection is a task of binary classification in machine learning.In this project, the main aim is to identify the authenticity of the currency notes by using several supervised machine learning algorithms to build models that distinguish between genuine and counterfeit(fake) banknotes. We analyze these algorithms to choose the best candidate, and then try to further optimize the algorithm to best model the data. Our goal with this implementation is to accurately predict whether a currency note is genuine or fake.

Data processing and Data extraction is performed by implementing machine learning algorithms and image processing to acquire the final result and accuracy. Fake currency detection using image processing is a computer vision project that aims to automatically detect counterfeit money using image analysis techniques.

Various machine learning algorithms are proposed by image processing that gives 99.9% accuracy for the fake identity of the currency. Detection and recognition methods over the algorithms include entities like colour, shape, paper width, image filtering on the note.

This system is designed such that any person can use it easily and detect the authenticity of the currency he has by using the visual features of the currency.

DATASET DESCRIPTION

The dataset required for our project ‘Fake currency detection’ is taken from UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/banknote+authentication). We have used the same dataset, but manually included title line, as the title line that describes the names of all features was missing.

The dataset contains these five input characteristics:

1. The variance of the image transformed into wavelets
2. The asymmetry(skewness) of the image transformed into wavelets
3. Kurtosis of the image transformed into wavelets
4. Image entropy
5. Target value

The target value is 0 for fake banknotes and 1 for real banknotes.

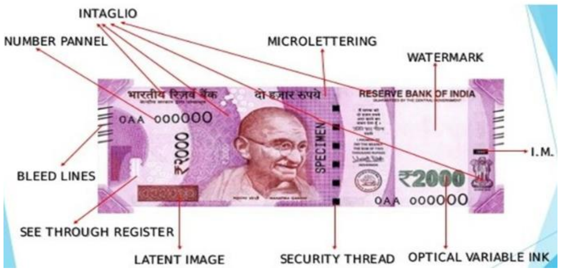
There are 1372 rows and 5 attributes(columns) and no missing or null values in this dataset.

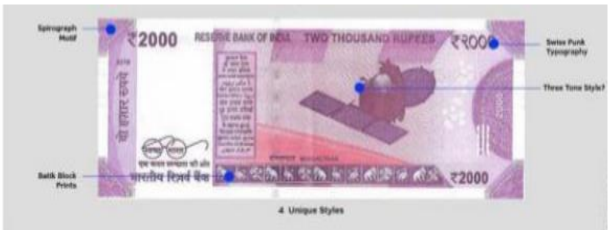
Variance of image refers to the squares of the standard deviations, in the values of the input or output images. **Skewness** is a measure of symmetry, or more precisely, the lack of symmetry. A distribution, or data set, is symmetric if it looks the same to the left and right of the center point. **Kurtosis** is a measure of whether the data are peaked or flat relative to a normal distribution. Data sets with high kurtosis tend to have a distinct peak near the mean whereas data sets with low kurtosis tend to have a flat top near the mean. The **entropy** or average information of an image is a measure of the degree of randomness in the image.

For image processing the dataset is two images. One is for real notes and the other is for fake notes. This model uses a grayscale image, segmentation, and feature extraction. The characteristics used to authenticate currency notes are Security Thread, Serial Number, Latent image, Watermark, Identification Mark.

Methodology

1. Image Acquisition: The image is provided to the model. There are two images – a note which we want to detect and a corresponding real note.
2. RGB to GRAYSCALE: The RGB image is converted into a GRAYSCALE image.
3. Segmentation: The image is segmented to crop Gandhi Ji image and thin strip image.
4. Feature Measurement: Feature measurement is done to measure the number of lines on a thin strip.
5. Finding Correlation: We find a correlation between Gandhi Ji’s image on the original note and a fake note. If the result is greater than 0.5 then we will consider it legitimate otherwise the currency is fake.
6. Classification: Finally, we classify the image as real or fake.





ALGORITHM DESCRIPTION

1.**Naive Bayes Algorithm**

Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.

A Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Even if these features are related to each other, a Naive Bayes classifier would consider all of these properties independently when calculating the probability of a particular outcome.

A Naive Bayesian model is easy to build and useful for massive datasets. It's simple and is known to outperform even highly sophisticated classification methods.

Naive Bayes is used for :

1.Filtering spam messages

2.Recommendation systems such as Netflix

3.Classify a news article about technology, politics, or sports

4.Sentiment analysis on social media

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

Algorithm

1. Convert the given dataset into frequency tables.
2. Generate likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Implementation

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

```
accuracy = float(n_fake_notes)/n_records

# Calculate F-Score with beta = 2
precision = accuracy
recall = 1
beta = 2
fscore = (1 + beta ** 2) * precision * recall /(beta ** 2 * precision + recall)

# Print the results
print ("Naive Predictor: [Accuracy score: {:.4f}, F-score: {:.4f}]".format(accuracy, fscore))
```

2. Decision Tree Algorithm

A decision tree is a supervised learning algorithm that is mainly used to solve the classification problems but can also be used for solving the regression problems. It can work with both categorical variables and continuous variables. It shows a tree-like structure that includes nodes and branches, and starts with the root node that expand on further branches till the leaf node. The **internal node** is used to represent the features of the dataset, branches show the decision rules, **and** leaf nodes represent the outcome of the problem. However, it is prone to overfitting if the tree is grown too deep.

Decision trees are commonly used for:

1.Building knowledge management platforms

2.Selecting a flight to travel

3. Forecasting predictions and identifying possibilities in various domains

Algorithm for Decision tree

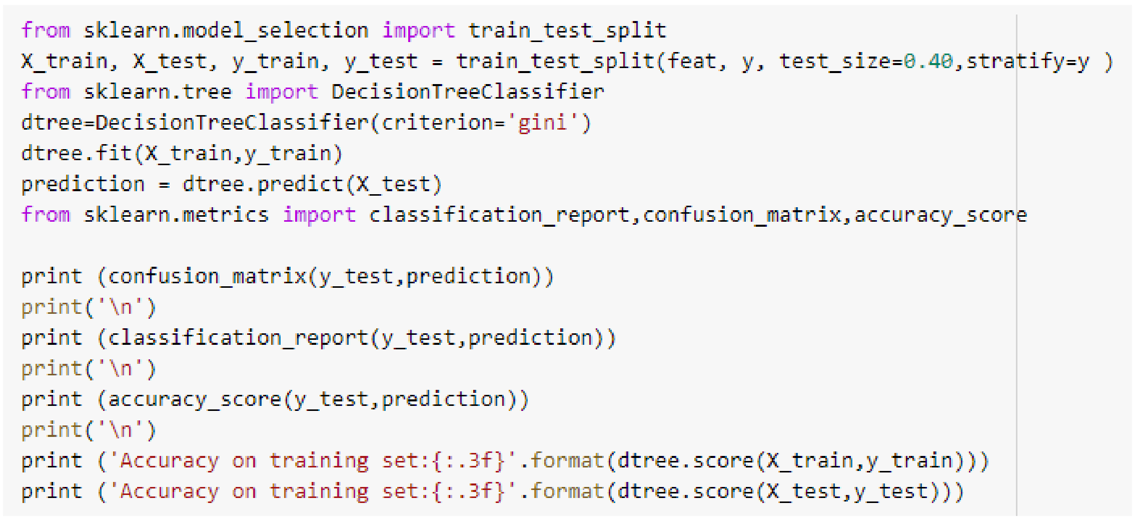
Step-1: Begin the tree with the root node, say S, which contains the complete dataset.

Step-2: Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**

Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.



```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(feat, y, test_size=0.40,stratify=y )
from sklearn.tree import DecisionTreeClassifier
dtree=DecisionTreeClassifier(criterion='gini')
dtree.fit(X_train,y_train)
prediction = dtree.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score

print (confusion_matrix(y_test,prediction))
print('\n')
print (classification_report(y_test,prediction))
print('\n')
print (accuracy_score(y_test,prediction))
print('\n')
print ('Accuracy on training set:{:.3f}'.format(dtree.score(X_train,y_train)))
print ('Accuracy on training set:{:.3f}'.format(dtree.score(X_test,y_test)))
```

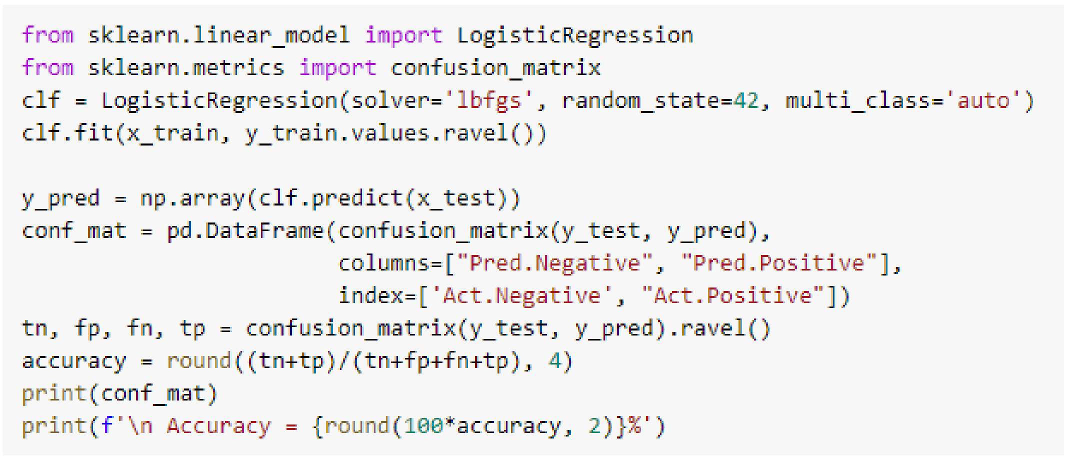
3. Logistic Regression

Logistic Regression algorithm is often used in the **binary classification problems** where the events in these cases commonly result in either of the two values, pass or fail, true or false.

It is best suited for situations where there is a need to predict probabilities that the dependent variable will fall into one of the two categories of the response.

Logistic regression is similar to the linear regression except how they are used, such as Linear regression is used to solve the regression problem and predict continuous values, whereas Logistic regression is used to solve the classification problem and used to predict the discrete values. Instead of fitting the best fit line, it forms an S-shaped curve that lies between 0 and 1. The S-shaped curve is also known as a logistic function that uses the concept of the threshold. Any value above the threshold will tend to 1, and below the threshold will tend to 0.

Common use cases for this algorithm would be to identify whether the given handwriting matches to the person in question, will the prices of oil go up in coming months.



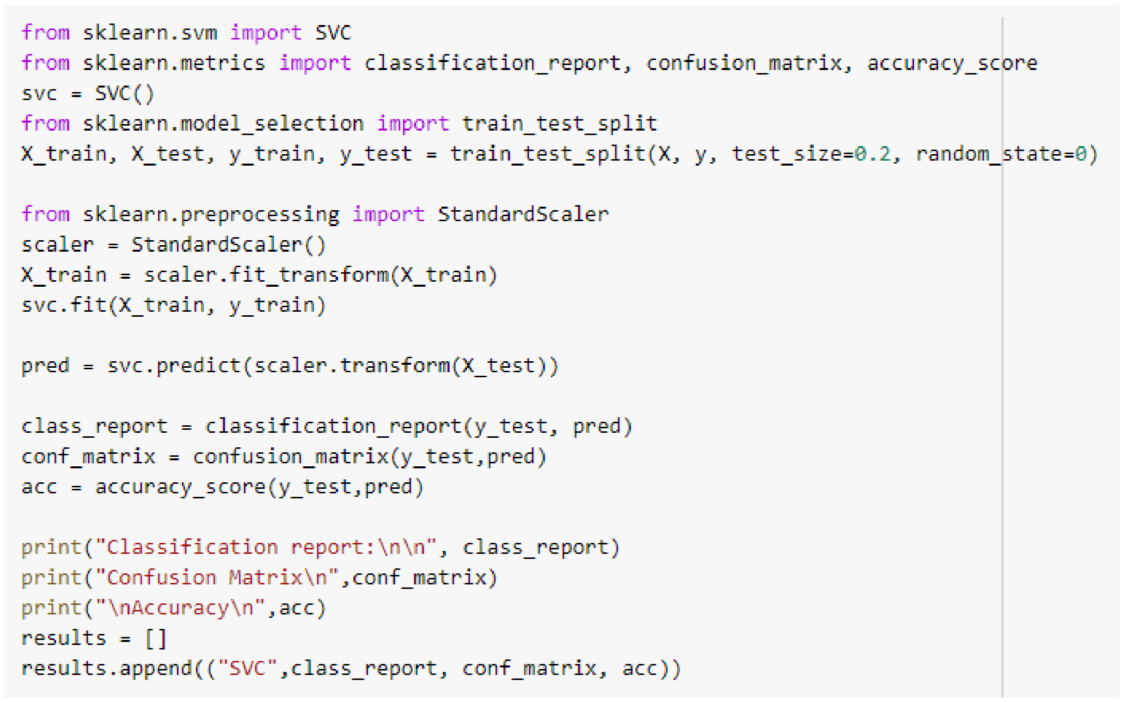
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
clf = LogisticRegression(solver='lbfgs', random_state=42, multi_class='auto')
clf.fit(x_train, y_train.values.ravel())

y_pred = np.array(clf.predict(x_test))
conf_mat = pd.DataFrame(confusion_matrix(y_test, y_pred),
                        columns=["Pred.Negative", "Pred.Positive"],
                        index=['Act.Negative', "Act.Positive"])
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
accuracy = round((tn+tp)/(tn+fp+fn+tp), 4)
print(conf_mat)
print(f'\n Accuracy = {round(100*accuracy, 2)}%')
```

4.**SVM (Support Vector Machine) Algorithm**

A support vector machine or SVM is a supervised learning algorithm that can also be used for classification and regression problems. However, it is primarily used for classification problems. The goal of SVM is to create a hyperplane or decision boundary that can segregate datasets into different classes.The data points that help to define the hyperplane are known as **support vectors**, and hence it is named as support vector machine algorithm. SVMs can handle non-linearly separable data by using a technique called the kernel trick, which maps the input data into a higher-dimensional space where a linear boundary can be found.

SVM algorithm is a method of a classification algorithm in which we plot raw data as points in an n-dimensional space (where n is the number of features). The value of each feature is then tied to a particular coordinate, making it easy to classify the data. Lines called classifiers can be used to split the data and plot them on a graph.

Some real-life applications of SVM are face detection, image classification, Drug discovery, etc.



```
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
svc = SVC()
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, random_state=0)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
svc.fit(X_train, y_train)

pred = svc.predict(scaler.transform(X_test))

class_report = classification_report(y_test, pred)
conf_matrix = confusion_matrix(y_test,pred)
acc = accuracy_score(y_test,pred)

print("Classification report:\n\n", class_report)
print("Confusion Matrix\n",conf_matrix)
print("\nAccuracy\n",acc)
results = []
results.append(("SVC",class_report, conf_matrix, acc))
```

5.Random Forest Algorithm

Random forest is the supervised learning algorithm that can be used for both classification and regression problems in machine learning. It is an ensemble learning technique that provides the predictions by combining the multiple classifiers and improve the performance of the model.

It contains multiple decision trees for subsets of the given dataset, and find the average to improve the predictive accuracy of the model. A random-forest should contain 64-128 trees. The greater number of trees leads to higher accuracy of the algorithm.

To classify a new dataset or object, each tree gives the classification result and based on the majority votes, the algorithm predicts the final output.Random forest is a fast algorithm, and can efficiently deal with the missing & incorrect data

Random Forests applications can be found in :

1.Fraud detection for bank accounts, credit card

2.Detect and predict the drug sensitivity of a medicine

3.Identify a patient's disease by analyzing their medical records

Algorithm

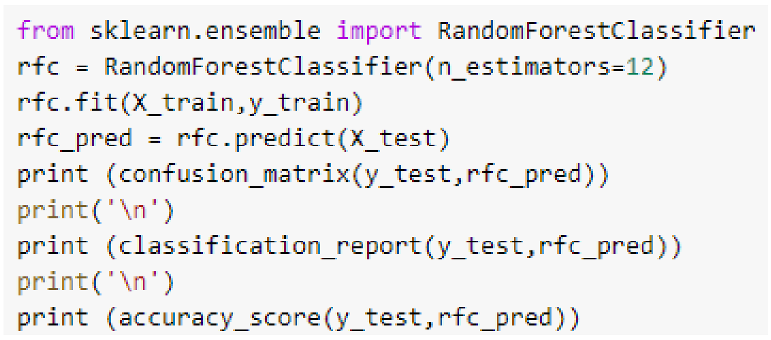
Step 1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees we want to build.

Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that has the majority.

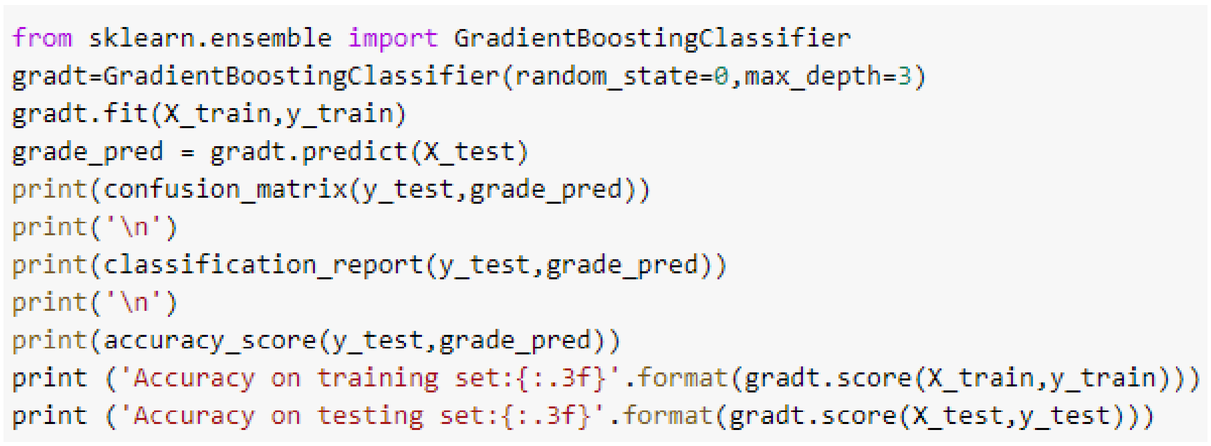


```python
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=12)
rfc.fit(X_train,y_train)
rfc_pred = rfc.predict(X_test)
print (confusion_matrix(y_test,rfc_pred))
print('\n')
print (classification_report(y_test,rfc_pred))
print('\n')
print (accuracy_score(y_test,rfc_pred))
```

6.Gradient Boosting algorithm

Boosting is a technique for ensemble ML algorithms converting weak learners to strong learners. Boosting algorithms are required when data is abundant, and we seek to reduce the bias and variance in supervised learning. It is a popular boosting algorithm.

Gradient Boosting algorithm is used for classification and regression problems by building a prediction model typically in an iterative manner such as the decision trees. It improves the weak learners by training it on the errors of the strong learners resulting in an overall accurate learner.



```
from sklearn.ensemble import GradientBoostingClassifier
gradt=GradientBoostingClassifier(random_state=0,max_depth=3)
gradt.fit(X_train,y_train)
grade_pred = gradt.predict(X_test)
print(confusion_matrix(y_test,grade_pred))
print('\n')
print(classification_report(y_test,grade_pred))
print('\n')
print(accuracy_score(y_test,grade_pred))
print ('Accuracy on training set:{:.3f}'.format(gradt.score(X_train,y_train)))
print ('Accuracy on testing set:{:.3f}'.format(gradt.score(X_test,y_test)))
```

7.K-nearest Neighbors

K-nearest neighbors is a supervised ML algorithm **used for both regression and classification problems**.

It is usually implemented for pattern recognition, this algorithm first stores, and identifies the distance between all inputs in the data using a distance function, selects the k specified inputs closest to query and outputs:

The most frequent label (for classification)

The average value of k nearest neighbors (for regression)

This algorithm works by assuming the similarities between the new data point and available data points. Based on these similarities, the new data points are put in the most similar categories. KNN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.

Applications include :

1.Fingerprint detection

2.Credit rating

3.Forecasting the stock market

Algorithm

Step-1: Select the number K of the neighbors.

Step-2: Calculate the Euclidean distance of **K number of neighbors**

Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.

Step-4: Among these k neighbors, count the number of the data points in each category.

Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.

```python
from sklearn.neighbors import KNeighborsClassifier
acc_scores = []
fbeta_scores = []
k_range = range(1,16)
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train,y_train)
    y_pred = knn.predict(X_test)
    acc_scores.append(accuracy_score(y_test,y_pred))
    fbeta_scores.append(fbeta_score(y_test, y_pred, beta=2))
for i in range(len(acc_scores)):
    print ("{}: Accuracy is {}, f-score is {}".format(i+1,acc_scores[i], fbeta_scores[i]))
```

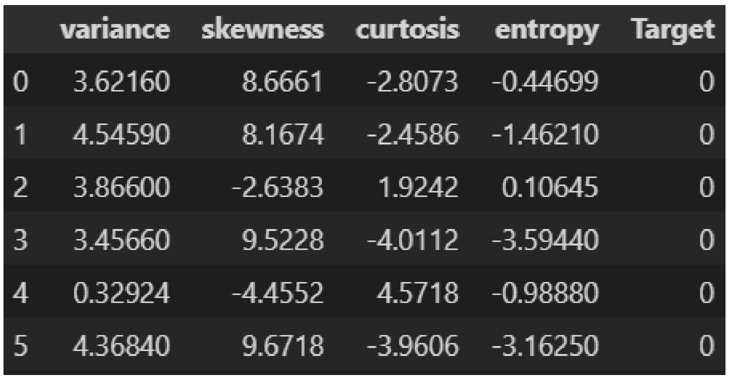
8. Principal Component Analysis (PCA)

PCA is a technique for dimensionality reduction and is often used in machine learning and data analysis. The code imports PCA from the sklearn.decomposition library and creates an instance of the PCA class with n_components=3, which means that it will reduce the dimensionality of the input data down to 3 principal components. The fit method is used to fit the PCA model to the input data, and the transform method is used to transform the input data to the new, lower-dimensional space. The resulting transformed data is then plotted using the matplotlib library, where the x-axis represents the first principal component, the y-axis represents the second principal component and the colour of the points represent the class label y.

```python
from sklearn.decomposition import PCA
pca = PCA(n_components = 2, random_state = 0)

transf = pca.fit_transform(X)

plt.scatter(x = transf[:,0], y = transf[:,1])
plt.title("Dataset after transformation with PCA", fontsize = 18)
plt.show()
```

RESULTS AND INFERENCES

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import display
import seaborn as sns

%matplotlib inline

data = pd.read_csv("bank_notes.csv")
display(data.head(n=6))

classes = data['Target']
features = data.drop('Target', axis=1)
```



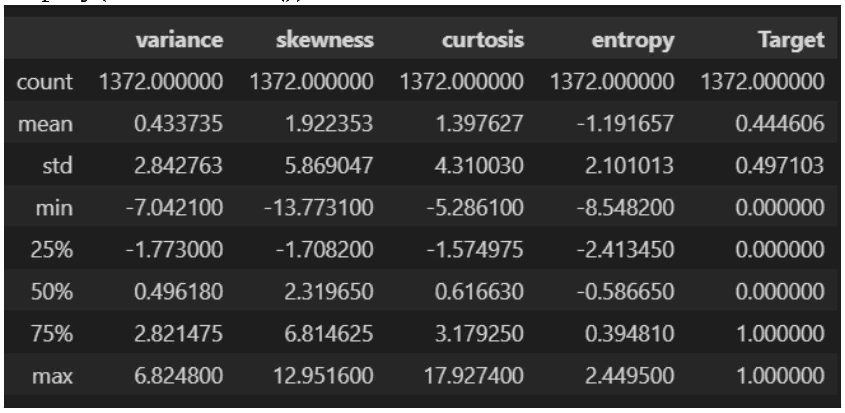
	variance	skewness	curtosis	entropy	Target
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0
5	4.36840	9.6718	-3.9606	-3.16250	0

EXPLORATORY DATA ANALYSIS

```
n_records = len(data)
n_fake_notes = len(data[data['Target'] == 0])
n_real_notes = len(data[data['Target'] == 1])
print ("Total number of records: {}".format(n_records))
print ("Total number of fake notes: {}".format(n_fake_notes))
print ("Total number of real notes: {}".format(n_real_notes))
missing_values = data.isnull().sum().sum()
if missing_values == 0:
    print ("\nThere are no missing values in the dataset")
else:
    print("\nThe dataset has {} missing values".format(missing_values))
```

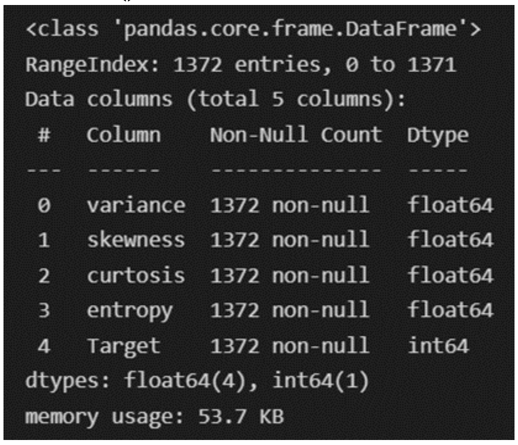
```
Total number of records: 1372
 Total number of fake notes: 762
Total number of real notes: 610
 There are no missing values in the dataset
```

display(data.describe())



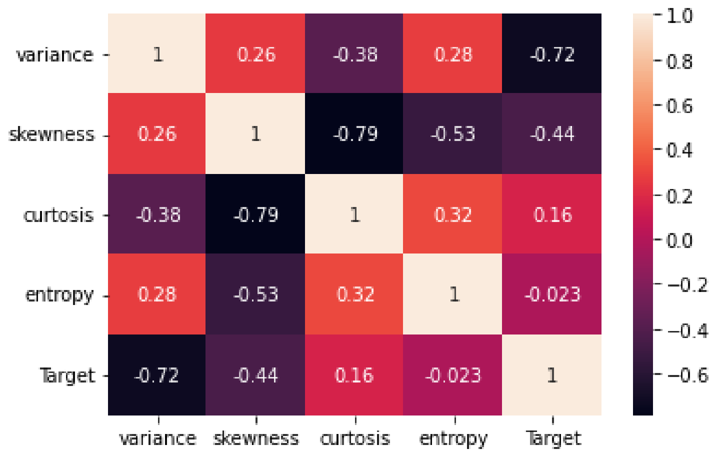
	variance	skewness	curtosis	entropy	Target
count	1372.000000	1372.000000	1372.000000	1372.000000	1372.000000
mean	0.433735	1.922353	1.397627	-1.191657	0.444606
std	2.842763	5.869047	4.310030	2.101013	0.497103
min	-7.042100	-13.773100	-5.286100	-8.548200	0.000000
25%	-1.773000	-1.708200	-1.574975	-2.413450	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

data.info()



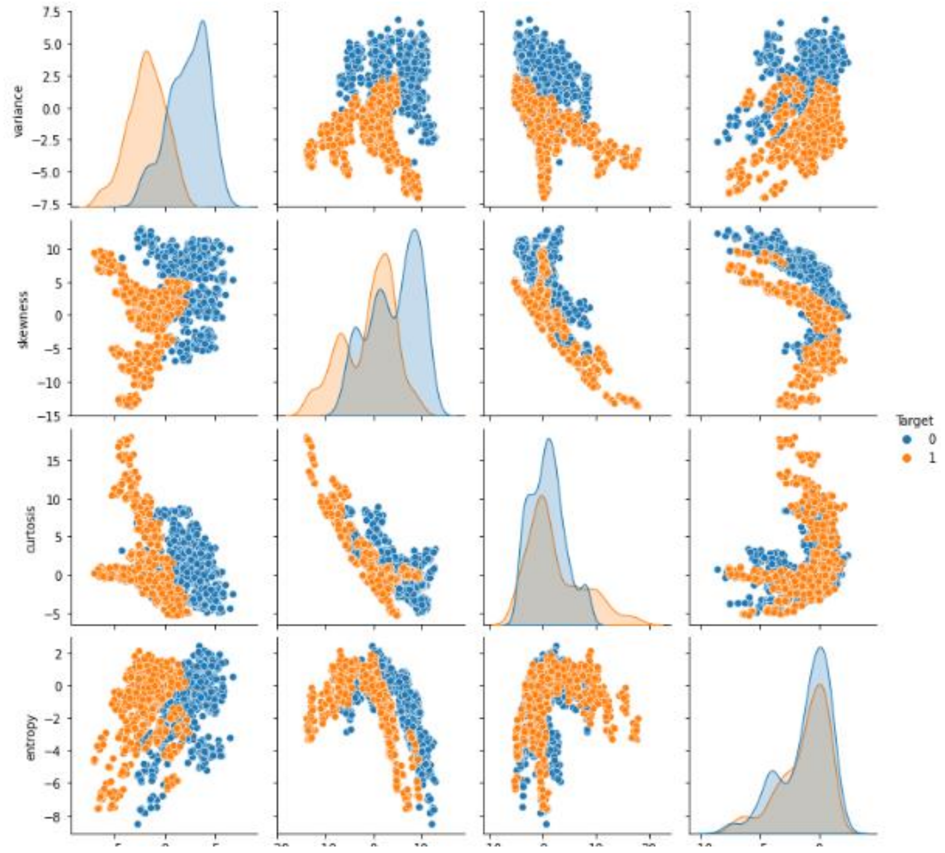
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1372 entries, 0 to 1371
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   variance  1372 non-null   float64
 1   skewness  1372 non-null   float64
 2   curtosis  1372 non-null   float64
 3   entropy   1372 non-null   float64
 4   Target    1372 non-null   int64  
dtypes: float64(4), int64(1)
memory usage: 53.7 KB
```

heatmp = data.corr()
sns.heatmap(heatmp,annot=True)

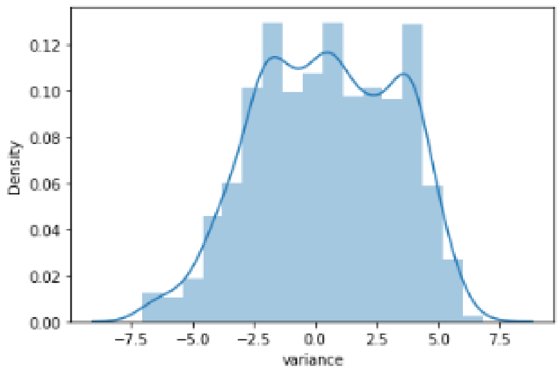


	variance	skewness	curtosis	entropy	Target
variance	1	0.26	-0.38	0.28	-0.72
skewness	0.26	1	-0.79	-0.53	-0.44
curtosis	-0.38	-0.79	1	0.32	0.16
entropy	0.28	-0.53	0.32	1	-0.023
Target	-0.72	-0.44	0.16	-0.023	1

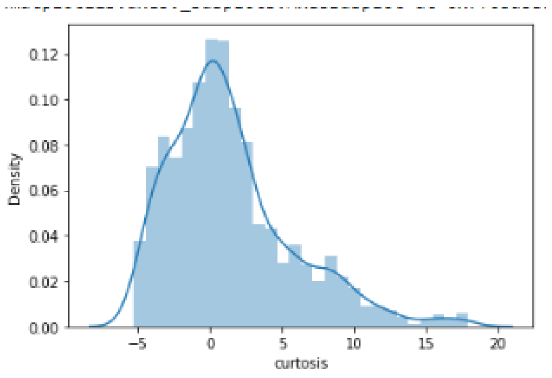
sns.pairplot(data,hue='Target')



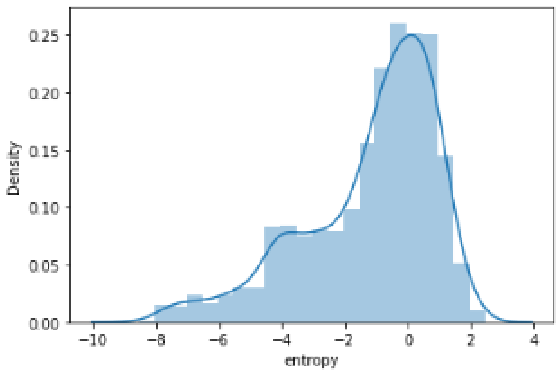
sns.distplot(data.variance)



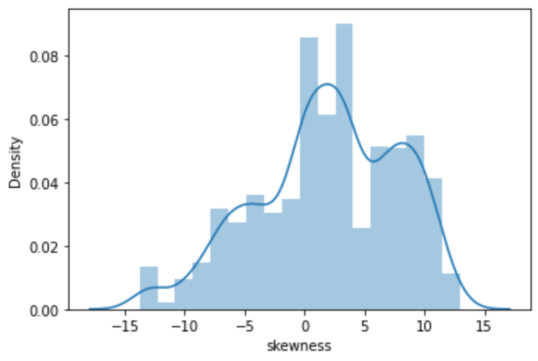
sns.distplot(data.curtosis)

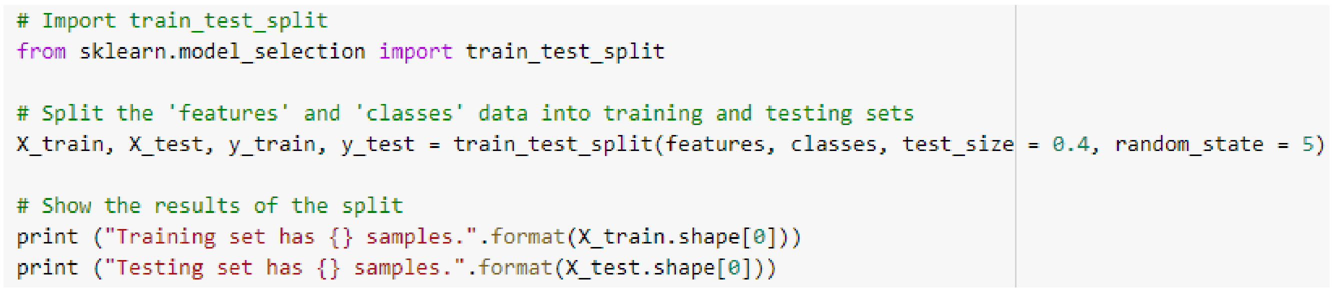


sns.distplot(data.entropy)



sns.distplot(data.skewness)





```
# Import train_test_split
from sklearn.model_selection import train_test_split

# Split the 'features' and 'classes' data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, classes, test_size = 0.4, random_state = 5)

# Show the results of the split
print ("Training set has {} samples.".format(X_train.shape[0]))
print ("Testing set has {} samples.".format(X_test.shape[0]))
```

```
Training set has 823 samples.
Testing set has 549 samples.
```

Performance of supervised learning classification algorithms

Naïve Bayes Classifier

```
Naive Predictor: [Accuracy score: 0.5554, F-score: 0.8620]
```

Decision tree

Confusion Matrix:-

```
[[295  10]
 [ 10 234]]
```

Accuracy, precision, recall and f1 score of algorithm:

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       305
           1       0.96      0.96      0.96       244

    accuracy                           0.96       549
   macro avg       0.96      0.96      0.96       549
weighted avg       0.96      0.96      0.96       549
```

```
0.9635701275045537
```

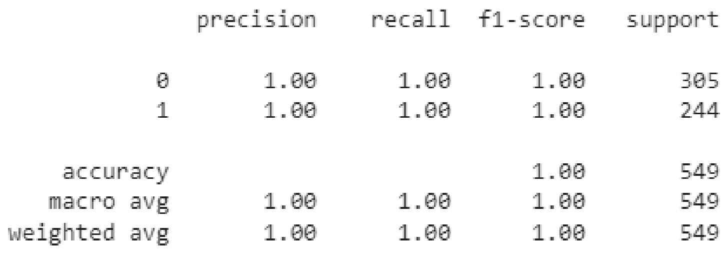
```
Accuracy on training set:1.000
Accuracy on testing set:0.964
```

Gradiant boosting

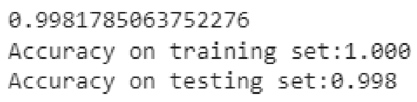
Confusion Matrix:-

```
[[304   1]
 [  0 244]]
```

Accuracy, precision, recall and f1 score of algorithm:

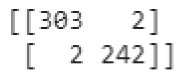


```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       305
           1       1.00      1.00      1.00       244

    accuracy                           1.00       549
   macro avg       1.00      1.00      1.00       549
weighted avg       1.00      1.00      1.00       549
```



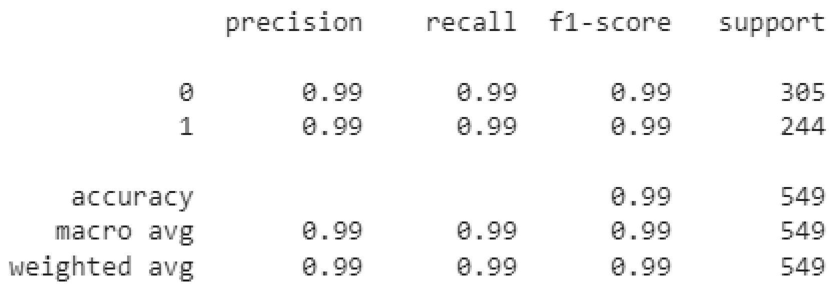
```
0.9981785063752276
Accuracy on training set:1.000
Accuracy on testing set:0.998
```

Random forest

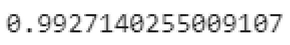


```
[[303   2]
 [  2 242]]
```

Accuracy, precision, recall and f1 score of algorithm:

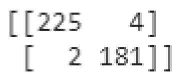


```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       305
           1       0.99      0.99      0.99       244

    accuracy                           0.99       549
   macro avg       0.99      0.99      0.99       549
weighted avg       0.99      0.99      0.99       549
```

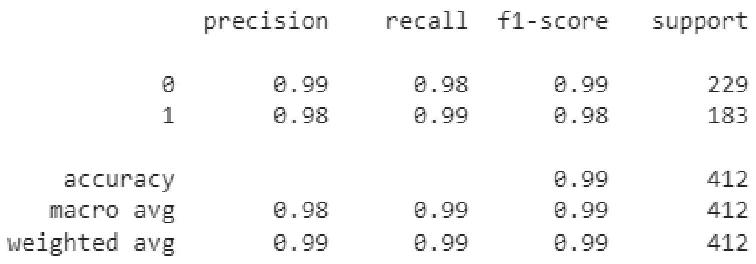


```
0.9927140255009107
```

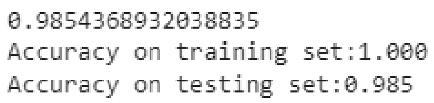
Standard Scaler



```
[[225   4]
 [  2 181]]
```



```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       229
           1       0.98      0.99      0.98       183

    accuracy                           0.99       412
   macro avg       0.98      0.99      0.99       412
weighted avg       0.99      0.99      0.99       412
```



```
0.9854368932038835
Accuracy on training set:1.000
Accuracy on testing set:0.985
```

SVM

```
Classification report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       157
           1       1.00      1.00      1.00       118

    accuracy                           1.00       275
   macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275

Confusion Matrix
 [[157   0]
 [  0 118]]

Accuracy
 1.0
```

KNN

```
1: Accuracy is 0.9981785063752276, f-score is 0.999213217938631
2: Accuracy is 0.9981785063752276, f-score is 0.999213217938631
3: Accuracy is 0.9981785063752276, f-score is 0.999213217938631
4: Accuracy is 0.9981785063752276, f-score is 0.999213217938631
5: Accuracy is 0.9981785063752276, f-score is 0.999213217938631
6: Accuracy is 0.9981785063752276, f-score is 0.999213217938631
7: Accuracy is 0.9981785063752276, f-score is 0.999213217938631
8: Accuracy is 0.9981785063752276, f-score is 0.999213217938631
9: Accuracy is 0.9981785063752276, f-score is 0.999213217938631
10: Accuracy is 0.9981785063752276, f-score is 0.999213217938631
11: Accuracy is 0.9890710382513661, f-score is 0.995297805642633
12: Accuracy is 0.9890710382513661, f-score is 0.995297805642633
13: Accuracy is 0.9890710382513661, f-score is 0.995297805642633
14: Accuracy is 0.9890710382513661, f-score is 0.995297805642633
15: Accuracy is 0.9890710382513661, f-score is 0.995297805642633
```

Logistic regression

```
              Pred.Negative  Pred.Positive
Act.Negative            187              6
Act.Positive              0            173

 Accuracy = 98.36%
```

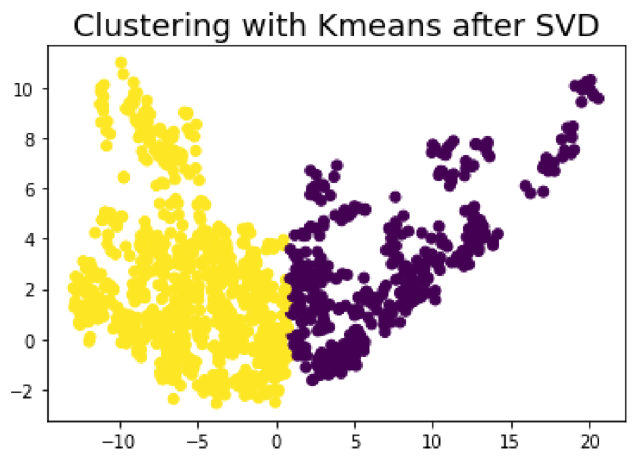
```
new_banknote = np.array([4.5, -8.1, 2.4, 1.4], ndmin=2)
new_banknote = scalar.transform(new_banknote)
print(f'Prediction:  Class{clf.predict(new_banknote)[0]}')
print(f'Probability [0/1]:  {clf.predict_proba(new_banknote)[0]}')
```

```
Prediction:  Class0
Probability [0/1]:  [0.61112576 0.38887424]
```

RESULT-The given banknote is genuine or real as probability of real currency note(0.611) is higher than fake note(0.388).

K-means clustering

```
from sklearn.cluster import KMeans

km = KMeans(n_clusters = 2)
c = km.fit_predict(transf)

plt.scatter(x = transf[:,0], y = transf[:,1], c = c)
plt.title("Clustering with Kmeans after SVD", fontsize = 18)
plt.show()
```



PCA

```
Text(0, 0.5, 'second PC')
```

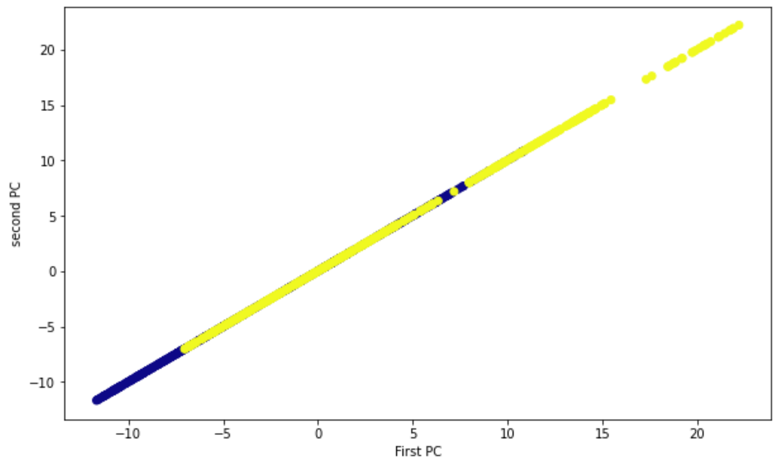


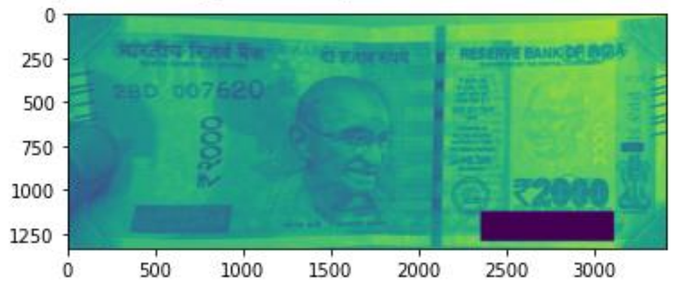
IMAGE PROCESSING

```
import cv2
import matplotlib.pyplot as plt
import numpy as np

A = cv2.imread('Real.jpg')
P = cv2.imread('fake.jpg')
plt.imshow(A)
```

```
<matplotlib.image.AxesImage at 0x7f2281e3e910>
```

```
p = cv2.cvtColor(P, cv2.COLOR_BGR2GRAY)
plt.imshow(a)
```

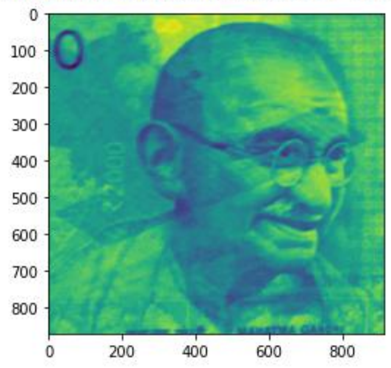
```
<matplotlib.image.AxesImage at 0x7f2281e319a0>
```



Mahatma Gandhi portraits of Real and fake notes

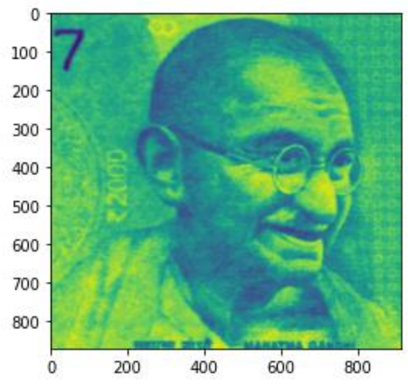
```
a2tr = a[330:1200, 1016:1927]
plt.imshow(a2tr)
```

```
<matplotlib.image.AxesImage at 0x7f227fd87e20>
```



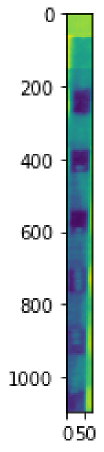
```
b2tr = p[170:1040, 716:1627]
plt.imshow(b2tr)
```

```
<matplotlib.image.AxesImage at 0x7f227fd647f0>
```



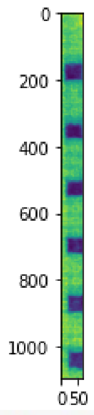
```
print(a.shape)
a2_str = a[5:1100, 2080:2151]
plt.imshow(a2_str)
```

```
(1332, 3416)
<matplotlib.image.AxesImage at 0x7f227fcc22e0>
```



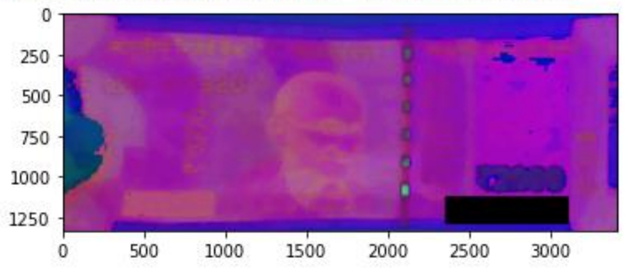
```
print(p.shape)
p2_str = p[5:1100, 1666:1729]
plt.imshow(p2_str)
```

```
(1100, 3000)
<matplotlib.image.AxesImage at 0x7f227fc898b0>
```



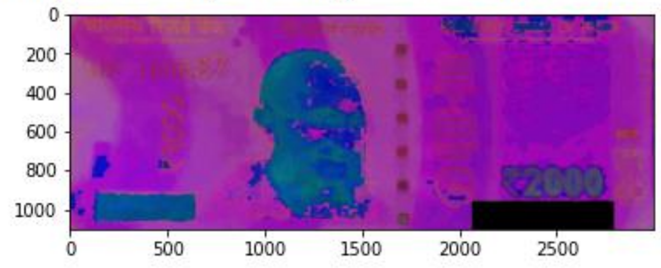
```
plt.imshow(hsvImageReal)
```

```
<matplotlib.image.AxesImage at 0x7f227fc54b20>
```

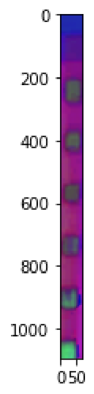


```
plt.imshow(hsvImageFake)
```

```
<matplotlib.image.AxesImage at 0x7f227fc27e80>
```

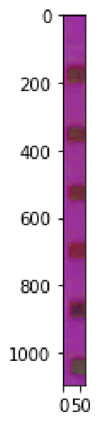


```
<matplotlib.image.AxesImage at 0x7f227fb7aeb0>
```



```
croppedImageFake = hsvImageFake[5:1100, 1666:1729]
plt.imshow(croppedImageFake)
```

```
<matplotlib.image.AxesImage at 0x7f227fb4c580>
```



```
satThresh = 0.3
valThresh = 0.9
g = croppedImageReal[:,:,1]>satThresh
h = croppedImageReal[:,:,2] < valThresh

g1 = croppedImageFake[:,:,1]>satThresh
h1 = croppedImageFake[:,:,2] < valThresh
BWImageReal = g&h
BWImageFake = g1&h1
```

```
def bwareaopen(img, min_size, connectivity=8):
    num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(
        img, connectivity=connectivity)

    for i in range(num_labels):
        label_size = stats[i, cv2.CC_STAT_AREA]

        if label_size < min_size:
            img[labels == i] = 0

    return img
```

```
binr2 = cv2.threshold(p2_str, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
kernel2 = np.ones((3, 3), np.uint8)
invert2 = cv2.bitwise_not(binr2)
BWImageCloseFake = cv2.morphologyEx(invert2, cv2.MORPH_GRADIENT, kernel2)
```

```
binr = cv2.threshold(a2_str, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
kernel = np.ones((3, 3), np.uint8)
invert = cv2.bitwise_not(binr)
BWImageCloseReal = cv2.morphologyEx(invert, cv2.MORPH_GRADIENT, kernel)
```

```
areaopenReal = bwareaopen(BWImageCloseReal, 15);
areaopenFake = bwareaopen(BWImageCloseFake, 15);
bw = areaopenReal
labels = np.zeros(bw.shape)
countReal = cv2.connectedComponentsWithStats(bw, labels,8);
```

```
def corr2(A, B):

    A_mA = A - A.mean(1)[:, None]
    B_mB = B - B.mean(1)[:, None]

    ssA = (A_mA**2).sum(1)
    ssB = (B_mB**2).sum(1)

    return np.dot(A_mA, B_mB.T) / np.sqrt(np.dot(ssA[:, None],ssB[None]))
```

```
co=corr2 (a2tr, b2tr)
if (co.any()>=0.5):
    print ('correlevance of gandhi > 0.5')
    if (countReal[0] == countFake[0] ):
        print ('currency is legitimate')
else:
    print ('correlevance of gandhi < 0.5')
    print ('currency is fake')
```

```
correlevance of gandhi > 0.5
```

RESULT – This shows that the given banknote which was subjected to image processing techniques is real as correlevance of image of Gandhi on the note is higher than 0.5

Inferences

In this project, detection of fake Indian currency note is done by using both machine learning algorithms as well as image processing principle. This will be the low-cost system. The system will work for denomination of 10, 20, 50, 100, 500, and 2000 for Indian currency. The system also provides accurate and valid results. The process of detection of fake note will be quick and easy.

The performance of different Algorithms are as follows

Algorithm Name	**Accuracy %age**
Naive Bayes Classifier	0.5554
Decision Tree	0.9636
Random Forest	0.9927
SVM	1.0000
Logistic Regression	0.9836
KNN	0.9912

From this table we infer that

1.SVM and Random forest have shown the best performance with high accuracy

2.All supervised learning methods have very good accuracy on testing data

3.Naive Bayes Classifier has very low accuracy and cannot be used for distinguishing between real and fake notes.

4. Using image processing we find that extraordinary results can be completed in less time.It has advantages such as simplicity and high performance and high accuracy.

Limitations

Data Availability

Less number of attributes

Complexity of the task

Variability of fake notes

Image Quality

Overfitting

Future Work

We can upscale this model by providing more data to the algorithms to analyse different counterfeits and improve the accuracy of the models. The results of this project can be used to develop a practical solution for detecting fake currency in real-world scenarios.

Our future work will focus on faster and more accurate fake currency identification with the use of modern image processing algorithms.