```
#opening the DataFrame
import pandas as pd
df = pd.read_csv('BankNote_Authentication.csv')
display(df)

classes = df['class']
features = df.drop('class', axis=1)
```

```
import visuals as vs
```

```
%load_ext autoreload
%autoreload 2
```

on the next cells we will illustrate our dataset and his features, it will help to understand the data distribution

```
vs.distribution(df)
```

```
vs.scatter(df)
```

```
#dimension of the DataFrame
print("Number of rows: {}".format(df.shape[0]))
print("Number of columns: {}\n".format(df.shape[1]))

n_records = len(df)
n_fake_notes = len(df[classes == 0])
n_real_notes = len(df[classes == 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))

#counting the number of missing values in each column
missing_values = df.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))

print(df.isnull().sum())
```

```
#Index object representing the column labels of the DataFrame
df.columns
```

```
#Showing information about the DataFrame
display(df.describe())
```

```
#setting maplotlib to show plots in the notebook and ingoring warnings
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Here we illustrate the division of Variance v Curtosis since it is very easy to understand and to see the difference between them

```
colors = {'0': 'red', '1': 'green'}

plt.scatter(df.variance, df.curtosis, alpha=0.5,
c=df['class'].apply(lambda x: colors[str(x)]))

plt.title('Scatter Plot of Variance v Curtosis')
plt.xlabel('Variance')
plt.ylabel('Curtosis')
plt.show()
```

There is no obvious cluster in spherical shapes. We had to check many K's values but we found that 9 is the optimal as we can see in the next cell

```
# k-mean implementation on dataset with a loop of iterations to check
whether k mean is stable or not

import numpy as np
from sklearn.cluster import KMeans

n_iter = 9
fig, ax = plt.subplots(3, 3, figsize=(10,10))
ax = np.ravel(ax)
for i in range(n_iter):
  km = KMeans(n_clusters=2,max_iter=3)
  km.fit(df)
  centroids=km.cluster_centers_
  ax[i].scatter(df[km.labels_== 0]['variance'], df[km.labels_ == 0]
['skewness'],label='cluster 1')
  ax[i].scatter(df[km.labels_ == 1]['variance'], df[km.labels_ == 1]
['skewness'],label='cluster 2')
  ax[i].scatter(centroids[:, 0], centroids[:, 1],c='r', marker='*',
s=100, label='centroid')
  ax[i].legend()
  plt.tight_layout();
```

After running K-Means 9 times, the results we got are very similar, which means the K-Means is stable.

```
# cloning df into df1 and keeping only 2 feature

df1=df.copy()
df1.drop(['curtosis','entropy','class'],axis=1,inplace=True)
df1.head()
```

Here we illustrate the division of Variance v Skewness since it is very easy to understand and to see the difference between them

```python
clusters = KMeans(2)

clusters.fit(df1)
df1['clusterid'] = clusters.labels_

colors = {'0': 'red', '1': 'green'}

plt.scatter(df1.variance, df1.skewness, alpha=0.5,
c=df1['clusterid'].apply(lambda x: colors[str(x)]))

plt.title('Scatter Plot of Variance v Skewness')
plt.xlabel('Variance')
plt.ylabel('Skewness')
plt.show()

#getting centroids of cluster
clusters.cluster_centers_

df1.head()

# calculating descriptive statistics for each cluster
df1.groupby( 'clusterid' ).describe()

from sklearn.preprocessing import StandardScaler

#normalization the data, 0 mean and 1 variance
scaler = StandardScaler()
scaled_df1 = scaler.fit_transform( df1[["variance", "skewness"]] )

scaled_df1=pd.DataFrame(scaled_df1,columns=['variance','skewness'])
scaled_df1

# reproducibility of results and switching ID's

clusters_new = KMeans( 2, random_state=42 )
clusters_new.fit(scaled_df1)
df1["clusterid_new"] = clusters_new.labels_
df1.head()

colors = {'0': 'red', '1': 'green'}

#plt.scatter(df1.variance, df1.skewness, alpha=0.5,
c=df1['clusterid'].apply(lambda x: colors[str(x)]))

plt.scatter(df1.variance,df1.skewness,alpha=0.5,
c=df1['clusterid_new'].apply(lambda x: colors[str(x)]))
plt.show()

#visualizing the data with correct labels
plt.scatter(df['variance'],df['skewness'],c=df['class'])
plt.xlabel('Variance')
```

```
plt.ylabel('Skewness')
plt.colorbar(label='class')
plt.show()
```

```
# new centroids of clusters

clusters_new.cluster_centers_
```

```
df1["clusterid_new"] = df1["clusterid_new"].map({0: 1, 1: 0})
```

```
df1.shape
```

```
df1=df1.reset_index()
df1
```

```
# evaluate the accuracy of a clustering algorithm that assigns cluster
labels

correct=0
for i in range(0,1371):
  if df['class'][i]==df1['clusterid_new'][i]:
   correct=correct+1

print(correct/1372)
```

Using PCA model to reduce the number of features into 2 columns

```
df.head()
```

```
#normalization the data, 0 mean and 1 variance
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_df2 = scaler.fit_transform( df[["variance",
"skewness","curtosis","entropy"]])
```

```
scaled_df2=pd.DataFrame(scaled_df2,columns=['variance','skewness',"cur
tosis","entropy"])
scaled_df2
```

```
# reducing dimensionality

from sklearn.decomposition import PCA

pca=PCA(n_components=2)
pca.fit(scaled_df2)
PCA_df=pd.DataFrame(pca.transform(scaled_df2),columns=(['col1','col2']
))
PCA_df.head()
```

```
plt.figure(figsize=(5,5))
plt.scatter(PCA_df['col1'],PCA_df['col2'])
```

```
plt.title('Scatter Plot of col1 vs col2')
plt.show()

#determine the optimal number of clusters for KMeans

cluster_range=range(1,11)
cluster_errors=[]
for num_clusters in cluster_range:
    clusters=KMeans(num_clusters)
    clusters.fit(PCA_df)
    cluster_errors.append(clusters.inertia_)
plt.figure(figsize=(6,4))
plt.plot(cluster_range,cluster_errors,marker='o')
plt.title('Elbow method to find the number of clusters')
plt.show()

# performing hierarchical clustering on a dataset represented by the
PCA transformed dataframe PCA_df.

from sklearn.cluster import AgglomerativeClustering

AC=AgglomerativeClustering(n_clusters=4)
AC.fit(PCA_df)

yhat_AC=AC.fit_predict(PCA_df)
yhat_AC

PCA_df['Clusters']=yhat_AC
PCA_df

plt.scatter(PCA_df['col1'],PCA_df['col2'],c=PCA_df['Clusters'])
plt.show()
```

Training the Model with the dataset

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from keras import regularizers

model = Sequential()


# load dataset
df = pd.read_csv('BankNote_Authentication.csv')

# split data into training and testing sets
```

```python
X_train, X_test, y_train, y_test =
train_test_split(df.drop('class',axis=1), df['class'], test_size=0.2)

# build the model
model = Sequential()
model.add(Dense(128, activation='relu', input_dim=X_train.shape[1]))
model.add(Dropout(0.2))
model.add(Dense(64, input_dim=64, activation='relu',
kernel_regularizer=regularizers.l2(0.01)))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))

# compile the model
optimizer = Adam(lr=0.001)
model.compile(loss='binary_crossentropy', optimizer=optimizer,
metrics=['accuracy'])

# set up a checkpoint to save the best model during training
checkpoint = ModelCheckpoint('model.h5', monitor='val_accuracy',
save_best_only=True, mode='max')

# train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=300,
validation_data=(X_test, y_test), callbacks=[checkpoint], verbose = 0)

# evaluate the model on the testing set
loss, accuracy = model.evaluate(X_test, y_test, verbose = 0)


# save the final model
model.save('model.h5')

print("Model saved into model.h5 file")


def detect_image(predicted_class,img):
    # Load image
    image = cv2.imread(img)

    # Check if image has been loaded correctly
    if image is None:
        print("Error: Could not load image")
        return

    # Convert image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray_blur = cv2.GaussianBlur(gray, (21, 21), 0)

    fig, ax = plt.subplots(figsize=(10, 10))
```

```python
    heatmap = ax.imshow(gray_blur, cmap='inferno')
    fig.colorbar(heatmap, ax=ax)

    # Display the heatmap
    plt.show()

    # Apply adaptive thresholding to get binary image
    thresh = cv2.adaptiveThreshold(gray, 255,
cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV, 11, 2)

    # Find contours in the binary image
    ret, thresh = cv2.threshold(gray, 127, 255, 0)
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

    #contours, hierarchy = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # Draw contours on original image
        # Calculate spread of largest contour
    max_contour = max(contours, key=cv2.contourArea)
    x, y, w, h = cv2.boundingRect(max_contour)
    spread = w / h

    # Draw contours on original image
    img_copy = image.copy()
    if predicted_class<=0.5:
        colors_tuple = (0, 0, 255)
    else:
        colors_tuple = (0, 255, 0)
    cv2.drawContours(img_copy, contours, -1, colors_tuple, 2)
    print("Number of contours found:", len(contours))
    print("Spread of largest contour:", spread)


    # Display the image with detected contours
    cv2.namedWindow("Detected Contours", cv2.WINDOW_NORMAL)
    cv2.resizeWindow("Detected Contours", img_copy.shape[1],
img_copy.shape[0])
    cv2.imshow("Detected Contours", img_copy)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

import cv2
import numpy as np
from keras.models import load_model
from sklearn.neighbors import KNeighborsClassifier
from scipy.stats import *
from keras.models import load_model
from tensorflow import keras
```

```python
import h5py
import matplotlib.pyplot as plt



def preprocess_image(image):
    # Load the trained model
    model = load_model('model.h5', compile=False)

    # Resize the image to 64x64 pixels
    resized_image = cv2.resize(image, (64, 64))
    # Convert the image to grayscale
    gray_image = cv2.cvtColor(resized_image, cv2.COLOR_RGB2GRAY)
    # Flatten the image to a 1D array
    flattened_image = gray_image.flatten()
    # Normalize the image by dividing each pixel value by 255
    normalized_image = flattened_image / 255.0
    # Add two extra zeros to the end of the flattened image to make it
a 1D array of four features
    feature_vector = np.concatenate([normalized_image, [0, 0]])


    var = np.var(flattened_image)/1372
    skewness = skew(flattened_image.reshape(-1))
    kurt = kurtosis(flattened_image.reshape(-1))
    hist, _ = np.histogram(flattened_image, bins=256)
    entropy_val = entropy(hist)

    # Predict the class label for the image
    img_features = [[var, skewness, kurt, entropy_val]]
    predicted_class = model.predict(img_features)

    # Print the predicted class label
    print("Predicted Class Label:", predicted_class)
    print("Predicted Class Probabilities 0 for fake|1 for original")
    print("var:", var)
    print("ske:", skewness)
    print("kurt Class Label:", kurt)
    print("entropy_val:", entropy_val)

    # output the prediction
    if predicted_class <= 0.75:
        print("Fake note")
    else:
        print("Real note")

    detect_image(predicted_class,path)
    ##############################
```

```python
    return predicted_class


# Load the image
path = "Example.jpg"
image = cv2.imread(path)
# Preprocess the image dand get reulst
feature_vector = preprocess_image(image)
```