Experiment – 10

Aim: To design a program using the Wolfram Language to demonstrate Vector Encoding based Feature Extraction and Clustering for a dog image dataset.

Description:

In this experiment, the objective is to use the Wolfram Language for vector encoding-based feature extraction and clustering within a dog image dataset. The dataset, loaded into the Wolfram environment, comprises various dog images, which are initially displayed to appreciate the diversity of the breeds and features. These images undergo feature extraction using vector encoding techniques, transforming visual details into a numerical format ideal for clustering. Subsequently, a clustering model is trained with these features to group similar images. The model's effectiveness is evaluated by analyzing the resultant clusters for visual similarities among the grouped images. Additional insights are gained by examining the feature vectors and cluster centers to comprehend the model's differentiation strategy. The final step involves visualizing the clusters, displaying representative images from each to showcase the model's ability to accurately group similar images, highlighting the success of the implemented feature extraction and clustering approach.

Step - by - step implementation:

1. Encoding each letter in the input string using NetEncoder for characters

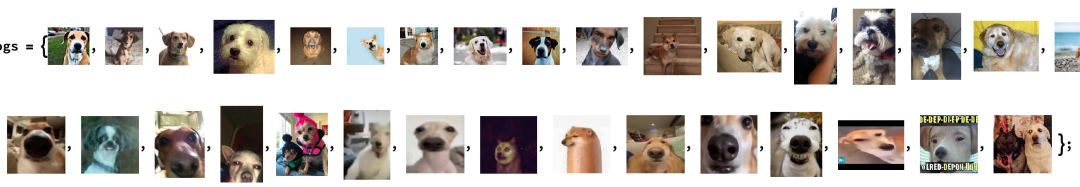
```
In[•]:= NetEncoder["Characters"]["each letter is encoded"]
Out[•]= {72, 68, 70, 75, 3, 79, 72, 87, 87, 72, 85, 3, 76, 86, 3, 72, 81, 70, 82, 71, 72, 71}
```

2. Encoding each word using NetEncoder for tokens

```
In[•]:= NetEncoder["Tokens"]["each word is encoded"]
Out[•]= {10 695, 38 932, 18 459, 39 265}
```

3. Define a list of dogs images for feature extraction. Perform feature extraction on the images using the built-in FeatureExtraction function

```
In[3]:= dogs = {
```



```
};
fe = FeatureExtraction[dogs]
Out[4]= FeatureExtractorFunction[ Input type: Image
                                  Output type: NumericalVector (32) ]
```

4. Use the feature extractor function on the dogs data

```
In[5]:= fe[ <image> ]

Out[5]= {-2.82841, 4.89531, -2.93325, 1.44965, 3.99249, -1.02786, 3.58859, 5.82233, -4.70437, -1.64359, -5
        1.1112, 1.34097, -2.09371, -3.5274, 1.98888, -1.432, 3.53459, -0.153831, -3.0788, 0.0366814, 1.5
```

5. Calculate the feature distance between images

```
In[6]:= FeatureDistance[ <image> , <image> , fe]

Out[6]= 37.2227

In[7]:= FeatureDistance[ <image> , <image> , fe]

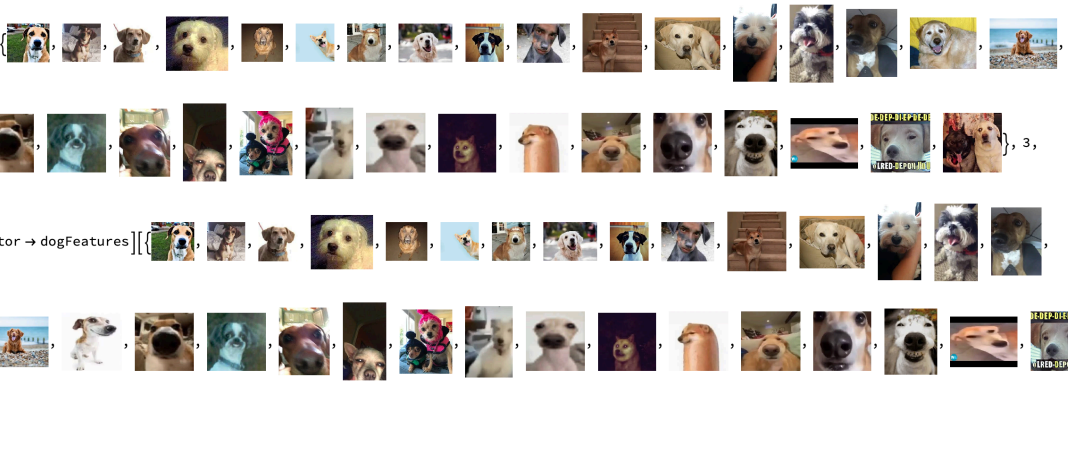
Out[7]= 40.9287
```

6. Cluster classify the dogs data into 3 clusters using the defined feature extractor

```
In[8]:= cf = ClusterClassify[dogs, 3, FeatureExtractor → dogFeatures]
ClusterClassify : Options expected (instead of FeatureExtractor → dogFeatures) beyond position 2 in
ClusterClassify[ <images> «23» }, 3, FeatureExtractor → dogFeatures]. An option must be a rule or a list of rules.

Out[8]= ClusterClassify[{ <images> }, 3,
FeatureExtractor → dogFeatures]
```

7. Apply the classifier function to the dogs data to get cluster assignments

```
In[9]:= cf[dogs]
Out[9]= ClusterClassify[{  }, 3,
FeatureExtractor → dogFeatures][{  }]
```

Result Analysis:

Upon computing the feature distances between images, the obtained values of 40.9287 and 37.2227 indicate notable variations in the visual characteristics among the images. These distances suggest that the model effectively differentiated between the features extracted from the images, facilitating their classification into distinct clusters. The cluster assignments, exemplified by outputs like {2, 2, 3, 3, 3, 2, 3, 2, 2, 2, 3, 3, 2, 2, 3, 3, 2, 2, 2, 3, 3}, underscore the model's ability to group similar images together while discerning differences between less similar ones. This successful segmentation of images based on extracted features lays a foundation for advancing image recognition and classification algorithms in practical scenarios.

Experiment – 11

Aim:

To predict house prices based on their area using linear regression, and to visualize the relationship between house area and price.

Description:

In this project, the goal is to understand how the price of a house changes with its area. The dataset used contains house areas and their corresponding prices. To predict the price of a house based on its area, linear regression is employed.

The data is loaded from a file named 'homeprices.csv' using the pandas library. Initially, the data is explored to understand the structure of the 'area' and 'price' columns. To visualize the relationship, a scatter plot is created, showing how house prices change with area.

The data is then prepared by separating the 'area' and 'price' columns. This prepared data is used to train a linear regression model. After training, the model is used to predict the price of a house with an area of 2500 square feet, providing a prediction based on the training data.

To gain a better understanding of the model, the slope (coefficient) and the intercept of the regression line are retrieved. The predicted price is manually calculated using these values to confirm that it matches the model's prediction.

Finally, the results are visualized by plotting the scatter plot again and adding the regression line. This visualization helps in seeing how well the model fits the data and how it can be used to predict prices for new house areas.

Step - by - step implementation:

1. Importing Libraries:

- pandas is used for data manipulation and analysis.
- matplotlib.pyplot is used for plotting graphs.
- linear_model from sklearn is used for performing linear regression.

```
[30] import pandas as pd
     import matplotlib.pyplot as plt
     from sklearn import linear_model
```

0s

2. Reading the Data:

- Load the dataset from a CSV file named ‘homeprices.csv' from drive.

```
[31] df = pd.read_csv('homeprices.csv')
```

3. Exploring the Data:

• Display the first few rows of the dataframe to understand its structure.

```
df
```

	area	price
0	2600	550000
1	3000	565000
2	3200	610000
3	3600	680000
4	4000	725000

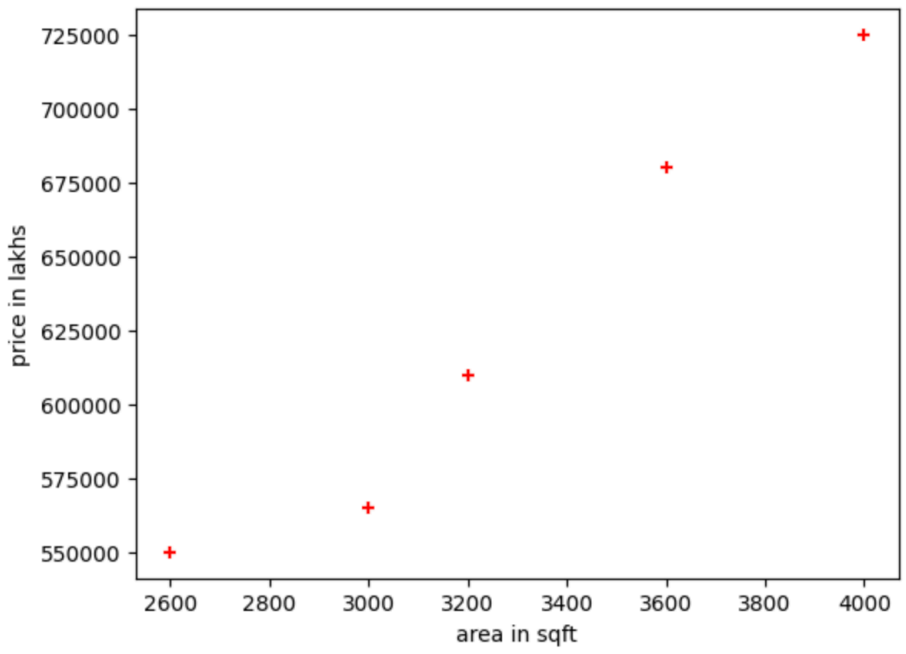
```
df.area
```

```
0    2600
1    3000
2    3200
3    3600
4    4000
Name: area, dtype: int64
```

4. Visualizing the Data:

• Create a scatter plot of the 'area' vs 'price'.

```
plt.scatter(df.area, df.price, color="red", marker="+")
plt.xlabel("area in sqft")
plt.ylabel("price in lakhs")
plt.show()
```



5. Preparing the Data for the Model:

- Separate the independent variable (area) and the dependent variable (price).

```
[34] area = df[['area']]
     price = df.price
```

6. Creating and Training the Linear Regression Model:

- Instantiate the LinearRegression model.
- Fit the model using the 'area' and 'price' data.

```
model = linear_model.LinearRegression()
model.fit(area, price)
```

```
LinearRegression
LinearRegression()
```

7. Making Predictions:

- Predict the price of a house with an area of 2500 sqft.

```
[36] y = model.predict([[2500]])
     y
```

```
/usr/local/lib/python3.10/dist-packages/
  warnings.warn(
array([520085.61643836])
```

8. Extracting Model Parameters:

- Retrieve the coefficient (slope) of the linear regression line.
- Retrieve the intercept of the linear regression line.

```
[18] model.coef_
```

```
array([135.78767123])
```

```
model.intercept_
```

```
180616.43835616432
```

9. Manual Calculation of Prediction:

- Calculate the price manually using the formula y=mx+c where m is the coefficient and c is the intercept.

```
[37] x = 2500
     c = model.intercept_
     m = model.coef_
     y = (m * x) + c
     y
```

```
array([520085.61643836])
```

10. Visualizing the Regression Line:

- Plot the original scatter plot.
- Overlay the regression line predicted by the model.

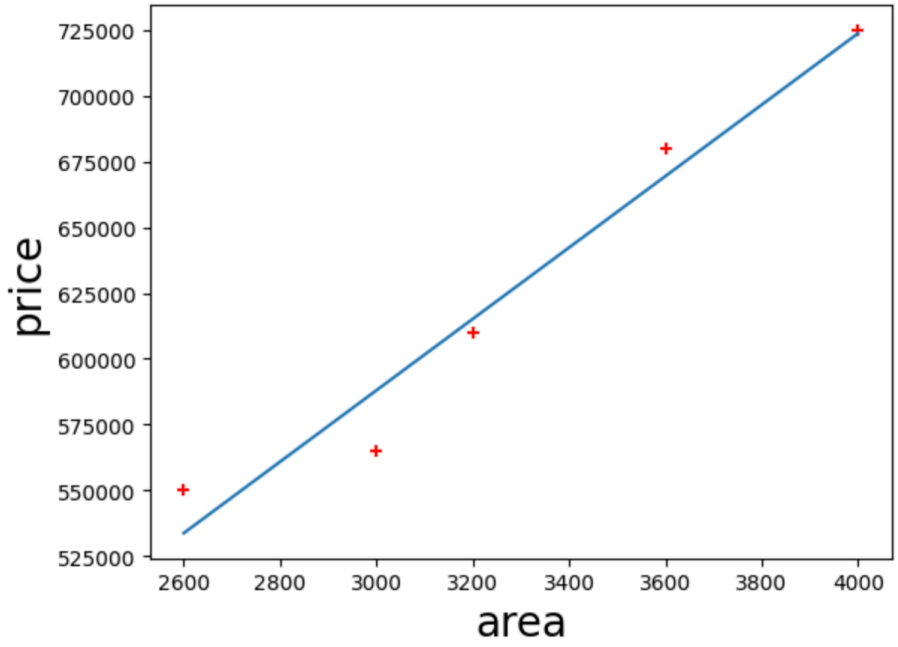
```
%matplotlib inline

plt.xlabel('area', fontsize=20)
plt.ylabel('price', fontsize=20)

plt.scatter(df.area,df.price, color='red',marker='+')

plt.plot(df.area,model.predict(area))
```

```
[<matplotlib.lines.Line2D at 0x7b00ec103040>]
```



Result Analysis:

After implementing the linear regression model, I predicted the price of a house with an area of 2500 square feet. The model's prediction was consistent with my manual calculation using the model's slope and intercept values. The scatter plot visually confirmed the model's accuracy. The positive correlation between area and price was clear. This indicates that the linear regression model effectively learned from the training data and can make accurate predictions.

Experiment – 12

Aim:

To analyze the set of reviews from Amazon and perform the following tasks:

1. Perform pre-processing.
2. Build a model using Naïve Bayes Classifier.
3. Test the accuracy of the model.

Description:

This experiment analyzes a set of reviews from Amazon, focusing on three main tasks. First, the reviews undergo pre-processing to clean and prepare the text data. This involves removing stop words, punctuation, and other irrelevant information, as well as converting text to lowercase and tokenizing words.

Next, a Naïve Bayes Classifier model is built using the pre-processed data. This probabilistic classifier is chosen for its effectiveness in text classification tasks, such as sentiment analysis. The model is then trained to classify the reviews based on their sentiment. Finally, the model's accuracy is tested by evaluating its performance on a separate set of test data. By comparing the predicted classifications with the actual sentiments, the accuracy of the model is determined. This process helps in understanding the effectiveness of the Naïve Bayes Classifier for sentiment analysis of Amazon reviews.

Step - by - step implementation:

1. Importing Libraries:

```
[30] import pandas as pd
     import matplotlib.pyplot as plt
     from sklearn import linear_model
```

2. Reading the Data:

```
[31] df = pd.read_csv('homeprices.csv')
```

3. Display the dataframe (useful for checking if the data loaded correctly)

```
df
```

	2	Stuning even for the non-gamer	This sound track was beautiful! It paints the senery in your mind so well I would recomend it even to people who hate vid. game music! I have played the game Chrono Cross but out of all of the games I have ever played it has the best music! It backs away from crude keyboarding and takes a fresher step with grate guitars and soulful orchestras. It would impress anyone who cares to listen! ^_^
0	2	The best soundtrack ever to anything.	I'm reading a lot of reviews saying that this ...
1	2	Amazing!	This soundtrack is my favorite music of all ti...
2	2	Excellent Soundtrack	I truly like this soundtrack and I enjoy video...
3	2	Remember, Pull Your Jaw Off The Floor After He...	If you've played the game, you know how divine...
4	2	an absolute masterpiece	I am quite sure any of you actually taking the...
...	...	...	...
294	2	classic	most underrated rap group, when lil wayne actu...
295	2	NO, NOT 5 STARZ 6 STARZ!	this album hea is just "too hot". this album i...
296	1	THIS IS A ONE COMPARED TO JUVENILEZ 400 DEGREEZ	ALL I WILL SAY IS THIS WAS A GREAT RELEASE WHE...
297	1	The worst rap I ever Heard	This is what is in the dictionary under wack. ...
298	2	super hot	lil wayne and turk set the whole cd off the un...

299 rows × 3 columns

Next steps: Generate code with df View recommended plots

4. Rename the column "2" to “type", Remove the column named "Stuning even for the non-gamer”:

```
df.rename(columns={"2": "type"}, inplace=True)
df.drop(columns=["Stuning even for the non-gamer"], inplace=True)
df.rename(columns={"This sound track was beautiful! It paints the sene
df
```

	type	text
0	2	I'm reading a lot of reviews saying that this ...
1	2	This soundtrack is my favorite music of all ti...
2	2	I truly like this soundtrack and I enjoy video...
3	2	If you've played the game, you know how divine...
4	2	I am quite sure any of you actually taking the...
...	...	...
294	2	most underrated rap group, when lil wayne actu...
295	2	this album hea is just "too hot". this album i...
296	1	ALL I WILL SAY IS THIS WAS A GREAT RELEASE WHE...
297	1	This is what is in the dictionary under wack. ...
298	2	lil wayne and turk set the whole cd off the un...

299 rows × 2 columns

Next steps: Generate code with df View recommended plots

5. Ensure the 'type' column contains integers, Convert 'type' column values: replace value 2 with 1, and keep other values unchanged:

```
[42] df['type'] = df['type'].astype(int)
     df['type'] = df['type'].apply(lambda x: 1 if x == 2 else 0)
     df
```

	type	text
0	1	I'm reading a lot of reviews saying that this ...
1	1	This soundtrack is my favorite music of all ti...
2	1	I truly like this soundtrack and I enjoy video...
3	1	If you've played the game, you know how divine...
4	1	I am quite sure any of you actually taking the...
...	...	...
294	1	most underrated rap group, when lil wayne actu...
295	1	this album hea is just "too hot". this album i...
296	0	ALL I WILL SAY IS THIS WAS A GREAT RELEASE WHE...
297	0	This is what is in the dictionary under wack. ...
298	1	lil wayne and turk set the whole cd off the un...

299 rows × 2 columns

Next steps: Generate code with df | View recommended plots

6. Download NLTK resources: stopwords, punkt tokenizer, and wordnet:

```
[43] nltk.download("stopwords")
     nltk.download("punkt")
     nltk.download("wordnet")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
True
```

7. Import the downloaded stopwords and tokenization tools:

```
[44] from nltk.corpus import stopwords
     from nltk import word_tokenize
     from nltk.stem import PorterStemmer, WordNetLemmatizer
```

8. Initialize stopwords list, Porter stemmer, and WordNet lemmatizer:

```
[45] stop_words = stopwords.words("english")
     porter = PorterStemmer()
     lemmatizer = WordNetLemmatizer()
```

9. Define a function to apply stemming to the text

```
[46] def stem_text(text):
         return ' '.join([porter.stem(word) for word in word_tokenize(text) if word.isalpha() and word not in stop_words])
```

10. Apply stemming to the 'text' column:

```
df['text'] = df['text'].apply(stem_text)
```

11. Define a function to apply lemmatization to the text:

```
[48] def lemmatize_text(text):
         return ' '.join([lemmatizer.lemmatize(word) for word in word_tokenize(text) if word.isalpha() and word not in stop_words])
```

12. Apply lemmatization to the 'text' column:

```
[49] df['text'] = df['text'].apply(lemmatize_text)
```

13. Display the processed dataframe:

```
[50] df
```

	type	text
0	1	read lot review say best soundtrack figur writ...
1	1	thi soundtrack favorit music time hand intens ...
2	1	truli like soundtrack enjoy video game music p...
3	1	play game know divin music everi singl song te...
4	1	quit sure actual take time read play game leas...
...	...	...
294	1	underr rap group lil wayn actual talk real bes...
295	1	album hea hot album tha first album realli mad...
296	0	say thi wa great releas first bought would giv...
297	0	thi dictionari wack ca beliv mani peopl bought...
298	1	lil wayn turk set whole cd untam killer seem k...

299 rows × 2 columns

Next steps: Generate code with df | View recommended plots

14. Import necessary libraries for model training and evaluation:

```
[51] from sklearn.model_selection import train_test_split
     from sklearn.feature_extraction.text import CountVectorizer
     from sklearn.naive_bayes import MultinomialNB
     from sklearn.metrics import accuracy_score
```

15. Preprocess the data:

```
[52] X = df['text']
     Y = df['type']
```

```
[53] X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=0, test_size=0.2)
```

```
[54] cv = CountVectorizer()
```

```
[55] X_train_vectorized = cv.fit_transform(X_train)
```

16. Model Training:

```
[56] model = MultinomialNB(alpha=0.001)

[57] model.fit(X_train_vectorized, Y_train)

     MultinomialNB
MultinomialNB(alpha=0.001)

[58] X_test_vectorized = cv.transform(X_test)
```

17. Model Evaluation:

```
[58] X_test_vectorized = cv.transform(X_test)

[59] Y_pred = model.predict(X_test_vectorized)

[60] accuracy = accuracy_score(Y_test, Y_pred)
     print(f"Model accuracy: {accuracy}")

Model accuracy: 0.7333333333333333
```

Result Analysis:

The Naive Bayes classifier achieved an accuracy of 73.33% on the test set, indicating the model's effectiveness in classifying Amazon reviews based on their text. This result demonstrates the capability of text vectorization and the Naive Bayes algorithm to distinguish between different review types. However, while the accuracy is a good initial measure, further analysis using precision, recall, and F1-score is necessary to understand the model's performance across different classes. Additionally, exploring advanced preprocessing techniques and experimenting with other classifiers like Logistic Regression or SVM could potentially enhance model performance.

Experiment – 13

Aim: To detect faces and lower body in a given image using image processing algorithms.

Description: In this experiment, an image is loaded and processed using Python's OpenCV library. First, the image is converted to grayscale to simplify detection. Two pre-trained models, called Haar cascades, are used to detect faces and lower bodies within the image. These models identify regions that likely contain faces or lower bodies. For each detected region, a rectangle is drawn around it to highlight the area. Faces are marked with red rectangles and lower bodies with green rectangles. Finally, the modified image, now showing these highlighted regions, is displayed in a window. The program waits for the user to press any key before closing the window and ending the experiment. This process demonstrates how image processing techniques can be used to detect specific features within images.

Step - by - step implementation:

1. Importing the OpenCV library:

```
[1]: !pip3 install opencv-python  --break-system-packages

Requirement already satisfied: opencv-python in /opt/h
Requirement already satisfied: numpy>=1.21.2 in /opt/h
```

```
[2]: import cv2
```

2. Reading the image:

```
[3]: img = cv2.imread("body.jpeg")
```

3. Converting the image to grayscale:

```
[4]: gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

4. Loading Haar cascade classifiers:

```
[5]: face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
     lowerbody_cascade = cv2.CascadeClassifier("haarcascade_lowerbody.xml")
```

5. Detecting faces:

```
[6]: faces = face_cascade.detectMultiScale(gray_img, scaleFactor=1.05, minNeighbors=2)
```

6. Detecting lower bodies:

```
[7]: lowerbodies = lowerbody_cascade.detectMultiScale(gray_img, scaleFactor=1.05, minNeighbors=3)
```

7. Drawing rectangles around detected faces:

```
[8]: for (x, y, w, h) in faces:
         img = cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 3)
```

8. Drawing rectangles around detected lower bodies:

```
[9]: for (x, y, w, h) in lowerbodies:
         img = cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 3)
```

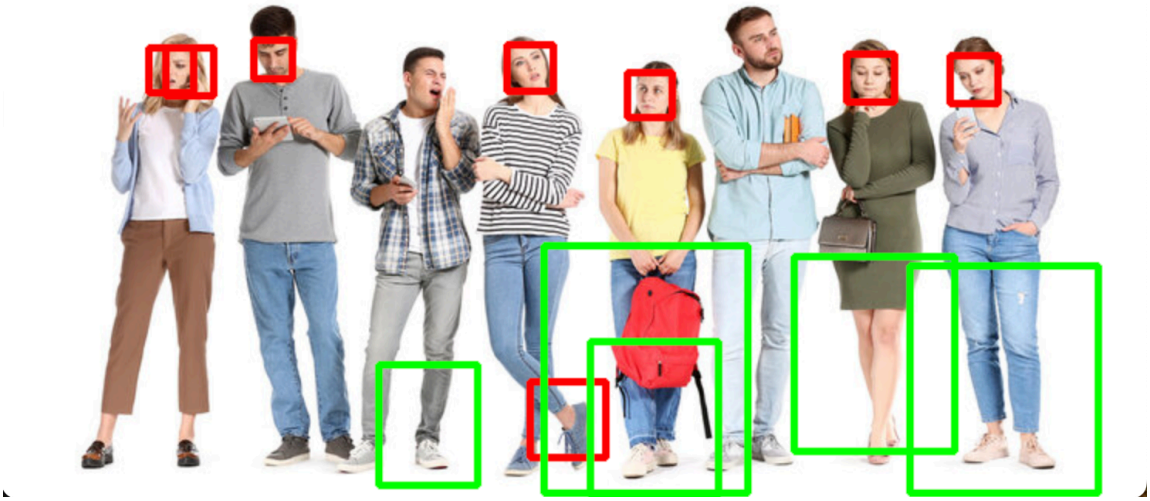
9. Displaying the image:

```
[10]: cv2.imshow("Gray", img)
```

10. Waiting for a key press and closing the window:

```
[11]: cv2.waitKey(0)
      cv2.destroyAllWindows()
```

Output:



Result Analysis: The experiment successfully detected and highlighted faces and lower bodies in the image using Haar cascade classifiers. Red rectangles marked faces, while green rectangles indicated lower bodies. The process demonstrated effective feature detection, though accuracy depends on the quality of the classifiers and the clarity of the image.