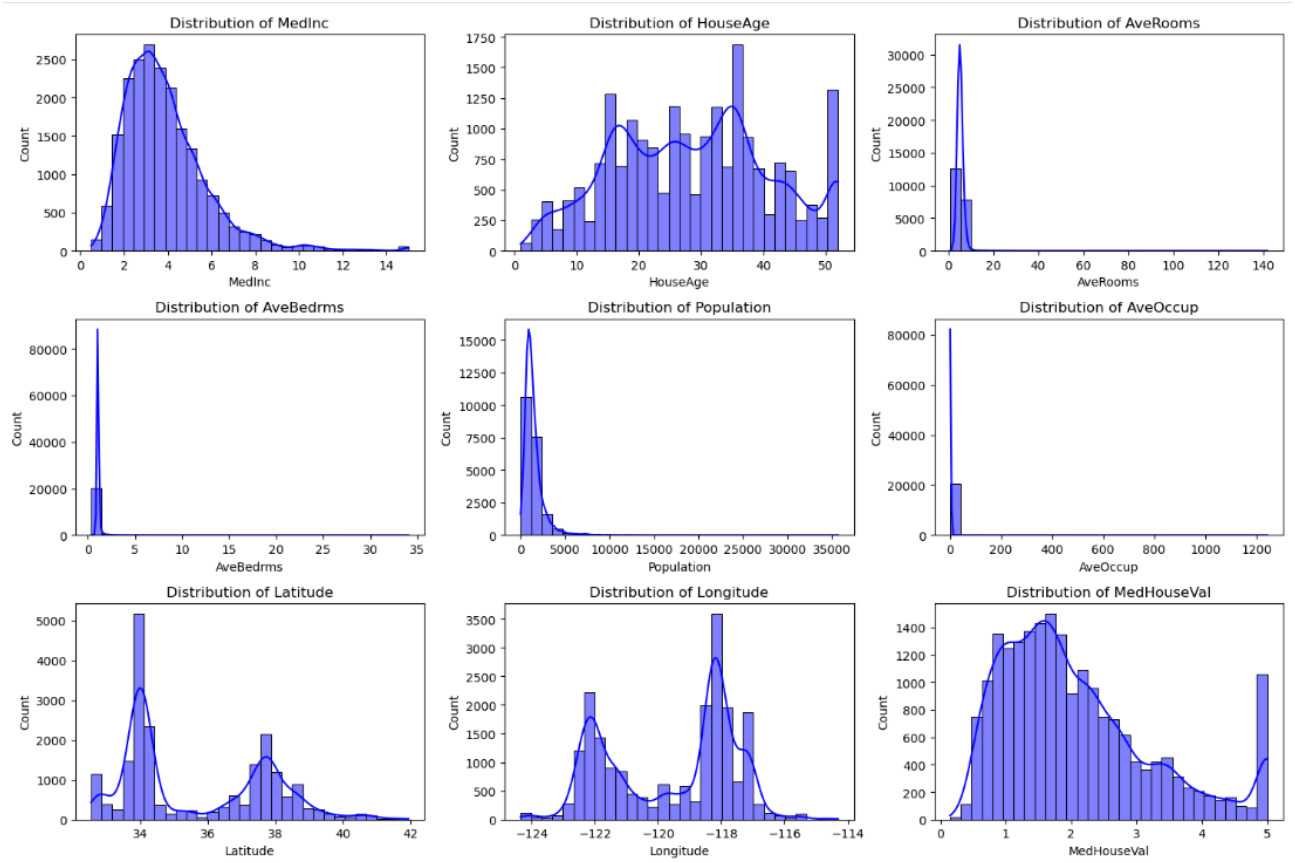
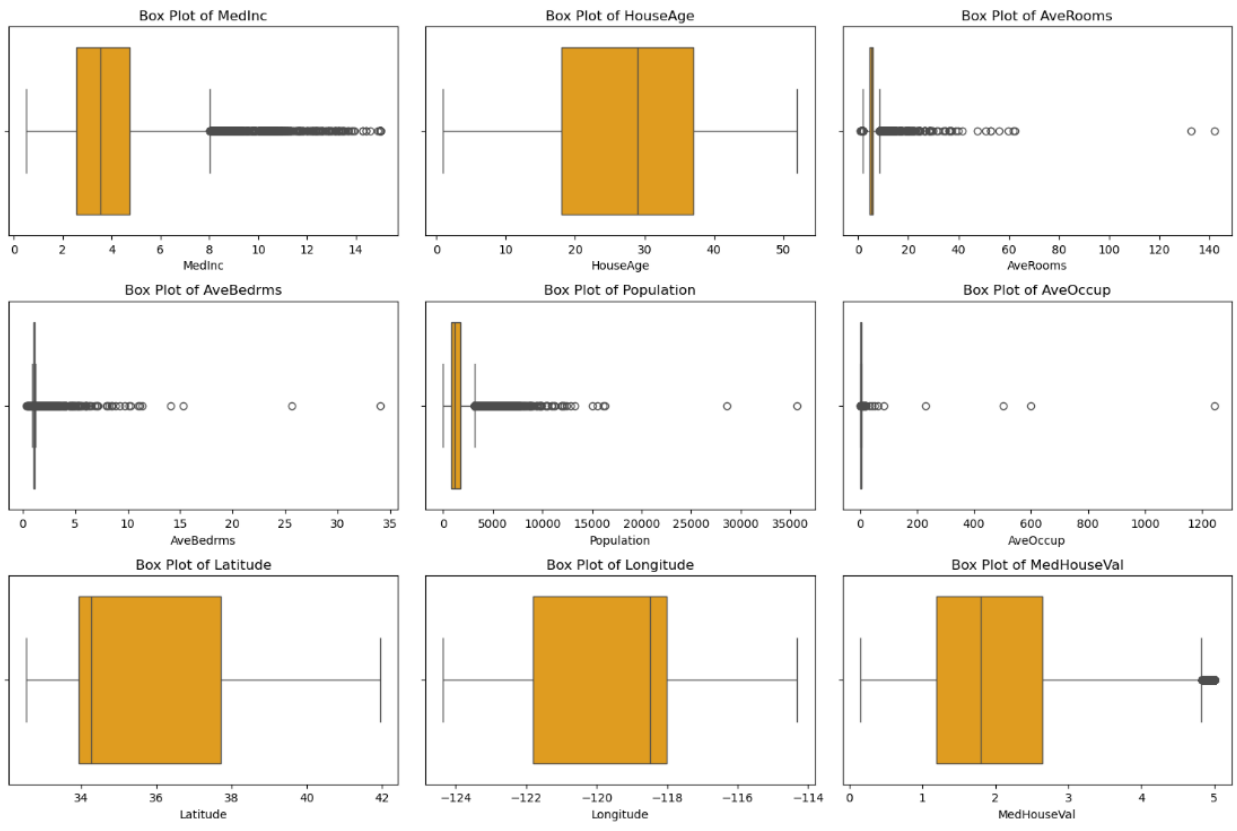
6. PROCEDURE / PROGRAMME:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing

# Step 1: Load the California Housing dataset
data = fetch_california_housing(as_frame=True)
housing_df = data.frame

# Step 2: Create histograms for numerical features
numerical_features =
housing_df.select_dtypes(include=[np.number]).columns

# Plot histograms
plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features):
    plt.subplot(3, 3, i + 1)
    sns.histplot(housing_df[feature], kde=True, bins=30,
color='blue')
    plt.title(f'Distribution of {feature}')
plt.tight_layout()
plt.show()

# Step 3: Generate box plots for numerical features
plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features):
    plt.subplot(3, 3, i + 1)
    sns.boxplot(x=housing_df[feature], color='orange')
    plt.title(f'Box Plot of {feature}')
plt.tight_layout()
plt.show()
```

```
# Step 4: Identify outliers using the IQR method
print("Outliers Detection:")
outliers_summary = {}
for feature in numerical_features:
    Q1 = housing_df[feature].quantile(0.25)
    Q3 = housing_df[feature].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = housing_df[(housing_df[feature] < lower_bound) |
(housing_df[feature] > upper_bound)]
    outliers_summary[feature] = len(outliers)
    print(f"{feature}: {len(outliers)} outliers")

# Optional: Print a summary of the dataset
print("\nDataset Summary:")
print(housing_df.describe())
```

OUTPUT:


Distribution of MedInc
Count
MedInc
Distribution of HouseAge
Count
HouseAge
Distribution of AveRooms
Count
AveRooms
Distribution of AveBedrms
Count
AveBedrms
Distribution of Population
Count
Population
Distribution of AveOccup
Count
AveOccup
Distribution of Latitude
Count
Latitude
Distribution of Longitude
Count
Longitude
Distribution of MedHouseVal
Count
MedHouseVal


Box Plot of MedInc
MedInc
Box Plot of HouseAge
HouseAge
Box Plot of AveRooms
AveRooms
Box Plot of AveBedrms
AveBedrms
Box Plot of Population
Population
Box Plot of AveOccup
AveOccup
Box Plot of Latitude
Latitude
Box Plot of Longitude
Longitude
Box Plot of MedHouseVal
MedHouseVal

```
Outliers Detection:
MedInc: 681 outliers
HouseAge: 0 outliers
AveRooms: 511 outliers
AveBedrms: 1424 outliers
Population: 1196 outliers
AveOccup: 711 outliers
Latitude: 0 outliers
Longitude: 0 outliers
MedHouseVal: 1071 outliers

Dataset Summary:
              MedInc      HouseAge      AveRooms     AveBedrms    Population  \
count  20640.000000  20640.000000  20640.000000  20640.000000  20640.000000
mean       3.870671     28.639486      5.429000      1.096675   1425.476744
std        1.899822     12.585558      2.474173      0.473911   1132.462122
min        0.499900      1.000000      0.846154      0.333333      3.000000
25%        2.563400     18.000000      4.440716      1.006079    787.000000
50%        3.534800     29.000000      5.229129      1.048780   1166.000000
75%        4.743250     37.000000      6.052381      1.099526   1725.000000
max       15.000100     52.000000    141.909091     34.066667  35682.000000

            AveOccup      Latitude     Longitude   MedHouseVal
count  20640.000000  20640.000000  20640.000000  20640.000000
mean       3.070655     35.631861   -119.569704      2.068558
std       10.386050      2.135952      2.003532      1.153956
min        0.692308     32.540000   -124.350000      0.149990
25%        2.429741     33.930000   -121.800000      1.196000
50%        2.818116     34.260000   -118.490000      1.797000
75%        3.282261     37.710000   -118.010000      2.647250
max     1243.333333     41.950000   -114.310000      5.000010
```

1. EXPERIMENT NO: 2

2. TITLE: correlation matrix to understand the relationships between pairs of features

3. LEARNING OBJECTIVES:

4. AIM: Develop a program to compute the correlation matrix to understand the relationships between pairs of features. Visualize the correlation matrix using a heatmap to know which variables have strong positive/negative correlations. Create a pair plot to visualize pairwise relationships between features. Use California Housing dataset.

5. THEORY

California Housing dataset

**Data Set Characteristics: **

Number of Instances: 20640

Number of Attributes: 8 numeric, predictive attributes and the target

1. Longitude: A measure of how far west a house is; a higher value is farther west
2. Latitude: A measure of how far north a house is; a higher value is farther north
3. Housing Median Age: Median age of a house within a block; a lower number is a newer building
4. Total Rooms: Total number of rooms within a block
5. Total Bedrooms: Total number of bedrooms within a block
6. Population: Total number of people residing within a block
7. Households: Total number of households, a group of people residing within a home unit, for a block
8. Median Income: Median income for households within a block of houses (measured in tens of thousands of US Dollars)
9. Median House Value: Median house value for households within a block (measured in US Dollars)
10. Ocean Proximity: Location of the house w.r.t ocean/sea

The target variable is the median house value for California districts, expressed in hundreds of thousands of dollars ($100,000).

This dataset was derived from the 1990 U.S. census, using one row per census block group. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people).

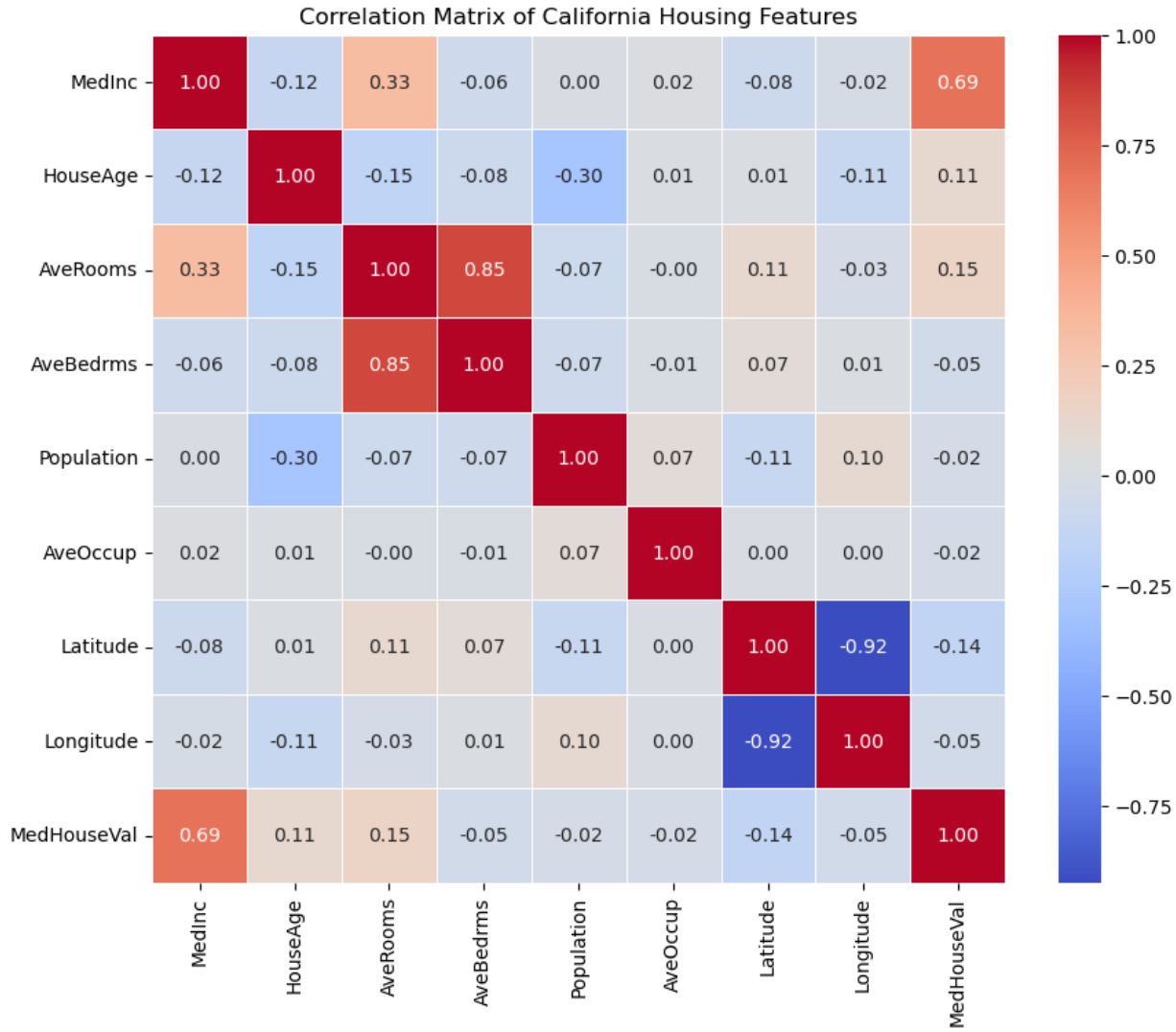
A household is a group of people residing within a home. Since the average number of rooms and bedrooms in this dataset are provided per household, these columns may take surprisingly large values for block groups with few households and many empty houses, such as vacation resorts.

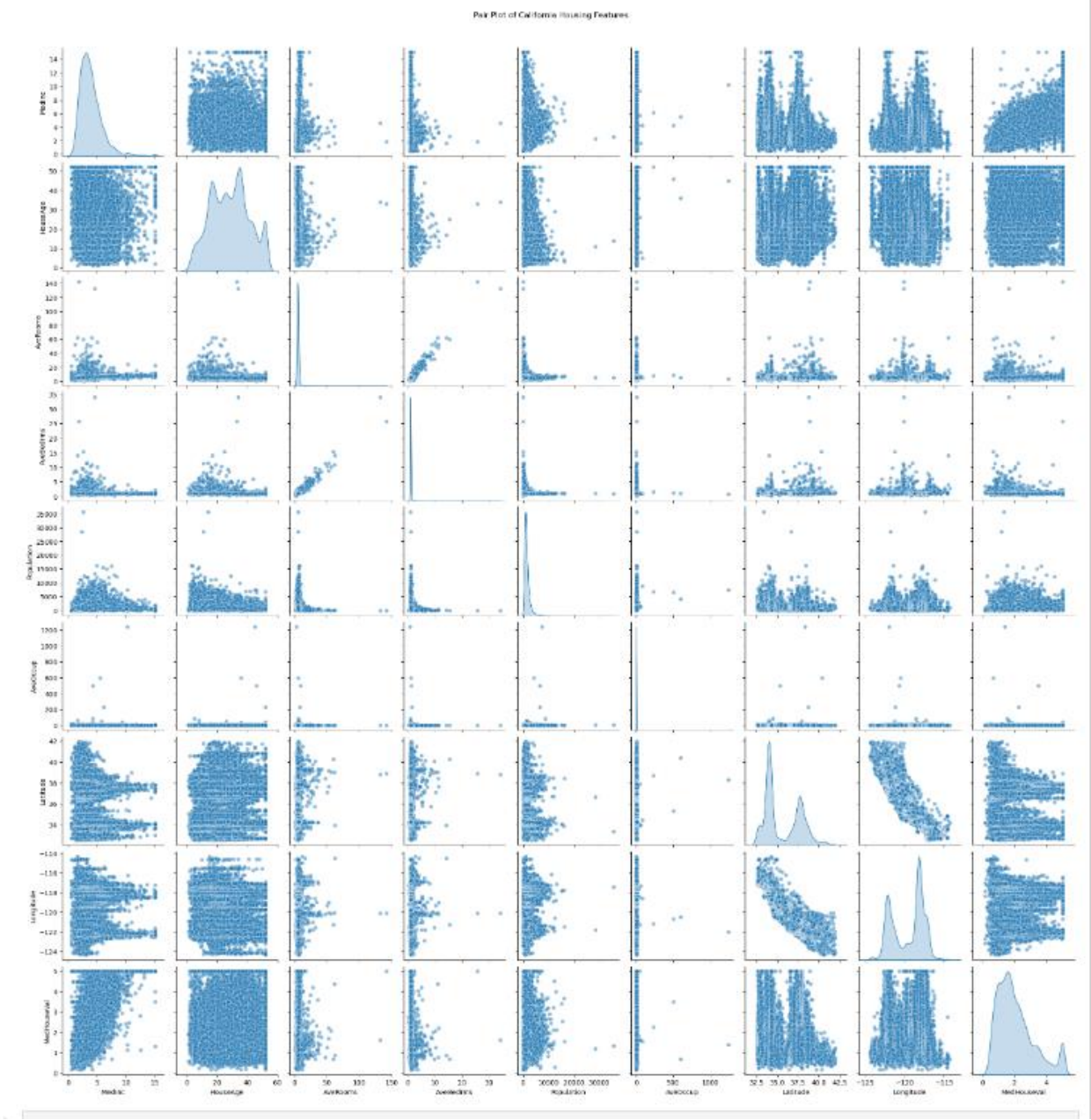
It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

6. PROCEDURE / PROGRAMME:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing

# Step 1: Load the California Housing Dataset
california_data = fetch_california_housing(as_frame=True)
data = california_data.frame

# Step 2: Compute the correlation matrix
correlation_matrix = data.corr()

# Step 3: Visualize the correlation matrix using a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix of California Housing Features')
plt.show()

# Step 4: Create a pair plot to visualize pairwise relationships
sns.pairplot(data, diag_kind='kde', plot_kws={'alpha': 0.5})
plt.suptitle('Pair Plot of California Housing Features', y=1.02)
plt.show()
```

OUTPUT:


Correlation Matrix of California Housing Features
MedInc
HouseAge
AveRooms
AveBedrms
Population
AveOccup
Latitude
Longitude
MedHouseVal
1.00 -0.12 0.33 -0.06 0.00 0.02 -0.08 -0.02 0.69
-0.12 1.00 -0.15 -0.08 -0.30 0.01 0.01 -0.11 0.11
0.33 -0.15 1.00 0.85 -0.07 -0.00 0.11 -0.03 0.15
-0.06 -0.08 0.85 1.00 -0.07 -0.01 0.07 0.01 -0.05
0.00 -0.30 -0.07 -0.07 1.00 0.07 -0.11 0.10 -0.02
0.02 0.01 -0.00 -0.01 0.07 1.00 0.00 0.00 -0.02
-0.08 0.01 0.11 0.07 -0.11 0.00 1.00 -0.92 -0.14
-0.02 -0.11 -0.03 0.01 0.10 0.00 -0.92 1.00 -0.05
0.69 0.11 0.15 -0.05 -0.02 -0.02 -0.14 -0.05 1.00
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75


Pair Plot of California Housing Features
MedInc
HouseAge
AveRooms
AveBedrms
Population
AveOccup
Latitude
Longitude
MedHouseVal

1. EXPERIMENT NO: 3

2. TITLE: Reduce Data Dimensionality using PCA – Python

3. LEARNING OBJECTIVES:

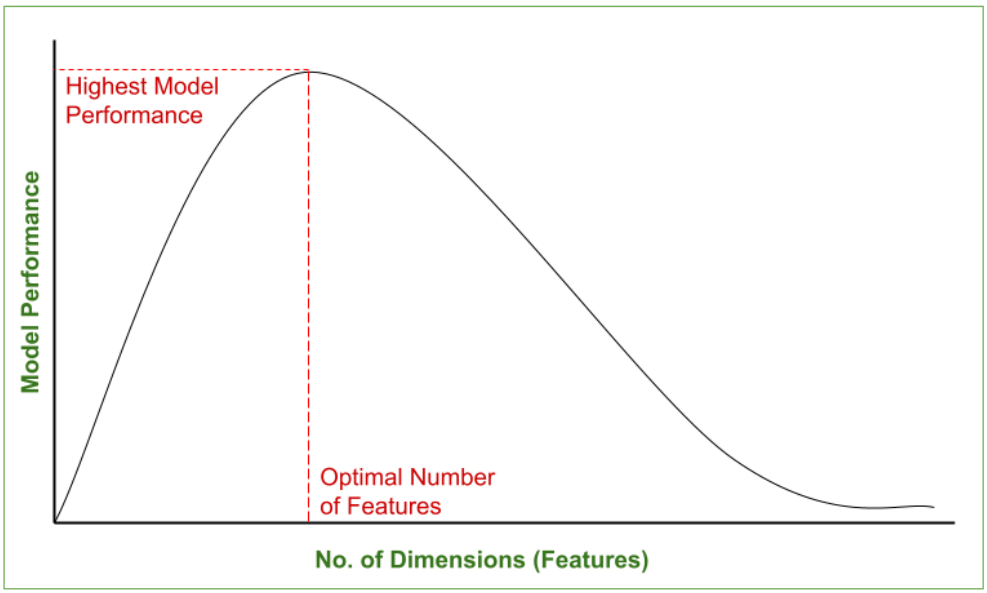
4. AIM: Develop a program to implement Principal Component Analysis (PCA) for reducing the dimensionality of the Iris dataset from 4 features to 2.

5. THEORY

Reduce Data Dimensionality using PCA – Python

The advancements in Data Science and Machine Learning have made it possible for us to solve several complex regression and classification problems. However, the performance of all these ML models depends on the data fed to them. Thus, it is imperative that we provide our ML models with an optimal dataset. Now, one might think that the more data we provide to our model, the better it becomes – however, it is not the case. If we feed our model with an excessively large dataset (with a large no. of features/columns), it gives rise to the problem of overfitting, wherein the model starts getting influenced by outlier values and noise. This is called the Curse of Dimensionality.

The following graph represents the change in model performance with the increase in the number of dimensions of the dataset. It can be observed that the model performance is best only at an option dimension, beyond which it starts decreasing.



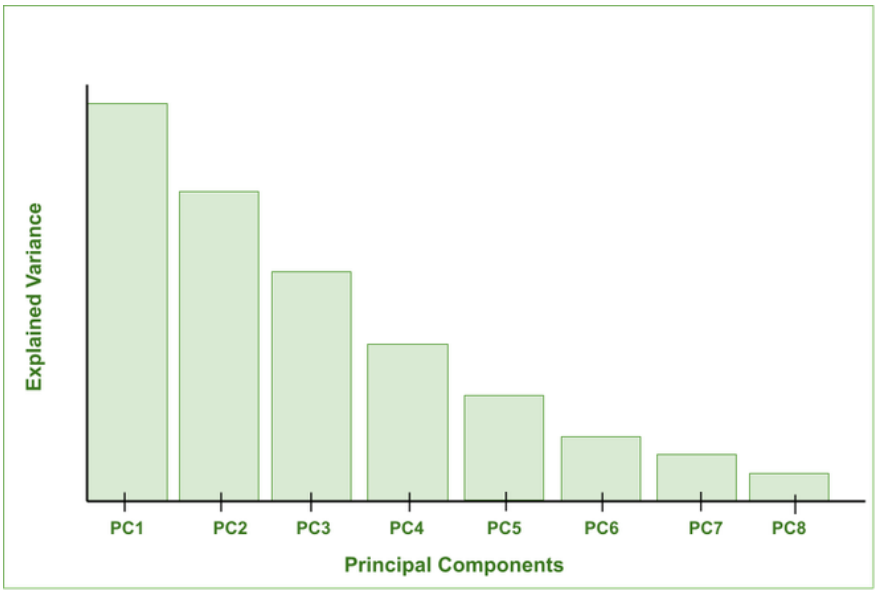
Dimensionality Reduction is a statistical/ML-based technique wherein we try to reduce the number of features in our dataset and obtain a dataset with an optimal number of dimensions.

One of the most common ways to accomplish Dimensionality Reduction is Feature Extraction, wherein we reduce the number of dimensions by mapping a higher dimensional feature space to a lower-dimensional feature space. The most popular technique of Feature Extraction is Principal Component Analysis (PCA)

Principal Component Analysis (PCA)

As stated earlier, Principal Component Analysis is a technique of feature extraction that maps a higher dimensional feature space to a lower-dimensional feature space. While reducing the number of dimensions, PCA ensures that maximum information of the original dataset is retained in the dataset with the reduced no. of dimensions and the co-relation between the newly obtained Principal Components is minimum. The new features obtained after applying PCA are called Principal Components and are denoted as PCi (i=1,2,3…n). Here, (Principal Component-1) PC1 captures the maximum information of the original dataset, followed by PC2, then PC3 and so on.

The following bar graph depicts the amount of Explained Variance captured by various Principal Components. (The Explained Variance defines the amount of information captured by the Principal Components).

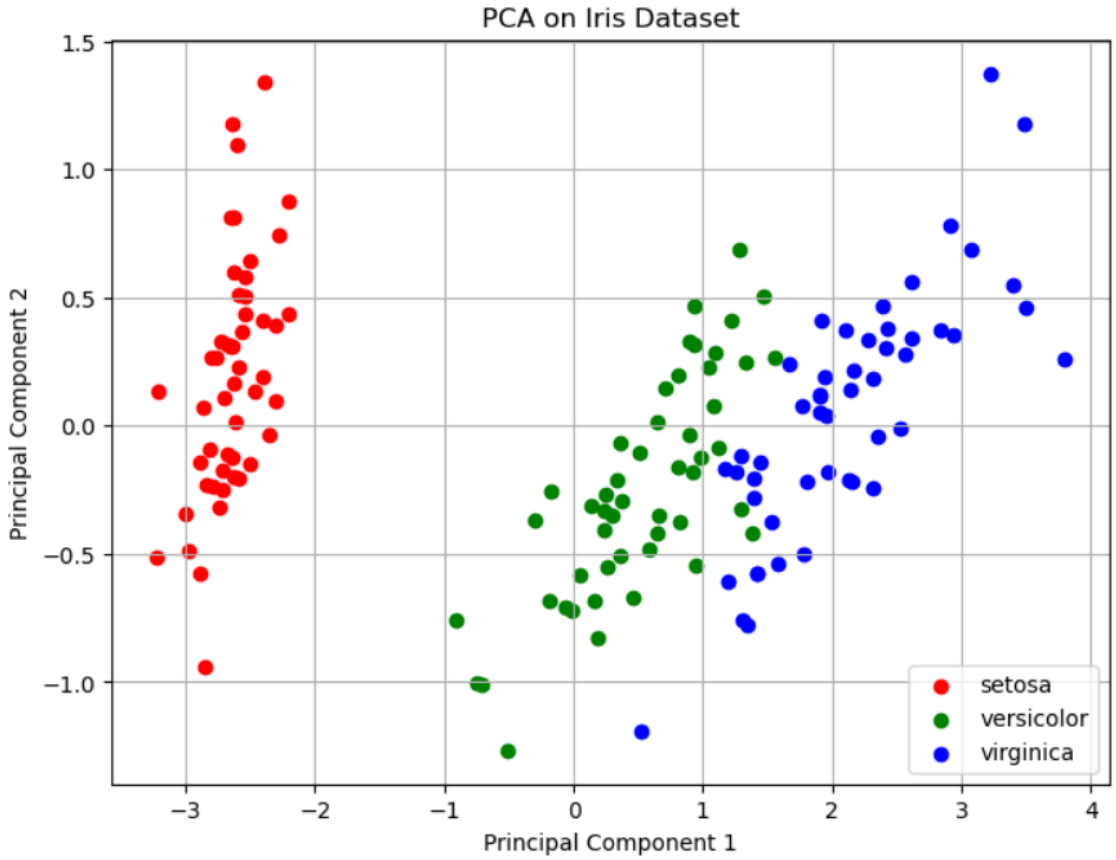


6. PROCEDURE / PROGRAMME :

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = load_iris()
data = iris.data
labels = iris.target
label_names = iris.target_names
```

```
# Convert to a DataFrame for better visualization
iris_df = pd.DataFrame(data, columns=iris.feature_names)


# Perform PCA to reduce dimensionality to 2
pca = PCA(n_components=2)
data_reduced = pca.fit_transform(data)


# Create a DataFrame for the reduced data
reduced_df = pd.DataFrame(data_reduced, columns=['Principal
Component 1', 'Principal Component 2'])
reduced_df['Label'] = labels


# Plot the reduced data
plt.figure(figsize=(8, 6))
colors = ['r', 'g', 'b']
for i, label in enumerate(np.unique(labels)):
    plt.scatter(
        reduced_df[reduced_df['Label'] == label]['Principal
Component 1'],
        reduced_df[reduced_df['Label'] == label]['Principal
Component 2'],
        label=label_names[label],
        color=colors[i]
    )

plt.title('PCA on Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.grid()
plt.show()
```

OUTPUT:


PCA on Iris Dataset
Principal Component 2
Principal Component 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
3
4
setosa
versicolor
virginica

1. EXPERIMENT NO: 4

2. TITLE: **FIND-S ALGORITHM**

3. LEARNING OBJECTIVES:
   a. Make use of Data sets in implementing the machine learning algorithms.
   b. Implement ML concepts and algorithms in Python

4. AIM: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.

5. THEORY:

- The concept learning approach in machine learning, can be formulated as "Problem of searching through a predefined space of potential hypotheses for the hypothesis that best fits the training examples".
- Find-S algorithm for concept learning is one of the most basic algorithms of machine learning.

Find-S Algorithm

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
   For each attribute constraint a i in h :
   If the constraint a i in h is satisfied by x then do nothing
   Else replace a i in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

- It is Guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples.
- Also Notice that negative examples are ignored.

Limitations of the Find-S algorithm:

- No way to determine if the only final hypothesis (found by Find-S) is consistent with data or there are more hypothesis that is consistent with data.
- Inconsistent sets of training data can mislead the finds algorithm as it ignores negative data samples.
- A good concept learning algorithm should be able to backtrack the choice of hypothesis found so that the resulting hypothesis can be improved over time. Unfortunately, Find-S provide no such method.

DATA SETS

sky	airTemp	humidity	wind	water	forecast	enjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

6. PROCEDURE / PROGRAMME :

```
import numpy as np
import pandas as pd
data=pd.read_csv('finds.csv')
print('Data',data)
def train(concepts,target):
        specific_h=concepts[0]
        print('\nspecific1\n',specific_h)
        for i,h in enumerate(concepts):
            print('i',i)
            print('h',h)
            if target[i]=="Yes":
                for x in range(len(specific_h)):
                    print('x',x)
                    print('specific',specific_h)
                    if h[x]==specific_h[x]:
                        pass
                    else:
                        specific_h[x]="?"
        return specific_h
concepts=np.array(data.iloc[:,0:-1])
target=np.array(data.iloc[:,-1])
print('\nConcept\n',concepts)
print('Target',target)
print(train(concepts,target))
```

OUTPUT:

```
Data      sky airTemp humidity    wind water forecast enjoySport
0  Sunny    Warm   Normal  Strong  Warm     Same        Yes
1  Sunny    Warm     High  Strong  Warm     Same        Yes
2  Rainy    Cold     High  Strong  Warm   Change         No
3  Sunny    Warm     High  Strong  Cool   Change        Yes
Concept
 [['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
```

```
Target ['Yes' 'Yes' 'No' 'Yes']

specific1
 ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
i 0
h ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
x 0
specific ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
x 1
specific ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
x 2
specific ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
x 3
specific ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
x 4
specific ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
x 5
specific ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
i 1
h ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
x 0
specific ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
x 1
specific ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
x 2
specific ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
x 3
specific ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
x 4
specific ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
x 5
specific ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
i 2
h ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
i 3
h ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']
x 0
specific ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
x 1
specific ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
x 2
specific ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
x 3
specific ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
x 4
specific ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
x 5
specific ['Sunny' 'Warm' '?' 'Strong' '?' 'Same']
['Sunny' 'Warm' '?' 'Strong' '?' '?']
```

1. EXPERIMENT NO: 5

2. TITLE: **k-Nearest Neighbour algorithm**

3. LEARNING OBJECTIVES:

4. AIM: Develop a program to implement k-Nearest Neighbour algorithm to classify the randomly generated 100 values of x in the range of [0,1]. Perform the following based on dataset generated.
    1. Label the first 50 points {x1,……,x50} as follows: if (xi ≤ 0.5), then xi ε Class1, else xi ε Class1
    2. Classify the remaining points, x51,……,x100 using KNN. Perform this for k=1,2,3,4,5,20,30

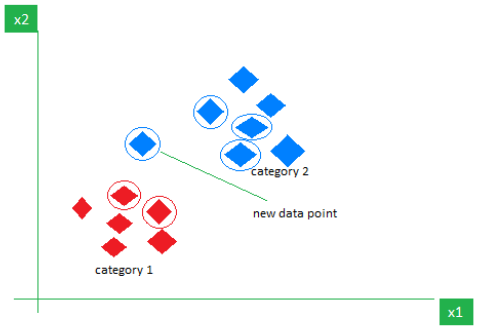
5. THEORY: K-Nearest Neighbor(KNN) Algorithm

K-Nearest Neighbors (KNN) is a simple way to classify things by looking at what's nearby. Imagine a streaming service wants to predict if a new user is likely to cancel their subscription (churn) based on their age. They checks the ages of its existing users and whether they churned or stayed. If most of the "K" closest users in age of new user canceled their subscription KNN will predict the new user might churn too. The key idea is that users with similar ages tend to have similar behaviors and KNN uses this closeness to make decisions.

Getting Started with K-Nearest Neighbors

K-Nearest Neighbors is also called as a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification it performs an action on the dataset.

As an example, consider the following table of data points containing two features:



The new point is classified as Category 2 because most of its closest neighbors are blue squares. KNN assigns the category based on the majority of nearby points.

The image shows how KNN predicts the category of a new data point based on its closest neighbours.

1. The red diamonds represent Category 1 and the blue squares represent Category 2.
2. The new data point checks its closest neighbours (circled points).
3. Since the majority of its closest neighbours are blue squares (Category 2) KNN predicts the new data point belongs to Category 2.

KNN works by using proximity and majority voting to make predictions.

What is 'K' in K Nearest Neighbour ?

In the k-Nearest Neighbours (k-NN) algorithm k is just a number that tells the algorithm how many nearby points (neighbours) to look at when it makes a decision.

Example:

Imagine you're deciding which fruit it is based on its shape and size. You compare it to fruits you already know.

- If k = 3, the algorithm looks at the 3 closest fruits to the new one.
- If 2 of those 3 fruits are apples and 1 is a banana, the algorithm says the new fruit is an apple because most of its neighbours are apples.

How to choose the value of k for KNN Algorithm?

The value of k is critical in KNN as it determines the number of neighbors to consider when making predictions. Selecting the optimal value of k depends on the characteristics of the input data. If the dataset has significant outliers or noise a higher k can help smooth out the predictions and reduce the influence of noisy data. However choosing very high value can lead to underfitting where the model becomes too simplistic.

Statistical Methods for Selecting k:

- Cross-Validation: A robust method for selecting the best k is to perform k-fold cross-validation. This involves splitting the data into k subsets training the model on some subsets and testing it on the remaining ones and repeating this for each subset. The value of k that results in the highest average validation accuracy is usually the best choice.
- Elbow Method: In the elbow method we plot the model's error rate or accuracy for different values of k. As we increase k the error usually decreases initially. However after a certain point the error rate starts to decrease more slowly. This point where the curve forms an "elbow" that point is considered as best k.
- Odd Values for k: It's also recommended to choose an odd value for k especially in classification tasks to avoid ties when deciding the majority class.

Distance Metrics Used in KNN Algorithm

KNN uses distance metrics to identify nearest neighbour, these neighbours are used for classification and regression task.

To identify nearest neighbour we use below distance metrics:

1. **Euclidean Distance**
   Euclidean distance is defined as the straight-line distance between two points in a plane or space. You can think of it like the shortest path you would walk if you were to go directly from one point to another.

   distance(x,Xi)=∑j=1d(xj–Xij)2]

2. **Manhattan Distance**
   This is the total distance you would travel if you could only move along horizontal and vertical lines (like a grid or city streets). It's also called "taxicab distance" because a taxi can only drive along the grid-like streets of a city.

   $d(x,y)=\sum i=1n|xi-yi|$

3. **Minkowski Distance**
   Minkowski distance is like a family of distances, which includes both Euclidean and Manhattan distances as special cases.

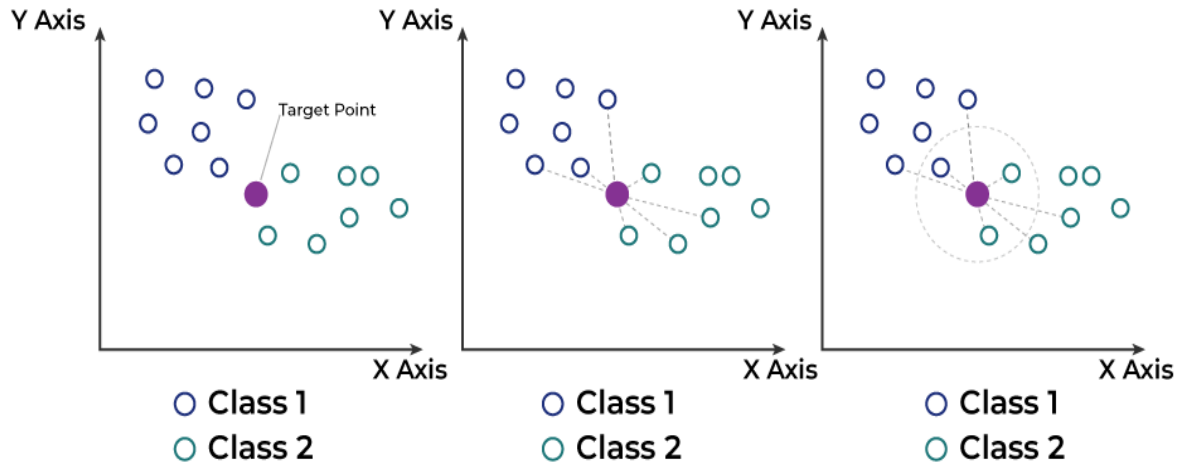
   $d(x,y)=(\sum i=1n(xi-yi)p)p1$

From the formula above we can say that when p = 2 then it is the same as the formula for the Euclidean distance and when p = 1 then we obtain the formula for the Manhattan distance.

So, you can think of Minkowski as a flexible distance formula that can look like either Manhattan or Euclidean distance depending on the value of p

Working of KNN algorithm

The K-Nearest Neighbors (KNN) algorithm operates on the principle of similarity where it predicts the label or value of a new data point by considering the labels or values of its K nearest neighbors in the training dataset.



Step-by-Step explanation of how KNN works is discussed below:

Step 1: Selecting the optimal value of K
K represents the number of nearest neighbors that needs to be considered while making prediction.

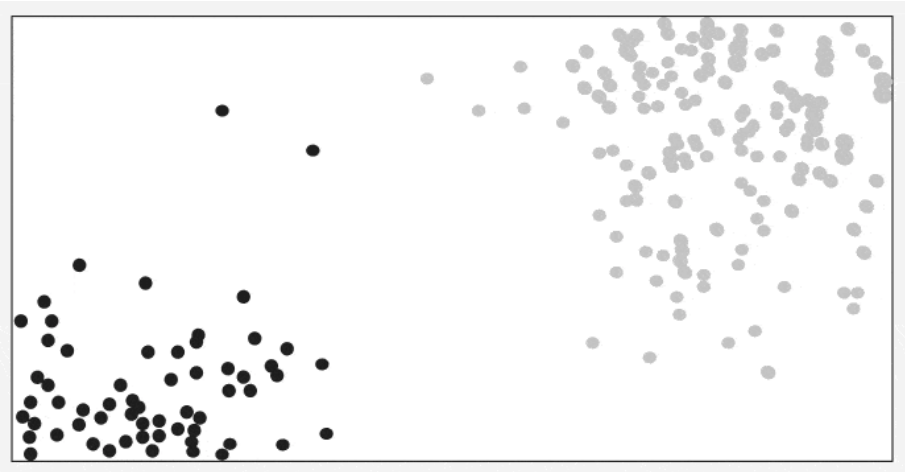
Step 2: Calculating distance
To measure the similarity between target and training data points Euclidean distance is used. Distance is calculated between data points in the dataset and target point.

Step 3: Finding Nearest Neighbors
The k data points with the smallest distances to the target point are nearest neighbors.

Step 4: Voting for Classification or Taking Average for Regression

When you want to classify a data point into a category (like spam or not spam), the K-NN algorithm looks at the K closest points in the dataset. These closest points are called neighbors. The algorithm then looks at which category the neighbors belong to and picks the one that appears the most. This is called majority voting.

In regression, the algorithm still looks for the K closest points. But instead of voting for a class in classification, it takes the average of the values of those K neighbors. This average is the predicted value for the new point for the algorithm.



It shows how a test point is classified based on its nearest neighbors. As the test point moves the algorithm identifies the closest 'k' data points i.e 5 in this case and assigns test point the majority class label that is grey label class here.

6. PROCEDURE / PROGRAMME :

```
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter

data = np.random.rand(100)

labels = ["Class1" if x <= 0.5 else "Class2" for x in data[:50]]



def euclidean_distance(x1, x2):
    return abs(x1 - x2)

def knn_classifier(train_data, train_labels, test_point, k):
```

```
    distances = [(euclidean_distance(test_point, train_data[i]),
train_labels[i]) for i in range(len(train_data))]

    distances.sort(key=lambda x: x[0])
    k_nearest_neighbors = distances[:k]

    k_nearest_labels = [label for _, label in k_nearest_neighbors]

    return Counter(k_nearest_labels).most_common(1)[0][0]

train_data = data[:50]
train_labels = labels

test_data = data[50:]

k_values = [1, 2, 3, 4, 5, 20, 30]
print("--- k-Nearest Neighbors Classification ---")
print("Training dataset: First 50 points labeled based on the rule
(x <= 0.5 -> Class1, x > 0.5 -> Class2)")
print("Testing dataset: Remaining 50 points to be classified\n")

results = {}

for k in k_values:
    print(f"Results for k = {k}:")
    classified_labels = [knn_classifier(train_data, train_labels,
test_point, k) for test_point in test_data]
    results[k] = classified_labels

    for i, label in enumerate(classified_labels, start=51):
        print(f"Point x{i} (value: {test_data[i - 51]:.4f}) is
classified as {label}")
    print("\n")

print("Classification complete.\n")
```

```
for k in k_values:
    classified_labels = results[k]
    class1_points = [test_data[i] for i in range(len(test_data)) if
classified_labels[i] == "Class1"]
    class2_points = [test_data[i] for i in range(len(test_data)) if
classified_labels[i] == "Class2"]

    plt.figure(figsize=(10, 6))
    plt.scatter(train_data, [0] * len(train_data), c=["blue" if
label == "Class1" else "red" for label in train_labels],
                label="Training Data", marker="o")
    plt.scatter(class1_points, [1] * len(class1_points), c="blue",
label="Class1 (Test)", marker="x")
    plt.scatter(class2_points, [1] * len(class2_points), c="red",
label="Class2 (Test)", marker="x")

    plt.title(f"k-NN Classification Results for k = {k}")
    plt.xlabel("Data Points")
    plt.ylabel("Classification Level")
    plt.legend()
    plt.grid(True)
    plt.show()
```

OUTPUT:

```
--- k-Nearest Neighbors Classification ---
Training dataset: First 50 points labeled based on the rule (x <=
0.5 -> Class1, x > 0.5 -> Class2)
Testing dataset: Remaining 50 points to be classified

Results for k = 1:
Point x51 (value: 0.4186) is classified as Class1
Point x52 (value: 0.1913) is classified as Class1
Point x53 (value: 0.9719) is classified as Class2
Point x54 (value: 0.6504) is classified as Class2
Point x55 (value: 0.2149) is classified as Class1
```