INDEX

Expt No	Date	Name of the experiment	Page No	Performance		Viva Voce	Total Marks	Signature of the faculty
				Lab ( )	Record ( )			
1a		Implement data crawlers Beautiful Soup, Ixml and scrapy						
1b		Implement data analyzing for iris data set using SciPy						
1c		Implement graph generator using NetworkX for covid data set						
1d		Implement 3D visualization for iris dataset using matplotlib						
1e		Implement Resource Description Frame work for smart store data by DBpedia						
1f		Create two-dimensional random dataset and implement Data Frames in python with various slicing methods						
1g		Create two-dimensional random dataset and implement Data Frames in python with various slicing methods						
2a		Implement the python code to compute the population proportions for the given problem.						
2b		Wine Quality Prediction						
2c		Digit Recognition using CNN in Python						
3		House Price Prediction						
4		Movie Recommendation Engine						
5		Market Segmentation using K-means Clustering on India Census Data						

Ex No: 1a Date:	**Implement data crawlers Beautiful Soup, Ixml and scrapy**

AIM:

To Implement data crawlers Beautiful Soup, Ixml and scrapy.

ALGORITHM:

- **Step 1:** Identify target websites and the data to be extracted.
- **Step 2:** Choose the appropriate library (Beautiful Soup, lxml, or Scrapy) based on the complexity of the website structure and your familiarity with the library.
- **Step 3:** Implement code to navigate through the website's HTML/XML structure and extract desired data using library-specific methods.
- **Step 4:** Clean and process the extracted data as needed (e.g., removing HTML tags, converting to desired data types).
- **Step 5:** Store the extracted data in a structured format (e.g., CSV, JSON, database) for further analysis or use.

PROGRAM:

```
import requests
from bs4 import BeautifulSoup

# Define the URL to crawl
url = 'https://www.linkedin.com'

# Send a GET request to the URL
response = requests.get(url)

# Parse the HTML content using BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')

# Extract information from the parsed HTML
# For example, find all the links on the page
links = soup.find_all('a')

# Print the extracted links
for link in links:
    print(link.get('href'))
```

OUTPUT:

#main-content
/legal/user-agreement?trk=linkedin-tc_auth-button_user-agreement
/legal/privacy-policy?trk=linkedin-tc_auth-button_privacy-policy
/legal/cookie-policy?trk=linkedin-tc_auth-button_cookie-policy
/?trk=guest_homepage-basic_nav-header-logo
https://www.linkedin.com/pulse/topics/home/?trk=guest_homepage-basic_guest_nav_menu_articles
https://www.linkedin.com/pub/dir/+/+?trk=guest_homepage-basic_guest_nav_menu_people
https://www.linkedin.com/learning/search?trk=guest_homepage-basic_guest_nav_menu_learning
https://www.linkedin.com/jobs/search?trk=guest_homepage-basic_guest_nav_menu_jobs
https://www.linkedin.com/games?trk=guest_homepage-basic_guest_nav_menu_games
https://www.linkedin.com/signup?trk=guest_homepage-basic_nav-header-join
https://www.linkedin.com/login?fromSignIn=true&trk=guest_homepage-basic_nav-header-signin
https://www.linkedin.com/login
/legal/user-agreement?trk=linkedin-tc_auth-button_user-agreement
/legal/privacy-policy?trk=linkedin-tc_auth-button_privacy-policy
/legal/cookie-policy?trk=linkedin-tc_auth-button_cookie-policy
https://www.linkedin.com/signup
https://www.linkedin.com/pulse/topics/marketing-s2461/
https://www.linkedin.com/pulse/topics/public-administration-s3697/
https://www.linkedin.com/pulse/topics/healthcare-s282/
https://www.linkedin.com/pulse/topics/engineering-s166/
https://www.linkedin.com/pulse/topics/it-services-s57547/
https://www.linkedin.com/pulse/topics/sustainability-s932/
https://www.linkedin.com/pulse/topics/business-administration-s50111/
https://www.linkedin.com/pulse/topics/telecommunications-s314/
https://www.linkedin.com/pulse/topics/hr-management-s50359/
https://www.linkedin.com/pulse/topics/home/

Department of AIML		
Average Performance	25	
Average Record	15	
Average Viva	10	
Total	50	

RESULT:

The above program is executed successfully.

Ex No: 1b Date:	**Implement data analyzing for iris data set using SciPy**

AIM

Implement data analyzing for iris data set using SciPy

ALGORITHM:

- **Step 1:** Load the iris dataset into memory.
- **Step 2:** Utilize SciPy functions to calculate summary statistics (mean, median, standard deviation, etc.) for each feature.
- **Step 3:** Perform data visualization using Matplotlib or Seaborn to explore relationships between features.
- **Step 4:** Apply statistical tests (e.g., t-test, ANOVA) to compare feature distributions across different classes.
- **Step 5:** Interpret the results and draw conclusions about the dataset based on the analysis performed.

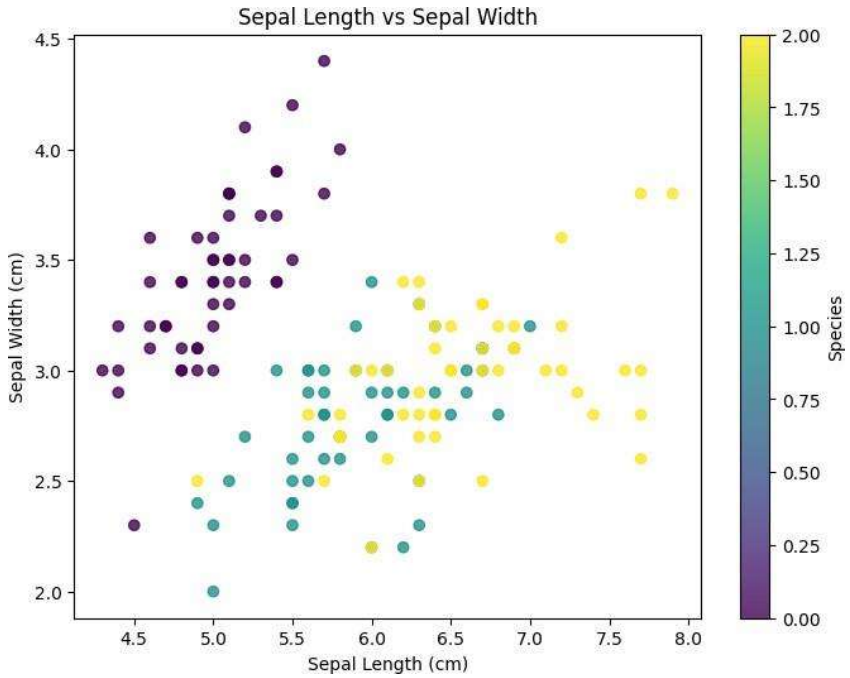
PROGRAM:

```
import numpy as np
import pandas as pd
from scipy.stats import describe
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()

# Convert the dataset to a pandas DataFrame
iris_df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                       columns= iris['feature_names'] + ['target'])

# Display the first few rows of the dataset
print(iris_df.head())

# Summary statistics
summary_stats = describe(iris_df.drop(columns='target'))
print(summary_stats)

# Visualization
# Scatter plot of sepal length vs sepal width
plt.figure(figsize=(8, 6))
plt.scatter(iris_df['sepal length (cm)'], iris_df['sepal width (cm)'],
```

```
        c=iris_df['target'], cmap='viridis', alpha=0.8)
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.title('Sepal Length vs Sepal Width')
plt.colorbar(label='Species')
plt.show()
```

OUTPUT:

Feature	Min	Max	Mean	Variance	Skewness	Kurtosis
Sepal Length (cm)	4.3	7.9	5.84	0.686	0.312	-0.574
Sepal Width (cm)	2.0	4.4	3.06	0.190	0.316	0.181
Petal Length (cm)	1.0	6.9	3.76	3.116	-0.272	-1.396
Petal Width (cm)	0.1	2.5	1.20	0.581	-0.102	-1.336



Department of AIML		
Average Performance	25	
Average Record	15	
Average Viva	10	
Total	50	

RESULT:

The above program is executed successfull

Ex No: 1c Date:	**Implement graph generator using NetworkX for covid data set**

AIM:

Implement graph generator using NetworkX for covid data set

ALGORITHM:

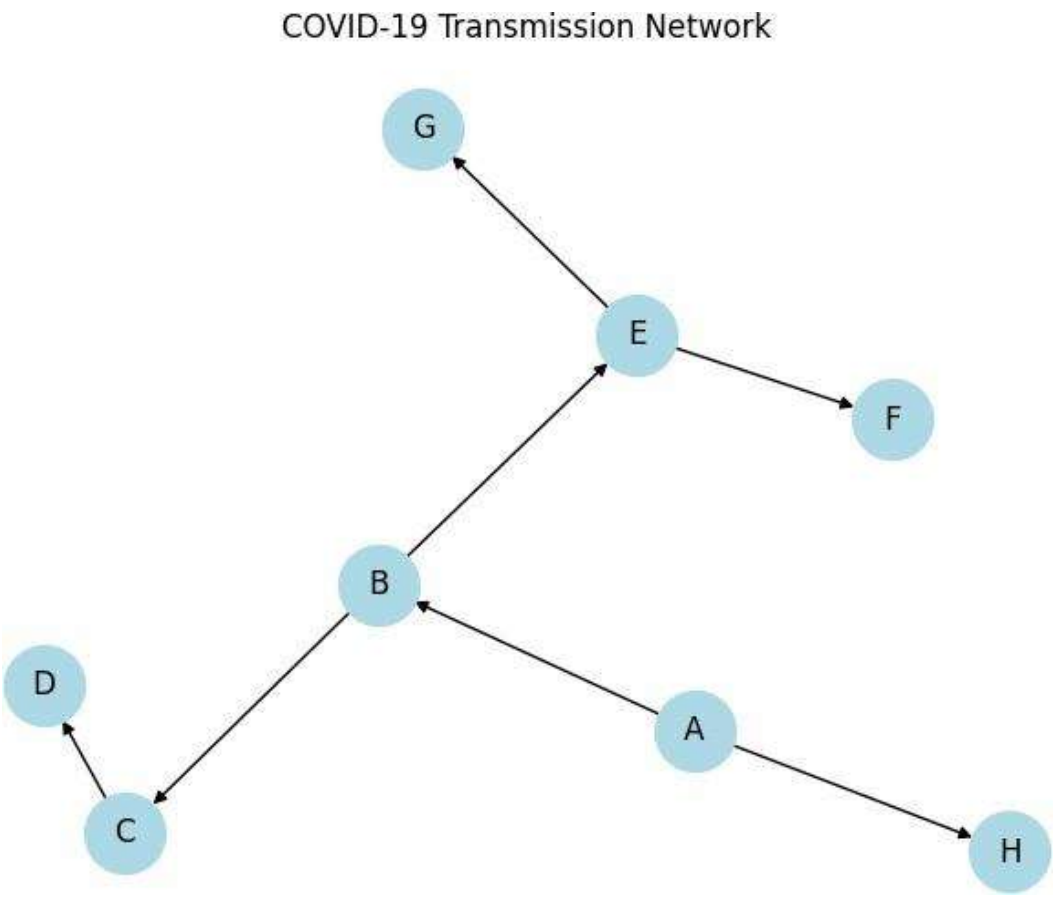
- **Step 1:** Obtain the COVID dataset containing relevant data points and relationships.
- **Step 2:** Create an empty graph object using NetworkX.
- **Step 3:** Add nodes to the graph representing entities (e.g., countries, regions) from the dataset.
- **Step 4:** Add edges to the graph representing relationships (e.g., transmission paths) between nodes based on the dataset.
- **Step 5:** Visualize the graph using NetworkX's built-in visualization tools or external libraries for further analysis and interpretation.

PROGRAM:

```
import networkx as nx
import matplotlib.pyplot as plt

# Assuming you have a dataset containing COVID-19 transmission information
# For demonstration, let's create a sample dataset
covid_data = [
    {'infected_by': 'A', 'infected': 'B'},
    {'infected_by': 'B', 'infected': 'C'},
    {'infected_by': 'C', 'infected': 'D'},
    {'infected_by': 'B', 'infected': 'E'},
    {'infected_by': 'E', 'infected': 'F'},
    {'infected_by': 'E', 'infected': 'G'},
    {'infected_by': 'A', 'infected': 'H'},
]

# Create a directed graph
G = nx.DiGraph()

# Add nodes and edges to the graph based on the COVID-19 transmission data
for record in covid_data:
    infected_by = record['infected_by']
    infected = record['infected']
    G.add_edge(infected_by, infected)

# Draw the graph
pos = nx.spring_layout(G)  # Position nodes using Fruchterman-Reingold force-directed
algorithm
```

```
nx.draw(G, pos, with_labels=True, node_size=1000, node_color='lightblue', font_size=12)
plt.title('COVID-19 Transmission Network')
plt.show()
```

OUTPUT:



Department of AIML		
Average Performance	25	
Average Record	15	
Average Viva	10	
Total	50	

RESULT:

The above program is executed successfully.

Ex No: 1d Date:	**Implement 3D visualization for iris dataset using matplotlib**

AIM:

Implement 3D visualization for iris dataset using matplotlib

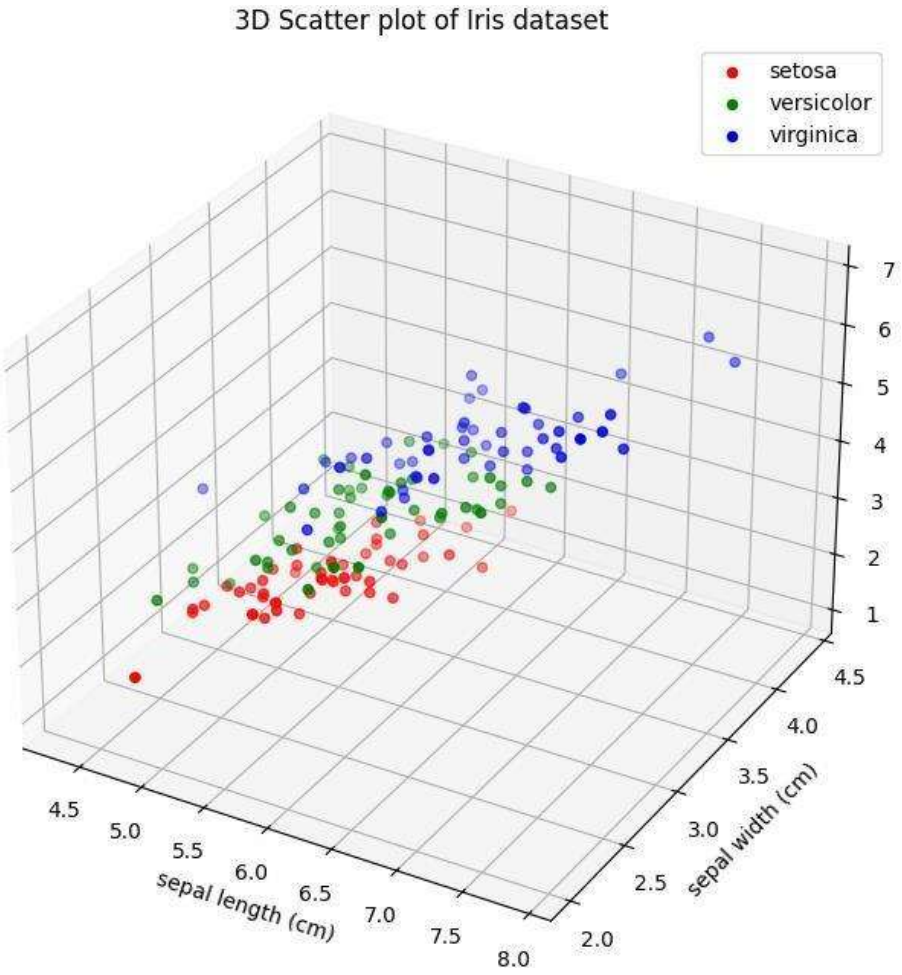
ALGORITHM:

- **Step 1:** Load the iris dataset into memory.
- **Step 2:** Select three features from the dataset to represent along the x, y, and z axes.
- **Step 3:** Create a 3D scatter plot using Matplotlib, with the selected features mapped to the corresponding axes.
- **Step 4:** Customize the plot appearance (e.g., colour, marker size) to enhance readability and interpretability.
- **Step 5:** Interpret the 3D visualization to identify patterns or clusters within the iris dataset.

PROGRAM:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.datasets import load_iris
import numpy as np

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Define the features to use for the 3D plot (for example, the first three features)
feature_names = iris.feature_names[:3]
X_3d = X[:, :3]

# Create a 3D plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot
for c, i, target_name in zip('rgb', [0, 1, 2], iris.target_names):
    ax.scatter(X_3d[y == i, 0], X_3d[y == i, 1], X_3d[y == i, 2], c=c, label=target_name)

ax.set_xlabel(feature_names[0])
ax.set_ylabel(feature_names[1])
ax.set_zlabel(feature_names[2])
ax.set_title('3D Scatter plot of Iris dataset')
ax.legend()
```

plt.show()



OUTPUT:

Department of AIML		
Average Performance	**25**	
Average Record	**15**	
Average Viva	**10**	
Total	**50**	

RESULT:

The above program is executed successfully.

Ex No: 1e Date:	**Implement Resource Description Frame work for smart store data by DBpedia**

AIM:

Implement Resource Description Frame work for smart store data by DBpedia

ALGORITHM:

- **Step 1:** Define the schema for the RDF graph representing smart store data (e.g., entities, attributes, relationships).
- **Step 2:** Query DBpedia for relevant knowledge related to smart stores and retail domain.
- **Step 3:** Convert the retrieved DBpedia data into RDF triples and integrate them into the smart store RDF graph.
- **Step 4:** Map the smart store data from your dataset to RDF triples and add them to the RDF graph.
- **Step 5:** Validate and store the RDF graph in a suitable RDF storage system for further querying and analysis.

PROGRAM:

```
from rdflib import Graph, Namespace, URIRef, Literal
from rdflib.namespace import RDF, RDFS

# Create a new RDF graph
g = Graph()

# Define namespaces
dbp = Namespace("http://dbpedia.org/resource/")
dbo = Namespace("http://dbpedia.org/ontology/")
ex = Namespace("http://example.org/smartstore/")

# Define individual resources
smart_store = ex["SmartStore"]
location = dbp["Location"]
product = ex["Product"]
service = ex["Service"]

# Add triples to the graph
g.add((smart_store, RDF.type, dbo.Store))
g.add((smart_store, RDFS.label, Literal("Smart Store")))

# Add more information about the smart store
g.add((smart_store, dbo.location, location))
g.add((smart_store, dbo.offers, product))
g.add((smart_store, dbo.offers, service))
```

```
# Add more triples based on your requirements
# Serialize the RDF graph to a file
g.serialize(destination='smart_store.rdf', format='xml')
```

OUTPUT:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
   xmlns:ns1="http://dbpedia.org/ontology/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
>
  <rdf:Description rdf:about="http://example.org/smartstore/SmartStore">
    <rdf:type rdf:resource="http://dbpedia.org/ontology/Store"/>
    <rdfs:label>Smart Store</rdfs:label>
    <ns1:location rdf:resource="http://dbpedia.org/resource/Location"/>
    <ns1:offers rdf:resource="http://example.org/smartstore/Product"/>
    <ns1:offers rdf:resource="http://example.org/smartstore/Service"/>
  </rdf:Description>
</rdf:RDF>
```

Department of AIML		
Average Performance	25	
Average Record	15	
Average Viva	10	
Total	50	

RESULT:

The above program is executed successfully.

Ex No: 1f Date:	**Create one dimensional random dataset and implement Series in python using slicing methods**

AIM:

Create one dimensional random dataset and implement Series in python using slicing methods

ALGORITHM:

- **Step 1:** Generate a random one-dimensional dataset using NumPy.
- **Step 2:** Import the Pandas library and create a Series object from the generated dataset.
- **Step 3:** Perform data manipulation tasks such as sorting, filtering, or descriptive statistics on the Series.
- **Step 4:** Use slicing methods to extract subsets of the Series based on index labels or positions.
- **Step 5:** Validate the results of slicing operations and analyse the extracted subsets for further insights.

PROGRAM:

```
import pandas as pd
import numpy as np

# Create a one-dimensional random dataset
np.random.seed(0)  # For reproducibility
data = np.random.randn(10)  # Generate 10 random numbers from a standard normal distribution

# Create a Series from the random data
series = pd.Series(data)

# Print the original Series
print("Original Series:")
print(series)
print()

# Implement slicing methods
print("Slicing Methods:")
# Slicing by index
print("Slice by index (2nd to 5th elements):")
print(series[1:5])  # Note: Python indexing is zero-based
print()

# Slicing by label
print("Slice by label (index 1 to 4):")
```

```
print(series.loc[1:4])  # Using loc to slice by label
print()

# Slicing by position
print("Slice by position (2nd to 5th elements):")
print(series.iloc[2:5])  # Using iloc to slice by position
```

OUTPUT:

```
Original Series:
0    1.764052
1    0.400157
2    0.978738
3    2.240893
4    1.867558
5   -0.977278
6    0.950088
7   -0.151357
8   -0.103219
9    0.410599
dtype: float64

Slicing Methods:
Slice by index (2nd to 5th elements):
1    0.400157
2    0.978738
3    2.240893
4    1.867558
dtype: float64

Slice by label (index 1 to 4):
1    0.400157
2    0.978738
3    2.240893
4    1.867558
dtype: float64

Slice by position (2nd to 5th elements):
2    0.978738
3    2.240893
4    1.867558
dtype: float64
```

Department of AIML		
Average Performance	25	
Average Record	15	
Average Viva	10	
Total	50	

RESULT:

The above program is executed successfully.

Ex No: 1g Date:	**Create two-dimensional random dataset and implement Data Frames in python with various slicing methods**

AIM:

Create two-dimensional random dataset and implement Data Frames in python with various slicing methods

ALGORITHM:

- **Step 1:** Generate a random two-dimensional dataset using NumPy.
- **Step 2:** Import the Pandas library and create a Data Frame object from the generated dataset.
- **Step 3:** Perform data cleaning and preprocessing tasks such as handling missing values or transforming data types.
- **Step 4:** Utilize Data Frame methods to conduct exploratory data analysis (EDA), including summary statistics, correlation analysis, and data visualization.
- **Step 5:** Apply various slicing methods (e.g., loc, iloc) to extract specific rows, columns, or subsets of the Data Frame for further analysis or visualization.

PROGRAM:

```
import pandas as pd
import numpy as np

# Create a two-dimensional random dataset
np.random.seed(0)  # For reproducibility
data = np.random.randn(5, 3)  # Generate 5 rows and 3 columns of random numbers from a standard normal distribution

# Create a DataFrame from the random data
df = pd.DataFrame(data, columns=['A', 'B', 'C'])

# Print the original DataFrame
print("Original DataFrame:")
print(df)
print()

# Implement slicing methods
print("Slicing Methods:")
# Slicing by column label
print("Slice by column label ('B' column):")
print(df['B'])  # Slice a single column
print()

# Slicing by multiple column labels
print("Slice by multiple column labels ('A' and 'C' columns):")
```

```
print(df[['A', 'C']])  # Slice multiple columns
print()

# Slicing by row index
print("Slice by row index (2nd row):")
print(df.iloc[1])  # Slice a single row
print()

# Slicing by row range using iloc
print("Slice by row range (2nd to 4th rows):")
print(df.iloc[1:4])  # Slice rows using index range
print()

# Slicing by row range using loc
print("Slice by row range with labels (1st to 3rd rows):")
print(df.loc[0:2])  # Slice rows using label range (inclusive)
```

OUTPUT:

Original DataFrame:

```
          A         B         C
0  1.764052  0.400157  0.978738
1  2.240893  1.867558 -0.977278
2  0.950088 -0.151357 -0.103219
3  0.410599  0.144044  1.454274
4  0.761038  0.121675  0.443863
```

Slicing Methods:

Slice by column label ('B' column):

```
0    0.400157
1    1.867558
2   -0.151357
3    0.144044
4    0.121675
Name: B, dtype: float64
```

Slice by multiple column labels ('A' and 'C' columns):

```
          A         C
0  1.764052  0.978738
1  2.240893 -0.977278
2  0.950088 -0.103219
3  0.410599  1.454274
4  0.761038  0.443863
```

Slice by row index (2nd row):

```
A    2.240893
B    1.867558
C   -0.977278
Name: 1, dtype: float64
```

Slice by row range (2nd to 4th rows):

```
          A         B         C
1  2.240893  1.867558 -0.977278
2  0.950088 -0.151357 -0.103219
3  0.410599  0.144044  1.454274
```

Slice by row range with labels (1st to 3rd rows):

```
          A         B         C
0  1.764052  0.400157  0.978738
1  2.240893  1.867558 -0.977278
2  0.950088 -0.151357 -0.103219
```

Department of AIML		
Average Performance	25	
Average Record	15	
Average Viva	10	
Total	50	

RESULT:

The above program is executed successfully.

Ex No: 2a Date:	**Implement the python code to compute the population proportions for the given problem.**

AIM:

Implement the python code to compute the population proportions for the given problem.

ALGORITHM:

- **Step 1:** Receive a dictionary data where keys represent categories and values represent population counts.
- **Step 2:** Initialize a variable total population to zero.
- **Step 3:** Store the computed proportion for each category in the dictionary proportions.
- **Step 4:** Return the dictionary proportions containing category names as keys and their respective population proportions as values.
- **Step 5:** Call this function with appropriate arguments to compute population proportions for a specific problem.

PROGRAM:

```
import numpy as np
import scipy.stats as stats

# Sample sizes and counts
n1 = 400  # Sample size last year
n2 = 380  # Sample size this year
x1 = 166  # Number of guests preferring non-smoking rooms last year
x2 = 205  # Number of guests preferring non-smoking rooms this year

# Compute population proportions
p1 = x1 / n1
p2 = x2 / n2

# Compute pooled proportion and standard error
p_pool = (x1 + x2) / (n1 + n2)
se = np.sqrt(p_pool * (1 - p_pool) * (1 / n1 + 1 / n2))

# Compute z-score
z = (p1 - p2) / se

# Compute p-value
p_value = 2 * (1 - stats.norm.cdf(abs(z)))  # Two-tailed test

# Set significance level
alpha = 0.01

# Compare p-value with significance level
```

```
if p_value < alpha:
    print("Reject the null hypothesis.")
    print("There is sufficient evidence to conclude that there is a difference in the proportion of guests preferring non-smoking rooms between last year and this year.")
else:
    print("Fail to reject the null hypothesis.")
    print("There is not enough evidence to conclude that there is a difference in the proportion of guests preferring non-smoking rooms between last year and this year.")

print("p-value:", p_value)
```

OUTPUT:

Reject the null hypothesis.
There is sufficient evidence to conclude that there is a difference in the proportion of guests preferring non-smoking rooms between last year and this year.
p-value: 0.0005026329784247885

Department of AIML		
Average Performance	25	
Average Record	15	
Average Viva	10	
Total	50	

RESULT:

The above program is executed successfully.

Ex No: 2b Date:	Wine Quality Prediction

AIM:

Develop a Convolutional Neural Network (CNN) model to predict the quality of wine based on its features.

ALGORITHM:

- **Step 1:** Preprocess the data by handling missing values, scaling numerical features, and encoding categorical variables.
- **Step 2:** Compile the model with an appropriate loss function
- **Step 3:** Feed the training data into the model and adjust the weights using backpropagation to minimize the loss function.
- **Step 4:** Use the testing dataset to assess the model's ability to generalize to new samples.
- **Step 5:** Develop an interface (e.g., web application) to accept input features of a wine sample and provide predicted quality as output.

PROGRAM:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Load the dataset
data = pd.read_csv("WineQT.csv")

if 'Id' in data.columns:
    data.drop('Id', axis=1, inplace=True)

missing_values = data.isnull().sum()
print("Missing Values:")
print(missing_values)

X = data.drop('quality', axis=1)
y = data['quality']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the dataset into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Model selection and training
model = LinearRegression()
model.fit(X_train, y_train)

y_pred_train = model.predict(X_train)
train_rmse = mean_squared_error(y_train, y_pred_train, squared=False)
print("\nTrain RMSE:", train_rmse)

y_pred_test = model.predict(X_test)
test_rmse = mean_squared_error(y_test, y_pred_test, squared=False)
print("Test RMSE:", test_rmse)

y_pred_test = model.predict(X_test)

# Compare actual vs. predicted wine quality
results = pd.DataFrame({'Actual Quality': y_test, 'Predicted Quality': y_pred_test})
print("Actual vs. Predicted Wine Quality:")
print(results.head(10))  # Display the first 10 samples

# Calculate RMSE on testing data
test_rmse = mean_squared_error(y_test, y_pred_test, squared=False)
print("\nTest RMSE:", test_rmse)
```

OUTPUT:

```
Missing Values:
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64

Train RMSE: 0.4151109255429171
Test RMSE: 0.3800324502627751
Actual vs. Predicted Wine Quality:
      Actual Quality  Predicted Quality
158                5           5.376391
1081               6           4.812752
291                5           5.268227
538                6           5.088265
367                6           6.096687
793                8           6.604335
128                5           5.364173
56                 5           5.069788
448                6           5.784611
422                5           5.323821

Test RMSE: 0.3800324502627751
```

DATASET:

A	B	C	D	E	F	G	H	I	J	K	L	M
fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	Id
7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5	0
7.8	0.88	0	2.6	0.098	25	67	0.9968	3.2	0.68	9.8	5	1
7.8	0.76	0.04	2.3	0.092	15	54	0.997	3.26	0.65	9.8	5	2
11.2	0.28	0.56	1.9	0.075	17	60	0.998	3.16	0.58	9.8	6	3
7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5	4
7.4	0.66	0	1.8	0.075	13	40	0.9978	3.51	0.56	9.4	5	5
7.9	0.6	0.06	1.6	0.069	15	59	0.9964	3.3	0.46	9.4	5	6
7.3	0.65	0	1.2	0.065	15	21	0.9946	3.39	0.47	10	7	7
7.8	0.58	0.02	2	0.073	9	18	0.9968	3.36	0.57	9.5	7	8
6.7	0.58	0.08	1.8	0.097	15	65	0.9959	3.28	0.54	9.2	5	10
5.6	0.615	0	1.6	0.089	16	59	0.9943	3.58	0.52	9.9	5	12
7.8	0.61	0.29	1.6	0.114	9	29	0.9974	3.26	1.56	9.1	5	13
8.5	0.28	0.56	1.8	0.092	35	103	0.9969	3.3	0.75	10.5	7	16
7.9	0.32	0.51	1.8	0.341	17	56	0.9969	3.04	1.08	9.2	6	19
7.6	0.39	0.31	2.3	0.082	23	71	0.9982	3.52	0.65	9.7	5	21
7.9	0.43	0.21	1.6	0.106	10	37	0.9966	3.17	0.91	9.5	5	22
8.5	0.49	0.11	2.3	0.084	9	67	0.9968	3.17	0.53	9.4	5	23
6.9	0.4	0.14	2.4	0.085	21	40	0.9968	3.43	0.63	9.7	6	24
6.3	0.39	0.16	1.4	0.08	11	23	0.9955	3.34	0.56	9.3	5	25
7.6	0.41	0.24	1.8	0.08	4	11	0.9962	3.28	0.59	9.5	5	26
7.1	0.71	0	1.9	0.08	14	35	0.9972	3.47	0.55	9.4	5	28
7.8	0.645	0	2	0.082	8	16	0.9964	3.38	0.59	9.8	6	29
6.7	0.675	0.07	2.4	0.089	17	82	0.9958	3.35	0.54	10.1	5	30
8.3	0.655	0.12	2.3	0.083	15	113	0.9966	3.17	0.66	9.8	5	32
5.2	0.32	0.25	1.8	0.103	13	50	0.9957	3.38	0.55	9.2	5	34
7.8	0.645	0	5.5	0.086	5	18	0.9986	3.4	0.55	9.6	6	35
7.8	0.6	0.14	2.4	0.086	3	15	0.9975	3.42	0.6	10.8	6	36

Department of AIML		
Average Performance	25	
Average Record	15	
Average Viva	10	
Total	50	

RESULT:

The above program is executed successfully.

Ex No: 2c Date:	Digit Recognition using CNN in Python

AIM:

Develop a Convolutional Neural Network (CNN) model to accurately recognize handwritten digits.

ALGORITHM:

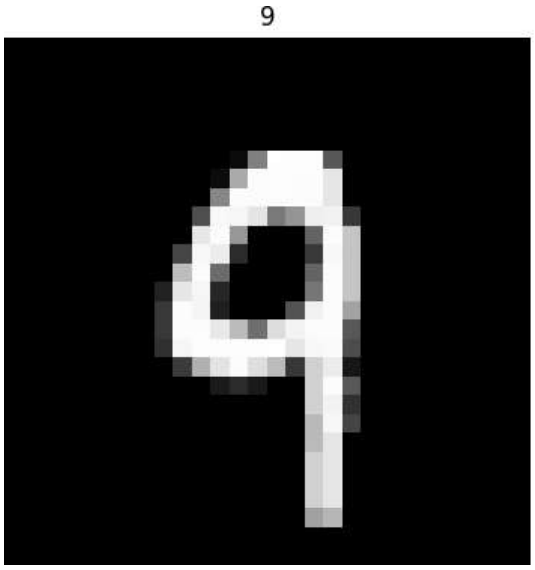
- **Step 1:** Preprocess the images by resizing them to a uniform size, normalizing pixel values, and converting them to grayscale if necessary.
- **Step 2:** Flatten the output of the convolutional layers and add one or more dense layers to classify the extracted features.
- **Step 3:** Feed the training images into the model and adjust the weights using backpropagation to minimize the categorical cross-entropy loss.
- **Step 4:** Calculate evaluation metrics such as accuracy, precision, recall, and F1-score to measure the model's performance.
- **Step 5:** Develop an interface (e.g., web application or API) to accept images containing handwritten digits as input and provide predicted digit labels as output.

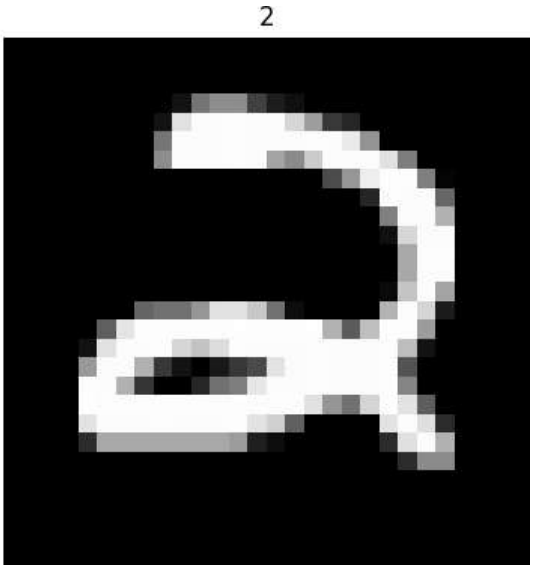
PROGRAM:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf

train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

print("train", train.shape)
print("test", test.shape)

y_train = train.iloc[:,:1]
x_train = train.iloc[:,1:]
x_test = test
print("x_train shape: ", x_train.shape)
print("y_train shape: ", y_train.shape)
print("x_test shape: ", x_test.shape)

plt.figure(figsize=(10, 10))
for i in range(15):
    plt.subplot(5, 5, i + 1)
```

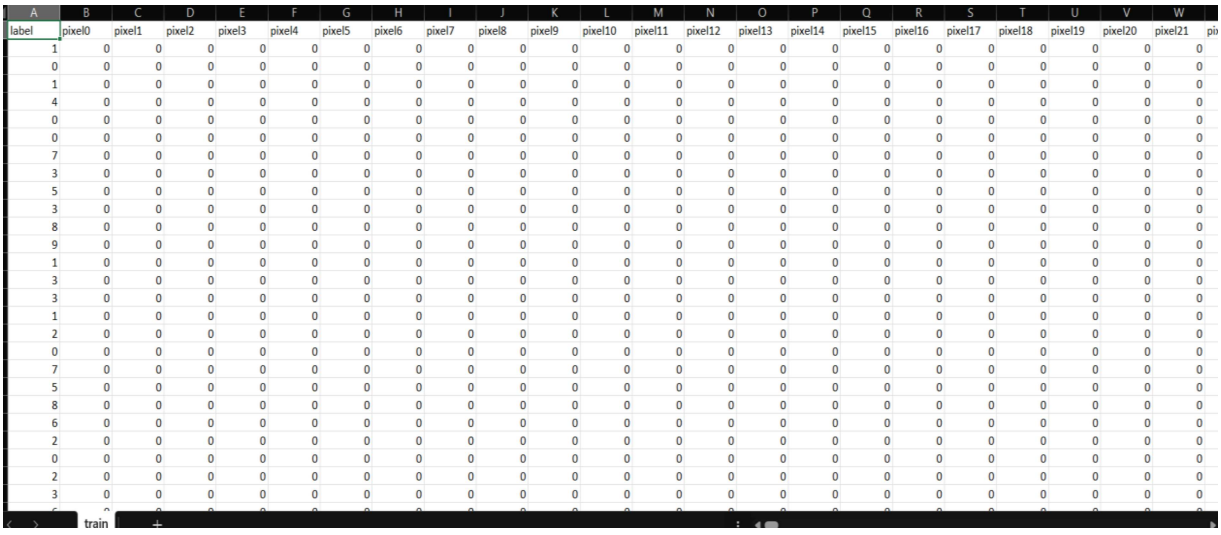
```
    plt.imshow(train.iloc[i, 1:].values.reshape(28, 28), cmap="gray")
    plt.title(train.iloc[i, 0])
    plt.axis("off")

print(f"We have {len(train)} images in the training set and {len(test)} images in the test set.")
print(f"The images are {28} x {28} pixels and have {len(train.columns) - 1} features.")
import random
random_images = random.sample(range(0, len(train)), 2)

# plot the random images
plt.figure(figsize=(10, 10))
for i, image in enumerate(random_images):
    plt.subplot(2, 2, i + 1)
    plt.imshow(train.iloc[image, 1:].values.reshape(28, 28), cmap="gray")
    plt.title(train.iloc[image, 0])
    plt.axis("off")
```

OUTPUT:





DATASET:



label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel14	pixel15	pixel16	pixel17	pixel18	pixel19	pixel20	pixel21
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Department of AIML		
Average Performance	25	
Average Record	15	
Average Viva	10	
Total	50	

RESULT:

The above program is executed successfully.

Ex No: 3 Date:	House Price Prediction

AIM:

Write code to predict house prices based on several parameters available in the Housing and Urban Development of TN dataset using least squares linear regression.

ALGORITHM:

- **Step 1:** Preprocess the data by handling missing values, encoding categorical variables, and scaling numerical features.
- **Step 2:** Analyse the dataset and identify important features such as square footage, number of bedrooms/bathrooms, location, and amenities.
- **Step 3:** Train the chosen model using the training dataset, tuning hyperparameters as necessary to optimize performance.
- **Step 4:** Conduct cross-validation to ensure the model's generalizability and reliability on unseen data.
- **Step 5:** Implement a monitoring system to track the model's performance metrics and detect any drift or degradation in prediction accuracy over time.

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

data = pd.read_csv("Housing.csv")
categorical_cols = ['mainroad', 'guestroom', 'basement',
                    'hotwaterheating', 'airconditioning', 'prefarea', 'furnishingstatus']
data_encoded = pd.get_dummies(data, columns=categorical_cols, drop_first=True)
X = data_encoded.drop(columns=["price"])
y = data_encoded["price"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
rf_model = RandomForestRegressor(
    n_estimators=500,
    max_depth=10,
    min_samples_split=5,
    min_samples_leaf=2,
    random_state=42
)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
print("Predicted Prices:")
for sqft_value, predicted_price in zip(X_test['area'], y_pred):
    print(f"Square Footage: {sqft_value}, Predicted Price: {int(predicted_price)}")
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\nMean Squared Error:", mse)
print("R-squared:", r2)
print("Model Score on Test Set:", rf_model.score(X_test, y_test))
data_encoded['Predicted Price'] = rf_model.predict(X)
data_encoded.to_csv("updated_housing_dataset.csv", index=False)
```

OUTPUT:

```
Predicted Prices:
Square Footage: 400.0, Predicted Price: 4872.083333333333
Square Footage: 950.0, Predicted Price: 6485.0
Square Footage: 1200.0, Predicted Price: 7449.916666666666
Square Footage: 800.0, Predicted Price: 12172.166666666668
Square Footage: 900.0, Predicted Price: 13751.138888888885
Square Footage: 700.0, Predicted Price: 9629.166666666668
Square Footage: 967.0, Predicted Price: 11643.000000000002
Square Footage: 990.0, Predicted Price: 11162.5
Square Footage: 600.0, Predicted Price: 5947.976190476192
Square Footage: 700.0, Predicted Price: 9629.166666666668
Square Footage: 650.0, Predicted Price: 7174.583333333332
Square Footage: 750.0, Predicted Price: 9947.333333333336
Square Footage: 600.0, Predicted Price: 5947.976190476192
Square Footage: 750.0, Predicted Price: 9947.333333333336
Square Footage: 1400.0, Predicted Price: 13496.0
Square Footage: 450.0, Predicted Price: 4050.0
Square Footage: 900.0, Predicted Price: 13751.138888888885
Square Footage: 900.0, Predicted Price: 13751.138888888885
Square Footage: 650.0, Predicted Price: 7174.583333333332
Square Footage: 850.0, Predicted Price: 13441.166666666668
Square Footage: 1100.0, Predicted Price: 11549.166666666664
Square Footage: 750.0, Predicted Price: 9947.333333333336
Square Footage: 800.0, Predicted Price: 12172.166666666668
Square Footage: 2000.0, Predicted Price: 12176.83
...
Square Footage: 1750.0, Predicted Price: 9030.0
Mean Squared Error: 2011266315670.2124
R-squared: 0.602089551110955
Model Score on Test Set: 0.602089551110955
```

DATASET:

price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterh	airconditic	parking	prefarea	furnishingstatus
13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished
10850000	7500	3	3	1	yes	no	yes	no	yes	2	yes	semi-furnished
10150000	8580	4	3	4	yes	no	no	no	yes	2	yes	semi-furnished
10150000	16200	5	3	2	yes	no	no	no	no	0	no	unfurnished
9870000	8100	4	1	2	yes	yes	yes	no	yes	2	yes	furnished
9800000	5750	3	2	4	yes	yes	no	no	yes	1	yes	unfurnished
9800000	13200	3	1	2	yes	no	yes	no	yes	2	yes	furnished
9681000	6000	4	3	2	yes	yes	yes	yes	no	2	no	semi-furnished
9310000	6550	4	2	2	yes	no	no	no	yes	1	yes	semi-furnished
9240000	3500	4	2	2	yes	no	no	yes	no	2	no	furnished
9240000	7800	3	2	2	yes	no	no	no	no	0	yes	semi-furnished
9100000	6000	4	1	2	yes	no	yes	no	no	2	no	semi-furnished
9100000	6600	4	2	2	yes	yes	yes	no	yes	1	yes	unfurnished
8960000	8500	3	2	4	yes	no	no	no	yes	2	no	furnished
8890000	4600	3	2	2	yes	yes	no	no	yes	2	no	furnished
8855000	6420	3	2	2	yes	no	no	no	yes	1	yes	semi-furnished
8750000	4320	3	1	2	yes	no	yes	yes	no	2	no	semi-furnished

Department of AIML		
Average Performance	25	
Average Record	15	
Average Viva	10	
Total	50	

RESULT:

The above program is executed successfully.

Ex No: 4 Date:	Movie Recommendation Engine

AIM:

Build a movie recommendation engine by applying collaborative filtering and topic modelling techniques

ALGORITHM:

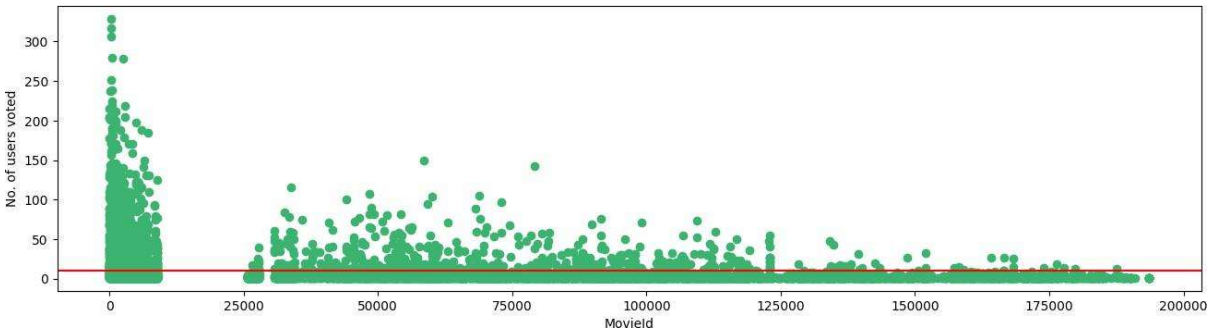
- **Step 1:** Preprocess the data by handling missing values, removing duplicates, and encoding categorical variables.
- **Step 2:** Compute user-item similarity matrices using techniques like cosine similarity or Pearson correlation coefficient.
- **Step 3:** Preprocess movie descriptions or reviews by tokenizing, lemmatizing, and removing stop words.
- **Step 4:** Weight the recommendations from each technique based on their relevance and novelty to the user.
- **Step 5:** Deploy the recommendation engine as a web service or API that accepts user preferences and provides personalized movie recommendations in real-time.

PROGRAM:

```
import pandas as pd
import numpy as np
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt
import seaborn as sns
movies = pd.read_csv("Movie_Recommendation\\movies.csv")
ratings = pd.read_csv("Movie_Recommendation\\ratings.csv")

print(movies.head())
ratings.head()
final_dataset = ratings.pivot(index='movieId',columns='userId',values='rating')
final_dataset.head()
final_dataset.fillna(0,inplace=True)
final_dataset.head()
no_user_voted = ratings.groupby('movieId')['rating'].agg('count')
no_movies_voted = ratings.groupby('userId')['rating'].agg('count')
f,ax = plt.subplots(1,1,figsize=(16,4))
# ratings['rating'].plot(kind='hist')
plt.scatter(no_user_voted.index,no_user_voted,color='mediumseagreen')
plt.axhline(y=10,color='r')
plt.xlabel('MovieId')
plt.ylabel('No. of users voted')
plt.show()
```

```
final_dataset = final_dataset.loc[no_user_voted[no_user_voted > 10].index,:]
final_dataset=final_dataset.loc[:,no_movies_voted[no_movies_voted > 50].index]
final_dataset
sample = np.array([[0,0,3,0,0],[4,0,0,0,2],[0,0,0,0,1]])
sparsity = 1.0 - ( np.count_nonzero(sample) / float(sample.size) )
print(sparsity)
csr_sample = csr_matrix(sample)
print(csr_sample)
csr_data = csr_matrix(final_dataset.values)
final_dataset.reset_index(inplace=True)
knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)
knn.fit(csr_data)
def get_movie_recommendation(movie_name):
    n_movies_to_reccomend = 10
    movie_list = movies[movies['title'].str.contains(movie_name)]
    if len(movie_list):
        movie_idx= movie_list.iloc[0]['movieId']
        movie_idx = final_dataset[final_dataset['movieId'] == movie_idx].index[0]
        distances , indices =
knn.kneighbors(csr_data[movie_idx],n_neighbors=n_movies_to_reccomend+1)
        rec_movie_indices =
sorted(list(zip(indices.squeeze().tolist(),distances.squeeze().tolist())),key=lambda x: x[1])[:0:-
1]
        recommend_frame = []
        for val in rec_movie_indices:
            movie_idx = final_dataset.iloc[val[0]]['movieId']
            idx = movies[movies['movieId'] == movie_idx].index
            recommend_frame.append({'Title':movies.iloc[idx]['title'].values[0],'Distance':val[1]}
)
        df = pd.DataFrame(recommend_frame,index=range(1,n_movies_to_reccomend+1))
        return df
    else:
        return "No movies found. Please check your input"
get_movie_recommendation('Iron Man')
get_movie_recommendation('Memento')
```

OUTPUT:

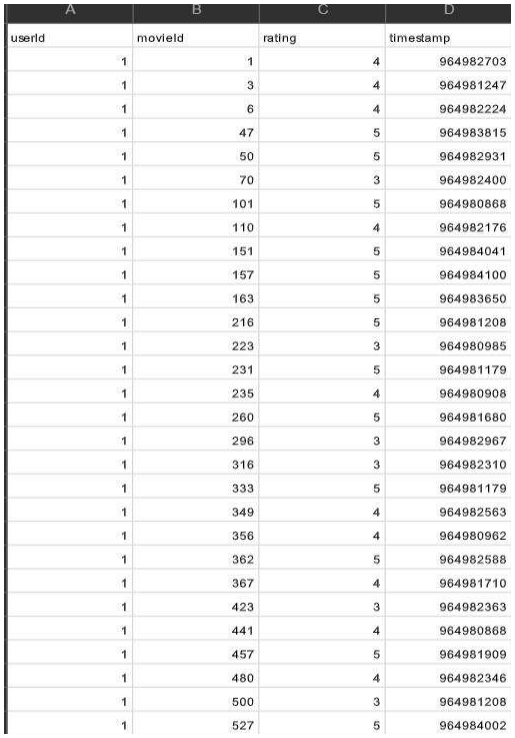


Title	Distance	
1	American Beauty (1999)	0.389346
2	American History X (1998)	0.388615
3	Pulp Fiction (1994)	0.386235
4	Lord of the Rings: The Return of the King, The...	0.371622
5	Kill Bill: Vol. 1 (2003)	0.350167
6	Lord of the Rings: The Two Towers, The (2002)	0.348358
7	Eternal Sunshine of the Spotless Mind (2004)	0.346196
8	Matrix, The (1999)	0.326215
9	Lord of the Rings: The Fellowship of the Ring,...	0.316777
10	Fight Club (1999)	0.272380

.

DATASET:

A	B	C
movieId	title	genres
1	Toy Story (1995)	Adventure|Animatior
2	Jumanji (1995)	Adventure|Children|F
3	Grumpier Old Men (1	Comedy|Romance
4	Waiting to Exhale (1	Comedy|Drama|Rom
5	Father of the Bride P	Comedy
6	Heat (1995)	Action|Crime|Thrille
7	Sabrina (1995)	Comedy|Romance
8	Tom and Huck (1995	Adventure|Children
9	Sudden Death (1995	Action
10	GoldenEye (1995)	Action|Adventure|Th
11	American President,	Comedy|Drama|Rom
12	Dracula: Dead and L	Comedy|Horror
13	Balto (1995)	Adventure|Animatior
14	Nixon (1995)	Drama
15	Cutthroat Island (199	Action|Adventure|Ro
16	Casino (1995)	Crime|Drama
17	Sense and Sensibilit	Drama|Romance
18	Four Rooms (1995)	Comedy
19	Ace Ventura: When I	Comedy
20	Money Train (1995)	Action|Comedy|Crim
21	Get Shorty (1995)	Comedy|Crime|Thrill
22	Copycat (1995)	Crime|Drama|Horror|
23	Assassins (1995)	Action|Crime|Thrille
24	Powder (1995)	Drama|Sci-Fi
25	Leaving Las Vegas (1	Drama|Romance
26	Othello (1995)	Drama
27	Now and Then (1995	Children|Drama
28	Persuasion (1995)	Drama|Romance
29	City of Lost Children,	Adventure|Drama|Fa



A	B	C	D
userId	movieId	rating	timestamp
1	1	4	964982703
1	3	4	964981247
1	6	4	964982224
1	47	5	964983815
1	50	5	964982931
1	70	3	964982400
1	101	5	964980868
1	110	4	964982176
1	151	5	964984041
1	157	5	964984100
1	163	5	964983650
1	216	5	964981208
1	223	3	964980985
1	231	5	964981179
1	235	4	964980908
1	260	5	964981680
1	296	3	964982967
1	316	3	964982310
1	333	5	964981179
1	349	4	964982563
1	356	4	964980962
1	362	5	964982588
1	367	4	964981710
1	423	3	964982363
1	441	4	964980868
1	457	5	964981909
1	480	4	964982346
1	500	3	964981208
1	527	5	964984002

Department of AIML		
Average Performance	25	
Average Record	15	
Average Viva	10	
Total	50	

RESULT:

The above program is executed successfully.

Ex No: 5 Date:	**Market Segmentation using K-means Clustering on India Census Data**

AIM:

Create market segments using the India Census dataset and by applying the k-means clustering method.

ALGORITHM:

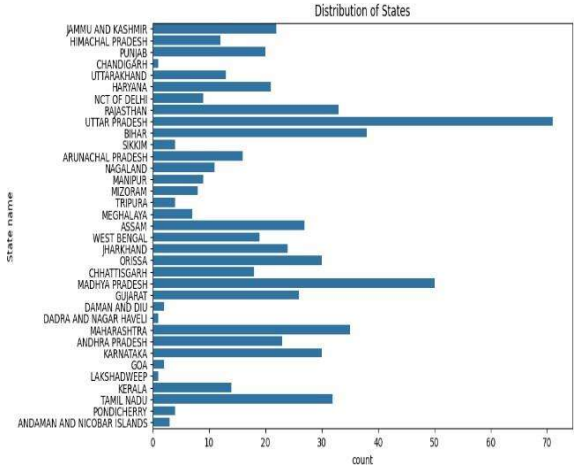
- **Step 1:** Preprocess the dataset by handling missing values, encoding categorical variables, and scaling numerical features if necessary.
- **Step 2:** Standardize the selected features to ensure that they have similar scales and variances, which is important for the k-means algorithm.
- **Step 3:** Choose the value of K that maximizes clustering performance while minimizing within-cluster variance.
- **Step 4:** Initialize K cluster centroids randomly or using a smart initialization technique like K-means++.
- **Step 5:** Visualize the clusters using techniques like scatter plots, box plots, or parallel coordinate plots to identify patterns and differences between segments.

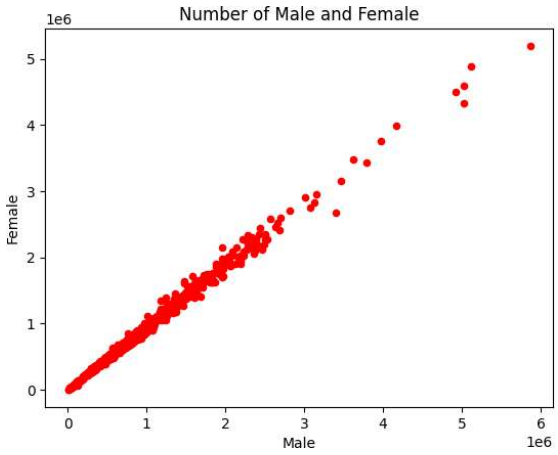
PROGRAM:

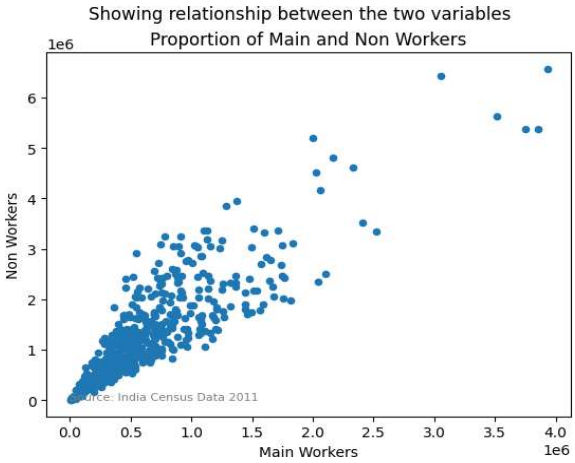
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv('India Census\\india-districts-census-2011.csv')
print(data.shape)
print(data.columns)
print(data.info())
print(data.describe())

plt.figure(figsize=(10, 6))
sns.countplot(data['State name'])
plt.title('Distribution of States')
plt.show()
data.plot(kind='scatter', x='Male', y='Female', color='red')
plt.title('Number of Male and Female')
plt.xlabel('Male')
plt.ylabel('Female')

plt.show()
import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame containing the India Census data
```

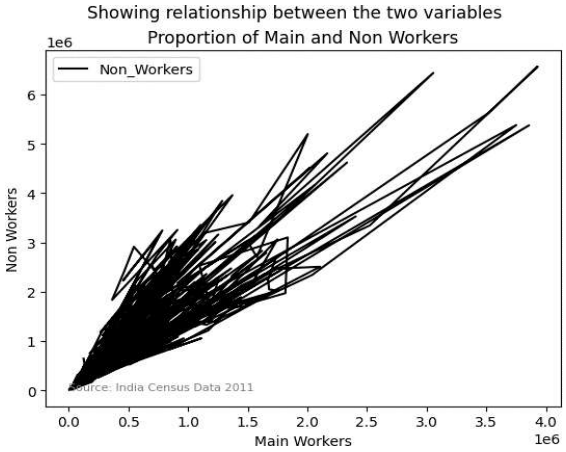
```
# Scatter plot
data.plot(kind='scatter', x='Main_Workers', y='Non_Workers', )
plt.title('Proportion of Main and Non Workers')
plt.xlabel('Main Workers')
plt.ylabel('Non Workers')
plt.suptitle('Showing relationship between the two variables')  # Corrected from 'subtitle' to 'suptitle'
plt.text(0, 0, 'Source: India Census Data 2011', fontsize=8, color='gray')  # Corrected from 'caption' to 'text'
plt.show()

# Line plot
data.plot(kind='line', x='Main_Workers', y='Non_Workers', color='black')
plt.title('Proportion of Main and Non Workers')
plt.xlabel('Main Workers')
plt.ylabel('Non Workers')
plt.suptitle('Showing relationship between the two variables')  # Corrected from 'subtitle' to 'suptitle'
plt.text(0, 0, 'Source: India Census Data 2011', fontsize=8, color='gray')  # Corrected from 'caption' to 'text'
plt.show()
```

OUTPUT:


Distribution of States
JAMMU AND KASHMIR
HIMACHAL PRADESH
PUNJAB
CHANDIGARH
UTTARAKHAND
HARYANA
NCT OF DELHI
RAJASTHAN
UTTAR PRADESH
BIHAR
SIKKIM
ARUNACHAL PRADESH
NAGALAND
MANIPUR
MIZORAM
TRIPURA
MEGHALAYA
ASSAM
WEST BENGAL
JHARKHAND
ORISSA
CHHATTISGARH
MADHYA PRADESH
GUJARAT
DAMAN AND DIU
DADRA AND NAGAR HAVELI
MAHARASHTRA
ANDHRA PRADESH
KARNATAKA
GOA
LAKSHADWEEP
KERALA
TAMIL NADU
PONDICHERRY
ANDAMAN AND NICOBAR ISLANDS
State name
0 10 20 30 40 50 60 70
count


Number of Male and Female
1e6
Female
0 1 2 3 4 5
Male
0 1 2 3 4 5 6
1e6


Showing relationship between the two variables
Proportion of Main and Non Workers
1e6
Non Workers
0 1 2 3 4 5 6
Source: India Census Data 2011
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0
Main Workers
1e6


Showing relationship between the two variables
Proportion of Main and Non Workers
1e6
Non_Workers
Non Workers
0 1 2 3 4 5 6
Source: India Census Data 2011
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0
Main Workers
1e6

DATASET:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	District code	State name	District name	Population	Male	Female	Literate	Male_Literate	Female_Literate	SC	Male_SC	Female_SC	ST	Male_ST
1	1	JAMMU AND KASHM	Kupwara	870354	474190	396164	439654	282823	156831	1048	1046	2	70352	36913
2	2	JAMMU AND KASHM	Badgam	753745	398041	355704	335649	207741	127908	368	343	25	23912	12383
3	3	JAMMU AND KASHM	Leh(Ladakh)	133487	78971	54516	93770	62834	30936	488	444	44	95857	47543
4	4	JAMMU AND KASHM	Kargil	140802	77785	63017	86236	56301	29935	18	12	6	122336	62652
5	5	JAMMU AND KASHM	Punch	476835	251899	224936	261724	163333	98391	556	406	150	176101	90274
6	6	JAMMU AND KASHM	Rajouri	642415	345351	297064	364109	224469	139640	48157	25170	22987	232815	121374
7	7	JAMMU AND KASHM	Kathua	616435	326109	290326	389204	228499	160705	141224	74644	66580	53307	27693
8	8	JAMMU AND KASHM	Baramula	1008039	534733	473306	545149	337170	207979	1476	1451	25	37705	20237
9	9	JAMMU AND KASHM	Bandipore	392232	207680	184552	185979	117058	68921	392	375	17	75374	39398
10	10	JAMMU AND KASHM	Srinagar	1236829	651124	585705	748584	431746	316838	1068	995	73	8935	5021
11	11	JAMMU AND KASHM	Ganderbal	297446	158720	138726	143276	90581	52695	117	105	12	61070	32554
12	12	JAMMU AND KASHM	Pulwama	560440	293064	267376	293958	178326	115632	402	397	5	22607	11837
13	13	JAMMU AND KASHM	Shupiyan	266215	136480	129735	136500	80355	56145	43	43	0	21820	11311
14	14	JAMMU AND KASHM	Anantnag	1078692	559767	518925	545532	324417	221115	1826	1811	15	116006	60990
15	15	JAMMU AND KASHM	Kulgam	424483	217620	206863	209085	125052	84033	21	11	10	26525	13888
16	16	JAMMU AND KASHM	Doda	409936	213641	196295	219083	138620	80463	53408	27209	26199	39216	20377
17	17	JAMMU AND KASHM	Ramban	283713	149132	134581	124065	82938	41127	13920	7168	6752	39772	20940
18	18	JAMMU AND KASHM	Kishtwar	230696	120165	110531	107508	68700	38806	14307	7322	6985	38149	19889
19	19	JAMMU AND KASHM	Udhampur	554985	296784	258201	322354	197543	124811	138569	72093	66476	56309	29142
20	20	JAMMU AND KASHM	Reasi	314667	166461	148206	150542	93937	56605	37757	19657	18100	88365	46330
21	21	JAMMU AND KASHM	Jammu	1529958	813821	716137	1137135	641916	495219	377991	197610	180381	69193	36323
22	22	JAMMU AND KASHM	Samba	318898	169124	149774	228139	130312	97827	91635	47920	43915	17573	9188
23	23	HIMACHAL PRADES	Chamba	519080	261320	257760	323842	186064	137778	111690	56154	55536	135500	67900
24	24	HIMACHAL PRADES	Kangra	1510075	750591	759484	1152640	606443	546197	319385	159697	159688	84564	41745
25	25	HIMACHAL PRADES	Lahul AND Spiti	31564	16588	14976	21845	12897	8948	2235	1154	1081	25707	12748
26	26	HIMACHAL PRADES	Kullu	437903	225452	212451	307672	174550	133122	122659	62686	59973	16822	8493
27	27	HIMACHAL PRADES	Mandi	999777	498065	501712	723747	393669	330078	293739	147250	146489	12787	6345

Department of AIML		
Average Performance	25	
Average Record	15	
Average Viva	10	
Total	50	

RESULT:

The above program is executed successfully.