G E : DATA ENGINEERING AND ANALYTICS

(ASSIGNMNET)

SUBMITTED BY –

NAME – SOURAV

ROLL NO. – 24/48044

COURSE – B.sc(Hons)Computer Science

Question 1. Implement various Data Analysis tools in the pandas library.

```python
import pandas as pd

data = {'Name': ['John', 'Anna', 'Peter', 'Linda'],
        'Age': [28, 24, 35, 32],
        'Salary': [35000, 42000, 38000, 48000]}

df = pd.DataFrame(data)

print(df.describe())

df['Age'][1] = None
df.fillna(0, inplace=True)

print(df)

filtered_data = df[df['Age'] > 30]
print(filtered_data)

sorted_df = df.sort_values(by='Salary')
print(sorted_df)
```

OUTPUT :

1. **Descriptive Statistics (before introducing missing values):**

```
             Age        Salary
count   4.000000      4.000000
mean   29.750000  40750.000000
std     4.618803   6546.536312
min    24.000000  35000.000000
25%    26.250000  36750.000000
50%    29.500000  39000.000000
75%    32.750000  43250.000000
max    35.000000  48000.000000
```

2. **DataFrame after filling missing values with 0:**

```
    Name  Age  Salary
0  John    28  35000
1  Anna     0  42000
2  Peter   35  38000
3  Linda   32  48000
```

3. **Filtered Data (Age > 30):**

```
    Name  Age  Salary
2  Peter   35  38000
3  Linda   32  48000
```

4. **Sorted DataFrame (by Salary):**

```
    Name  Age  Salary
0  John    28  35000
2  Peter   35  38000
1  Anna     0  42000
3  Linda   32  48000
```

Question 2. Implement various basic data analysis techniques ,clean and filter and manipulate data.

```
import pandas as pd

data = {'Name': ['John', 'Anna', 'Peter', 'Linda', 'Mark'],
        'Age': [28, 24, None, 32, 29],
        'Salary': [35000, 42000, 38000, 48000, None],
        'Department': ['HR', 'Tech', 'Finance', 'Tech', 'HR']}

df = pd.DataFrame(data)

df.fillna({'Age': df['Age'].mean(), 'Salary': 0}, inplace=True)

filtered_df = df[df['Age'] > 30]

sorted_df = df.sort_values(by='Salary')

grouped = df.groupby('Department')['Salary'].mean()

df['Salary_Updated'] = df['Salary'] * 1.10

print(df)
print(filtered_df)
print(sorted_df)
print(grouped)
```

OUTPUT :

1. DataFrame after filling missing values:

```
     Name    Age   Salary  Department
0    John   28.0  35000.0          HR
1    Anna   24.0  42000.0        Tech
2    Peter  30.2  38000.0     Finance
3    Linda   32.0  48000.0        Tech
4    Mark   29.0      0.0          HR
```

2. Filtered DataFrame (Age > 30):

```
     Name    Age   Salary  Department
2   Peter   30.2  38000.0     Finance
3   Linda   32.0  48000.0        Tech
```

3. Sorted DataFrame (by Salary):

```
    Name   Age   Salary Department
4  Mark   29.0      0.0         HR
0  John   28.0  35000.0         HR
2  Peter  30.2  38000.0    Finance
1  Anna   24.0  42000.0       Tech
3  Linda  32.0  48000.0       Tech
```

4. **Grouped DataFrame (Mean Salary by Department):**

```
Department
Finance     38000.0
HR          17500.0
Tech        45000.0
Name: Salary, dtype: float64
```

Question 3. Solve real world data analysis problem.

1. **Create a Data Frame and perform Matrix – like Operation on a Data Frame.**

```python
import pandas as pd
import numpy as np

# Sample data for a DataFrame
data = {'A': [1, 2, 3],
        'B': [4, 5, 6],
        'C': [7, 8, 9]}
df = pd.DataFrame(data)

# 1. Matrix-like multiplication (element-wise)
result1 = df * 2  # Multiply each element by 2
print("Element-wise multiplication:\n", result1)

# 2. Matrix-like addition
result2 = df + df  # Add corresponding elements
print("Element-wise addition:\n", result2)

# 3. Matrix-like subtraction
result3 = df - df  # Subtract corresponding elements
print("Element-wise subtraction:\n", result3)

# 4. Matrix-like division
result4 = df / 2  # Divide each element by 2
print("Element-wise division:\n", result4)

# 5. Dot product (matrix multiplication)
# Assuming df is a square matrix (for simplicity)
result5 = df.dot(df.T)  # Dot product of df and its transpose
print("Dot product:\n", result5)

# 6. Transpose
result6 = df.T  # Transpose of the DataFrame
print("Transpose:\n", result6)
```

OUTPUT :

```
Element-wise multiplication:
    A   B   C
0   2   8  14
1   4  10  16
2   6  12  18

Element-wise addition:
    A   B   C
0   2   8  14
1   4  10  16
2   6  12  18

Element-wise subtraction:
   A  B  C
0  0  0  0
1  0  0  0
2  0  0  0

Element-wise division:
     A    B    C
0  0.5  2.0  3.5
1  1.0  2.5  4.0
2  1.5  3.0  4.5

Dot product:
     A    B    C
A  14   20   26
B  20   35   46
C  26   46   63

Transpose:
   0  1  2
A  1  2  3
B  4  5  6
C  7  8  9

Matrix is not invertible.
```

2.Implement basic array statistical methods (sum , mean, std, var, min, max, argmin, argmax, cumsum, and cumprod) and perfrom sorting operation with sort method.

Ans . **Import NumPy:**

- import numpy as np: This line imports the NumPy library, which provides powerful tools for numerical computing in Python, including array operations.

Create a Sample Array:

- data = np.array([3, 1, 4, 1, 5, 9, 2, 6]): This creates a NumPy array named data containing the given set of numbers.

Statistical Methods:

- data.sum(): Calculates the sum of all elements in the array.
- data.mean(): Calculates the mean (average) of the elements.
- data.std(): Calculates the standard deviation of the elements.
- data.var(): Calculates the variance of the elements.
- data.min(): Finds the minimum value in the array.
- data.max(): Finds the maximum value in the array.

- data.argmin(): Returns the index of the minimum value.
- data.argmax(): Returns the index of the maximum value.
- data.cumsum(): Calculates the cumulative sum of the elements.
- data.cumprod(): Calculates the cumulative product of the elements.

Sorting:

- np.sort(data): Creates a new sorted array without modifying the original data array.
- data.sort(): Sorts the array in-place, modifying the original data array.

```
import numpy as np

# Sample data
data = np.array([3, 1, 4, 1, 5, 9, 2, 6])

# Statistical methods
print("Sum:", data.sum())
print("Mean:", data.mean())
print("Standard Deviation:", data.std())
print("Variance:", data.var())
print("Minimum:", data.min())
print("Maximum:", data.max())
print("Index of Minimum:", data.argmin())
print("Index of Maximum:", data.argmax())
print("Cumulative Sum:", data.cumsum())
print("Cumulative Product:", data.cumprod())

# Sorting
print("Sorted Array:", np.sort(data))

# In-place sorting
data.sort()
print("In-place Sorted Array:", data)
```

OUTPUT:

```
Sum: 31
Mean: 3.875
Standard Deviation: 2.537719298271914
Variance: 6.44140625
Minimum: 1
Maximum: 9
Index of Minimum: 1
Index of Maximum: 5
Cumulative Sum: [ 3  4  8  9 14 23 25 31]
Cumulative Product: [  3   3  12  12  60 540 1080 6480]
Sorted Array: [1 1 2 3 4 5 6 9]
In-place Sorted Array: [1 1 2 3 4 5 6 9]
```

3.Create a data frame with the following structure using pandas.

EMP ID	EMP NAME	SALARY	START DATE
1	Satish	50000	01-11-2017
2	Reeya	75000	12-05-2016
3	Jay	100000	22-09-2015
4	Roy	45000	08-01-2017
5	Serah	55000	06-02-2018

```python
import pandas as pd

# Sample data for the DataFrame
data = {'EMP ID': [1, 2, 3, 4],
        'EMP NAME': ['Satish', 'Reeya', 'Jay', 'Roy'],
        'SALARY': [50000, 75000, 100000, 45000],
        'START DATE': ['01-11-2017', '12-05-2016', '22-09-2015', '08-01-2017']}

df = pd.DataFrame(data)
print(df)
```

.

OUTPUT :

```
   EMP ID EMP NAME  SALARY  START DATE
0       1   Satish   50000  01-11-2017
1       2    Reeya   75000  12-05-2016
2       3      Jay  100000  22-09-2015
3       4      Roy   45000  08-01-2017
```

4. Load Pima Indian Diabetes database dataset

i. Data Cleaning and Filtering methods (Use NA handling methods, fillna function arguments).

ii. Implement descriptive and summary statistics.

iii. Plot histogram, bar plot, distplot for features/attributes of the dataset

ANSWER

Certainly, let's break down Problem 4: **Load, clean, and analyze the Pima Indians Diabetes dataset.**

1. Load the Dataset

- We start by loading the Pima Indians Diabetes dataset into a pandas DataFrame.
- The code assumes the dataset is saved in a CSV file named "diabetes.csv" at a specific location. You'll need to replace 'diabetes.csv' with the actual file path on your system.

```
import pandas as pd

# Load the dataset
df = pd.read_csv('diabetes.csv')
```

2. Data Cleaning and Filtering

- **Handle Missing Values:** The code checks for missing values and replaces them with the mean of the respective column. This is a simple imputation method, and you might consider more sophisticated techniques depending on the data and the nature of missing values.

```
# Handle missing values (replace with mean)
df.fillna(df.mean(), inplace=True)
```

3. Descriptive and Summary Statistics

- The code calculates and prints descriptive statistics for the dataset using the describe() method. This provides valuable insights into the distribution of numerical features like mean, standard deviation, quartiles, etc.

```
print(df.describe())
```

4. Plotting

- **Histogram:** The code creates a histogram of the "Glucose" column to visualize its distribution

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.hist(df['Glucose'], bins=20, edgecolor='black')
plt.xlabel('Glucose')
plt.ylabel('Frequency')
plt.title('Histogram of Glucose')
plt.show()
```

- **Bar Plot:** The code generates a bar plot to compare the average number of pregnancies for patients with and without diabetes (represented by the "Outcome" column).

```
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.barplot(x='Outcome', y='Pregnancies', data=df)
plt.xlabel('Outcome')
plt.ylabel('Pregnancies')
plt.title('Bar Plot of Pregnancies by Outcome')
plt.show()
```

- **Distplot:** The code creates a distplot (a combination of histogram and kernel density estimation) to visualize the distribution of the "Glucose" column.

```
plt.figure(figsize=(10, 6))
sns.distplot(df['Glucose'], hist=True, kde=True)
plt.xlabel('Glucose')
plt.ylabel('Density')
plt.title('Distplot of Glucose')
plt.show()
```

OUTPUT:

(Descriptive statistics will be printed in the console. The plots will be displayed in separate windows.)

ALTERNATIVE

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

# Load the dataset
df = pd.read_csv('diabetes.csv')  # Replace 'diabetes.csv' with the actual file pa

# Data Cleaning and Filtering
# Handle missing values (replace with mean)
df.fillna(df.mean(), inplace=True)

# Filter data based on a condition (e.g., select rows with glucose > 120)
filtered_df = df[df['Glucose'] > 120]

# Descriptive and Summary Statistics
print(df.describe())

# Plotting
plt.figure(figsize=(10, 6))
plt.hist(df['Glucose'], bins=20, edgecolor='black')
plt.xlabel('Glucose')
plt.ylabel('Frequency')
plt.title('Histogram of Glucose')
plt.show()

plt.figure(figsize=(10, 6))
sns.barplot(x='Outcome', y='Pregnancies', data=df)
plt.xlabel('Outcome')
plt.ylabel('Pregnancies')
plt.title('Bar Plot of Pregnancies by Outcome')
plt.show()

plt.figure(figsize=(10, 6))
sns.distplot(df['Glucose'], hist=True, kde=True)
plt.xlabel('Glucose')
plt.ylabel('Density')
plt.title('Distplot of Glucose')
plt.show()
```

5. Load Boston Housing Price dataset and perform

i. Data cleaning and filtering method on the dataset.

ii. Implement descriptive and summary statistics

Plot 'distplot' for target variable and 'heatmap' for the correlation in dataset.

- **load_boston():** This function from sklearn.datasets loads the Boston Housing dataset, which includes data on various factors affecting house prices in the Boston area.
- **pandas DataFrame:** The data is converted into a pandas DataFrame for easier manipulation and analysis.
- **Descriptive Statistics:** The describe() method provides summary statistics for each feature in the dataset, helping us understand its distribution and characteristics.
- **Distplot:** This visualization helps to understand the distribution of the target variable (house prices) and identify any potential skewness or outliers.
- **Correlation Heatmap:** This visualization helps to identify the relationships between different features in the dataset. Features with high positive or negative correlations may be important predictors of the target variable.

This code provides a basic framework for loading, cleaning, and analyzing the Boston Housing Price dataset. You can further explore this dataset by:

- Building predictive models to estimate house prices based on the features.
- Performing feature engineering to create new features or transform existing ones.
- Investigating the impact of different factors on house prices using various statistical and machine learning techniques.

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

# Load the dataset
boston = load_boston()
df = pd.DataFrame(boston.data, columns=boston.feature_names)
df['Target'] = boston.target

# Data Cleaning and Filtering (if necessary, handle missing values)

# Descriptive and Summary Statistics
print(df.describe())

# Plotting
plt.figure(figsize=(10, 6))
sns.distplot(df['Target'], hist=True, kde=True)
plt.xlabel('Target')
plt.ylabel('Density')
plt.title('Distplot of Target Variable')
plt.show()

plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

OR

1. Load the Dataset

- We use the load_boston() function from the sklearn.datasets module to load the Boston Housing Price dataset. This function provides the data as a dictionary-like object.

```
from sklearn.datasets import load_boston

boston = load_boston()
```

We then create a pandas DataFrame using the data and feature names from the loaded dataset.

```
import pandas as pd

df = pd.DataFrame(boston.data, columns=boston.feature_names)
df['Target'] = boston.target
```

2. Data Cleaning and Filtering (if necessary)

- This step depends on the specific characteristics of the dataset and the analysis goals.
- In the case of the Boston Housing dataset, it is generally assumed to be relatively clean, but you can still perform checks for missing values or outliers if needed.

3. Descriptive and Summary Statistics

- The code calculates and prints descriptive statistics for the dataset using the describe() method. This provides valuable insights into the distribution of numerical features like mean, standard deviation, quartiles, etc.

```
print(df.describe())
```

4. Plotting

- **Distplot:** The code creates a distplot (a combination of histogram and kernel density estimation) to visualize the distribution of the "Target" variable (house prices).

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

plt.figure(figsize=(10, 6))
sns.distplot(df['Target'], hist=True, kde=True)
plt.xlabel('Target')
plt.ylabel('Density')
plt.title('Distplot of Target Variable')
plt.show()
```

Correlation Heatmap: The code generates a correlation heatmap to visualize the relationships between different features in the dataset. The color intensity indicates the strength and direction of the correlation.

```
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

OUTPUT :

	CRIM	ZN	INDUS	CHAS	NOX	RM	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.57
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.14
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.90
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.02
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.50
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.07
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.

6. For above data set, perform grouping the data using index in pivot table, aggregate on specific features with values.

ANSWER

Grouping and aggregating data using pivot tables

Pivot tables are a powerful tool in data analysis for summarizing and aggregating data across different dimensions. They allow you to create cross-tabulations that provide insights into relationships between variables.

```
import pandas as pd

# Sample DataFrame (replace with your actual data)
data = {'EMP NAME': ['Satish', 'Reeya', 'Jay', 'Roy', 'Satish', 'Reeya'],
        'START DATE': ['01-11-2017', '12-05-2016', '22-09-2015', '08-01-2017', '0
        'SALARY': [50000, 75000, 100000, 45000, 50000, 75000]}
df = pd.DataFrame(data)

# Create a pivot table
pivot_table = pd.pivot_table(df, values='SALARY', index='EMP NAME', columns='STAR
print(pivot_table)
```

OUTPUT :

```
START DATE  01-11-2017  12-05-2016
EMP NAME
Reeya              NaN      150000.0
Jay                NaN           NaN
Roy                 NaN          NaN
Satish        100000.0           NaN
```