SRM Institute of Science and Technology

Department of Computer Applications

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Circular – 2024-25

BCA DS 1st Sem

Fundamentals of Data Science (UDS24102J)

Lab Manual

Lab 1: Write a Python script to print a statement

Title: Printing a Statement in Python

Aim: To write a basic Python script to display a simple text message.

Procedure:

Open a text editor or a Python IDE (Integrated Development Environment).

Type the Python code to print the desired statement using the `print()` function.

Save the file with a `.py` extension (e.g., `hello.py`).

Run the script from the command line by typing `python hello.py` or execute it within the IDE.

Source Code:

```
print("Hello, Data Science World!")
```

Input: (None)

Expected Output:

```
Hello, Data Science World!
```

Lab 2: Perform Analysis on Simple Dataset for Data Science and Business Intelligence Applications

Title: Simple Dataset Analysis for Data Science and Business Intelligence

Aim: To perform basic analysis on a simple dataset. *(Note: This lab requires a specific dataset. I'll provide a sample and analysis. You'll need to adapt it to your actual dataset.)*

Procedure:

1. Obtain a simple dataset (e.g., a CSV file with sales data, student grades, etc.). For this example, assume a CSV file named "sales_data.csv" with columns: `Product`, `Sales`, `Region`.
2. Import the pandas library in Python.
3. Read the dataset into a pandas DataFrame.
4. Perform basic analysis:
   - Calculate summary statistics (mean, median, etc.) for numerical columns.
   - Count the occurrences of unique values in categorical columns.
   - Group data and aggregate to find totals or averages.

Source Code:

```
import pandas as pd

# Load the dataset
data = {'Product': ['A', 'B', 'A', 'C', 'B', 'A', 'C'],
        'Sales': [100, 150, 200, 120, 180, 250, 130],
        'Region': ['North', 'South', 'North', 'East', 'South', 'West',
'East']}
df = pd.DataFrame(data)

# Basic Analysis
print("Summary Statistics for Sales:")
print(df['Sales'].describe())

print("\nValue Counts for Product:")
print(df['Product'].value_counts())

print("\nGrouped Sales by Region:")
print(df.groupby('Region')['Sales'].sum())
```

Input: A CSV file named "sales_data.csv" (or your dataset). For the code above, the input is the dictionary that is converted to a dataframe.

Expected Output:

```
Summary Statistics for Sales:
count      7.000000
mean     161.428571
std       49.635722
min      100.000000
25%      125.000000
50%      150.000000
75%      190.000000
max      250.000000
Name: Sales, dtype: float64
```

```
Value Counts for Product:
A    3
B    2
C    2
Name: Product, dtype: int64

Grouped Sales by Region:
Region
East     250
North    300
South    330
West     250
Name: Sales, dtype: int64
```

Lab 3: Write a Python Program for swapping two numbers and write a Python script for performing subset(), aggregate() functions on iris dataset.

Title: Swapping Numbers and Dataset Manipulation with Iris

Aim: To write a Python program to swap two numbers and use `subset()` and `aggregate()`-like operations on the iris dataset. *(Note: Python doesn't have direct `subset()` and `aggregate()` functions in the same way some other languages might. Pandas provides equivalent functionality. I'll use Pandas for this.)*

Procedure:

1. **Swapping Numbers:**
   - Get two numbers as input.
   - Use a temporary variable or Python's tuple assignment to swap the values.
   - Print the swapped numbers.
2. **Iris Dataset Analysis:**
   - Load the iris dataset (you can use scikit-learn's built-in dataset or load from a CSV).
   - Use Pandas to perform operations similar to `subset()` (filtering) and `aggregate()` (grouping and summarizing).

Source Code:

```
import pandas as pd
from sklearn.datasets import load_iris

# Swapping Numbers
def swap_numbers(a, b):
    """Swaps two numbers."""
    temp = a
    a = b
    b = temp
    return a, b

# Get input
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
swapped_num1, swapped_num2 = swap_numbers(num1, num2)
print(f"Swapped numbers: a = {swapped_num1}, b = {swapped_num2}")

# Iris Dataset Analysis
iris = load_iris()
iris_df = pd.DataFrame(data=iris['data'], columns=iris['feature_names'])
iris_df['target'] = iris['target']  # Add the target variable

# Subset (Filtering) - like operation
setosa_df = iris_df[iris_df['target'] == 0]  # Get only setosa species
print("\nSubset of Iris data (Setosa):")
print(setosa_df.head())

# Aggregate - like operation
average_measurements = iris_df.groupby('target').mean()
print("\nAverage measurements per species:")
print(average_measurements)
```

Input:

For swapping: Two integers entered by the user.

Expected Output:

```
Enter first number: 5
Enter second number: 10
Swapped numbers: a = 10, b = 5

Subset of Iris data (Setosa):
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
target
0                5.1             3.5              1.4             0.2       0
1                4.9             3.0              1.4             0.2       0
2                4.7             3.2              1.3             0.2       0
3                4.6             3.1              1.5             0.2       0
4                5.0             3.6              1.4             0.2       0

Average measurements per species:
       sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cm)  target
target
0                 5.006             3.428             1.462             0.246
0.0
1                 5.936             2.770             4.260             1.326
1.0
2                 6.588             2.974             5.552             2.026
2.0
```

Lab 4: Reading different types of data sets (.txt, .csv) from Web and disk and writing in file in specific disk location.

Title: Reading and Writing Data from/to Files and Web

Aim: To read data from `.txt` and `.csv` files, both from the web and local disk, and write data to a file on the disk.

Procedure:

1. **Reading from a local `.txt` file:**
   - Create a sample `.txt` file.
   - Use Python's `open()` function in read mode (`'r'`) to open the file.
   - Read the contents using `read()` or `readlines()`.
2. **Reading from a local `.csv` file:**
   - Create a sample `.csv` file.
   - Use the `pandas` library to read the CSV file into a DataFrame using `pd.read_csv()`.
3. **Reading from a web `.txt` file:**
   - Use the `requests` library to fetch the content from the URL.
   - Read the text content.
4. **Reading from a web `.csv` file:**
   - Use the `requests` library to fetch the CSV content from the URL.
   - Use `io.StringIO` to convert the string content into a file-like object.
   - Use `pandas` to read the content from the string buffer.
5. **Writing to a file:**
   - Use Python's `open()` function in write mode (`'w'`) to create/open a file.
   - Use the `write()` method to write data to the file.
   - Close the file.

Source Code:

```
import pandas as pd
import requests
import io

# 1. Reading from a local .txt file
with open("local_text_file.txt", "w") as f:
    f.write("This is a sample text file.\nIt has multiple lines.")
with open("local_text_file.txt", "r") as file:
    local_text_content = file.read()
print("Content of local .txt file:")
print(local_text_content)

# 2. Reading from a local .csv file
data = {'col1': [1, 2, 3], 'col2': [4, 5, 6]}
df = pd.DataFrame(data)
df.to_csv("local_csv_file.csv", index=False) #save the df
local_csv_data = pd.read_csv("local_csv_file.csv")
print("\nContent of local .csv file:")
print(local_csv_data)

# 3. Reading from a web .txt file
web_text_url = "https://www.gutenberg.org/files/100/100-0.txt"  # Example:
Project Gutenberg's Moby Dick
try:
    response = requests.get(web_text_url)
    response.raise_for_status()  # Raise an exception for bad status codes
    web_text_content = response.text[:200]  # Read only the first 200
characters
    print("\nContent of web .txt file (first 200 chars):")
    print(web_text_content)
except requests.exceptions.RequestException as e:
    print(f"Error reading from web .txt: {e}")
    web_text_content = ""

# 4. Reading from a web .csv file
web_csv_url = "https://raw.githubusercontent.com/pandas-
dev/pandas/main/doc/data/titanic.csv"  # Example: Titanic dataset
try:
    response = requests.get(web_csv_url)
    response.raise_for_status()
    csv_content = response.text
    csv_file = io.StringIO(csv_content)  # Create a file-like object from the
string
    web_csv_data = pd.read_csv(csv_file)
    print("\nContent of web .csv file (first 5 rows):")
    print(web_csv_data.head())
except requests.exceptions.RequestException as e:
    print(f"Error reading from web .csv: {e}")
    web_csv_data = pd.DataFrame()

# 5. Writing to a file
output_file_path = "output_file.txt"
with open(output_file_path, "w") as outfile:
    outfile.write("This data will be written to a new file.\n")
    outfile.write(f"Content from web txt file: {web_text_content[:50]}...\n")
#first 50
    outfile.write(f"Local csv data:\n{local_csv_data.to_string()}\n")
print(f"\nData written to {output_file_path}")
```

Input:

6. Local `.txt` file: "local_text_file.txt" (created by the script)
7. Local `.csv` file: "local_csv_file.csv" (created by the script)
8. Web `.txt` file: Content from "https://www.gutenberg.org/files/100/100-0.txt"
9. Web `.csv` file: Content from "https://raw.githubusercontent.com/pandas-dev/pandas/main/doc/data/titanic.csv"

Expected Output:

```
Content of local .txt file:
This is a sample text file.
It has multiple lines.

Content of local .csv file:
   col1  col2
0     1     4
1     2     5
2     3     6

Content of web .txt file (first 200 chars):
The Project Gutenberg eBook of Moby Dick; or The Whale, by Herman Melville

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it

Content of web .csv file (first 5 rows):
   PassengerId  Survived  Pclass  ...  Cabin Embarked  Unnamed: 0
```

```
0            1         0       3  ...    NaN        S         NaN
1            2         1       1  ...    C85        C         NaN
2            3         1       3  ...    NaN        S         NaN
3            4         1       1  ...   C123        S         NaN
4            5         0       3  ...    NaN        S         NaN

[5 rows x 13 columns]

Data written to output_file.txt
```

Lab 5: Install Python and apply all basic python functions and perform Numerical Array Processing using NumPy

Title: Python Basics and NumPy for Numerical Array Processing

Aim: To install Python, apply basic Python functions, and perform numerical array processing using the NumPy library.

Procedure:

1. **Install Python:**
   - Download the latest version of Python from the official website (python.org).
   - Run the installer and follow the instructions. Make sure to add Python to your system's PATH during installation.
2. **Install NumPy:**
   - Open a command prompt or terminal.
   - Use pip (Python's package installer) to install NumPy: `pip install numpy`
3. **Basic Python Functions:**
   - Write a Python script to demonstrate the use of basic functions like `print()`, `len()`, `type()`, and string manipulation functions.
4. **NumPy Array Processing:**
   - Import the NumPy library.
   - Create NumPy arrays using `np.array()`, `np.arange()`, `np.zeros()`, etc.
   - Perform array operations: arithmetic, indexing, slicing, reshaping, and broadcasting.

Source Code:

```
import numpy as np

# Basic Python Functions
def demonstrate_basic_functions():
    """Demonstrates basic Python functions."""
    text = "Hello, Python!"
    print(f"Text: {text}")
    print(f"Length of text: {len(text)}")
    print(f"Type of text: {type(text)}")
    print(f"Uppercase text: {text.upper()}")

# NumPy Array Processing
def demonstrate_numpy():
    """Demonstrates NumPy array processing."""
    # Create arrays
    arr1 = np.array([1, 2, 3, 4, 5])
    arr2 = np.arange(0, 10, 2)  # Array from 0 to 10 (exclusive) with step 2
    arr3 = np.zeros((2, 3))     # 2x3 array of zeros

    print("\nNumPy Arrays:")
    print(f"arr1: {arr1}")
    print(f"arr2: {arr2}")
    print(f"arr3:\n{arr3}")

    # Array operations
    print("\nArray Operations:")
    print(f"arr1 + 5: {arr1 + 5}")
    print(f"arr1 * 2: {arr1 * 2}")
    print(f"arr1[1:4]: {arr1[1:4]}")  # Slicing
    print(f"arr1.reshape(5,1):\n{arr1.reshape(5,1)}")  # Reshape
    arr4 = np.array([10, 20, 30, 40, 50])
    print(f"arr1 + arr4: {arr1 + arr4}")  # Element-wise addition

# Call the functions
demonstrate_basic_functions()
demonstrate_numpy()
```

Input: (None)

Expected Output:

```
Text: Hello, Python!
Length of text: 13
Type of text: <class 'str'>
Uppercase text: HELLO, PYTHON!

NumPy Arrays:
arr1: [1 2 3 4 5]
arr2: [0 2 4 6 8]
arr3:
[[0. 0. 0.]
 [0. 0. 0.]]

Array Operations:
arr1 + 5: [ 6  7  8  9 10]
arr1 * 2: [ 2  4  6  8 10]
arr1[1:4]: [2 3 4]
arr1.reshape(5,1):
[[1]
 [2]
 [3]
 [4]
 [5]]
arr1 + arr4: [11 22 33 44 55]
```

Lab 6: Write a Python script to find basic descriptive statistics using summary, str, quartile function on mtcars & cars datasets

Title: Descriptive Statistics on mtcars and cars Datasets

Aim: To use Python (with pandas) to find basic descriptive statistics, similar to `summary()`, `str()`, and quartile functions in R, on the `mtcars` and `cars` datasets. *(Note: I'll use a readily available version of the mtcars dataset. You may need to adjust the file path if you have a local copy. I will create a simple 'cars' dataset.)*

Procedure:

1. **Import pandas.**
2. **Load the `mtcars` dataset.** I'll use a URL.
3. **Create a simple `cars` dataset as a dictionary and convert it into a DataFrame.**
4. **Use `df.describe()` to get summary statistics (like `summary()`).**
5. **Use `df.info()` to get information about the DataFrame (like `str()`).**
6. **Use `df.quantile()` to get quartiles.**

Source Code:

```
import pandas as pd

# Load the mtcars dataset
mtcars_url =
"https://gist.githubusercontent.com/seankross/a412dfb88889b9e084b6/raw/0ef46f
ec8e511482d381e1e608b70e7d06d6b573/mtcars.csv"
mtcars_df = pd.read_csv(mtcars_url)
mtcars_df = mtcars_df.set_index('model') # set model name as index

# Create a simple cars dataset
cars_data = {
    'model': ['Toyota Camry', 'Honda Civic', 'Ford Mustang', 'Chevrolet
Corvette'],
    'mpg': [34, 42, 21, 16],
    'horsepower': [203, 158, 310, 460],
    'price': [25000, 22000, 35000, 60000]
}
cars_df = pd.DataFrame(cars_data)

# Descriptive Statistics for mtcars
print("\nDescriptive Statistics for mtcars dataset:")
print(mtcars_df.describe())  # Equivalent to summary()

print("\nInformation about mtcars dataset:")
print(mtcars_df.info())      # Equivalent to str()

print("\nQuartiles for mtcars dataset:")
print(mtcars_df.quantile([0.25, 0.5, 0.75]))  # Quartiles

# Descriptive Statistics for cars
print("\nDescriptive Statistics for cars dataset:")
print(cars_df.describe())  # Equivalent to summary()

print("\nInformation about cars dataset:")
print(cars_df.info())      # Equivalent to str()

print("\nQuartiles for cars dataset:")
print(cars_df.quantile([0.25, 0.5, 0.75]))  # Quartiles
```

Input:

7. `mtcars` dataset from the provided URL.
8. `cars` dataset created within the script.

Expected Output:

```
Descriptive Statistics for mtcars dataset:
         mpg  cyl   disp   hp  drat     wt   qsec   vs   am  gear   carb
count   32.0   32.0   32.0   32.0   32.00   32.00  32.00   32.0   32.0   32.0
32.0
mean    20.1    6.2  230.7  146.7    3.60    3.22  17.84    0.4    0.4    3.7
2.8
std      6.0    1.8  123.9   68.6    0.53    0.98   1.79    0.5    0.5    0.7
1.6
min     10.4    4.0   71.1   52.0    2.76    1.51  14.50    0.0    0.0    3.0
1.0
25%     15.4    4.0  120.8   96.5    3.08    2.58  16.89    0.0    0.0    3.0
2.0
50%     19.2    6.0  196.3  123.0    3.70    3.33  17.71    0.0    0.0    4.0
2.0
75%     22.8    8.0  326.0  180.0    3.92    3.61  18.90    1.0    1.0    4.0
4.0
max     33.9    8.0  472.0  335.0    4.93    5.42  22.90    1.0    1.0    5.0
8.0

Information about mtcars dataset:
<class 'pandas.core.frame.DataFrame'>
Index: 32 entries, Mazda RX4 to Volvo 142E
Data columns (total 11 columns):
#   Column  Non-Null Count  Dtype
---  ------  --------------  -----
0   mpg     32 non-null     float64
1   cyl     32 non-null     int64
2   disp    32 non-null     float64
3   hp      32 non-null     int64
4   drat    32 non-null     float64
5   wt      32 non-null     float64
6   qsec    32 non-null     float64
7   vs      32 non-null     int64
8   am      32 non-null     int64
9   gear    32 non-null     int64
10  carb    32 non-null     int64
dtypes: float64(5), int64(6)
memory usage: 3.0 KB

Quartiles for mtcars dataset:
         mpg   cyl   disp    hp  drat    wt   qsec    vs    am  gear  carb
0.25  15.425   4.0  120.8   96.5  3.08  2.576  16.89  0.0  0.0   3.0   2.0
0.50  19.200   6.0  196.3  123.0   3.70   3.325   17.71  0.0  0.0    4.0   2.0
0.75  22.800   8.0  326.0  180.0   3.92   3.610   18.90  1.0  1.0    4.0   4.0

Descriptive Statistics for cars dataset:
        mpg  horsepower     price
count   4.0    4.000000     4.00000
mean   28.2  282.750000  35500.00000
std    11.2  136.621045  17078.25128
min    16.0  158.000000  22000.00000
25%    19.8  192.250000  24250.00000
50%    27.5  256.500000  30000.00000
75%    35.9  347.000000  41250.00000

Information about cars dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
```

```
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  ------      --------------  -----
0   model       4 non-null      object
1   mpg         4 non-null      int64
2   horsepower  4 non-null      int64
3   price       4 non-null      int64
dtypes: int64(3), object(1)
memory usage: 256.0 bytes

Quartiles for cars dataset:
       mpg  horsepower  price
0.25  19.75      192.25  24250.0
0.50  27.50      256.50  30000.0
0.75  35.90      347.00  41250.0
```

Lab 7: Find the correlation Matrix.

Title: Correlation Matrix Calculation

Aim: To calculate the correlation matrix for a given dataset using Python and pandas.

Procedure:

1. Import the pandas library.
2. Load the dataset into a pandas DataFrame. For this example, I'll use the `mtcars` dataset.
3. Use the `.corr()` method of the DataFrame to calculate the correlation matrix.
4. Print the correlation matrix.

Source Code:

```
import pandas as pd

# Load the mtcars dataset
mtcars_url =
"https://gist.githubusercontent.com/seankross/a412dfb88889b9e084b6/raw/0ef46f
ec8e511482d381e1e608b70e7d06d6b573/mtcars.csv"
mtcars_df = pd.read_csv(mtcars_url)
mtcars_df = mtcars_df.set_index('model')  # set model name as index

# Calculate the correlation matrix
correlation_matrix = mtcars_df.corr()

# Print the correlation matrix
print("Correlation Matrix of mtcars dataset:")
print(correlation_matrix)
```

Input: The `mtcars` dataset (from the URL).

Expected Output:

```
Correlation Matrix of mtcars dataset:
       mpg     cyl    disp      hp    drat      wt    qsec      vs      am
gear    carb
mpg     1.00  -0.85  -0.85  -0.78    0.68   -0.87    0.42    0.66    0.60
0.48  -0.55
cyl    -0.85    1.00   0.90   0.83   -0.70    0.78   -0.59   -0.81   -0.52
-0.49   0.53
disp   -0.85    0.90   1.00   0.79   -0.71    0.89   -0.43   -0.71   -0.59
-0.56   0.39
hp     -0.78    0.83   0.79    1.00   -0.45    0.66   -0.71   -0.72   -0.24
-0.13   0.75
drat    0.68   -0.70  -0.71   -0.45    1.00   -0.71    0.09    0.44    0.71
0.70  -0.09
wt     -0.87    0.78   0.89    0.66   -0.71    1.00   -0.17   -0.55   -0.69
-0.58   0.43
qsec    0.42   -0.59  -0.43   -0.71    0.09   -0.17    1.00    0.74   -0.23
-0.21  -0.66
vs      0.66   -0.81  -0.71   -0.72    0.44   -0.55    0.74    1.00    0.17
0.21  -0.57
am      0.60   -0.52  -0.59   -0.24    0.71   -0.69   -0.23    0.17    1.00
0.79   0.06
gear    0.48   -0.49  -0.56   -0.13    0.70   -0.58   -0.21    0.21    0.79
1.00   0.27
```

```
carb   -0.55    0.53   0.39    0.75   -0.09     0.43   -0.66   -0.57    0.06
0.27    1.00
```

Lab 8: Plot the correlation plot on dataset and visualize giving an overview of relationships among data on iris data.

Title: Correlation Plot and Visualization of Iris Dataset Relationships

Aim: To visualize the relationships between variables in the iris dataset using a correlation plot.

Procedure:

1. Import the necessary libraries: `pandas`, `matplotlib.pyplot`, and `seaborn`.
2. Load the iris dataset.
3. Calculate the correlation matrix using `df.corr()`.
4. Create a heatmap using `seaborn.heatmap()` to visualize the correlation matrix.
5. Add annotations to the heatmap to display the correlation coefficients.
6. Set the title of the plot.
7. Display the plot using `plt.show()`.

Source Code:

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

# Load the iris dataset
iris = load_iris()
iris_df = pd.DataFrame(data=iris['data'], columns=iris['feature_names'])
iris_df['target'] = iris['target']  # Add the target variable

# Calculate the correlation matrix
correlation_matrix = iris_df.corr()

# Create the correlation heatmap
plt.figure(figsize=(10, 8))  # Adjust figure size for better readability
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix of Iris Dataset')
plt.show()
```

Input: The iris dataset.

Expected Output: A heatmap showing the correlation coefficients between the features of the iris dataset. The heatmap will have values between -1 and 1, with colors indicating the strength and direction of the correlations (e.g., red for strong positive correlation, blue for strong negative correlation).

Lab 9: Install and perform a simple Exploratory Data Analysis using Pandas and Explore a Sample Dataset with it

Title: Exploratory Data Analysis (EDA) with Pandas

Aim: To perform basic exploratory data analysis (EDA) on a sample dataset using the pandas library.

Procedure:

1. **Install pandas** (if not already installed): `pip install pandas`
2. **Load a sample dataset.** For this example, I'll use the Titanic dataset from a URL.
3. **Explore the dataset using pandas functions:**

   `head()`: View the first few rows.

   `info()`: Get information about the data types and null values.

   `describe()`: Calculate descriptive statistics.

   `value_counts()`: Count the occurrences of unique values in a column.

   `isnull().sum()`: Count the null values in each column

Source Code:

```
import pandas as pd

# Load the Titanic dataset
titanic_url = "https://raw.githubusercontent.com/pandas-
dev/pandas/main/doc/data/titanic.csv"
titanic_df = pd.read_csv(titanic_url)

# Explore the dataset
print("First 5 rows of the Titanic dataset:")
print(titanic_df.head())

print("\nInformation about the Titanic dataset:")
print(titanic_df.info())

print("\nDescriptive statistics of the Titanic dataset:")
print(titanic_df.describe())

print("\nValue counts for the 'Survived' column:")
print(titanic_df['Survived'].value_counts())

print("\nNull values per column:")
print(titanic_df.isnull().sum())
```

Input: The Titanic dataset from the specified URL.

Expected Output:

```
First 5 rows of the Titanic dataset:
   PassengerId  Survived  Pclass  ...  Cabin Embarked  Unnamed: 0
0            1         0       3  ...    NaN        S         NaN
```

```
1            2         1       1  ...    C85        C         NaN
2            3         1       3  ...    NaN        S         NaN
3            4         1       1  ...   C123        S         NaN
4            5         0       3  ...    NaN        S         NaN

[5 rows x 13 columns]

Information about the Titanic dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 columns):
#   Column       Non-Null Count  Dtype
---  ------       --------------  -----
0   PassengerId  891 non-null    int64
1   Survived     891 non-null    int64
2   Pclass       891 non-null    int64
3   Name         891 non-null    object
4   Sex          891 non-null    object
5   Age          714 non-null    float64
6   SibSp        891 non-null    int64
7   Parch        891 non-null    int64
8   Ticket       891 non-null    object
9   Fare         891 non-null    float64
10  Cabin        204 non-null    object
11  Embarked     889 non-null    object
12  Unnamed: 0   0 non-null    float64
dtypes: float64(3), int64(5), object(5)
memory usage: 90.6+ KB

Descriptive statistics of the Titanic dataset:
       PassengerId    Survived      Pclass  ...        Fare  Unnamed: 0
count   891.000000  891.000000  891.000000  ...  891.000000         0.0
mean    446.000000    0.383838    2.308642  ...   32.204208         NaN
std     256.963452    0.486592    0.836071  ...   49.693429         NaN
min       1.000000    0.000000    1.000000  ...    0.000000         NaN
25%     223.500000    0.000000    2.000000  ...    7.910400         NaN
50%     446.000000    0.000000    3.000000  ...   14.454200         NaN
75%     668.500000    1.000000    3.000000  ...   31.000000         NaN
max     891.000000    1.000000    3.000000  ...  512.329200         NaN

[8 rows x 8 columns]

Value counts for the 'Survived' column:
0    549
1    342
Name: Survived, dtype: int64

Null values per column:
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
Unnamed: 0     891
dtype: int64
```

Lab 10: Install, Import Scikit Learn and Explore Iris Dataset with Pandas for ML Modelling

Title: Exploring Iris Dataset with Pandas for ML Modeling

Aim: To install scikit-learn, import it, and explore the Iris dataset using pandas in preparation for machine learning modeling.

Procedure:

1. **Install scikit-learn** (if not installed): `pip install scikit-learn`
2. **Import necessary libraries:** `pandas` and `sklearn.datasets`.
3. **Load the Iris dataset** using `load_iris()`.
4. **Convert the Iris data to a pandas DataFrame.**
5. **Explore the dataset:**

   Print the first few rows using `head()`.

   Print the column names.

   Print the shape of the DataFrame.

   Get descriptive statistics using `describe()`.

   Print the target variable distribution

Source Code:

```
import pandas as pd
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()
iris_df = pd.DataFrame(data=iris['data'], columns=iris['feature_names'])
iris_df['target'] = iris['target']  # Add the target variable

# Explore the dataset
print("First 5 rows of the Iris dataset:")
print(iris_df.head())

print("\nColumn names of the Iris dataset:")
print(iris_df.columns)

print("\nShape of the Iris dataset:")
print(iris_df.shape)

print("\nDescriptive statistics of the Iris dataset:")
print(iris_df.describe())

print("\nTarget variable distribution")
print(iris_df['target'].value_counts())
```

Input: The Iris dataset.

Expected Output:

```
First 5 rows of the Iris dataset:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  
target
0                5.1               3.5                1.4               0.2       0
1                4.9               3.0                1.4               0.2       0
2                4.7               3.2                1.3               0.2       0
3                4.6               3.1                1.5               0.2       0
4                5.0               3.6                1.4               0.2       0

Column names of the Iris dataset:
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)', 'target'],
      dtype='object')

Shape of the Iris dataset:
(150, 5)

Descriptive statistics of the Iris dataset:
       sepal length (cm)  sepal width (cm)  petal length (cm)  petal width 
(cm)  target
count         150.000000        150.000000         150.000000        
150.000000   150.0
mean            5.843333          3.057333           3.758000        
1.199333     1.0
std             0.828066          0.435866           1.765298        
0.762238     0.8
min             4.300000          2.000000           1.000000        
0.100000     0.0
25%             5.100000          2.800000           1.600000        
0.300000     0.0
50%             5.800000          3.000000           4.350000        
1.300000     1.0
75%             6.400000          3.300000           5.100000        
1.800000     2.0
max             7.900000          4.400000           6.900000        
2.500000     2.0

Target variable distribution
0    50
1    50
2    50
Name: target, dtype: int64
```

Lab 11: Explore all the Data Visualization Graphs and Find the outliers using plot.

Title: Data Visualization and Outlier Detection

Aim: To explore various data visualization graphs and identify outliers using plots. *(Note: This lab is quite broad. I'll focus on common techniques: box plots, scatter plots, and histograms. You can expand on this.)*

Procedure:

1. Import necessary libraries: `matplotlib.pyplot`, `pandas`, and `seaborn`.
2. Load a dataset. I'll use the `mtcars` dataset.
3. **Box plots:** Create box plots to visualize the distribution of numerical variables and identify potential outliers.
4. **Scatter plots:** Create scatter plots to visualize the relationship between two numerical variables and identify outliers in the relationship.
5. **Histograms:** Create histograms to visualize the frequency distribution of a single numerical variable.

Source Code:

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

# Load the mtcars dataset
mtcars_url =
"https://gist.githubusercontent.com/seankross/a412dfb88889b9e084b6/raw/0ef46f
ec8e511482d381e1e608b70e7d06d6b573/mtcars.csv"
mtcars_df = pd.read_csv(mtcars_url)
mtcars_df = mtcars_df.set_index('model')  # set model name as index

# 1. Box plots for outlier detection
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.boxplot(y=mtcars_df['mpg'])
plt.title('Box Plot of MPG')

plt.subplot(1, 2, 2)
sns.boxplot(y=mtcars_df['hp'])
plt.title('Box Plot of HP')
plt.show()

# 2. Scatter plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x=mtcars_df['mpg'], y=mtcars_df['hp'])
plt.title('Scatter Plot of MPG vs HP')
plt.show()

# 3. Histogram
plt.figure(figsize=(8, 6))
sns.histplot(mtcars_df['mpg'], kde=True)
plt.title('Histogram of MPG')
plt.show()
```

Input: The `mtcars` dataset.

Expected Output:

6. Box plots: Show the distribution of `mpg` and `hp`, highlighting any outliers as points outside the whiskers.
7. Scatter plot: Visualizes the relationship between `mpg` and `hp`, potentially showing outliers as points far from the general trend.
8. Histogram: Shows the frequency distribution of `mpg`.

Lab 12: Find the data distributions using box and Scatter plot.

Title: Data Distributions with Box and Scatter Plots

Aim: To visualize data distributions using box plots and scatter plots.

Procedure:

1. Import the necessary libraries: `matplotlib.pyplot`, `pandas`, and `seaborn`.
2. Load a dataset. I'll use the `mtcars` dataset.
3. Create box plots to visualize the distribution of individual numerical variables.
4. Create scatter plots to visualize the joint distribution of pairs of numerical variables.

Source Code:

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

# Load the mtcars dataset
mtcars_url =
"https://gist.githubusercontent.com/seankross/a412dfb88889b9e084b6/raw/0ef46f
ec8e511482d381e1e608b70e7d06d6b573/mtcars.csv"
mtcars_df = pd.read_csv(mtcars_url)
mtcars_df = mtcars_df.set_index('model')  # set model name as index

# 1. Box plots for individual distributions
plt.figure(figsize=(15, 8))
plt.subplot(2, 2, 1)
sns.boxplot(y=mtcars_df['mpg'])
plt.title('MPG Distribution')

plt.subplot(2, 2, 2)
sns.boxplot(y=mtcars_df['hp'])
plt.title('HP Distribution')

plt.subplot(2, 2, 3)
sns.boxplot(y=mtcars_df['wt'])
plt.title('Weight Distribution')

plt.subplot(2,2,4)
sns.boxplot(y=mtcars_df['qsec'])
plt.title('QSEC Distribution')
plt.show()


# 2. Scatter plots for joint distributions
plt.figure(figsize=(10, 8))
sns.scatterplot(x=mtcars_df['mpg'], y=mtcars_df['hp'])
plt.title('MPG vs HP')
plt.show()

plt.figure(figsize=(10,8))
sns.scatterplot(x=mtcars_df['wt'], y=mtcars_df['disp'])
plt.title('Weight vs Displacement')
plt.show()
```

Input: The `mtcars` dataset.

Expected Output:

5. Box plots: A series of box plots showing the distribution of `mpg`, `hp`, `wt`, and `qsec`.
6. Scatter plots: Two scatter plots, one showing the relationship between `mpg` and `hp`, and another showing the relationship between `wt` and `disp`.

Lab 13: Plot the histogram, bar chart and pie chart on sample data

Title: Basic Plots: Histogram, Bar Chart, and Pie Chart

Aim: To create and visualize data using histograms, bar charts, and pie charts.

Procedure:

1. Import the necessary libraries: `matplotlib.pyplot` and `pandas`.
2. Create a sample dataset. I'll create a simple dataset of sales data.
3. **Histogram:** Create a histogram to visualize the distribution of a numerical variable (e.g., sales amount).
4. **Bar chart:** Create a bar chart to visualize the count of a categorical variable (e.g., sales by region).
5. **Pie chart:** Create a pie chart to visualize the proportion of a categorical variable (e.g., sales by region as a percentage).

Source Code:

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

# Create a sample dataset
data = {'Region': ['North', 'South', 'East', 'West', 'North', 'South',
'East', 'West', 'North', 'South'],
        'Sales': [150, 200, 180, 220, 170, 210, 190, 230, 160, 240],
        'Category': ['A', 'B', 'A', 'C', 'B', 'A', 'C', 'B', 'A', 'C']}
df = pd.DataFrame(data)

# 1. Histogram
plt.figure(figsize=(8, 6))
plt.hist(df['Sales'], bins=5, edgecolor='black')  # Adjust bins as needed
plt.title('Sales Amount Distribution')
plt.xlabel('Sales Amount')
plt.ylabel('Frequency')
plt.show()

# 2. Bar chart
region_sales = df.groupby('Region')['Sales'].sum().reset_index()
plt.figure(figsize=(8, 6))
plt.bar(region_sales['Region'], region_sales['Sales'])
plt.title('Total Sales by Region')
plt.xlabel('Region')
plt.ylabel('Total Sales')
plt.show()

# 3. Pie chart
category_counts = df['Category'].value_counts()
plt.figure(figsize=(8, 6))
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%')
plt.title('Category Distribution')
plt.show()
```

Input: The sample dataset created in the code.

Expected Output:

6. Histogram: Shows the distribution of the `Sales` variable.

7. Bar chart: Shows the total sales for each `Region`.
8. Pie chart: Shows the proportion of each `Category`.

Lab 14: Install, Import Matplotlib. Explore all the Data Visualization Graphs and write a Python

Title: Data Visualization with Matplotlib

Aim: To install, import Matplotlib, and explore various data visualization graphs using it. *(Note: This is similar to Lab 11, but focusing specifically on Matplotlib. I'll provide a range of common Matplotlib plots.)*

Procedure:

1. **Install Matplotlib** (if not installed): `pip install matplotlib`
2. **Import Matplotlib:** `import matplotlib.pyplot as plt`
3. Load a dataset. I'll use the `mtcars` dataset.
4. Create various Matplotlib plots:

   Scatter plot: `plt.scatter()`

   Line plot: `plt.plot()`

   Bar chart: `plt.bar()`

   Histogram: `plt.hist()`

   Box plot: `plt.boxplot()`

Source Code:

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

# Load the mtcars dataset
mtcars_url =
"https://gist.githubusercontent.com/seankross/a412dfb88889b9e084b6/raw/0ef46f
ec8e511482d381e1e608b70e7d06d6b573/mtcars.csv"
mtcars_df = pd.read_csv(mtcars_url)
mtcars_df = mtcars_df.set_index('model')  # set model name as index

# 1. Scatter Plot
plt.figure(figsize=(8, 6))
plt.scatter(mtcars_df['mpg'], mtcars_df['hp'])
plt.title('MPG vs HP')
plt.xlabel('MPG')
plt.ylabel('HP')
plt.show()

# 2. Line Plot (example with a different data, or you could plot a series
from mtcars)
plt.figure(figsize=(8, 6))
years = [2015, 2016, 2017, 2018, 2019, 2020]
gdp = [12000, 12500, 13100, 14000, 14500, 15000]
plt.plot(years, gdp, marker='o')
plt.title('GDP over Time')
plt.xlabel('Year')
plt.ylabel('GDP (in $)')
plt.show()

# 3. Bar Chart
```

```
plt.figure(figsize=(8, 6))
cyl_counts = mtcars_df['cyl'].value_counts()
plt.bar(cyl_counts.index, cyl_counts.values)
plt.title('Car Count by Cylinder')
plt.xlabel('Cylinders')
plt.ylabel('Count')
plt.show()

# 4. Histogram
plt.figure(figsize=(8, 6))
plt.hist(mtcars_df['mpg'], bins=5, edgecolor='black')
plt.title('MPG Distribution')
plt.xlabel('MPG')
plt.ylabel('Frequency')
plt.show()

# 5. Box Plot
plt.figure(figsize=(8, 6))
plt.boxplot(mtcars_df['mpg'])
plt.title('MPG Box Plot')
plt.ylabel('MPG')
plt.show()
```

Input: The `mtcars` dataset.

Expected Output: A series of Matplotlib plots:

Scatter plot of MPG vs HP.

Line plot of GDP over time (using example data).

Bar chart of car counts by cylinder.

Histogram of MPG distribution.

Box plot of MPG.

Lab 15: Python program for customizing plot program for line chart

Title: Customizing Line Charts in Python

Aim: To write a Python program to create and customize line charts using Matplotlib.

Procedure:

1. Import the `matplotlib.pyplot` library.
2. Create sample data for the line chart (x and y values).
3. Create a basic line chart using `plt.plot()`.
4. Customize the chart:

   Add a title using `plt.title()`.

   Add x-axis and y-axis labels using `plt.xlabel()` and `plt.ylabel()`.

   Change the line color, style, and marker using arguments in `plt.plot()`.

   Add a legend using `plt.legend()`.

   Add gridlines using `plt.grid()`.

   Adjust the x-axis and y-axis limits using `plt.xlim()` and `plt.ylim()`.

5. Display the customized chart using `plt.show()`.

Source Code:

```
import matplotlib.pyplot as plt

# Sample data
years = [2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020]
production_a = [20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70]
production_b = [10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60]

# Create and customize the line chart
plt.figure(figsize=(12, 6))  # Adjust figure size
plt.plot(years, production_a, color='blue', linestyle='-', marker='o',
label='Product A')
plt.plot(years, production_b, color='green', linestyle='--', marker='s',
label='Product B')

plt.title('Production of Products A and B Over Time', fontsize=16)  # Set
title and font size
plt.xlabel('Year', fontsize=12)
plt.ylabel('Production (Units)', fontsize=12)
plt.legend(fontsize=10)
plt.grid(True, linestyle='--', alpha=0.7)  # Add gridlines
plt.xlim(2009, 2021)  # Set x-axis limits
plt.ylim(0, 80)  # Set y-axis limits

plt.xticks(years)  # Show all year ticks
plt.yticks(range(0, 81, 10)) # Show y-axis ticks

plt.show()
```

Input: The sample data for years and production of two products.

Expected Output: A line chart showing the production of Product A and Product B over the years 2010-2020, with customizations including:

6. Different colored lines and markers for each product.
7. A title, x-axis label, and y-axis label.
8. A legend to distinguish the products.
9. Gridlines.
10. Adjusted x-axis and y-axis limits.