16.

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.metrics import classification\_report, accuracy\_score

import matplotlib.pyplot as plt

# Load the dataset

file\_path = 'Admission\_Predict.csv'  # Replace with your file path if needed

admission\_data = pd.read\_csv(file\_path)

# Preprocess the data

# Drop 'Serial No.' as it is not relevant for prediction

admission\_data.drop(columns=["Serial No."], inplace=True)

# Convert the target variable into binary (0 or 1)

threshold = 0.75

admission\_data['Admit'] = (admission\_data['Chance of Admit '] >= threshold).astype(int)

admission\_data.drop(columns=["Chance of Admit "], inplace=True)

# Features and target

X = admission\_data.drop(columns=["Admit"])

y = admission\_data['Admit']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

# Train a Decision Tree Classifier

clf = DecisionTreeClassifier(random\_state=42)

clf.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = clf.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred)

print("Accuracy:", accuracy)

print("Classification Report:\n", report)

# Visualize the decision tree

plt.figure(figsize=(8, 10))

plot\_tree(clf,

          feature\_names=X.columns,

          class\_names=["Not Admitted", "Admitted"],

          filled=True,

          rounded=True,

          fontsize=10)

plt.title("Decision Tree Visualization")

plt.show()

15.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Create a synthetic dataset

df = pd.read\_csv('housing.csv')

print(df.columns)

print(df.iloc[:2])

# Features and target variable

X = df[['longitude']]

y = df['latitude']

# Split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train the model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Make predictions

y\_pred = model.predict(X\_test)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f"Model Coefficient(Slope): {model.coef\_[0]}")

print(f"Model Intercept : {model.intercept\_}")

print(f'Mean Squared Error: {mse}')

print(f'R² Score: {r2}')

# Visualize the results

plt.scatter(X, y, color='blue', label='Actual Prices')

plt.plot(X\_test, y\_pred, color='red', linewidth=2, label='Predicted Prices')

plt.title('longitude vs latitude')

plt.xlabel('Size (sq ft)')

plt.ylabel('Size (sq ft)')

plt.legend()

plt.show()

14.

# Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import matplotlib.pyplot as plt

import seaborn as sns

# Step 1: Load the Heart dataset

file\_path = 'heart.csv'  # Replace with your file path

data = pd.read\_csv(file\_path)

# Step 2: Separate features and target variable

X = data.drop(columns=["target"])  # Features

y = data["target"]  # Target

# Step 3: Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.2, random\_state=42)

# Step 4: Create and train the logistic regression model

model = LogisticRegression(max\_iter=2000, random\_state=42)

model.fit(X\_train, y\_train)

# Step 5: Make predictions

y\_pred = model.predict(X\_test)

# Step 6: Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

# Print the results

print(f"Accuracy: {accuracy \* 100:.2f}%")

print("Confusion Matrix:")

print(conf\_matrix)

print("Classification Report:")

print(class\_report)

# Step 7: Visualize the confusion matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=["No Disease", "Disease"],

            yticklabels=["No Disease", "Disease"]

           )

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.title('Confusion Matrix')

plt.show()

13.

import numpy as np

from scipy import stats

def hypothesis\_test(sample,population\_mean,alpha=0.05):

    sample\_mean=np.mean(sample)

    sample\_std=np.std(sample)

    #calculate length of sample

    n=len(sample)

    #calculate t-statistic

    t\_stat=(sample\_mean-population\_mean)/(sample\_std/np.sqrt(n))

    #degree of freedom

    df=n-1

    #calculate the critical t-value for two-tailed test

    t\_critical=stats.t.ppf(1-alpha/2,df)

    #calculate the p-value

    p\_value=2\*(1-stats.t.cdf(abs(t\_stat),df))

    #results

    print(f"Sample Mean :{sample\_mean:.4f}")

    print(f"T-statistic:{t\_stat:.4f}")

    print(f"Critical T-value:{t\_critical:.4f}")

    print(f"P-value:{p\_value:.4f}")

    #check if we reject or fail to reject the null hypothesis

    if abs(t\_stat)>t\_critical:

        print(f"Reject the null hypothesis at alpha={alpha}")

    else:

        print(f"fail to reject the null hypothesis at alpha={alpha}")

sample\_data=[56,57,59,58,60,62,63,61,58,59]

population\_mean=55

hypothesis\_test(sample\_data,population\_mean)

12.

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Step 1: Load the dataset

df = pd.read\_csv("iris.csv")

# Step 2: Inspect the dataset

print(df.head())

print(df.info())  # Check for missing values and data types

# Step 3: Clean the dataset

# Drop rows with missing values (if any)

df = df.dropna()

# Exclude non-numeric columns (e.g., 'species')

numeric\_df = df.select\_dtypes(include=['float64', 'int64'])

# Step 4: Calculate the correlation matrix

correlation\_matrix = numeric\_df.corr()

print("Correleation matrix" ,correlation\_matrix)

# Step 5: Visualize the correlation matrix using a heatmap

plt.figure(figsize=(8, 6))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f")

plt.title("Correlation Matrix Heatmap for Iris Dataset")

plt.show()

11.

#  11. For the dataset \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, Use Seaborn to create a histogram, box plot, and

#  heatmap.

#  a. Orange\_Telecom\_Churn\_Data.csv

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import  numpy as np

# Load the dataset

df = pd.read\_csv("archive (2).zip")

# Display the first few rows to understand the data

print(df.head())

print(df.columns)

# Histogram: Visualizing the distribution of a numeric column, e.g., 'total day minutes'

plt.figure(figsize=(8, 5))

sns.histplot(df['total\_day\_minutes'], kde=True, bins=30, color='blue')

plt.title('Histogram of Total Day Minutes')

plt.xlabel('Total Day Minutes')

plt.ylabel('Frequency')

plt.show()

# Box Plot: Checking for outliers in a numeric column, e.g., 'total day minutes'

plt.figure(figsize=(8, 5))

sns.boxplot(df['total\_day\_minutes'], color='orange')

plt.title('Box Plot of Total Day Minutes')

plt.xlabel('Total Day Minutes')

plt.show()

# Heatmap: Visualizing the correlation between numeric columns

# Compute the correlation matrix

# Select only numeric columns for the correlation matrix

numeric\_df = df.select\_dtypes(include=[np.number])

# Create a heatmap of correlations between numeric features

plt.figure(figsize=(10, 10))

correlation\_matrix = numeric\_df.corr()

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')

plt.title('Heatmap of Feature Correlations')

plt.show()

10.

# Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Step 1: Load the Dataset

# Replace 'iris.csv' with the actual file path

data = pd.read\_csv('iris.csv')

# Step 2: Display the first few rows and basic info

print("First 5 rows of the dataset:")

print(data.head())

print("\nDataset Info:")

print(data.info())

# Step 3: Check for missing values and clean data

print("\nMissing Values in Each Column:")

print(data.isnull().sum())

# Drop rows with missing values (if any)

data.dropna(inplace=True)

# Step 4: Perform Basic Exploratory Data Analysis (EDA)

print("\nSummary Statistics:")

print(data.describe())

# Value counts for a categorical column (if applicable, e.g., species)

if 'species' in data.columns:

    print("\nSpecies Value Counts:")

    print(data['species'].value\_counts())

# Step 5: Visualize Findings

# 5.1 Histogram: Distribution of Sepal Length

plt.figure(figsize=(8, 5))

plt.hist(data['sepal.length'], bins=10, color='blue', edgecolor='black')

plt.title('Distribution of Sepal Length')

plt.xlabel('Sepal Length (cm)')

plt.ylabel('Frequency')

plt.show()

# 5.2 Scatter Plot: Sepal Length vs Sepal Width

plt.figure(figsize=(8, 5))

plt.scatter(data['sepal.length'], data['sepal.width'], color='green')

plt.title('Sepal Length vs. Sepal Width')

plt.xlabel('Sepal Length (cm)')

plt.ylabel('Sepal Width (cm)')

plt.show()

# 5.3 Bar Chart: Species Distribution

species\_counts = data['species'].value\_counts()

plt.figure(figsize=(8, 5))

plt.bar(species\_counts.index, species\_counts.values, color='orange', edgecolor='black')

plt.title('Species Distribution')

plt.xlabel('Species')

plt.ylabel('Count')

plt.show()

# 5.4 Correlation Heatmap

# Select only numeric columns for the correlation matrix

numeric\_data = data.select\_dtypes(include=['number'])

# Correlation matrix

print("\nCorrelation Matrix:")

print(numeric\_data.corr())

# Heatmap for numeric data

plt.figure(figsize=(8, 6))

sns.heatmap(numeric\_data.corr(), annot=True, cmap='coolwarm', fmt='.2f')

plt.title('Feature Correlation Heatmap')

plt.show()

9.

import pandas as pd

import numpy as np

# Load the dataset

df = pd.read\_csv('heart.csv')

# Display the original DataFrame

print("Original DataFrame shape: ", df.shape)

print("Original DataFrame: \n", df.head(), "\n")

# Calculate mean and standard deviation for the 'age' column

mean\_age = np.mean(df['age'])

std\_age = np.std(df['age'])

# Calculate Z-scores manually

z\_score = (df['age'] - mean\_age) / std\_age

# Define the threshold for outliers

threshold = 2

# Filter rows where Z-score is less than or equal to the threshold

df\_cleaned = df[np.abs(z\_score) <= threshold]

# Display the cleaned DataFrame

print("Cleaned DataFrame shape: ", df\_cleaned.shape)

print("Cleaned DataFrame: \n", df\_cleaned.head())

8.

#Handling Missing Values:

# Create a DataFrame with some missing values.

# Use Python to handle missing values by dropping rows,

#filling with a specific value, and filling with the mean of the column.

# Display the DataFrame after each operation.

# Importing pandas as pd

import pandas as pd

# Importing numpy as np

import numpy as np

# Creating a DataFrame using the dictionary

df = pd.read\_csv('heart.csv')

# Display the original DataFrame

print("Original DataFrame:\n")

print(df)

print(df.isnull().sum())

# Using notnull() function to check for non-null values

not\_null\_df = df.notnull()

print("\nDataFrame showing non-null values:\n")

print(not\_null\_df)

# Dropping rows with missing values

df\_dropped = df.dropna()

print("\nDataFrame after dropping rows with missing values:\n")

print(df\_dropped)

# Filling missing values with a specific value for example 0

df\_filled\_zero = df.fillna(0)

print("\nDataFrame after filling missing values with 0:\n")

print(df\_filled\_zero)

# Filling missing values with the mean of each column

df\_filled\_mean = df.fillna(df.mean())

print("\nDataFrame after filling missing values with the mean of each column:\n")

print(df\_filled\_mean)

7.

import requests

import pandas as pd

# Fetch data from CoinGecko API

response = requests.get("https://api.coingecko.com/api/v3/coins/markets?vs\_currency=usd&order=market\_cap\_desc&per\_page=10&page=1")

# Parse JSON data from the API response

data = response.json()

# Convert the data into a DataFrame

df = pd.DataFrame(data)

# Operation 1: Display the first few rows of the DataFrame

print("First few rows of the DataFrame:\n", df.head(), "\n")

print(df.columns)

# Operation 2: Select and display the 'name' column

print("Cryptocurrency Names:\n", df['name'], "\n")

# Operation 3: Check for missing values in the DataFrame

print("Missing values in each column:\n", df.isnull().sum(), "\n")

# Operation 4: Filter cryptocurrencies with current price > 10000

filtered\_df = df[df['current\_price'] > 10000]

print("Cryptocurrencies with price > 10,000:\n", filtered\_df[['name', 'current\_price']], "\n")

# Operation 5: Get basic statistics of numerical columns

print("Basic statistics of numerical columns:\n", df.describe(), "\n")

6.

import requests

from bs4 import BeautifulSoup

import pandas as pd

# Define the URL of the Wikipedia page

url = 'https://en.wikipedia.org/wiki/List\_of\_countries\_by\_GDP\_(nominal)'

# Send a request to fetch the HTML content

response = requests.get(url)

soup = BeautifulSoup(response.content, 'html.parser')

# Find the table on the page

table = soup.find('table', {'class': 'wikitable'})

# Extract the header (column names)

headers = [th.get\_text(strip=True) for th in table.find\_all('th')]

print("Headers:", headers)

# Extract the rows

rows = []

for tr in table.find\_all('tr')[1:]:  # Skip the header row

    cells = [td.get\_text(strip=True) for td in tr.find\_all('td')]

    # Print each row to check if they match the number of columns

    if len(cells) == len(headers):

        rows.append(cells)

    else:

        print("Row does not match number of columns:", cells)

# Check if rows were successfully extracted

if not rows:

    print("No rows extracted!")

else:

    # Convert to DataFrame

    df = pd.DataFrame(rows, columns=headers)

    print("\nInitial DataFrame:\n", df.head())

    # Perform operations on the DataFrame

    # Rename Columns (adjust based on actual table structure)

    df.columns = ['Country', 'GDP IMF', 'GDP World Bank', 'GDP UN', 'Forecast', 'Year1', 'Estimate1', 'Year2', 'Estimate2', 'Year3']

    # 2. Handle Missing Data (drop rows with missing values)

    df\_cleaned = df.dropna()

    print("\nDataFrame after Dropping Rows with Missing Data:\n", df\_cleaned.head())

    # 3. Convert GDP column to numeric (assuming GDP columns contain commas)

    df\_cleaned['GDP IMF'] = pd.to\_numeric(df\_cleaned['GDP IMF'].str.replace(',', '', regex=False), errors='coerce')

    print("\nDataFrame with 'GDP IMF' Converted to Numeric:\n", df\_cleaned.head())

    # 4. Filter countries with GDP over a certain threshold (e.g., 1 trillion)

    filtered\_df = df\_cleaned[df\_cleaned['GDP IMF'] > 1000000000000]

    print("\nFiltered DataFrame (GDP > 1 Trillion):\n", filtered\_df.head())

    # 5. Sort countries by GDP in descending order

    sorted\_df = filtered\_df.sort\_values(by='GDP IMF', ascending=False)

    print("\nSorted DataFrame by GDP (Descending):\n", sorted\_df.head())

5.

import sqlite3

import pandas as pd

# Step 1: Set up SQLite database and create a table

# Connect to SQLite database

conn = sqlite3.connect("students\_database.db")

cursor = conn.cursor()

# Create a table for student information

cursor.execute("""

CREATE TABLE IF NOT EXISTS students (

    id INTEGER PRIMARY KEY,

    name TEXT NOT NULL,

    age INTEGER NOT NULL,

    grade TEXT NOT NULL,

    admission\_date TEXT NOT NULL,

    marks INTEGER NOT NULL

)

""")

student\_info= [

    ("Ananya Mishra", 20, "A", "2022-07-01", 85),

    ("Rohan Gupta", 22, "B", "2021-06-15", 78),

    ("Ishita Roy", 19, "A", "2023-01-20", 92),

    ("Kabir Khan", 21, "C", "2020-05-10", 65),

    ("Priya Sen", 20, "B", "2021-09-30", 74)]

# Insert sample data into the table

cursor.executemany("""

INSERT INTO students (name, age, grade, admission\_date, marks)

VALUES (?, ?, ?, ?, ?)

""",student\_info

)

# Commit changes and close the cursor

conn.commit()

# Step 2: Read data into a Pandas DataFrame

df = pd.read\_sql\_query("SELECT \* FROM students", conn)

print("Initial DataFrame:\n", df)

# Step 3: Perform 5 operations on the DataFrame

# 1. Filter students with marks greater than 80

high\_scorers = df[df['marks'] > 80]

print("\nStudents with marks > 80:\n", high\_scorers)

# 2. Calculate the average marks

avg\_marks = df['marks'].mean()

print("\nAverage marks:", avg\_marks)

# 3. Add a new column 'status' based on marks (>=50 is "Pass", otherwise "Fail")

df['status'] = ["Pass" if marks >= 50 else "Fail" for marks in df['marks']]

print("\nDataFrame with 'Pass/Fail' status:\n", df)

# 4. Count the number of students in each grade

students\_per\_grade = df['grade'].value\_counts()

print("\nNumber of students per grade:\n", students\_per\_grade)

# 5. Select only the names and marks of students

names\_and\_marks = df[['name', 'marks']]

print("\nNames and Marks of students:\n", names\_and\_marks)

# Step 4: Close the SQLite connection

conn.close()

4.

# Create a line plot and bar chart to visualize categorical data using Matplotlib. Use all the possible

#  operations on graphs.

import matplotlib.pyplot as plt

years=[2018,2019,2020,2021,2022,2023]

placed=[72,64,78,88,90,107]

plt.figure(figsize=(8,5))

plt.plot(years,placed,color='red',marker='\*',ls=':')

plt.xlabel("Years")

plt.ylabel("Placed Students")

plt.title("years vs Placed Students",fontsize=20,fontweight='bold')

plt.show()

plt.bar(years,placed,color='blue')

plt.xlabel("Years")

plt.ylabel("Placed Students")

plt.title("years vs Placed Students",fontsize=20,fontweight='bold')

plt.show()

3.

import pandas as pd

# 1. Create a DataFrame from a Dictionary of Lists

data = {

    'Name': ['Alice', 'Bob', 'Charlie', 'David'],

    'Age': [25, 30, 35, 40],

    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']

}

# Create DataFrame

df = pd.DataFrame(data)

# Display DataFrame

print("DataFrame:")

print(df)

# 2. Select a single column

names = df['Name']

print("\nNames Column:")

print(names)

# 3. Select multiple columns

name\_age = df[['Name', 'Age']]

print("\nName and Age Columns:")

print(name\_age)

# 4. Select a specific row

row = df.iloc[1]

print("\nRow at Index 1:")

print(row)

# 5. Select multiple rows

rows = df.iloc[1:3]

print("\nRows from Index 1 to 2:")

print(rows)

# 6. Filter rows based on a condition (Age > 30)

filtered\_data = df[df['Age'] > 30]

print("\nFiltered Data (Age > 30):")

print(filtered\_data)

# 7. Sort by 'Age' in ascending order

sorted\_by\_age = df.sort\_values(by='Age', ascending=True)

print("\nData Sorted by Age (Ascending):")

print(sorted\_by\_age)

# 8. Sort by multiple columns ('City' ascending, 'Age' descending)

sorted\_multi = df.sort\_values(by=['City', 'Age'], ascending=[True, False])

print("\nData Sorted by City and Age:")

print(sorted\_multi)

2.

def count\_words(filename):

    try:

        file=open("demo.txt","r")

        content=file.read()

        words=content.split()

        word\_count=len(words)

        return word\_count

    except FileNotFoundError:

        print(f"The file '{filename} was not found ")

        return None

filename='Demo.txt'

word\_count=count\_words(filename)

print(f"The number of words in '{filename}' is {word\_count}.")

1.

# Input string

text = input("Enter a string: ")

# Remove punctuation using a simple loop

punctuation = ".,!?;:'\"()[]{}<>-\_="

for char in punctuation:

    text = text.replace(char, " ")

# Convert to lowercase and split into words

words = text.lower().split()

# Dictionary to store word frequencies

word\_freq = {}

# Count frequencies

for word in words:

    if word in word\_freq:

        word\_freq[word] += 1

    else:

        word\_freq[word] = 1

# Display the result

print("Word Frequencies:")

for word, freq in word\_freq.items():

    print(f"{word}: {freq}")