EXPERIMENT-2

Aim of the experiment:

Theory:

Source Code:

squares = []

for i in range(1, 31):

squared = i \*\* 2

squares.append(squared)

print(squares[5:])

Output:

EXPERIMENT-3a

Source Code:

def reverse\_string\_for\_loop(string):

reversed\_string = ""

for i in range(len(string) - 1, -1, -1):

reversed\_string += string[i]

return reversed\_string

def reverse\_string\_while\_loop(string):

reversed\_string = ""

index = len(string) - 1

while index >= 0:

reversed\_string += string[index]

index -= 1

return reversed\_string

# Example usage: 1

original\_string = "Hello\_there"

reversed\_string\_for = reverse\_string\_for\_loop(original\_string)

reversed\_string\_while = reverse\_string\_while\_loop(original\_string)

print("Original String:", original\_string)

print("Reversed String (For Loop):", reversed\_string\_for)

print("Reversed String (While Loop):", reversed\_string\_while)

Output:

EXPERIMENT-3b

Source Code:

# Get input number from the user

num = int(input("Enter a number: "))

# Initialize a variable to store the sum of digits

sum\_of\_digits = 0

# While the number is greater than 0, repeat the following:

while num > 0:

# Extract the last digit of the number

digit = num % 10

# Add the extracted digit to the sum

sum\_of\_digits += digit

# Remove the last digit from the number

num = num // 10

# Print the calculated sum of digits

print("Sum of digits:", sum\_of\_digits)

EXPERIMENT-3c

Source Code:

# Get input number from the user

num = int(input("Enter a number: "))

# Initialize a variable to store the factorial

factorial = 1

# Calculate the factorial using a for loop

for i in range(1, num + 1):

factorial \*= i

# Print the calculated factorial

print("Factorial of", num, "is:", factorial)

Output:

EXPERIMENT-3d

Source Code:

# Get the number of terms from the user

num = int(input("Enter Number of terms: "))

# Initialize the first two Fibonacci numbers

num1 = 0

num2 = 1

# Print the first two Fibonacci numbers

print("Fibonacci Series: ", num1, num2, end=" ")

# Generate and print the remaining Fibonacci numbers

for i in range(2, num + 1):

# Calculate the next Fibonacci number

num3 = num1 + num2

# Print the next Fibonacci number

print(num3, end=" ")

# Update the first two numbers for the next iteration

num1 = num2

num2 = num3

Output:

EXPERIMENT-3e

Source Code:

# Get the number of rows from the user

n = int(input("Enter the number of rows: "))

# Print the pattern

for i in range(1, n + 1):

# Print leading spaces

print(" " \* (n - i), end="")

# Print asterisks

print("\* " \* i)

Output:

Exp\_4

Code

def max\_of\_three():

a = int(input("Enter the first number: "))

b = int(input("Enter the second number: "))

c = int(input("Enter the third number: "))

if a >= b and a >= c: 1

return a

elif b >= a and b >= c:

return b

else:

return c

# Example usage:

result = max\_of\_three()

print("The maximum number is:", result)

**exp\_5**

Source Code:

import random

# Define lists of story elements

characters = ["Caesar", "Roman", "Strawman"]

\_11\_plots = ["Once there used to be a king", "There was a strong man", "He used to rule the seas"]

\_12\_plots = ["Feared by all", "He was a great warrior", "He was a great leader"]

part\_story = ["He wanted to rule the land", "He had an ambition of great wealth in his mind", "He had a vision of great future"]

final\_part = ["So he pursued the plans he had set in his mind", "He was willing to do anything to get he what he sought", "So he aligned his forces to capture the land"]

# Generate the story

story = (

f"\nSo the story goes...\n\n"

f"+ {random.choice(\_11\_plots)} +\n"

f'"{random.choice(characters)}" +\n'

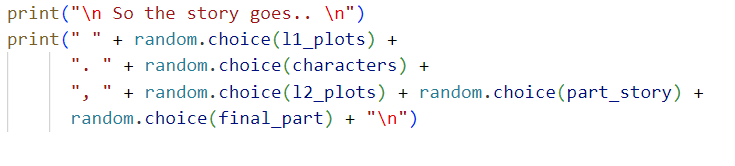
f'"{random.choice(part\_story)}" +\n'

f"{random.choice(final\_part)}\n"

)

# Print the story

print(story)



Output:

Exp-6

Source Code:

import pandas as pd

from faker import Faker

import random as rand

# Create a Faker object and set the seed

Fake = Faker()

Faker.seed(0)

# Set the random seed

rand.seed(0)

# Number of records to generate

num\_records = 100

# Initialize lists to store the data

roll\_no = []

name = []

age = []

gender = []

mpython = []

mjava = []

# Generate data

for i in range(num\_records):

roll\_no.append(Fake.unique.random\_int(0, 100))

name.append(Fake.unique.name())

age.append(Fake.random\_int(18, 22))

gender.append(Fake.random.choice(["Male", "Female"]))

mpython.append(Fake.random\_int(0, 100))

mjava.append(Fake.random\_int(0, 100))

# Create a DataFrame from the data

data = pd.DataFrame({

"Roll No": roll\_no,

"Name": name,

"Age": age,

"Gender": gender,

"Marks Python": mpython,

"Marks Java": mjava

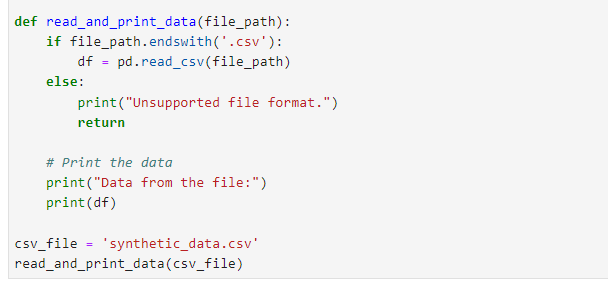
})

# Print the DataFrame

print(data)

# Save the DataFrame to a CSV file

data.to\_csv("Synthetic\_Data.csv")

Output:

EXPERIMENT - 7

Aim of the experiment:

Theory:

A>

import numpy as np

random\_array = np.random.randint(1, 51, size=(4, 5))

print("Random 2D array is:")

print(random\_array)

B>

import numpy as np

random\_array = np.random.randint(1, 51, size=(4, 5))

print("2D Array:")

print(random\_array)

total\_sum = np.sum(random\_array)

print(f"Sum of all elements: {total\_sum}")

C>

import numpy as np

random\_array = np.random.randint(1, 51, size=(4, 5))

print("2D Array:")

print(random\_array)

total\_sum = np.sum(random\_array)

print(f"Sum of all elements: {total\_sum}")

max\_value = np.max(random\_array)

print(f"Maximum value in the array: {max\_value}")

mean\_value = np.mean(random\_array)

print(f"Mean value of the array: {mean\_value}")

**D>**

import numpy as np

random\_array = np.random.randint(1, 51, size=(4, 5))

print("2D Array:")

print(random\_array)

mean\_value = np.mean(random\_array)

print(f"Mean of array elements: {mean\_value}")

**E>**

import numpy as np

random\_array = np.random.randint(1, 51, size=(4, 5))

print("2D Array:")

print(random\_array)

row\_sums = np.sum(random\_array, axis=1)

print("Sum of elements in each row:")

print(row\_sums)

**F>**

import numpy as np

random\_array = np.random.randint(1, 51, size=(4, 5))

print("Original 2D Array:")

print(random\_array)

transposed\_array = random\_array.T

print("Transposed 2D Array:")

print(transposed\_array)

G>

import numpy as np

random\_array = np.random.randint(1, 51, size=(4, 5))

print("2D Array:")

print(random\_array)

mask\_greater\_than\_25 = random\_array > 25

print("Boolean mask of elements > 25:")

print(mask\_greater\_than\_25)

filtered\_elements = random\_array[mask\_greater\_than\_25]

print("Elements greater than 25:")

print(filtered\_elements)

EXPERIMENT-8

**Aim of the experiment:** Perform Statistics and Data Visualization in python. Assume you have a .csv file containing 10 student details along with their marks in python, java and C language. Perform following operations on it.

* Print mean, standard deviation, minimum marks, maximum marks 1st quantile, 3rd quantile, maximum marks in each category.
* Plot a histogram plot for each subject.

**Code**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from faker import Faker

# Generate fake data

fake = Faker()

np.random.seed(42)

python\_marks = np.random.randint(60, 101, size=10)

names = [fake.name() for \_ in range(10)]

c\_marks = np.random.randint(60, 101, size=10)

java\_marks = np.random.randint(60, 101, size=10)

# Create a DataFrame

data = pd.DataFrame({

'Name': names,

'Python': python\_marks,

'C': c\_marks,

'Java': java\_marks

})

# Save the DataFrame to a CSV file

data.to\_csv('student\_marks.csv', index=False)

# Read the CSV file

data = pd.read\_csv('student\_marks.csv')

# Calculate statistics

statistics = {}

subjects = ['Python', 'Java', 'C']

for subject in subjects:

stats = {

'Mean': data[subject].mean(),

'Standard Deviation': data[subject].std(),

'Minimum Marks': data[subject].min(),

'1st Quantile': data[subject].quantile(0.25),

'Maximum Marks': data[subject].max(),

'3rd Quantile': data[subject].quantile(0.75)

}

statistics[subject] = stats

for subject, stats in statistics.items():

print(f"Statistics for {subject}:")

for stat, value in stats.items():

print(f"{stat}: {value}")

print()

# Create histograms

for subject in subjects:

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

plt.title(f'Histogram of {subject} Marks')

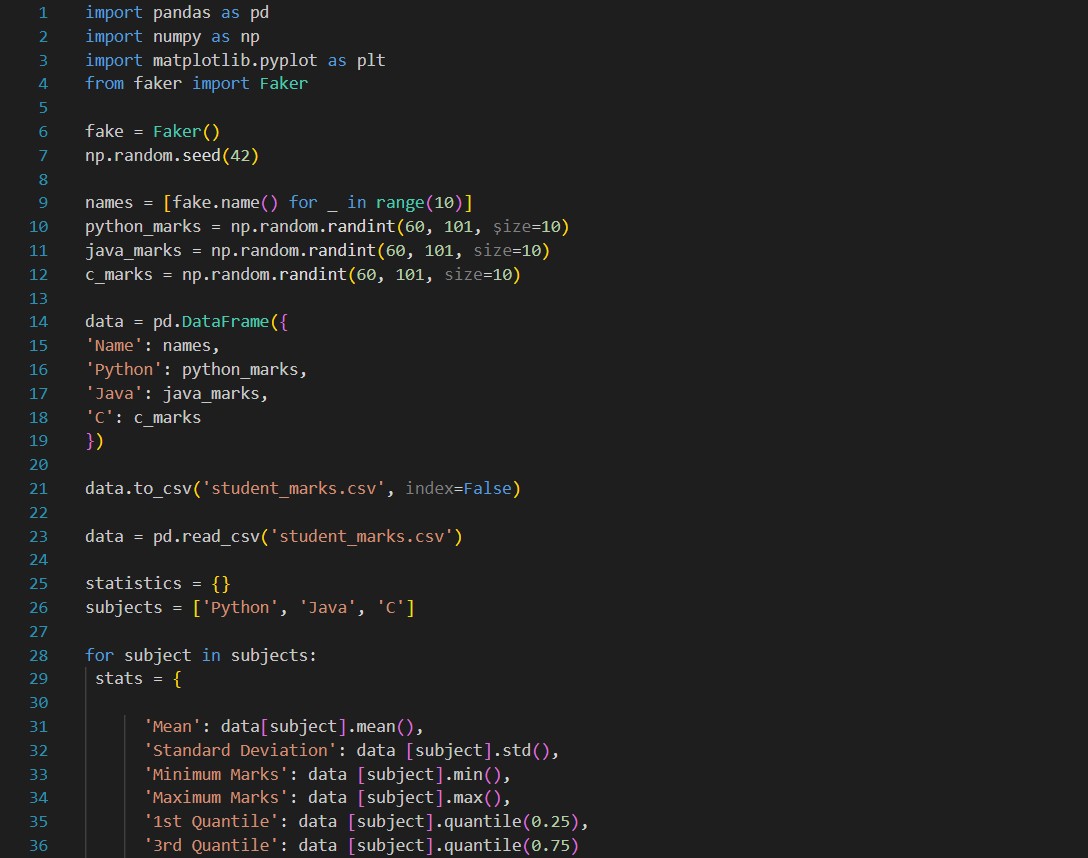
plt.hist(data[subject], bins=10, alpha=0.7, color='blue', edgecolor='black')

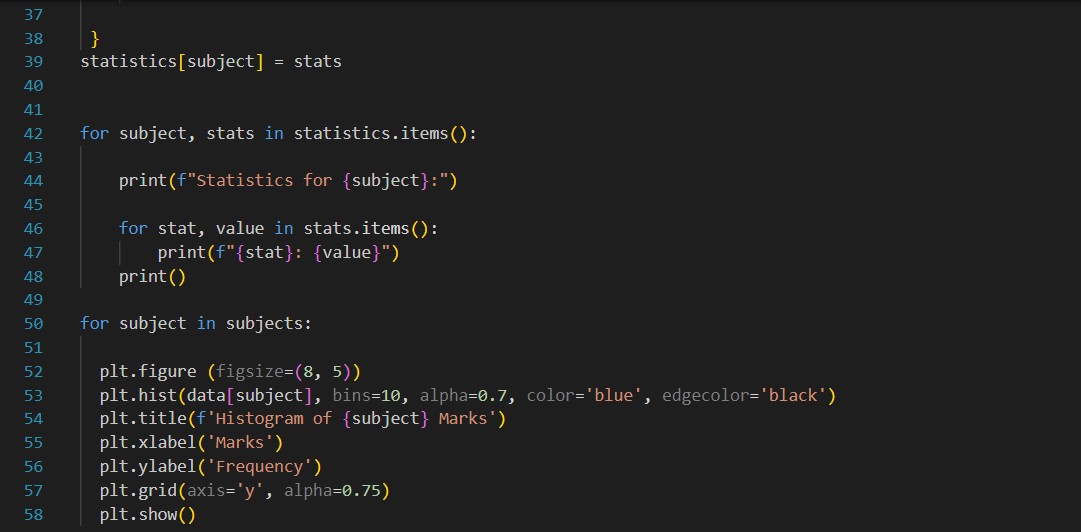
plt.xlabel('Marks')

plt.ylabel('Frequency')

plt.grid(axis='y', alpha=0.75)

plt.show()

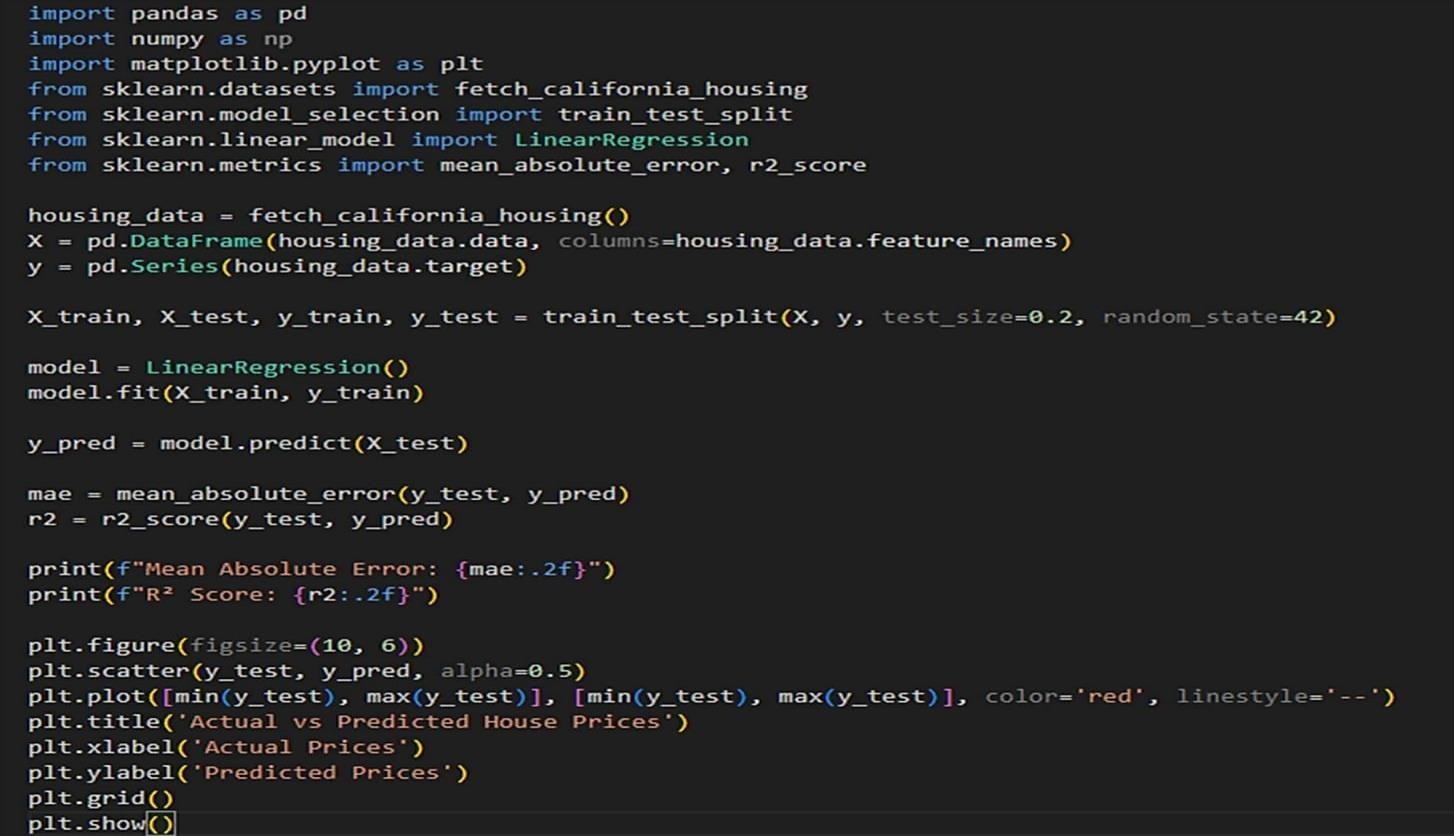




**EXPERIMENT NO – 9**

**Aim of the experiment:** Design a Python program to implement Linear Regression House price prediction using california\_housing from scikit-learn.

**Theory:**

****

**Source Code**

Code

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

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

from sklearn.linear\_model import LinearRegression

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

# Load California Housing Dataset

housing\_data = fetch\_california\_housing()

# Create DataFrames for features (X) and target (y)

X = pd.DataFrame(housing\_data.data, columns=housing\_data.feature\_names)

y = pd.Series(housing\_data.target)

# 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)

# Create a Linear Regression model

model = LinearRegression()

# Train the model on the training data

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

# Make predictions on the testing data

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

# Evaluate the model's performance

mae = mean\_absolute\_error(y\_test, y\_pred)

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

print(f"Mean Absolute Error: {mae:.2f}")

print(f"R² Score: {r2:.2f}")

# Visualize the actual vs. predicted house prices

plt.figure(figsize=(10, 6))

plt.scatter(y\_test, y\_pred, alpha=0.5)

plt.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], color='red', linestyle='--')

plt.title('Actual vs Predicted House Prices')

plt.xlabel('Actual Prices')

plt.ylabel('Predicted Prices')

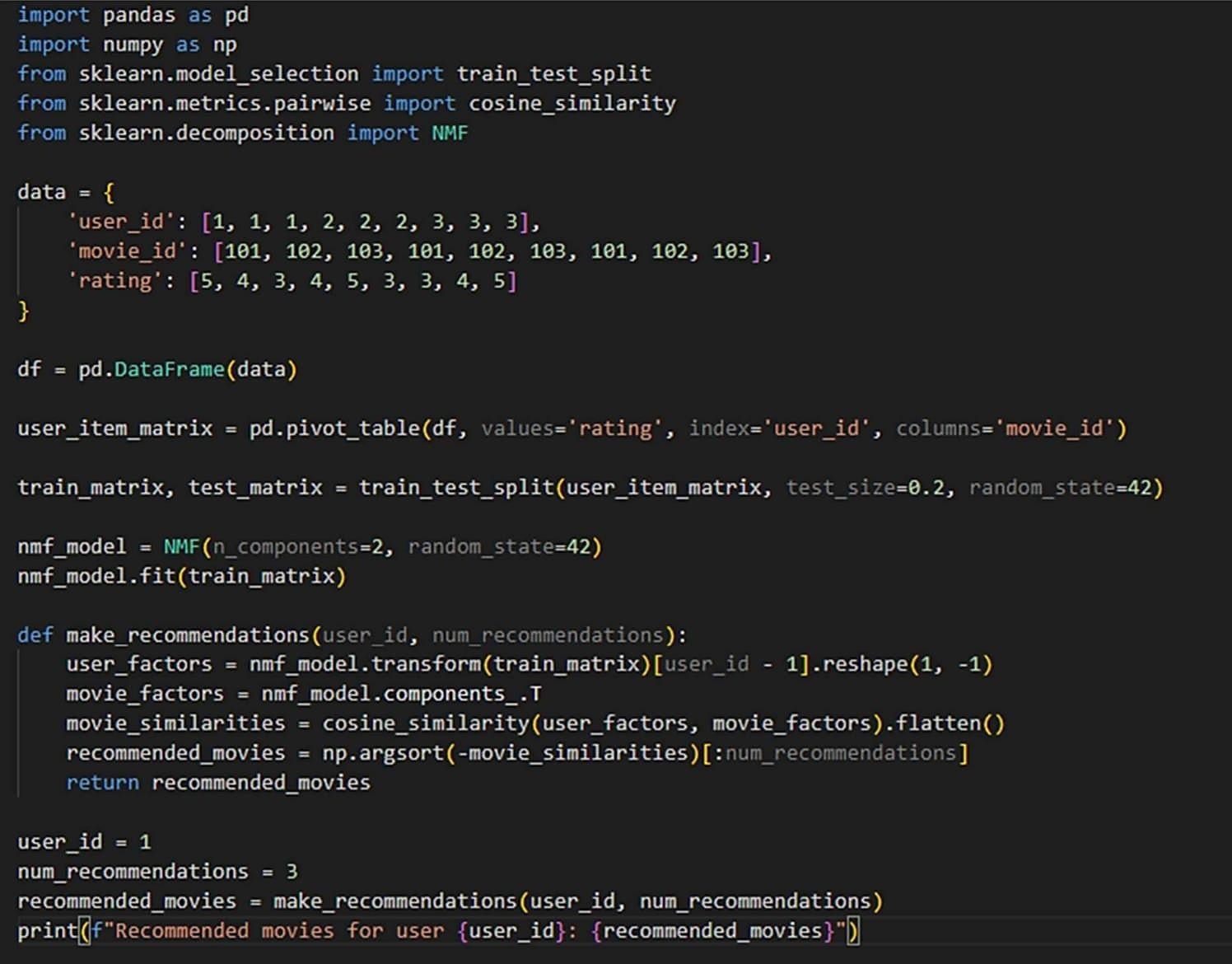
plt.grid()

plt.show()

**EXPERIMENT NO – 10**

**Aim of the experiment:** Design a Python program to create a recommender system.

**Theory:**

**Source Code:**

import pandas as pd

import numpy as np

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

from sklearn.metrics.pairwise import cosine\_similarity

from sklearn.decomposition import NMF

data = {

'user\_id': [1, 1, 1, 2, 2, 2, 3, 3, 3],

'movie\_id': [101, 102, 103, 101, 102, 103, 101, 102, 103],

'rating': [5, 4, 3, 4, 5, 3, 3, 4, 5]

}

df = pd.DataFrame(data)

user\_item\_matrix = pd.pivot\_table(df, values='rating', index='user\_id', columns='movie\_id')

train\_matrix, test\_matrix = train\_test\_split(user\_item\_matrix, test\_size=0.2, random\_state=42)

nmf\_model = NMF(n\_components=2, random\_state=42)

nmf\_model.fit(train\_matrix)

def make\_recommendations(user\_id, num\_recommendations):

user\_factors = nmf\_model.transform(train\_matrix)[user\_id - 1].reshape(1, -1)

movie\_factors = nmf\_model.components\_.T

movie\_similarities = cosine\_similarity(user\_factors, movie\_factors).flatten()

recommended\_movies = np.argsort(-movie\_similarities)[:num\_recommendations]

return recommended\_movies

user\_id = 1

num\_recommendations = 3

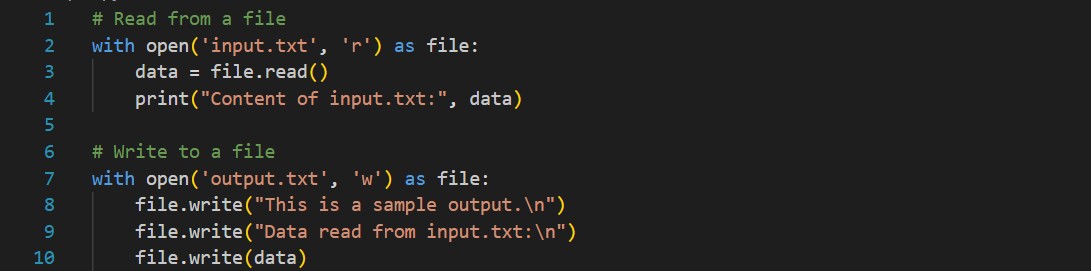
recommended\_movies = make\_recommendations(user\_id, num\_recommendations)

print(f"Recommended movies for user {user\_id}: {recommended\_movies}")

**EXPERIMENT NO – 11**

**Aim of the experiment:** Write a program in Python to read a text file and write a text file.

**Theory:**



**Code**

# Read from a file

with open('input.txt', 'r') as file:

data = file.read()

print("Content of input.txt:", data)

# Write to a file

with open('output.txt', 'w') as file:

file.write("This is a sample output.\n")

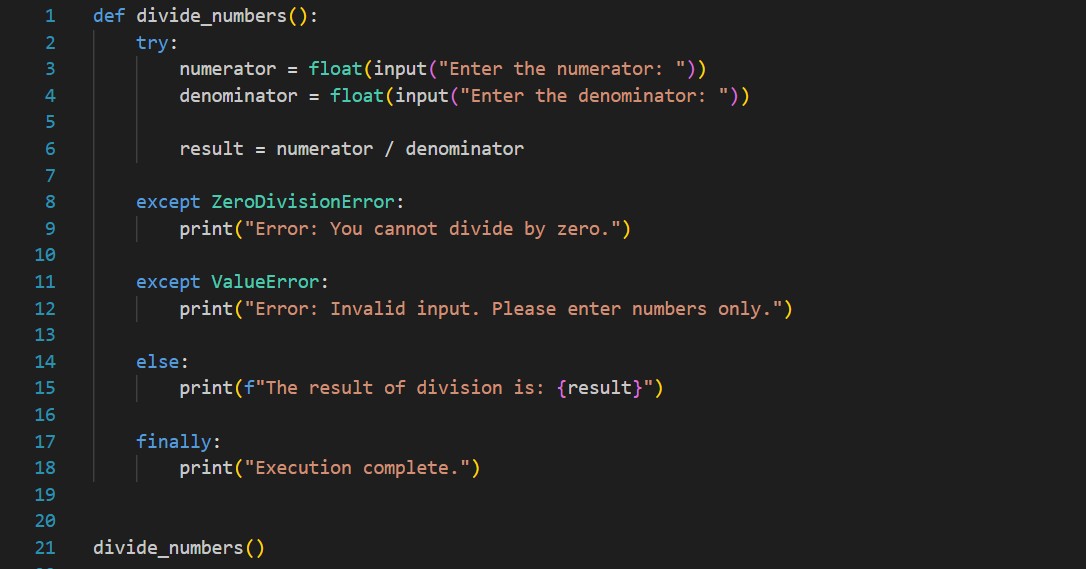
file.write("Data read from input.txt:\n")

file.write(data)

**EXPERIMENT NO – 12**

**Aim of the experiment:** Write a program in Python to implement exception handling.

**Theory:**



Code

def divide\_numbers():

try:

numerator = float(input("Enter the numerator: "))

denominator = float(input("Enter the denominator: "))

result = numerator 1 / denominator

print(f"The result of division is: {result}")

except ZeroDivisionError:

print("Error: You cannot divide by zero.")

except ValueError:

print("Error: Invalid input. Please enter numbers only.")

finally:

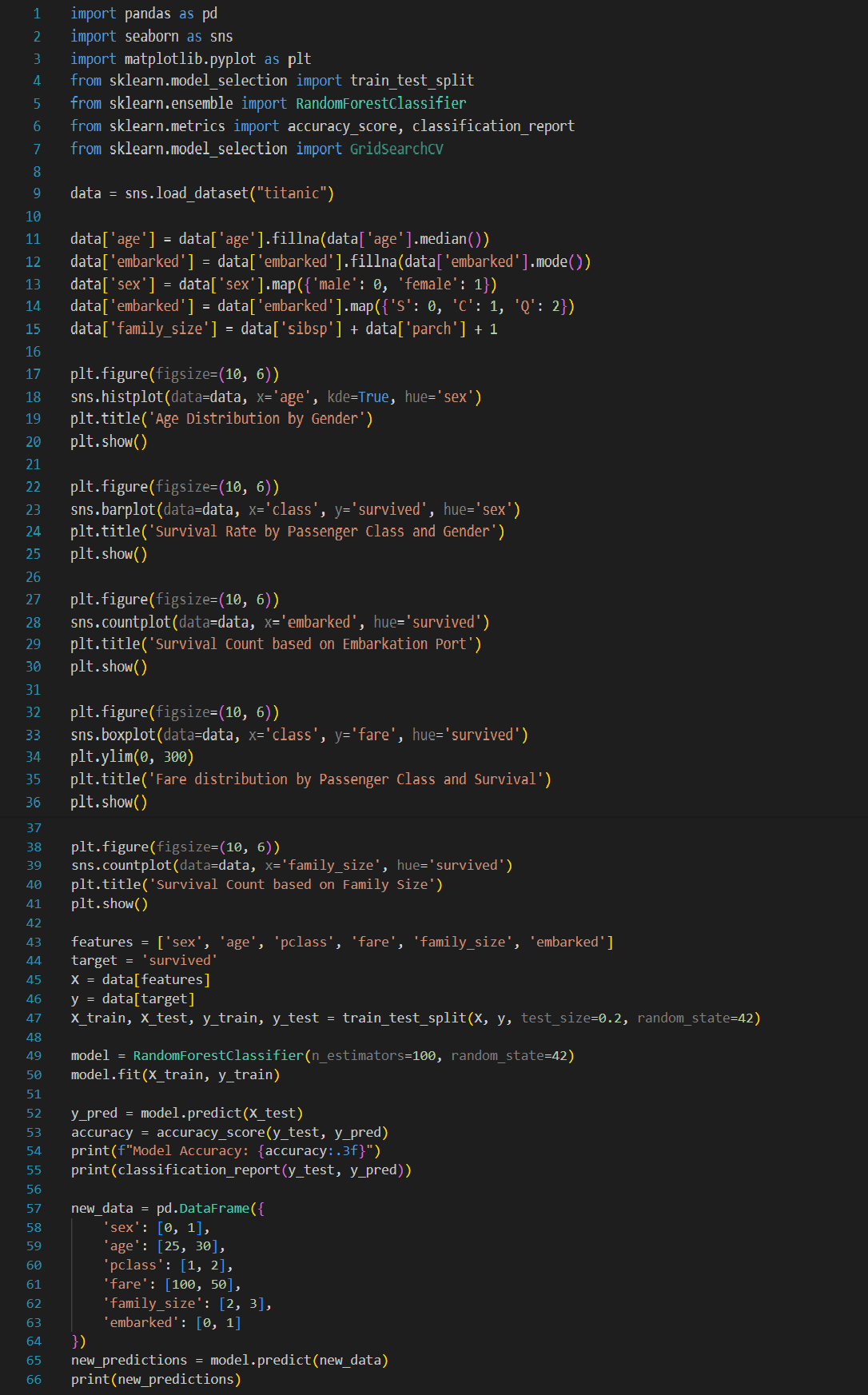
print("Execution complete.")

divide\_numbers()

**Experiment\_13**

**Aim of the experiment: Data Science Project: students can take any dataset of their choice (titanic / stock price prediction / credit card fraud detection, etc.) and show all the steps of the data science life cycle.**

**Theory:**



Code

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

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

from sklearn.ensemble import RandomForestClassifier

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

from sklearn.model\_selection import GridSearchCV

# Load Titanic dataset

data = sns.load\_dataset("titanic")

# Data preprocessing

data['age'] = data['age'].fillna(data['age'].median())

data['embarked'] = data['embarked'].fillna(data['embarked'].mode()[0])

data['sex'] = data['sex'].map({'male': 0, 'female': 1})

data['embarked'] = data['embarked'].map({'S': 0, 'C': 1, 'Q': 2})

data['family\_size'] = data['sibsp'] + data['parch'] + 1

# Exploratory Data Analysis (EDA)

# ... (Add your desired visualizations here)

# Feature selection and target variable

features = ['sex', 'age', 'pclass', 'fare', 'family\_size', 'embarked']

target = 'survived'

# Split data into training and testing sets

X = data[features]

y = data[target]

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

# Create and train a Random Forest Classifier

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

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

# Make predictions on the test set

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

# Evaluate the model

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

print("Model Accuracy:", accuracy)

print(classification\_report(y\_test, y\_pred))

# Example of making predictions on new data

new\_data = pd.DataFrame({

'sex': [0, 1],

'age': [25, 30],

'pclass': [1, 2],

'fare': [100, 50],

'family\_size': [2, 3],

'embarked': [0, 1]

})

new\_predictions = model.predict(new\_data)

print(new\_predictions)