DATA ENGINEERING

PYTHON ASSIGNMENT

1. Data Manipulation with Pandas:

1a. Given a dataset in the form of a dictionary, convert it to a DataFrame and perform

the following tasks:

import pandas as pd

data = {

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

'Age': [25, 30, None, 35, 40], # introducing missing value for demonstration

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

'Gender': ['F', 'M', 'M', 'M', 'F']

}

df = pd.DataFrame(data)

print("Initial DataFrame:")

print(df)

1. Remove rows with missing values.

Ans:

df\_cleaned = df.dropna()

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

print(df\_cleaned)

2. Group the data by 'Product' and calculate the total sales for each product.

Ans:

total\_sales = df.groupby('Product')['Sales'].sum().reset\_index()

print("\nTotal sales for each product:")

print(total\_sales)

3.Sort the results by total sales in descending order

Ans:

total\_sales\_sorted = total\_sales.sort\_values(by='Sales', ascending=False)

print("\nTotal sales sorted by total sales in descending order:")

print(total\_sales\_sorted)

1b.Create a pivot table that shows the sum of sales for each product, broken down

by month.

Ans:

import pandas as pd

data = {

'Product': ['A', 'B', 'A', 'B', 'A'],

'Sales': [100, 150, 200, 300, 250],

'Month': ['Jan', 'Feb', 'Jan', 'Feb', 'Mar']

}

df = pd.DataFrame(data)

print("Initial DataFrame:")

print(df)

pivot\_table = pd.pivot\_table(df, values='Sales', index='Product', columns='Month', aggfunc='sum')

print("\nPivot table showing sum of sales for each product, broken down by month:")

print(pivot\_table)

2. Data Cleaning:

2a.Write a function that takes a DataFrame with various types of data (numeric,

text, dates) and performs the following cleaning steps:

Ans:

import pandas as pd

import numpy as np

1.Replace all empty strings with NaN.

Ans:

def clean\_dataframe(df):

df.replace('', np.nan, inplace=True)

2.Fill numeric NaNs with the mean of their column.

Ans:

for col in df.select\_dtypes(include='number'):

df[col].fillna(df[col].mean(), inplace=True)

3.Convert all text to lowercase.

Ans:

df = df.apply(lambda x: x.str.lower() if x.dtype == 'object' else x)

return df

data = {

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

'Age': [25, 30, np.nan, 35, 40],

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

'Gender': ['F', 'M', 'M', 'M', 'F']

}

df = pd.DataFrame(data)

cleaned\_df = clean\_dataframe(df)

print("Cleaned DataFrame:")

print(cleaned\_df)

2b.Write a function that detects and removes outliers from a numeric column in a

DataFrame using the IQR method.

Ans:

import pandas as pd

def remove\_outliers(df, column):

Q1 = df[column].quantile(0.25)

Q3 = df[column].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

df\_cleaned = df[(df[column] >= lower\_bound) & (df[column] <= upper\_bound)]

return df\_cleaned

data = {

'Score': [85, 88, 84, 91, 70, 100, 99, 82, 81, 120, 79, 78]

}

df = pd.DataFrame(data)

cleaned\_df = remove\_outliers(df, 'Score')

print("Cleaned DataFrame after removing outliers:")

print(cleaned\_df)

3. Lambda Functions and Map-Reduce:

3a. Use a lambda function to ϐilter out even numbers from a list of integers.

Ans:

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

filtered = list(filter(lambda x: x % 2 != 0, numbers))

print("Original list:", numbers)

print("Filtered list (odd numbers only):", filtered)

3b.Use the reduce function to calculate the product of the remaining numbers.

Ans:

from functools import reduce

numbers = [1, 2, 3, 4, 5]

product = reduce(lambda x, y: x \* y, numbers)

print("List of numbers:", numbers)

print("Product of numbers:", product)

3c.Use a lambda function with the ϐilter function to remove words from a list that

are shorter than 4 characters. Then, use reduce to concatenate the remaining

words into a single string.

Ans:

from functools import reduce

words = ["mango", "banana", "pear", "kiwi", "orange"]

result\_string = reduce(lambda x, y: x + " " + y, filter(lambda word: len(word) >= 4, words))

print("List of words:", words)

print("Concatenated string:", result\_string)

data = {

'Date': ['2023-01-01','2023-01-02','2023-01-03', '2023-01-04','2023-01-05',

'2023-01-06','2023-01-07','2023-01-08','2023-01-09', '2023-01-10'] , 'Sales': [100, 120, 130, 150, 160, 180, 200, 210, 220, 230]

}df = pd.DataFrame(data)

#visualisation

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

plt.plot(df['Date'], df['Sales'], marker='o', color='g', label='Sales Trend')

4. Data Visualization:

4a.Using Matplotlib, create a line chart showing the trend of sales over time from

the given dataset.

Ans:

data = {

'Date': ['2023-01-01','2023-01-02','2023-01-03', '2023-01-04','2023-01-05',

'2023-01-06','2023-01-07','2023-01-08','2023-01-09', '2023-01-10'] , 'Sales': [100, 120, 130, 150, 160, 180, 200, 210, 220, 230]

}df = pd.DataFrame(data)

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

plt.plot(df['Date'], df['Sales'], marker='o', color='g', label='Sales Trend')



4b.Customize the chart with labels, title, and legend.

Ans:plt.title('Sales Trend Over Time')

plt.xlabel('Date')

plt.ylabel('Sales')

plt.legend()



4c.Create a scatter plot showing the relationship between two numerical columns

in a DataFrame. Add a trend line to the scatter plot.

Ans:

import numpy as np

np.random.seed(0)

x = np.random.randn(50)

y = 2 \* x + np.random.randn(50)

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

plt.scatter(x, y, color='g', label='Data Points')

m, b = np.polyfit(x, y, 1) plt.plot(x, m \* x + b, color='r', label='Trend Line')



5. Data Aggregation:

5a.Given a list of dictionaries representing transactions, write a function to

aggregate the total amount spent by each user.

Ans:from collections import defaultdict

def aggregate\_total\_spent(transactions):

# Create a defaultdict to store total amount spent by each user

total\_spent = defaultdict(float)

for transaction in transactions:

user = transaction['user']

amount\_spent = transaction['amount']

total\_spent[user] += amount\_spent

return total\_spent

transactions = [

{'user': 'Alice', 'amount': 100.0},

{'user': 'Bob', 'amount': 50.0},

{'user': 'Alice', 'amount': 200.0},

{'user': 'Charlie', 'amount': 75.0},

{'user': 'Bob', 'amount': 150.0},

{'user': 'Alice', 'amount': 50.0},

]

total\_spent = aggregate\_total\_spent(transactions)

for user, total\_amount in total\_spent.items():

print(f"User: {user}, Total Amount Spent: {total\_amount}")

5b.Write a function that calculates the moving average of the total amount spent by

each user over a speciϐied window size.

Ans:from collections import defaultdict, deque

def moving\_average\_total\_spent(transactions, window\_size):

user\_total = defaultdict(list)

moving\_avg = defaultdict(float)

for transaction in transactions:

user = transaction['user']

amount = transaction['amount']

user\_total[user].append(amount)

if len(user\_total[user]) > window\_size:

user\_total[user].pop(0)

total\_spent = sum(user\_total[user])

moving\_avg[user] = total\_spent / len(user\_total[user])

return dict(moving\_avg)

window\_size = 3

moving\_avg = moving\_average\_total\_spent(transactions, window\_size)

print("\nMoving average of total amount spent by each user:")

print(moving\_avg)

6. Exception Handling:

6a.Write a function that handles division by zero and returns a meaningful error

message when a division by zero occurs.

Ans:def safe\_divide(a, b):

try:

result = a / b

except ZeroDivisionError:

return "Error: Division by zero is not allowed."

else:

return result

print(safe\_divide(10, 2))

print(safe\_divide(5, 0))

6b.Write a function that takes a list of ϐile paths and attempts to open each one,

handling FileNotFoundError, PermissionError, and IOError, and logging the

results.

Ans:

import logging

def open\_files(file\_paths):

logging.basicConfig(filename='file\_opening.log', level=logging.INFO, format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d %H:%M:%S')

for file\_path in file\_paths:

try:

with open(file\_path, 'r') as file:

logging.info(f"Successfully opened file: {file\_path}")

except FileNotFoundError:

logging.error(f"FileNotFoundError: File not found: {file\_path}")

except PermissionError:

logging.error(f"PermissionError: Permission denied: {file\_path}")

except IOError:

logging.error(f"IOError: Unable to open file: {file\_path}")

except Exception as e:

logging.error(f"Unexpected error occurred while opening file {file\_path}: {str(e)}")

file\_paths = ['file1.txt', 'file2.txt', 'file3.txt', '/root/somefile.txt']

open\_files(file\_paths)

7. Working with Dates:

7a.Write a function that takes a list of date strings in various formats and converts

them to a standardized format (YYYY-MM-DD).

Ans:from datetime import datetime

def standardize\_date\_formats(date\_strings):

standardized\_dates = []

for date\_str in date\_strings:

try:

# Try different formats to parse the date string

date\_obj = datetime.strptime(date\_str, '%Y-%m-%d')

except ValueError:

try:

date\_obj = datetime.strptime(date\_str, '%m/%d/%Y')

except ValueError:

try:

date\_obj = datetime.strptime(date\_str, '%d-%b-%Y')

except ValueError:

try:

date\_obj = datetime.strptime(date\_str, '%Y-%b-%d')

except ValueError:

print(f"Skipping invalid date format: {date\_str}")

continue

standardized\_dates.append(date\_obj.strftime('%Y-%m-%d'))

return standardized\_dates

7b.Write a function that calculates the number of business days between two given

dates, excluding weekends and holidays.

Ans:

from datetime import datetime, timedelta

def is\_business\_day(date):

return date.weekday() < 5

def calculate\_business\_days(start\_date, end\_date, holidays=[]):

current\_date = start\_date

business\_days\_count = 0

while current\_date <= end\_date:

if is\_business\_day(current\_date) and current\_date not in holidays:

business\_days\_count += 1

current\_date += timedelta(days=1)

return business\_days\_count

date\_strings = ['2023-12-25', '12/31/2024', '15-Aug-2025', '2026-Sep-30', 'InvalidDate']

standardized\_dates = standardize\_date\_formats(date\_strings)

print("Standardized Dates:")

print(standardized\_dates)

8. ETL Process:

8a.Simulate an ETL process using Python that extracts data from a list of

dictionaries, transforms it by normalizing numeric ϐields, and loads it into a

Pandas DataFrame.

Ans:

import pandas as pd

data = [

{'id': 1, 'name': 'Alice', 'age': 25, 'salary': 5000},

{'id': 2, 'name': 'Bob', 'age': 30, 'salary': 7500},

{'id': 3, 'name': 'Charlie', 'age': 28, 'salary': 9000},

{'id': 4, 'name': 'David', 'age': 35, 'salary': 6000},

{'id': 5, 'name': 'Eve', 'age': 27, 'salary': 8000}

]

for entry in data:

entry['normalized\_salary'] = entry['salary'] / 100 # Transforming by normalizing salary (dividing by 100)

df = pd.DataFrame(data)

print("Transformed Data:")

print(df)

8b.Extend the ETL process to include a validation step that checks for data quality

issues (e.g., missing values, outliers) before loading the data into the DataFrame.

Ans:

import pandas as pd

data = [

{'id': 1, 'name': 'Alice', 'age': 25, 'salary': 5000},

{'id': 2, 'name': 'Bob', 'age': 30, 'salary': 7500},

{'id': 3, 'name': 'Charlie', 'age': None, 'salary': 9000}, # Simulate missing age

{'id': 4, 'name': 'David', 'age': 35, 'salary': 6000},

{'id': 5, 'name': 'Eve', 'age': 27, 'salary': 8000}

]

for entry in data:

if entry['age'] is None:

entry['age'] = 0

entry['normalized\_salary'] = entry['salary'] / 100

df = pd.DataFrame(data)

def validate\_data\_quality(df):

issues\_found = False

for column in df.columns:

if df[column].isnull().any():

issues\_found = True

print(f"Warning: Missing values found in column '{column}'")

if column == 'salary':

salary\_std = df['salary'].std()

salary\_mean = df['salary'].mean()

salary\_threshold = salary\_mean + 2 \* salary\_std

if (df['salary'] > salary\_threshold).any():

issues\_found = True

print("Warning: Outliers found in 'salary' column")

if not issues\_found:

print("No data quality issues found. Data is clean.")

print("Data Validation:")

validate\_data\_quality(df)

print("\nTransformed Data:")

print(df)

9. Data Normalization:

9a.Write a function that normalizes the values in a DataFrame column to a range

between 0 and 1.

Ans:import pandas as pd

def normalize\_column(df, column\_name):

column = df[column\_name]

minVal = column.min()

maxVal = column.max()

normalized\_values = (column - minVal) / (maxVal - minVal)

df[column\_name] = normalized\_values

return df

data = {

'A': [1, 2, 3, 4, 5],

'B': [10, 20, 30, 40, 50],

'C': [100, 200, 300, 400, 500]

}

df = pd.DataFrame(data)

normalized\_df = normalize\_column(df.copy(), 'B')

print("Normalized DataFrame (column B):")

print(normalized\_df)

9b.Write a function that standardizes the values in a DataFrame column (mean=0,

standard deviation=1).

Ans: import pandas as pd

def standardize\_column(df, column\_name):

column = df[column\_name]

mean\_val = column.mean()

stdVal = column.std()

standardized\_values = (column - mean\_val) / stdVal

df[column\_name] = standardized\_values

return df

data = {

'A': [1, 2, 3, 4, 5],

'B': [10, 20, 30, 40, 50],

'C': [100, 200, 300, 400, 500]

}

df = pd.DataFrame(data)

standardized\_df = standardize\_column(df.copy(), 'C')

print("\nStandardized DataFrame (column C):")

print(standardized\_df)

10. Advanced List Comprehensions:

10a.Given a list of numbers, create a new list containing the square roots of the even

numbers only, using list comprehension.

Ans:import math

def square\_roots\_of\_even\_numbers(numbers):

square\_roots = [math.sqrt(num) for num in numbers if num % 2 == 0]

return square\_roots

numbers = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

square\_roots\_even = square\_roots\_of\_even\_numbers(numbers)

print("Square roots of even numbers:", square\_roots\_even)

10b.Given a list of tuples representing (name, score), create a new list containing

names of students who scored above the average, using list comprehension.

Ans:

def names\_above\_average(scores):

if not scores:

return []

total\_score = sum(score for \_, score in scores)

average\_score = total\_score / len(scores)

above\_average\_names = [name for name, score in scores if score > average\_score]

return above\_average\_names

student\_scores = [("Alice", 80), ("Bob", 75), ("Charlie", 90), ("David", 85), ("Eve", 95)]

above\_average\_students = names\_above\_average(student\_scores)

print("Students who scored above the average:", above\_average\_students)