Part-B Questions

May 28, 2025

[1]:
```
'''
1. Develop a python program to create a class called as BankAccount with␣
 ↪attributes like CustName,
AccountNumber, Balance, TypeofAccount and Address. Write the methods for␣
 ↪withdraw(), deposit() and
displayDetails(). Create multiple objects and simulate the bank operations.
'''

class BankAccount:
    def __init__(self, CustName, AccountNumber, Balance, TypeofAccount,␣
 ↪Address):
        self.CustName = CustName
        self.AccountNumber = AccountNumber
        self.Balance = Balance
        self.TypeofAccount = TypeofAccount
        self.Address = Address

    def deposit(self, amount):
        self.Balance += amount
        print(f"{self.CustName} deposited {amount}. New Balance: {self.
 ↪Balance}")

    def withdraw(self, amount):
        if amount <= self.Balance:
            self.Balance -= amount
            print(f"{self.CustName} withdrew {amount}. New Balance: {self.
 ↪Balance}")
        else:
            print(f"{self.CustName} has insufficient balance.")

    def displayDetails(self):
        print(f"Name: {self.CustName}, Account: {self.AccountNumber}, Balance:␣
 ↪{self.Balance}, Type: {self.TypeofAccount}, Address: {self.Address}")


acc1 = BankAccount("Rahul", 101, 5000, "Savings", "Bangalore")
acc2 = BankAccount("Anita", 102, 10000, "Current", "Delhi")
```

```
acc1.deposit(2000)
acc1.withdraw(1000)
acc1.displayDetails()

acc2.withdraw(3000)
acc2.deposit(1500)
acc2.displayDetails()
```

```
Rahul deposited  2000. New Balance:  7000
Rahul withdrew  1000. New Balance:  6000
Name: Rahul, Account: 101, Balance:  6000, Type: Savings, Address: Bangalore
Anita withdrew  3000. New Balance:  7000
Anita deposited  1500. New Balance:  8500
Name: Anita, Account: 102, Balance:  8500, Type: Current, Address: Delhi
```

[2]:
```
'''
2. Illustrate web scrapping by developing a python program to extract the date␣
 ↪and time of a
GitHub repository created and also the language of the last five repositories.
'''
import requests
from collections import Counter
from dateutil.parser import parse

def get_github_repositories(github_user):
    response = requests.get(f"https://api.github.com/users/{github_user}/repos")

    if response.status_code != 200:
        print(f"Failed to retrieve data for user: {github_user}")
        return

    repos = response.json()
    dates = [parse(repo["created_at"]) for repo in repos]

    print(f"Creation Dates: {dates}")
    print(f"Repositories Created by Month: {Counter(date.month for date in␣
 ↪dates)}")
    print(f"Repositories Created by Weekday: {Counter(date.weekday() for date␣
 ↪in dates)}")

    last_5_repos = sorted(repos, key=lambda r: r["pushed_at"], reverse=True)[:5]
    print(f"Last 5 Repositories: {[repo['name'] for repo in last_5_repos]}")
    print(f"Languages of Last 5 Repositories: {[repo['language'] for repo in␣
 ↪last_5_repos]}")

# Example usage
```

```
get_github_repositories("Anjan-2006")  # Replace with any valid GitHub username
```

```
Creation Dates: [datetime.datetime(2025, 4, 17, 2, 6, 23, tzinfo=tzutc()),
datetime.datetime(2025, 1, 8, 13, 36, 43, tzinfo=tzutc()),
datetime.datetime(2025, 5, 28, 11, 23, 37, tzinfo=tzutc()),
datetime.datetime(2025, 5, 28, 11, 23, 56, tzinfo=tzutc()),
datetime.datetime(2025, 5, 18, 10, 52, 24, tzinfo=tzutc())]
Repositories Created by Month: Counter({5: 3, 4: 1, 1: 1})
Repositories Created by Weekday: Counter({2: 3, 3: 1, 6: 1})
Last 5 Repositories: ['python2', 'python1', 'youtubeDataAnalysis', 'git-repo-
practice', 'gps-navigation']
Languages of Last 5 Repositories: [None, None, 'EJS', 'HTML', 'TypeScript']
```

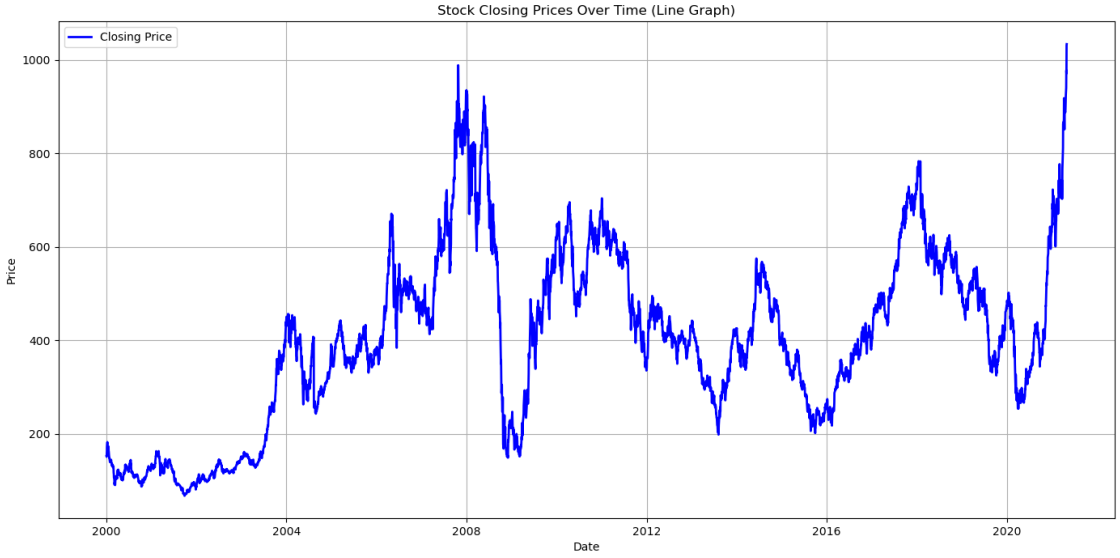
[5]:
```
'''
3. Create a sales report for a business based on product sales data stored in␣
 ↪an Excel file.
The Excel sheet contains the following columns:
    • Product: The name of the product.
    • Quantity: The number of units sold.
    • Price: The price per unit of the product.
    • Write a Python program using the openpyxl library to read the data from␣
 ↪the Excel file and
      compute the following:
        - Total Sales Amount, Total Quantity Sold, Average Price per Product,␣
 ↪Best-Selling Product,
          Most Expensive Product
'''

from openpyxl import load_workbook

wb = load_workbook("sales_data.xlsx")
ws = wb.active

sales_data = {}

for row in ws.iter_rows(min_row=2, values_only=True):
    product, quantity, price = row
    if product not in sales_data:
        sales_data[product] = {"quantity": 0, "total": 0, "price": price}
    sales_data[product]["quantity"] += quantity
    sales_data[product]["total"] += quantity * price

total_sales = sum(item["total"] for item in sales_data.values())
total_quantity = sum(item["quantity"] for item in sales_data.values())
average_price = sum(item["price"] for item in sales_data.values()) /␣
 ↪len(sales_data)
```

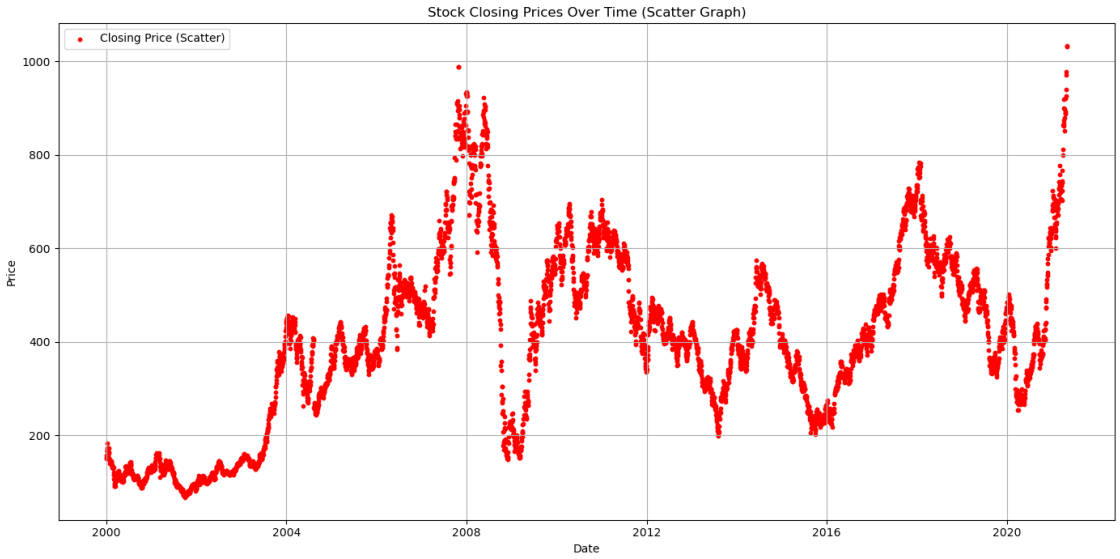
```
best_selling = max(sales_data.items(), key=lambda x: x[1]["quantity"])
most_expensive = max(sales_data.items(), key=lambda x: x[1]["price"])

print("\nSALES REPORT".upper())
print(f"Total Sales Amount:  {total_sales}")
print(f"Total Quantity Sold: {total_quantity} units")
print(f"Average Price per Product:  {average_price:.2f}")
print(f"Best-Selling Product: {best_selling[0]} ({best_selling[1]['quantity']}␣
 ↪units)")
print(f"Most Expensive Product: {most_expensive[0]}␣
 ↪( {most_expensive[1]['price']})")

'''
sales_data.xlsx contents
Product    Quantity    Price
Pen        100         1.5
Book       50          10
Bag        20          50
Notebook   75          5
'''
```

```
SALES REPORT
Total Sales Amount:  2025.0
Total Quantity Sold: 245 units
Average Price per Product:  16.62
Best-Selling Product: Pen (100 units)
Most Expensive Product: Bag ( 50)
```

[6]:
```
'''
4. Develop python program to load stock price data from a Kaggle CSV file and␣
 ↪create visualization
for closing prices over time. Using Matplotlib plot both line and scatter␣
 ↪graphs with full chart
features like labels, legends, and gridlines.
'''

import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv('TATASTEEL.csv')
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)

plt.figure(figsize=(14, 14))

# Line Graph
```

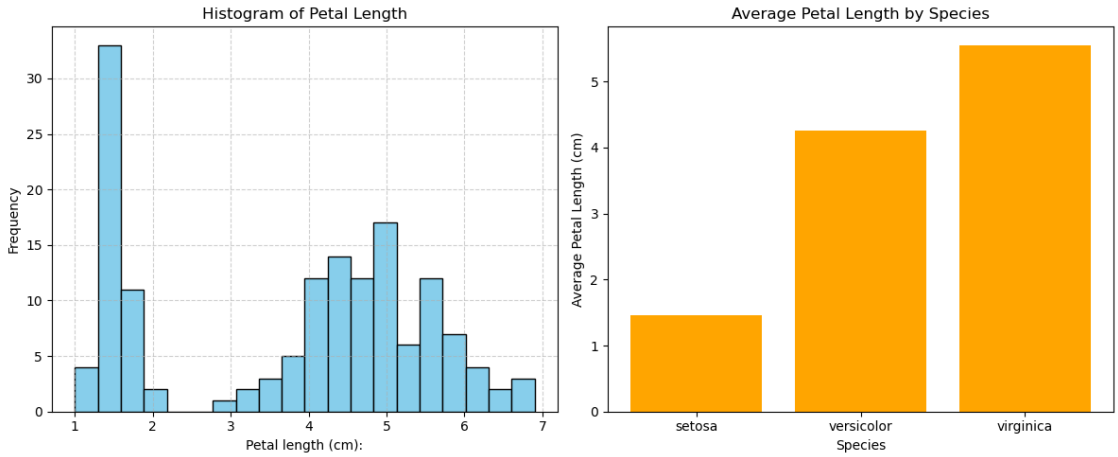
```
plt.subplot(2, 1, 1)
plt.plot(data['Close'], label='Closing Price', color='blue', linewidth=2)
plt.title('Stock Closing Prices Over Time (Line Graph)')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid(True)

# Scatter Graph
plt.subplot(2, 1, 2)
plt.scatter(data.index, data['Close'], color='red', s=10, label='Closing Price␣
 ↪(Scatter)')
plt.title('Stock Closing Prices Over Time (Scatter Graph)')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```





[7]:
```
'''
5. Develop Python program to visualize the Iris dataset by displaying a␣
 ↪histogram of petal lengths
and a bar graph of average petal length by species.
'''
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

iris = sns.load_dataset('iris')

plt.figure(figsize=(12, 5))
```

```
#----HISTOGRAM----
plt.subplot(1, 2, 1)
plt.hist(iris['petal_length'], bins=20, color='skyblue', edgecolor='black')
plt.title("Histogram of Petal Length")
plt.xlabel('Petal length (cm):')
plt.ylabel('Frequency')
plt.grid(True, linestyle='--', alpha=0.6)

#----BAR GRAPH----
plt.subplot(1, 2, 2)
average_petal_length = iris.groupby('species')['petal_length'].mean().
 ↪reset_index()
plt.bar(average_petal_length['species'], average_petal_length['petal_length'],␣
 ↪color='orange')
plt.title('Average Petal Length by Species')
plt.xlabel('Species')
plt.ylabel('Average Petal Length (cm)')

plt.tight_layout()
plt.show()
```



[8]:
```
'''
6. Create a Python program using SQLite to manage a users table with id, name,␣
 ↪and age.
Perform data insertion and retrieval of user information.
'''
import sqlite3  # Import SQLite module

# Establish Database Connection
conn = sqlite3.connect(':memory:')  # or use 'users.db' for file-based DB
```

```
cursor = conn.cursor()  # Create a cursor object

# Create Users Table
cursor.execute('''
CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    age INTEGER NOT NULL
)
''')

# Insert Sample Users
cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", ("Alice", 25))
cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", ("Bob", 30))
conn.commit()  # Commit changes

# Retrieve and Display User Information
cursor.execute("SELECT * FROM users")
rows = cursor.fetchall()

for row in rows:
    print(f"ID: {row[0]}, Name: {row[1]}, Age: {row[2]}")

# Close Database Connection
conn.close()
```

```
ID: 1, Name: Alice, Age: 25
ID: 2, Name: Bob, Age: 30
```