Assignment No:03

Q) Create a classes and objects for the project topic that you have selected.

1) Class:

```
class StockData(Base):
    __tablename__ = 'stock_data'

    Date = Column(DateTime, primary_key=True)
    Ticker = Column(String)
    Open = Column(Float)
    High = Column(Float)
    Low = Column(Float)
    Close = Column(Float)
    Volume = Column(Integer)
    Adj_Close = Column(Float)
    Change = Column(Float)

class StockAnalyzerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Stock Analyzer App")
        self.root.geometry("800x600")

        self.notebook = ttk.Notebook(root)
        self.notebook.pack(fill=tk.BOTH, expand=True)

        self.create_information_page()
        self.create_data_page()
        self.create_analyze_result_page()
        self.create_visualization_page()

        exit_button = tk.Button(root, text="Exit", command=root.destroy)
        exit_button.pack()


        self.engine = create_engine('sqlite:///stock_data.db', echo=False)
        Base.metadata.create_all(self.engine)
        self.Session = sessionmaker(bind=self.engine)


        self.ticker = ""


    def create_information_page(self):
        info_page = ttk.Frame(self.notebook)
        self.notebook.add(info_page, text="Information")


        label_ticker = tk.Label(info_page, text="Enter Ticker:")
        label_ticker.pack()


        self.ticker_entry = tk.Entry(info_page)
        self.ticker_entry.pack()


        retrieve_info_button = tk.Button(info_page, text="Retrieve Information",
command=self.retrieve_information)
        retrieve_info_button.pack()


        self.info_text = tk.Text(info_page, wrap=tk.WORD)
        self.info_text.pack(fill=tk.BOTH, expand=True)


    def create_data_page(self):
        data_page = ttk.Frame(self.notebook)
        self.notebook.add(data_page, text="Data")


        label_start_date = tk.Label(data_page, text="Start Date (YYYY-MM-DD):")
        label_start_date.pack()


        self.start_date_entry = tk.Entry(data_page)
        self.start_date_entry.pack()

        label_end_date = tk.Label(data_page, text="End Date (YYYY-MM-DD):")
        label_end_date.pack()

        self.end_date_entry = tk.Entry(data_page)
        self.end_date_entry.pack()

        retrieve_button = tk.Button(data_page, text="Retrieve Data", command=self.retrieve_data)
        retrieve_button.pack()

        download_data_button = tk.Button(data_page, text="Download Data",
command=self.download_data)
        download_data_button.pack()

        self.data_text = tk.Text(data_page, wrap=tk.WORD)
        self.data_text.pack(fill=tk.BOTH, expand=True)

    def create_visualization_page(self):
        visualization_page = ttk.Frame(self.notebook)
        self.notebook.add(visualization_page, text="Visualization")

        self.canvas = FigureCanvasTkAgg(plt.Figure(figsize=(8, 6)), master=visualization_page)
        self.canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)

    def create_analyze_result_page(self):
        analysis_page = ttk.Frame(self.notebook)
        self.notebook.add(analysis_page, text="Analyze result")

        self.analysis_text = tk.Text(analysis_page, wrap=tk.WORD)
        self.analysis_text.pack(fill=tk.BOTH, expand=True)

    def retrieve_information(self):
        self.ticker = self.ticker_entry.get().upper()

        try:
            stock_info = yf.Ticker(self.ticker).info
            self.info_text.delete(1.0, tk.END)
            self.info_text.insert(tk.END, f"Information for {self.ticker}:\n")
            self.info_text.insert(tk.END, f"Company Name: {stock_info['longName']}\n")
            self.info_text.insert(tk.END, f"Sector: {stock_info['sector']}\n")
            self.info_text.insert(tk.END, f"Industry: {stock_info['industry']}\n")
            self.info_text.insert(tk.END, f"Country: {stock_info['country']}\n")
            self.info_text.insert(tk.END, f"Exchange: {stock_info['exchange']}\n")
        except Exception as e:
            messagebox.showwarning("Invalid Ticker", f"Error retrieving information for {self.ticker}.")

    def retrieve_data(self):
        self.ticker = self.ticker_entry.get().upper()
        start_date_str = self.start_date_entry.get()
        end_date_str = self.end_date_entry.get()

        try:
            start_date = datetime.strptime(start_date_str, "%Y-%m-%d")
            end_date = datetime.strptime(end_date_str, "%Y-%m-%d")

            data = yf.download(self.ticker, start=start_date, end=end_date)
            if data is not None and not data.empty:
                data['Change'] = data['Adj Close'] / data['Adj Close'].shift(1) - 1

                self.data_text.delete(1.0, tk.END)
                self.data_text.insert(tk.END, f"Data for {self.ticker}:\n")
                self.data_text.insert(tk.END, f"Number of Rows: {len(data)}\n")
                self.data_text.insert(tk.END, f"Columns: {', '.join(data.columns)}\n\n")
                self.data_text.insert(tk.END, f"Data for {self.ticker}:\n")
                self.data_text.insert(tk.END, data.head())
                self.display_charts(data)
                self.analyze_stock(data)
            else:
                messagebox.showwarning("No Data", f"No data available for {self.ticker}.")
        except ValueError as e:
            messagebox.showwarning("Invalid Date Format", "Please enter valid date format (YYYY-MM-DD).")


    def analyze_stock(self, data):
        result_text = f"Results for {self.ticker}:\n"
        result_text += f"Total Return: {round(data['Change'].mean() * data['Change'].count() * 100, 2)}%\n"
        result_text += f"Standard Deviation: {round(np.std(data['Change']) * np.sqrt(data['Change'].count()), 4)}\n"
        result_text += f"Risk Return: {round((data['Change'].mean() / (np.std(data['Change']) * 100)), 4)}\n"

        result_text += f"50-day Moving Average: {round(data['Close'].rolling(window=50).mean().iloc[-1], 2)}\n"
        result_text += f"200-day Moving Average: {round(data['Close'].rolling(window=200).mean().iloc[-1], 2)}\n"

        momentum_period = 10
        data['Momentum'] = data['Close'] - data['Close'].shift(momentum_period)
        result_text += f"{momentum_period}-day Momentum: {round(data['Momentum'].iloc[-1], 2)}\n"

        self.analysis_text.delete(1.0, tk.END)
        self.analysis_text.insert(tk.END, result_text)

        self.store_data_in_database(data)

        self.result_text.delete(1.0, tk.END)
        self.result_text.insert(tk.END, result_text)
```

```
def store_data_in_database(self, data):
    Session = self.Session()
    for i, row in data.iterrows():
        stock_data = StockData(
            Date=row.name,
            Ticker=self.ticker,
            Open=row['Open'],
            High=row['High'],
            Low=row['Low'],
            Close=row['Close'],
            Volume=row['Volume'],
            Adj_Close=row['Adj Close'],
            Change=row['Change']
        )
        Session.merge(stock_data)

    Session.commit()
    Session.close()

def display_charts(self, data):
    self.display_line_chart(data)

    self.display_bar_chart(data)

    self.display_scatter_plot(data)

    self.display_box_plot(data)

    self.display_joint_bar_chart(data)

    self.display_technical_indicator_chart(data)
```

```
def display_line_chart(self, data):
    fig = self.canvas.figure
    fig.clear()

    ax = fig.add_subplot(231)
    ax.plot(data.index, data['Adj Close'], label='Adj Close')
    ax.set_title(f"{self.ticker_entry.get()} Adj Close Price Over Time(Line Chart)")
    ax.set_xlabel("Date")
    ax.set_ylabel("Adj Close Price")
    ax.legend()

def display_bar_chart(self, data):
    ax1 = self.canvas.figure.add_subplot(232)
    ax1.bar(data.index, data['Adj Close'], color='blue')
    ax1.set_title(f"{self.ticker} Adj Close Price Over Time(Bar Chart)")
    ax1.set_xlabel("Date")
    ax1.set_ylabel("Adj Close Price")

def display_scatter_plot(self, data):
    ax2 = self.canvas.figure.add_subplot(233)
    ax2.scatter(data['Open'], data['Close'], color='red')
    ax2.set_title(f"{self.ticker} Closing Price Over Opening Price(Scatter Plot)")
    ax2.set_xlabel("Opening Price")
    ax2.set_ylabel("Closing Price")

def display_box_plot(self, data):
    ax3 = self.canvas.figure.add_subplot(234)
    data[['Open', 'High', 'Low', 'Close', "Adj Close"]].plot(kind='box', ax=ax3)
    ax3.set_title(f"{self.ticker} Price (Box Plot)")
    ax3.set_ylabel("Price")

def display_joint_bar_chart(self, data):
    ax4 = self.canvas.figure.add_subplot(235)
```

```
        ax4.bar(data.index, data['Open'], label='Open')
        ax4.bar(data.index, data['Close'], label='Close', alpha=0.5)
        ax4.set_title(f"{self.ticker} Price Over Time(Joint Bar Chart)")
        ax4.set_xlabel("Date")
        ax4.set_ylabel("Price")
        ax4.legend()

    def display_technical_indicator_chart(self, data):
        ax5 = self.canvas.figure.add_subplot(236)
        indicator_data = self.simple_moving_average(data['Close'], window=50)
        ax5.plot(data.index, data['Close'], label='Close Price')
        ax5.plot(data.index, indicator_data, label='SMA (50)')
        ax5.set_title(f"{self.ticker} Price Over Time(Technical Indicator Chart)")
        ax5.set_xlabel("Date")
        ax5.set_ylabel("Price")
        ax5.legend()

        self.canvas.draw()

    def download_data(self):
        self.ticker = self.ticker_entry.get().upper()
        start_date_str = self.start_date_entry.get()
        end_date_str = self.end_date_entry.get()

        try:
            start_date = datetime.strptime(start_date_str, "%Y-%m-%d")
            end_date = datetime.strptime(end_date_str, "%Y-%m-%d")

            data = yf.download(self.ticker, start=start_date, end=end_date)

            file_path = filedialog.asksaveasfilename(defaultextension=".csv", filetypes=[("CSV files",
"*.csv")])
```

```
            if file_path:
                data.to_csv(file_path)
                messagebox.showinfo("Data Downloaded", f"Data for {self.ticker} has been downloaded to {file_path}.")
            else:
                messagebox.showinfo("Download Cancelled", "Data download cancelled.")
        except ValueError as e:
            messagebox.showwarning("Invalid Date Format", "Please enter valid date format (YYYY-MM-DD).")


    def simple_moving_average(self, data, window=50):
        return data.rolling(window=window).mean()
```

2) Object:

```
if __name__ == "__main__":
    root = tk.Tk()
    app = StockAnalyzerApp(root)
    root.mainloop()
```