|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE** | | | | | **DEPARTMENT OF COMPUTER SCIENCE ENGINEERING** | | | | |
| **Program Name:** B. Tech | | | | **Assignment Type: Lab** | | | **Academic Year:**2025-2026 | | |
| **Course Coordinator Name** | | | | Venkataramana Veeramsetty | | | | | |
| **Instructor(s) Name** | | | | |  | | --- | | Dr. V. Venkataramana (Co-ordinator) | | Dr. T. Sampath Kumar | | Dr. Pramoda Patro | | Dr. Brij Kishor Tiwari | | Dr.J.Ravichander | | Dr. Mohammand Ali Shaik | | Dr. Anirodh Kumar | | Mr. S.Naresh Kumar | | Dr. RAJESH VELPULA | | Mr. Kundhan Kumar | | Ms. Ch.Rajitha | | Mr. M Prakash | | Mr. B.Raju | | Intern 1 (Dharma teja) | | Intern 2 (Sai Prasad) | | Intern 3 (Sowmya) | | NS\_2 ( Mounika) | | | | | | |
| **Course Code** | | | 24CS002PC215 | **Course Title** | | AI Assisted Coding | | | |
| **Year/Sem** | | | II/I | **Regulation** | | R24 | | | |
| **Date and Day**  **of Assignment** | | | Week5 - Monday | **Time(s)** | |  | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | |  | | | |
| **AssignmentNumber: 9.1**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***Expected Time***  ***to complete*** |  |
|  | 1 | **Lab 9 – Documentation Generation: Automatic Documentation and Code Comments**  **Lab Objectives**   * To use AI-assisted coding tools for generating Python documentation and code comments. * To apply zero-shot, few-shot, and context-based prompt engineering for documentation creation. * To practice generating and refining docstrings, inline comments, and module-level documentation. * To compare outputs from different prompting styles for quality analysis.   **Task Description #1** (Documentation – Google-Style Docstrings for Python Functions)   * Task: Use AI to add Google-style docstrings to all functions in a given Python script. * Instructions:   + Prompt AI to generate docstrings without providing any input-output examples.   + Ensure each docstring includes:     - Function description     - Parameters with type hints     - Return values with type hints     - Example usage   + Review the generated docstrings for accuracy and formatting.   **Prompt**:  Please add Google‑style docstrings to all functions in the following Python script. Do not provide any input-output examples. Each docstring should include: function description, parameters with type hints, return values with type hints, and an example usage. Format appropriately according to Google-style guidelines.  **Code:**  def multiply\_numbers(num1: int, num2: int) -> int:    """Multiplies two numbers and returns the result.    Args:      num1: The first number to multiply.      num2: The second number to multiply.    Returns:      The product of num1 and num2.    """    return num1 \* num2  **Output:**  **Task Description #2** (Documentation – Inline Comments for Complex Logic)   * Task: Use AI to add meaningful inline comments to a Python program explaining only complex logic parts. * Instructions:   + Provide a Python script without comments to the AI.   + Instruct AI to skip obvious syntax explanations and focus only on tricky or non-intuitive code sections.   + Verify that comments improve code readability and maintainability.   **Prompt**:  Add inline comments to the Python scrip function—but only on complex or non‑obvious code sections. Skip commenting on clear or trivial lines. Keep comments concise and aligned (two spaces before `#`).  **Code:**  # Calculate the sum of squares of numbers from 1 to 10  def sum\_of\_squares():      total = 0      for i in range(1, 11):  # Loop through numbers from 1 to 10          total += i\*\*2       # Add the square of the current number to the total      return total            # Return the final sum  # Find the largest prime factor of a given number  def largest\_prime\_factor(n):      i = 2                   # Start checking divisibility from 2      largest\_factor = 1      while i \* i <= n:       # Continue while i squared is less than or equal to n          if n % i:           # Check if n is divisible by i              i += 1          # If not divisible, increment i          else:              n //= i         # If divisible, divide n by i (integer division)              largest\_factor = i # Update the largest factor      if n > 1:               # If after the loop, n is still greater than 1, it's a prime factor          largest\_factor = n  # Update the largest factor      return largest\_factor   # Return the largest prime factor  # Reverse a string  def reverse\_string(s):      return s[::-1]          # Use slicing to reverse the string  **Output:**  **Task Description #3** (Documentation – Module-Level Documentation)   * Task: Use AI to create a module-level docstring summarizing the purpose, dependencies, and main functions/classes of a Python file. * Instructions:   + Supply the entire Python file to AI.   + Instruct AI to write a single multi-line docstring at the top of the file.   + Ensure the docstring clearly describes functionality and usage without rewriting the entire code.   **Prompt**:  Add a module-level docstring to Python script. Summarize the module's purpose and functionality.List any external dependencies or imports.Describe the primary classes and functions defined within the module.  Do not rewrite the entire code; focus on summarizing its functionality and usage.  **Code:**  """A module for exploring and visualizing historical stock price data.  This module provides functions to download, process, and visualize historical  stock price data from Yahoo Finance using the yfinance library and analyze  stock trends using various technical indicators such as Simple Moving Average  (SMA), Exponential Moving Average (EMA), and Moving Average Convergence  Divergence (MACD).  Dependencies:  - yfinance: Used to download historical stock price data.  - pandas: Used for data manipulation and analysis.  - matplotlib.pyplot: Used for plotting and visualization.  - seaborn: Used for enhanced data visualization.  Classes:  - No custom classes are defined in this module.  Functions:  - download\_stock\_data(ticker, start\_date, end\_date): Downloads historical stock data for a given ticker symbol within a specified date range.  - calculate\_moving\_averages(data, window): Calculates the Simple Moving Average (SMA) and Exponential Moving Average (EMA) for a given data series.  - calculate\_macd(data, short\_window, long\_window, signal\_window): Calculates the Moving Average Convergence Divergence (MACD) and its signal line.  - plot\_stock\_data(data, title): Plots the closing price of the stock data.  - plot\_moving\_averages(data, title): Plots the closing price and calculated moving averages.  - plot\_macd(data, title): Plots the MACD line, signal line, and histogram.  """  import yfinance as yf  import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  def download\_stock\_data(ticker, start\_date, end\_date):      """Downloads historical stock data for a given ticker symbol within a specified date range."""      data = yf.download(ticker, start=start\_date, end=end\_date)      return data  def calculate\_moving\_averages(data, window):      """Calculates the Simple Moving Average (SMA) and Exponential Moving Average (EMA) for a given data series."""      data['SMA'] = data['Close'].rolling(window=window).mean()      data['EMA'] = data['Close'].ewm(span=window, adjust=False).mean()      return data  def calculate\_macd(data, short\_window, long\_window, signal\_window):      """Calculates the Moving Average Convergence Divergence (MACD) and its signal line."""      short\_ema = data['Close'].ewm(span=short\_window, adjust=False).mean()      long\_ema = data['Close'].ewm(span=long\_window, adjust=False).mean()      data['MACD'] = short\_ema - long\_ema      data['Signal\_Line'] = data['MACD'].ewm(span=signal\_window, adjust=False).mean()      data['MACD\_Histogram'] = data['MACD'] - data['Signal\_Line']      return data  def plot\_stock\_data(data, title):      """Plots the closing price of the stock data."""      plt.figure(figsize=(12, 6))      plt.plot(data['Close'], label='Closing Price')      plt.title(title)      plt.xlabel('Date')      plt.ylabel('Price')      plt.legend()      plt.grid(True)      plt.show()  def plot\_moving\_averages(data, title):      """Plots the closing price and calculated moving averages."""      plt.figure(figsize=(12, 6))      plt.plot(data['Close'], label='Closing Price')      plt.plot(data['SMA'], label='SMA', color='red')      plt.plot(data['EMA'], label='EMA', color='green')      plt.title(title)      plt.xlabel('Date')      plt.ylabel('Price')      plt.legend()      plt.grid(True)      plt.show()  def plot\_macd(data, title):      """Plots the MACD line, signal line, and histogram."""      plt.figure(figsize=(12, 8))      plt.plot(data['MACD'], label='MACD Line', color='blue')      plt.plot(data['Signal\_Line'], label='Signal Line', color='red')      plt.bar(data.index, data['MACD\_Histogram'], label='MACD Histogram', color='gray')      plt.title(title)      plt.xlabel('Date')      plt.ylabel('Value')      plt.legend()      plt.grid(True)      plt.show()  if \_\_name\_\_ == '\_\_main\_\_':      # Example Usage      ticker = "AAPL"      start\_date = "2022-01-01"      end\_date = "2023-01-01"      stock\_data = download\_stock\_data(ticker, start\_date, end\_date)      stock\_data = calculate\_moving\_averages(stock\_data, window=20)      stock\_data = calculate\_macd(stock\_data, short\_window=12, long\_window=26, signal\_window=9)      plot\_stock\_data(stock\_data, f'{ticker} Closing Price')      plot\_moving\_averages(stock\_data, f'{ticker} Moving Averages')      plot\_macd(stock\_data, f'{ticker} MACD')    **Output:**  IMG_256  IMG_256  IMG_256  **Task Description #4** (Documentation – Convert Comments to Structured Docstrings)   * Task: Use AI to transform existing inline comments into structured function docstrings following Google style. * Instructions:   + Provide AI with Python code containing inline comments.   + Ask AI to move relevant details from comments into function docstrings.   + Verify that the new docstrings keep the meaning intact while improving structure.   **Prompt**:  Convert the inline comments in the following Python code into structured function docstrings using the Google style. Ensure each docstring includes:  - A brief description of the function's purpose.  - Parameters with type hints.  - Return values with type hints.  - Example usage.  Do not add new comments; only transform existing ones into docstrings.  **Code:**  def add\_numbers(a: int, b: int) -> int:    """Adds two numbers.    Args:      a: The first number.      b: The second number.    Returns:      The sum of the two numbers.    Examples:      >>> add\_numbers(2, 3)      5    """    return a + b  def subtract\_numbers(a: int, b: int) -> int:    """Subtracts two numbers.    Args:      a: The first number.      b: The second number.    Returns:      The difference between the two numbers.    Examples:      >>> subtract\_numbers(5, 2)      3    """    return a - b  **Task Description #5** (Documentation – Review and Correct Docstrings)   * Task: Use AI to identify and correct inaccuracies in existing docstrings. * Instructions:   + Provide Python code with outdated or incorrect docstrings.   + Instruct AI to rewrite each docstring to match the current code behavior.   + Ensure corrections follow Google-style formatting.   **Code:**  import requests  import math  def add(a, b):      """Adds two numbers.      Args:          a: The first number.          b: The second number.      Returns:          The sum of `a` and `b`.      """      return a + b  def fetch\_data(url):      """Retrieves data from the given URL.      Args:          url: The URL to fetch data from.      Returns:          The JSON response from the URL.      """      response = requests.get(url)      return response.json()  def process\_file(file\_path):      """Processes the file at the given path.      Args:          file\_path: The path to the file.      Returns:          A list of lines from the file.      """      with open(file\_path, 'r') as file:          data = file.read()      return data.splitlines()  def calculate\_area(radius):      """Calculates the area of a circle.      Args:          radius: The radius of the circle.      Returns:          The area of the circle.      """      return math.pi \* radius \*\* 2  **Task Description #6** (Documentation – Prompt Comparison Experiment)   * Task: Compare documentation output from a vague prompt and a detailed prompt for the same Python function. * Instructions:   + Create two prompts: one simple (“Add comments to this function”) and one detailed (“Add Google-style docstrings with parameters, return types, and examples”).   + Use AI to process the same Python function with both prompts.   + Analyze and record differences in quality, accuracy, and completeness.   **Prompt**:  Generate a Python function that performs a specific task. First, provide a vague prompt to generate the function, then a detailed prompt to generate the same function with comprehensive Google-style docstrings.  **Code:**  def process\_text(text):    """Processes a given string by converting it to uppercase.    This function takes a string as input and returns a new string with all    characters converted to uppercase.    Args:      text: The input string to be processed.    Returns:      A new string with all characters converted to uppercase.    Examples:      >>> process\_text("hello world")      'HELLO WORLD'      >>> process\_text("Python")      'PYTHON'    """    return text.upper() | | | | | | Week5 - Monday |  |