# System Requirements Specification Index

For

## Python Basics and NumPy, Pandas

Restaurant ordering system 1.0

## IIHT Pvt. Ltd.

[fullstack@iiht.com](mailto:fullstack@iiht.com)

**Usecase on Python basics and numpy and pandas**

**Topic covered in the usecase**

* **Class & Object Creation →** RestaurantManager is a class that handles restaurant operations.
* Encapsulation → Data like menu, orders, and total\_orders are stored within the class.
* **Method Operations** → Functions like add\_order(), get\_sales\_analysis(), and get\_daily\_revenue\_stats().
* Dictionary (dict) → Stores the menu and orders.
* List (list) → Stores multiple orders.
* Tuple (tuple) → Stores daily specials.
* **Try-Except Blocks** → Used in add\_order(), \_save\_order\_to\_file(), and get\_sales\_analysis() to handle:
* Invalid menu items (KeyError).
* Empty orders (ValueError).
* File I/O errors (IOError).
* **File Handling**
* Writing to JSON → Stores order details in \_save\_order\_to\_file() using json.dump().
* Appending Data → Opens the file in append mode ("a") to log multiple orders.
* **NumPy for Statistical Analysis**
* Used in get\_daily\_revenue\_stats() for:
* np.sum() → Calculates total revenue.
* np.mean() → Computes average order value.
* np.min() & np.max() → Finds the lowest and highest order amounts.
* **Pandas for Data Analysis**
* Used in get\_sales\_analysis() for:
* DataFrame Creation (pd.DataFrame(self.orders))
* Data Explosion (.explode("items")) → Converts lists into separate rows.
* JSON Normalization (pd.json\_normalize()) → Extracts nested item details.
* Data Merging (pd.concat()) → Joins exploded data with order metadata

**Introduction**

**Bites Restaurant**

Welcome to  **Bites Restaurant** a newly opened digital restaurant! You have been hired as a software engineer to develop an efficient system for managing restaurant orders, analyzing sales, and calculating daily revenue statistics.

The restaurant serves various delicious meals, and your task is to create a **Restaurant Manager** system that can:

1. Store menu items and prices.
2. Process customer orders.
3. Apply discounts for daily specials.
4. Save orders to a file.
5. Analyze daily sales using **Pandas**.
6. Calculate revenue statistics using **NumPy**.
7. Validate customer names before processing orders

**Project structure**

restaurant\_management\_system/

│── 📂 src/ # Source code

│ │── restaurant\_manager.py # Main restaurant manager script

│ │── demo.py # Demo script

│ │── orders.json # JSON storage

│── 📂 tests/ # Test cases

│── 📂 docs/ # Documentation

│ │── Restaurant\_document.docx

│── custom.ih

In the above project structure, you are given the template to complete the restaurant manager.py , demo .py template code .

As a developer you are supposed to build the system.

**Create a RestaurantManager class** to handle:

* Menu storage
* Order processing with discounts
* Revenue calculations with NumPy
* Sales analysis using Pandas
* Order storage using JSON

**Develop functionalities** for:

* add\_order(customer\_name, items): Adds an order, applies a 10% discount on daily specials, and saves it.
* \_save\_order\_to\_file(order): Stores orders in orders.json.
* get\_daily\_revenue\_stats(): Uses NumPy to calculate total, average, min, and max revenue.
* get\_sales\_analysis(): Uses Pandas to analyze sales data.

**Test the system using demo.py**, which:

* Creates a RestaurantManager instance.
* Places orders for customers.
* Retrieves and prints revenue statistics.
* Displays a Pandas DataFrame for sales analysis.

**Questions to be answered**

1. What is the total cost of Order 1?
2. What is the total cost of Order 2?
3. What is the total daily revenue?
4. What is the average revenue per order?
5. What is the minimum order value recorded?
6. What is the maximum order value recorded?
7. What is the price of the Burger after applying the discount?
8. What is the outcome with the sales analysis?

**Execution Steps to Follow:**

1. All actions like build, compile, running application,running test cases will be through Command Terminal.
2. To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal
3. This editor Auto Saves the code
4. If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the

internal git/repository. Else the code will not be available in the next login.

1. These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
2. To setup environment:

You can run the application without importing any packages

1. To launch application:

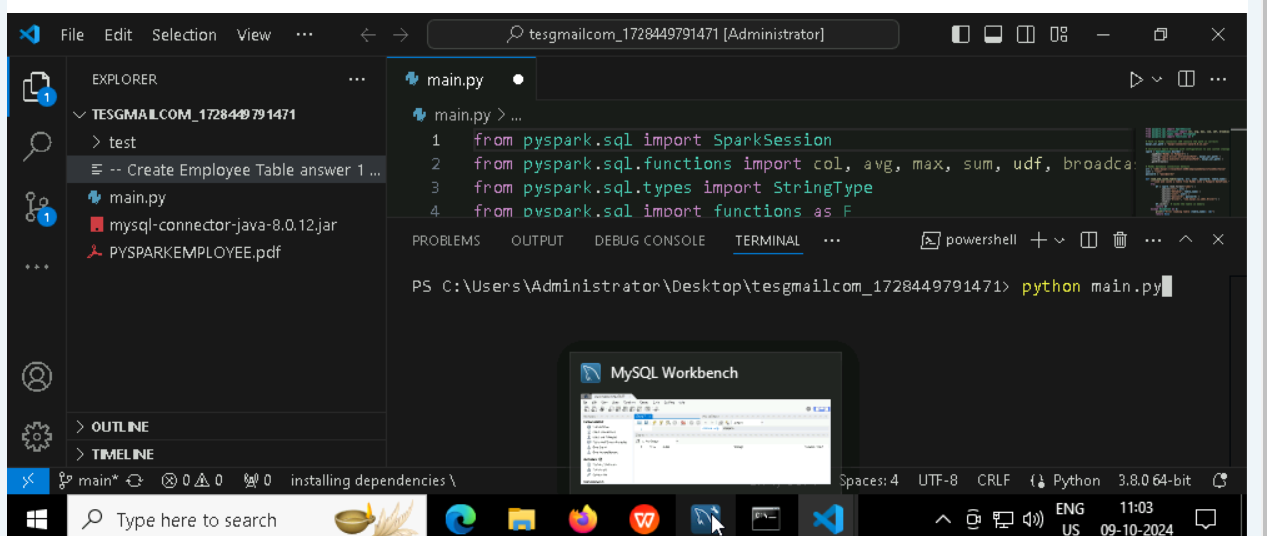
Python demo .py

1. To run Test cases:

python -m unittest

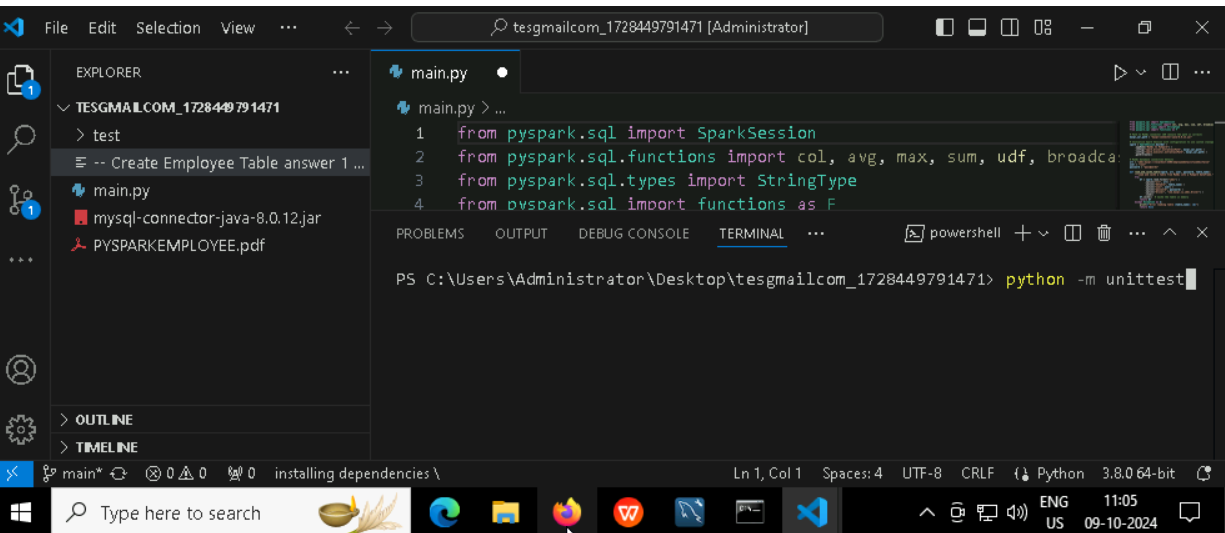
Before Final Submission also, you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository for code

**Screen shot to run the program**



**To run the application**

* **Python main.py**



**To run the testcase**

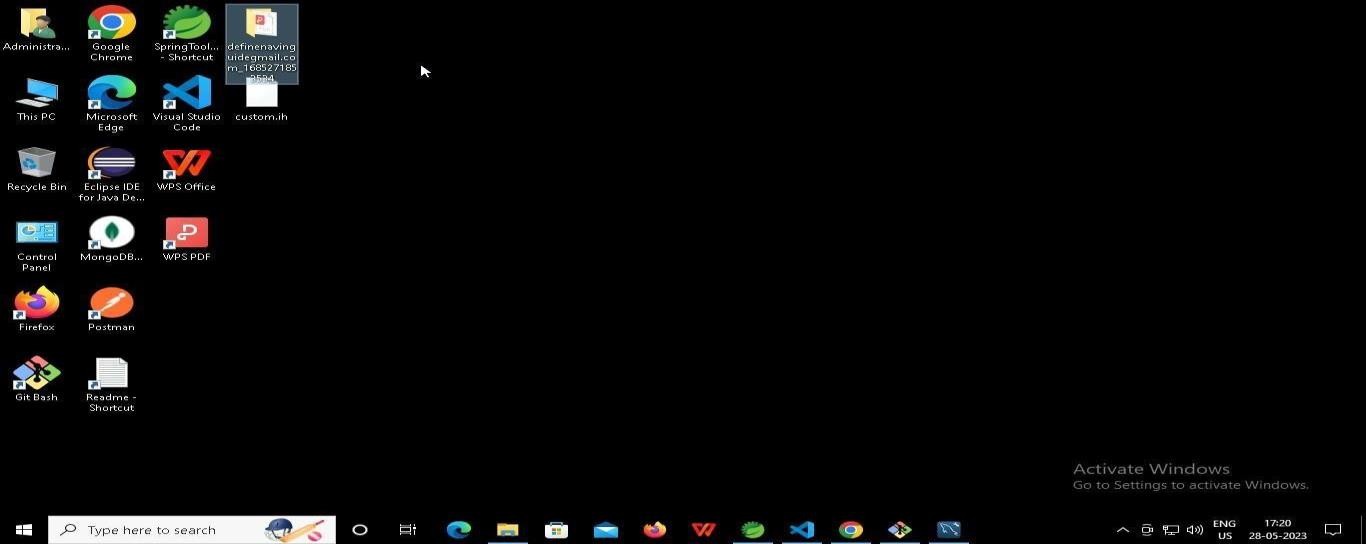
* **Python -m unittest**

**Screenshot to push the application to github**

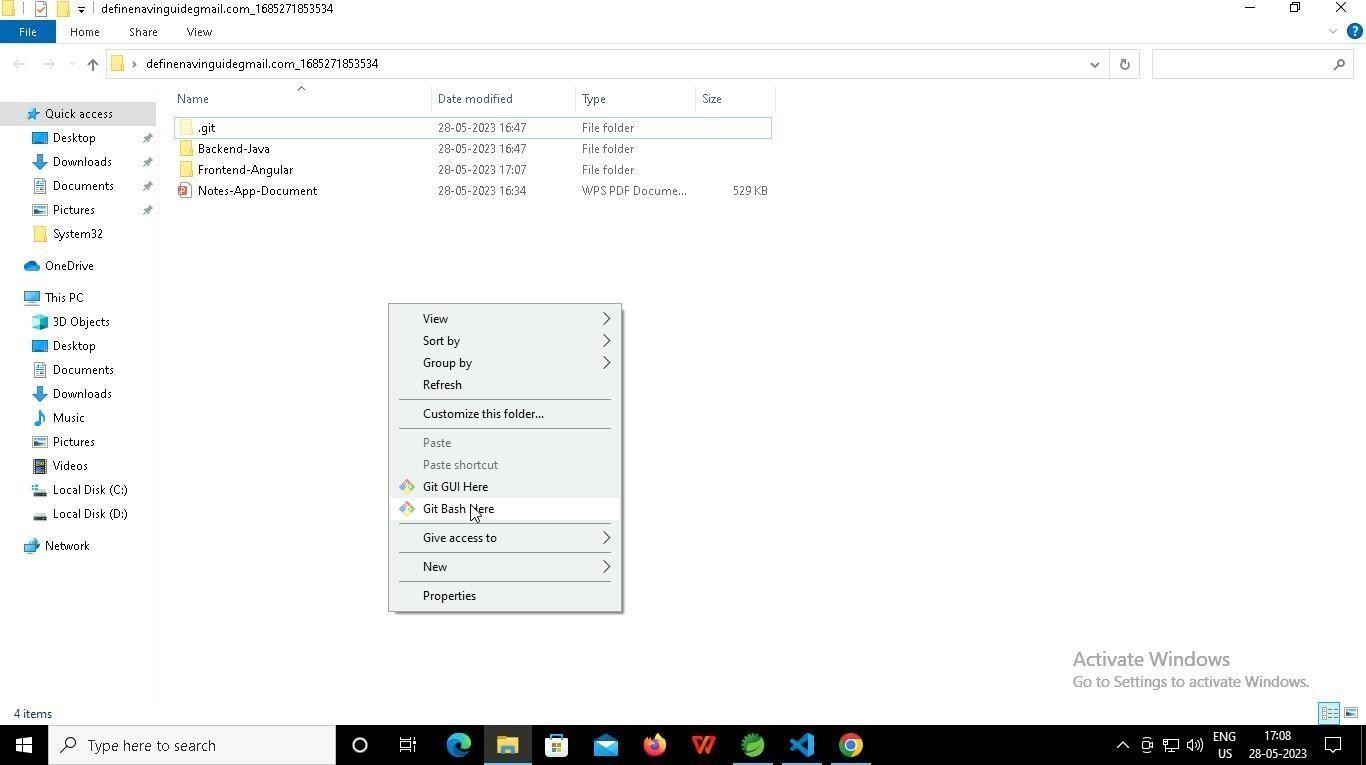
### -----x-----

#### You can run test cases as many numbers of times and at any stage of Development, to check how many test cases are passed/failed and accordingly refactor your code.

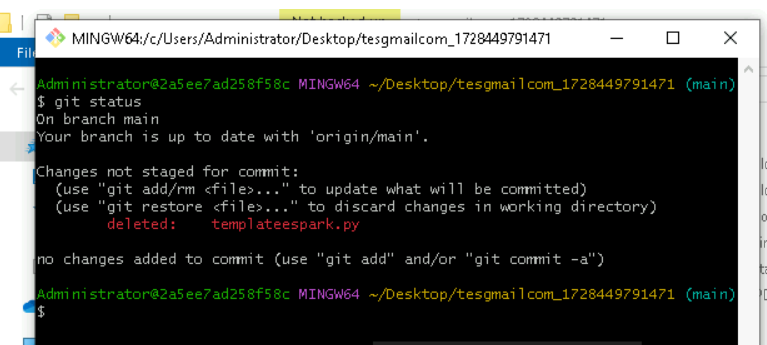
* 1. **Make sure before final submission you commit all changes to git**. For that open the project folder available on desktop



* + 1. **Right click in folder and open Git Bash**



* + 1. **In Git bash terminal, run following commands**
    2. **git status**



* + 1. **git add .**

A black screen with yellow and purple text

Description automatically generated

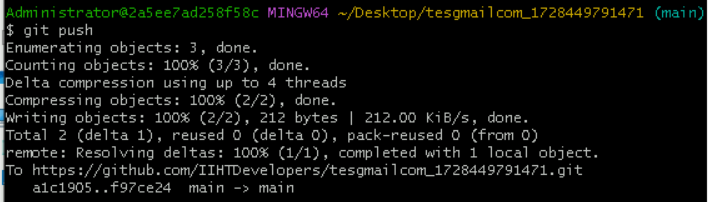
* + 1. git commit -m “First commit”

(You can provide any message every time you commit)

A screenshot of a computer

Description automatically generated

* + 1. **git push**



### After you have pushed your code Finally click on the final submission button

### 

### You should see a screen like this you will have to wait for the results . after getting this page you can leave the system

### A blue screen with white text Description automatically generated

### -----x-----