
# End-to-End Machine Learning Application with Streamlit

# 1.0 Project Objective

The objective of this project is to develop an end-to-end machine learning system with an interactive user interface for real-time predictions. The system will leverage the power of GitHub for version control and collaboration, Python for machine learning model development, Streamlit for creating an intuitive web interface, and Streamlit Community Cloud for easy deployment. The project aims to achieve the following:

## 1.1 Version Control Setup

The first step will involve creating a GitHub repository to manage and maintain version control throughout the project. This repository will serve as the central location for code collaboration, change tracking, and project documentation.

## 1.2 Local Environment Setup

The project will be developed in a local Python environment using VS Code as the integrated development environment (IDE). A virtual environment will be set up to manage dependencies and avoid conflicts with other Python projects. Essential libraries such as Jupyter, Pandas, Scikit-learn, and Streamlit will be installed to support model development and web application creation.

## 1.3 Model Development

The core of the project will involve building a machine learning model capable of making predictions based on input data. This process will be carried out in a Jupyter Notebook to facilitate interactive data exploration, preprocessing, and model evaluation. The model will be trained on a dataset, validated for accuracy, and optimized through hyperparameter tuning to enhance performance.

## 1.4 Streamlit UI Development

Following model development, an interactive web interface will be built using Streamlit. The interface will allow users to input their data, trigger predictions from the trained model, and view the results in real-time. This will make the model accessible and usable by non-technical users.

## 1.5 Deployment

Once the model and interface are developed, the project will be deployed using Streamlit Community Cloud, ensuring that the application is hosted online and easily accessible. The deployment process will involve generating a **requirements.txt** file to define the necessary Python libraries and versions, ensuring the application can run on the cloud without issues.

## 1.6 Outcome

The outcome of this project will be a fully functional machine learning application where:

- Users can interact with the model via a simple web interface built with Streamlit.
- The model can make real-time predictions on new data.
- The project will be easily reproducible and maintainable through GitHub version control and cloud-based deployment.
- By the end of this project, participants will have the skills to integrate machine learning models into a fully functional web application. They will gain practical experience in developing an end-to-end system, from data preprocessing and model training to deployment. Participants will also learn to ensure the system's scalability, maintainability, and modularity, while acquiring proficiency in version control with GitHub, environment management, and cloud-based deployment on Streamlit Community Cloud.

# Dataset Description

The **Chronic Kidney Disease (CKD)** dataset is designed to help in building machine learning models for predicting the likelihood of CKD in patients based on clinical, biochemical, and symptomatic features. The dataset comprises 30,000 patient records, modeled after real-world CKD case reports.

**Key Features:**

- **Clinical & Biochemical Variables:** Includes age, blood pressure, serum creatinine, albumin levels, blood glucose, hemoglobin, and other important clinical indicators related to kidney function.
- **Symptoms & Conditions:** Features include hypertension, diabetes mellitus, coronary artery disease, anemia, and edema, which are major factors influencing kidney health.
- **Urinary Indicators:** Includes urine-related features like specific gravity, pus cells, albumin, and sugar levels in the urine, which are critical for assessing kidney function.
- **Target Variable: ckd** (Chronic Kidney Disease), where **1** indicates the presence of CKD and **0** indicates its absence.

**Clinical Relevance:**

- This dataset allows for the prediction of CKD onset, particularly in patients with predisposing conditions like diabetes and hypertension.
- The variables selected are clinically significant and are used by healthcare professionals for diagnosing kidney disease.

**Use Cases:**

1. **Early Risk Detection:** Predict the onset of CKD, especially in diabetic and hypertensive patients.
2. **Remote Screening:** Integrate predictive models for community-level CKD screening through mobile applications.
3. **Hospital Decision Support:** Real-time alerts for kidney failure risks in healthcare settings.
4. **Medical Education:** Aid in teaching the clinical signs and risks of CKD.

**Data Collection:**

The dataset was sourced from EuropePMC, with raw abstracts scraped and clinical values extracted using Python-based scraping and NLP techniques. Some synthetic data records were generated based on medical literature to compensate for missing or unstructured data.

This dataset is an ideal resource for training robust classifiers, simulating diagnostic scenarios, and understanding the progression of CKD and its related comorbidities.

Dataset File: https://drive.google.com/file/d/1e1S2rNn0_Bjv3XrUthw_M7MFoGuYcQE1/view

Description: https://drive.google.com/file/d/1ayspq7NiOkMEFLYGVSNsawiXDNAPLejA/view

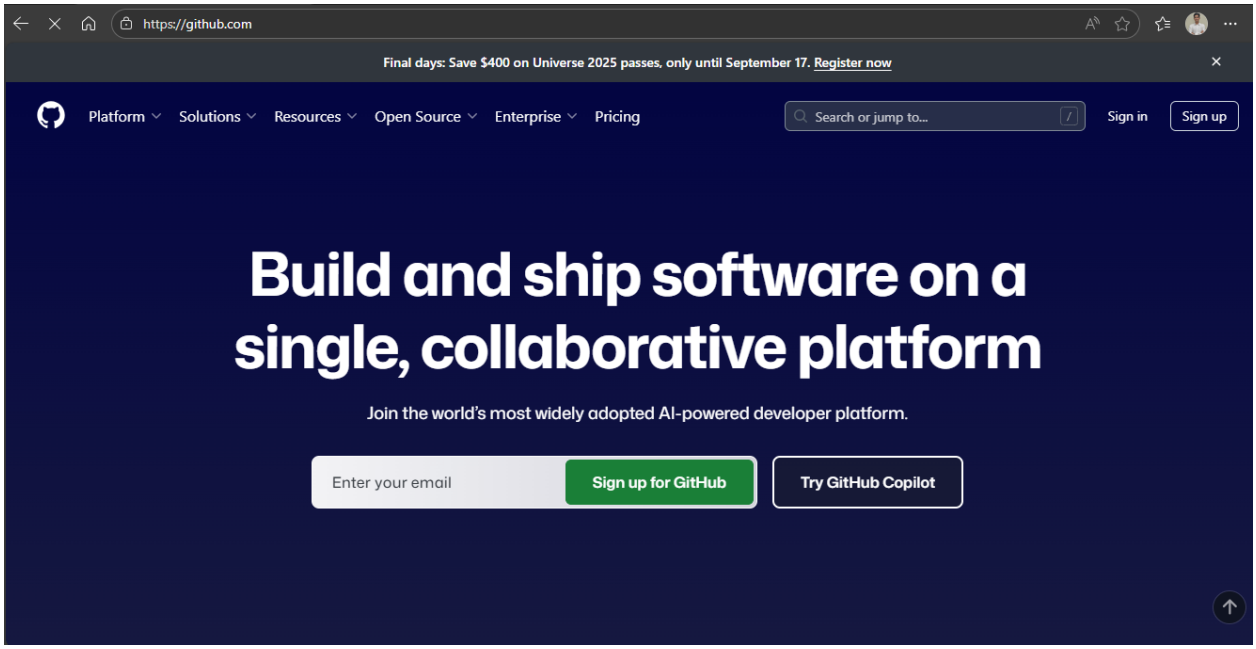
# 1. Version Control Setup

# Step 1. Version Control Setup

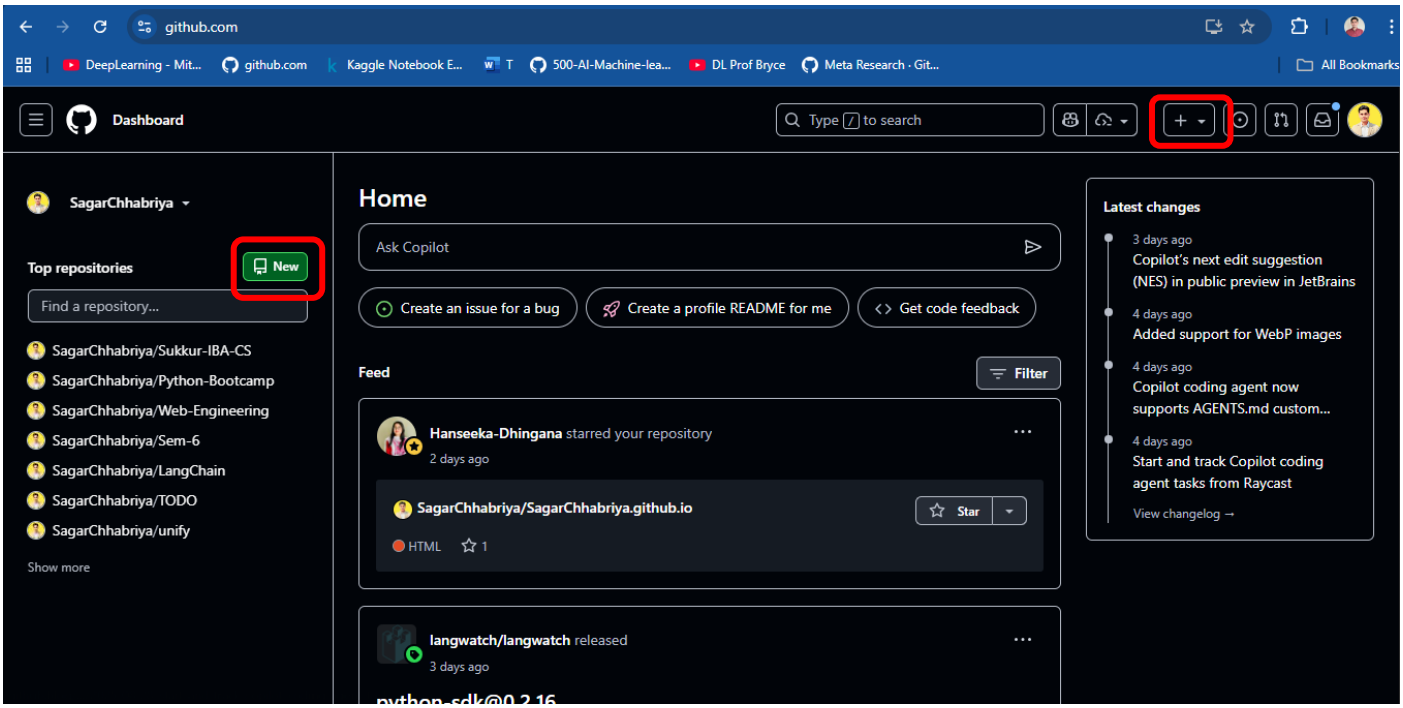
## 1.1 GitHub Account Creation

To manage and maintain version control for the project, GitHub will be used as the hosting platform. Follow the steps below to create and configure a GitHub repository:

1. Navigate to https://github.com.
2. Create a new GitHub account using your institutional email address (e.g., .edu) if available; alternatively, a Gmail address is also acceptable.



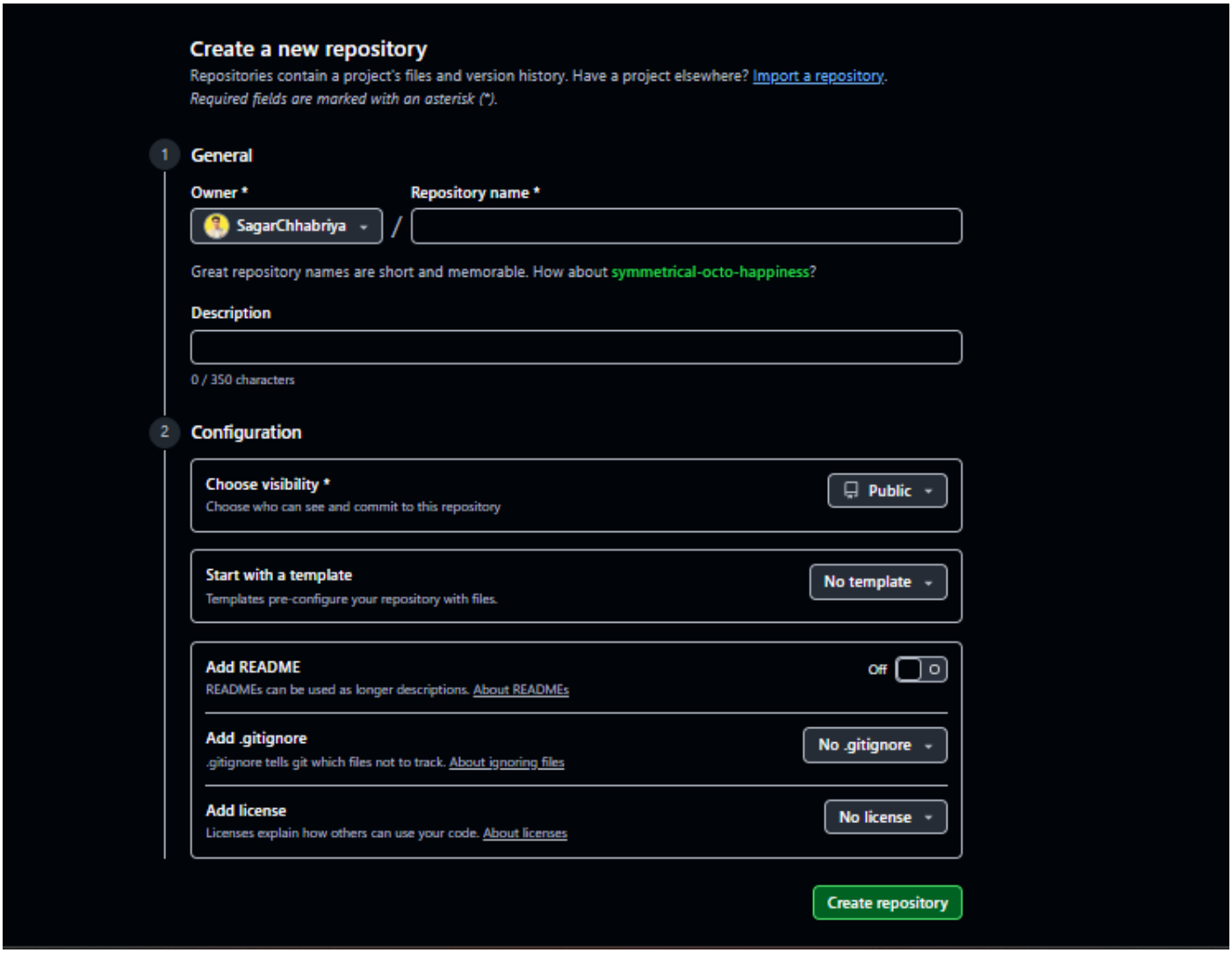

Once registered and logged in, the GitHub home screen will be displayed. If this is your first time using GitHub, the interface may appear minimal, as no repositories have been created yet.

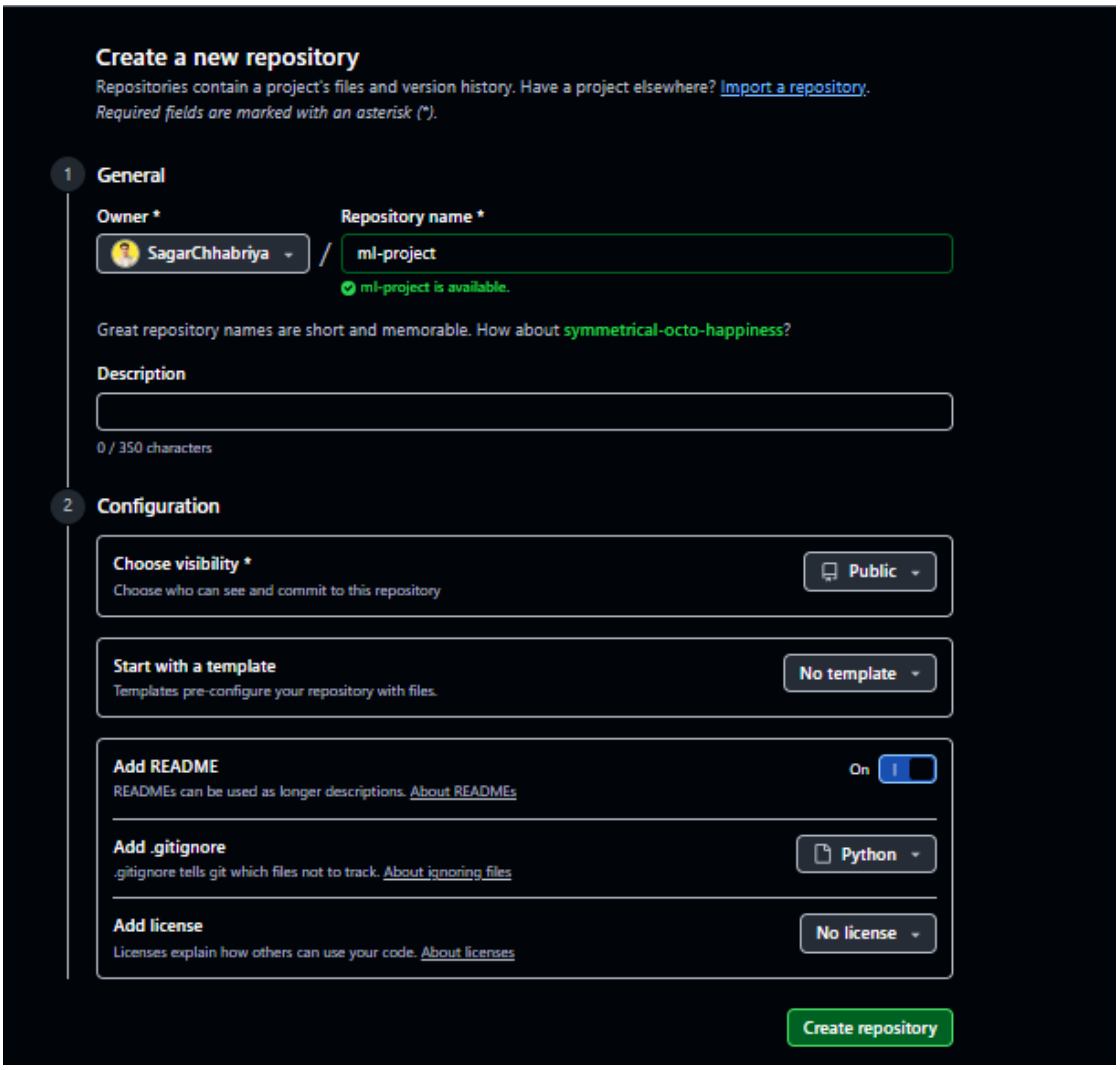


## 1.2 Creating a New Repository

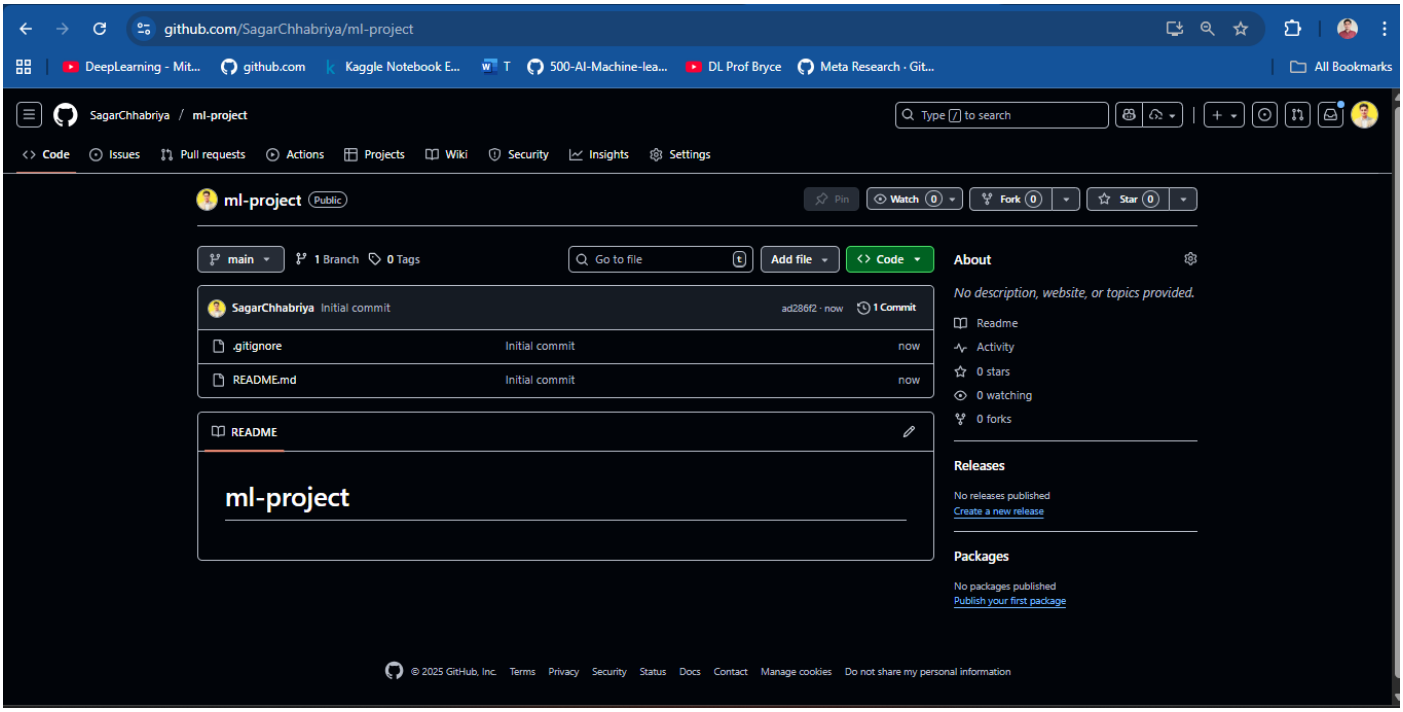
To create a new repository for the project:

1. On the GitHub homepage, click the green *New* button or click on the **"+"** icon in the top-right corner and select "New repository" from the dropdown.
2. Fill in the repository details as follows:
    - **Repository Name**: ml-project
    - **Description**: *(Optional; can be left blank)*
    - **Visibility**: Select **Public**
    - **Template**: Leave at default (no template)
    - **Initialize Repository**:
        - **Add a README file**: Enable this option
        - **Add .gitignore**:
            - Click on the .gitignore dropdown
            - Type and select **Python**
        - **License**: Leave at default (No license selected)
3. Click the green **"Create repository"** button to complete the setup.






Once the repository is successfully created, it will be initialized with a README.md and a Python-specific .gitignore file, making it ready for cloning and further development.



# 2. Local Setup (VS Code)

# Step 2. Local Setup (VS Code)

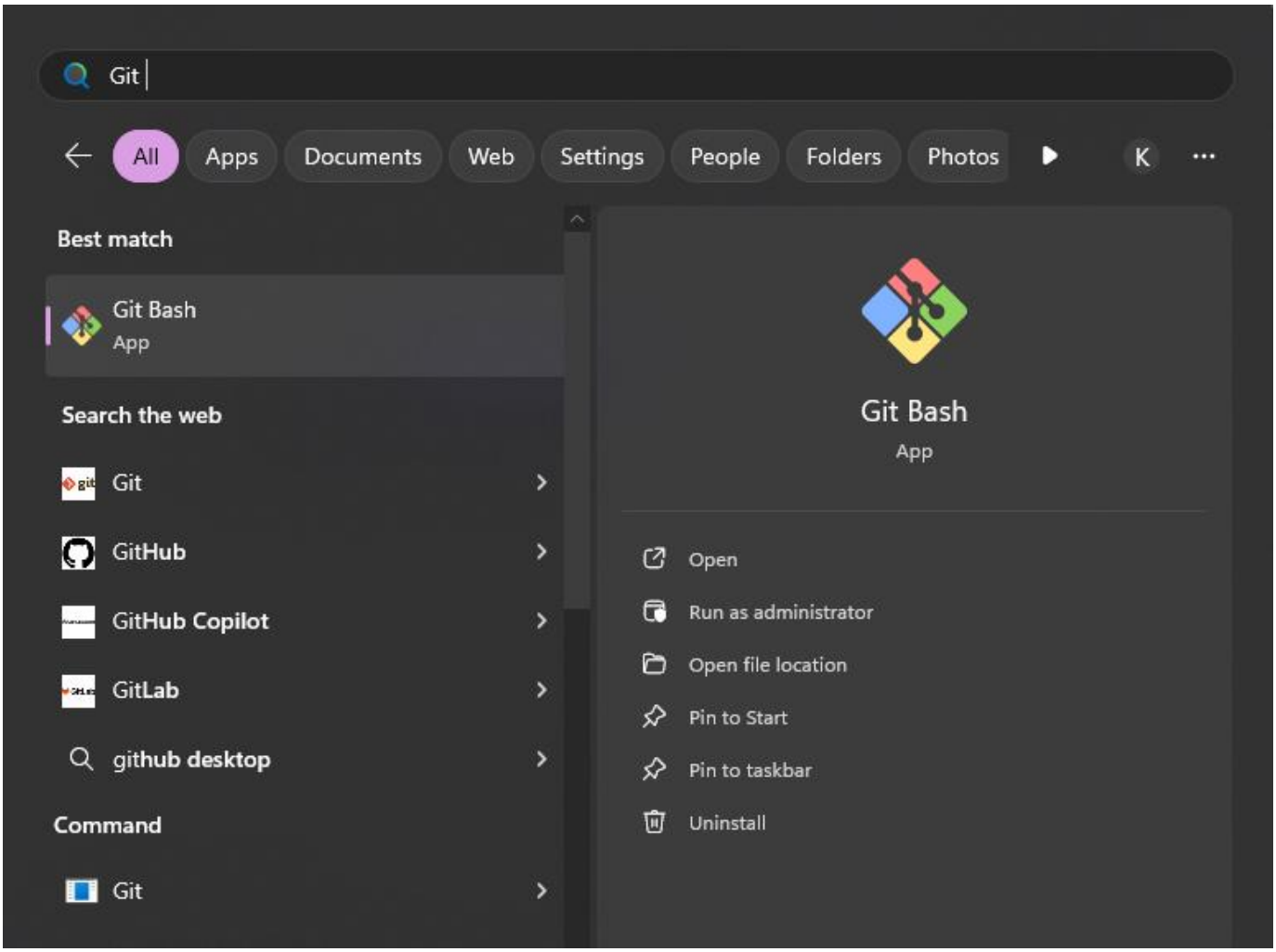
To set up the local development environment for this project, follow the steps below:

## 2.1 Install Required Software

1. **Python Installation**:
   Ensure that Python is installed on your local system. If not, download and install the latest stable version from Python's official website.
   - During installation, ensure to check the option **"Add Python to PATH"** to set the necessary environment variables automatically.
2. **Install Visual Studio Code (VS Code)**:
   Download and install VS Code, which will be used as the primary integrated development environment (IDE) for this project.
3. **Install Required Extensions for VS Code**:
   - Open VS Code and go to the **Extensions** tab on the left sidebar.
   - Install the following extensions:
     - **Python** (for Python development)
     - **Jupyter** (for working with Jupyter notebooks within VS Code)

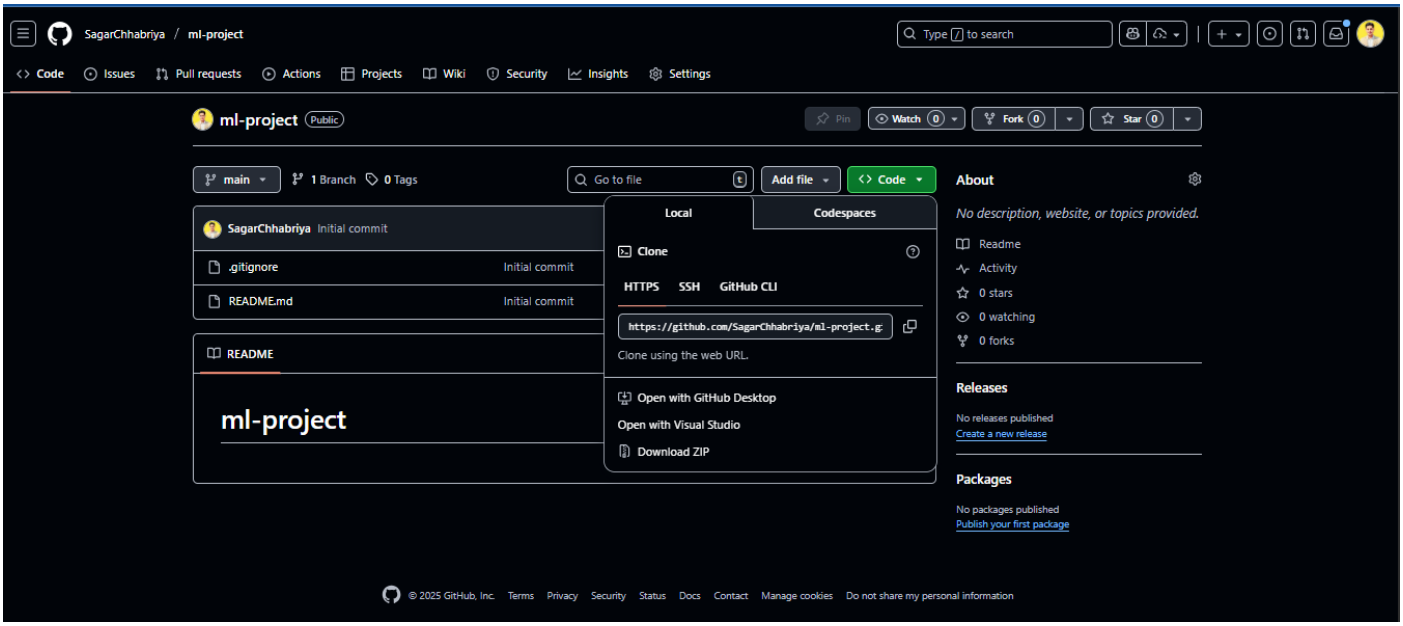

In my case it's already installed and it shows the disable option.

4. **Download and Install Git Bash** (if not already installed):
   Git Bash is a terminal emulator that allows you to run Git commands and other shell commands on Windows.
   Follow these steps to install it:
   - Go to the official Git Downloads page.
   - Select **Windows** and download the latest version of Git for Windows.
   - Run the installer and follow the prompts. Make sure to select the option to **Add Git Bash to your PATH** during installation.
   - After installation, you can open Git Bash by searching for it in the Windows Start menu.



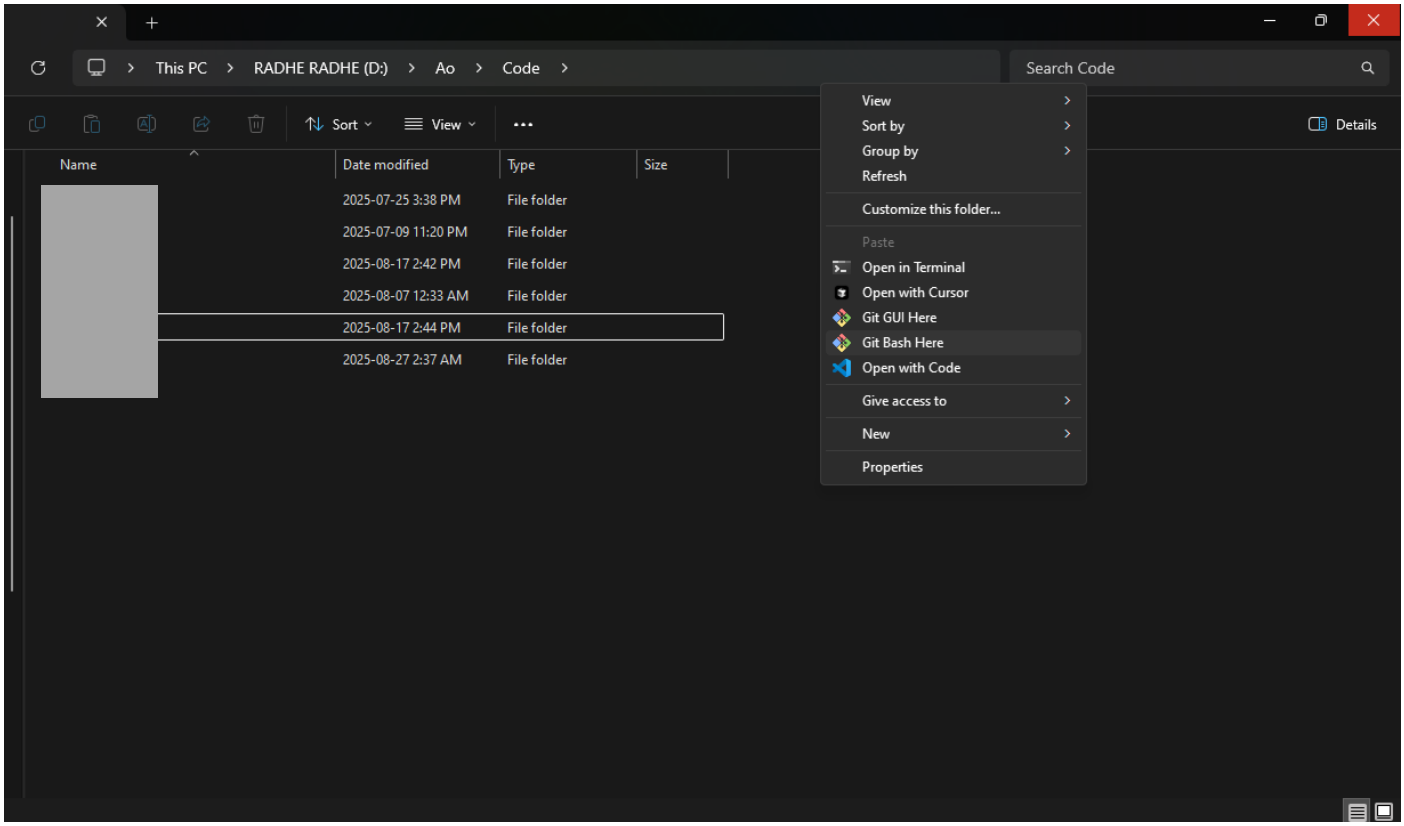

## 2.2 Clone the Repository

1. Go to the GitHub repository page that you created in Step 1.
2. Click the green **"Code"** button and copy the HTTPS clone link.

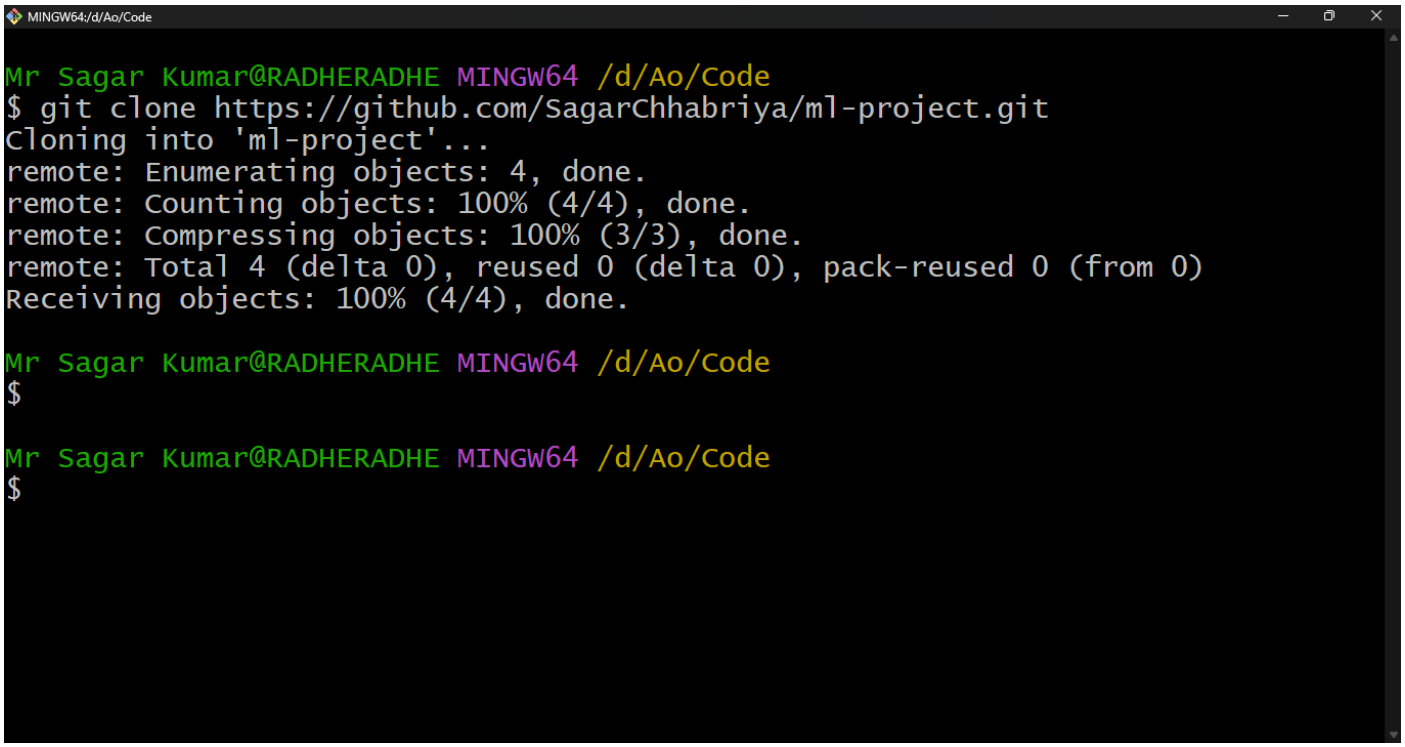


3. Open **Git Bash** (or your terminal of choice).

   Open any folder > Right Click > Git Bash Here




Type git clone <repo link > and hit enter.



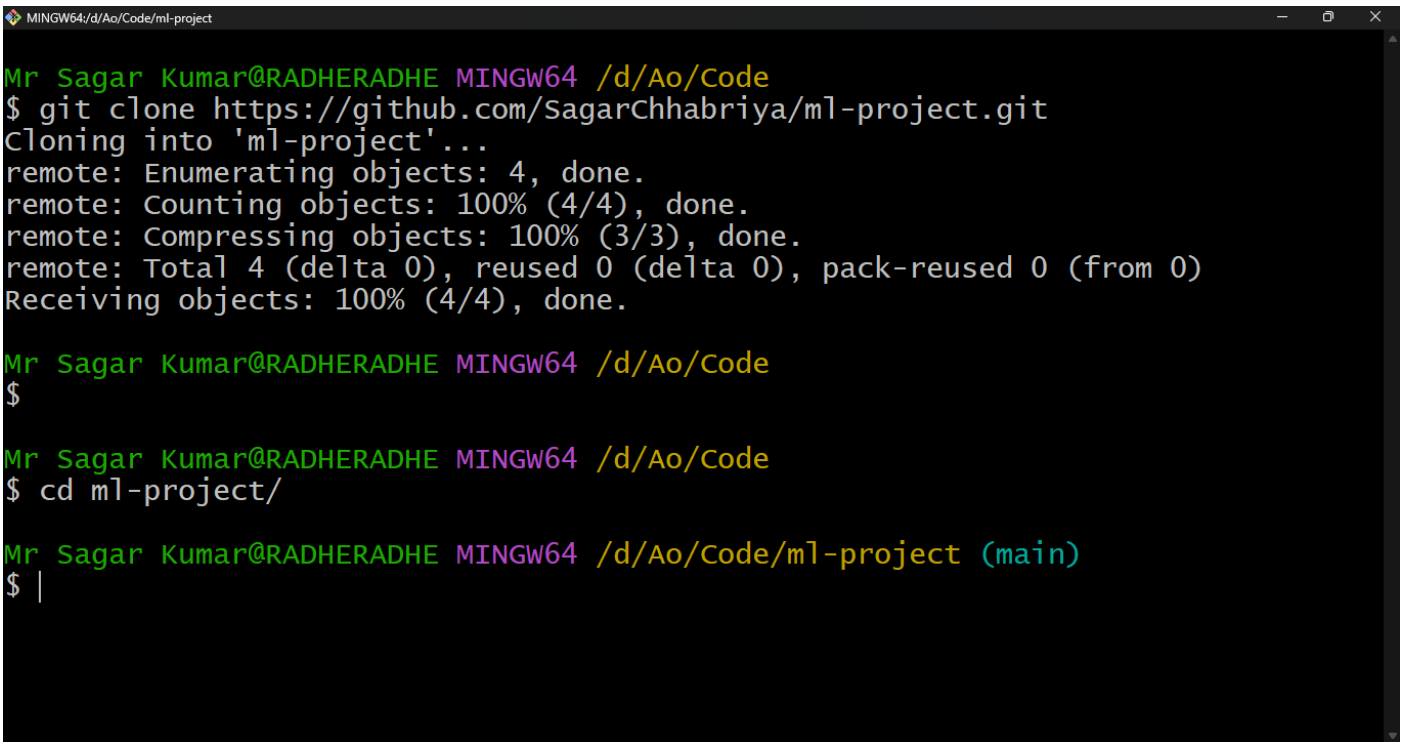

Note: You need to config the email and username with git bash for authorization after installing.

git config --global user.email “youemail@org.com”

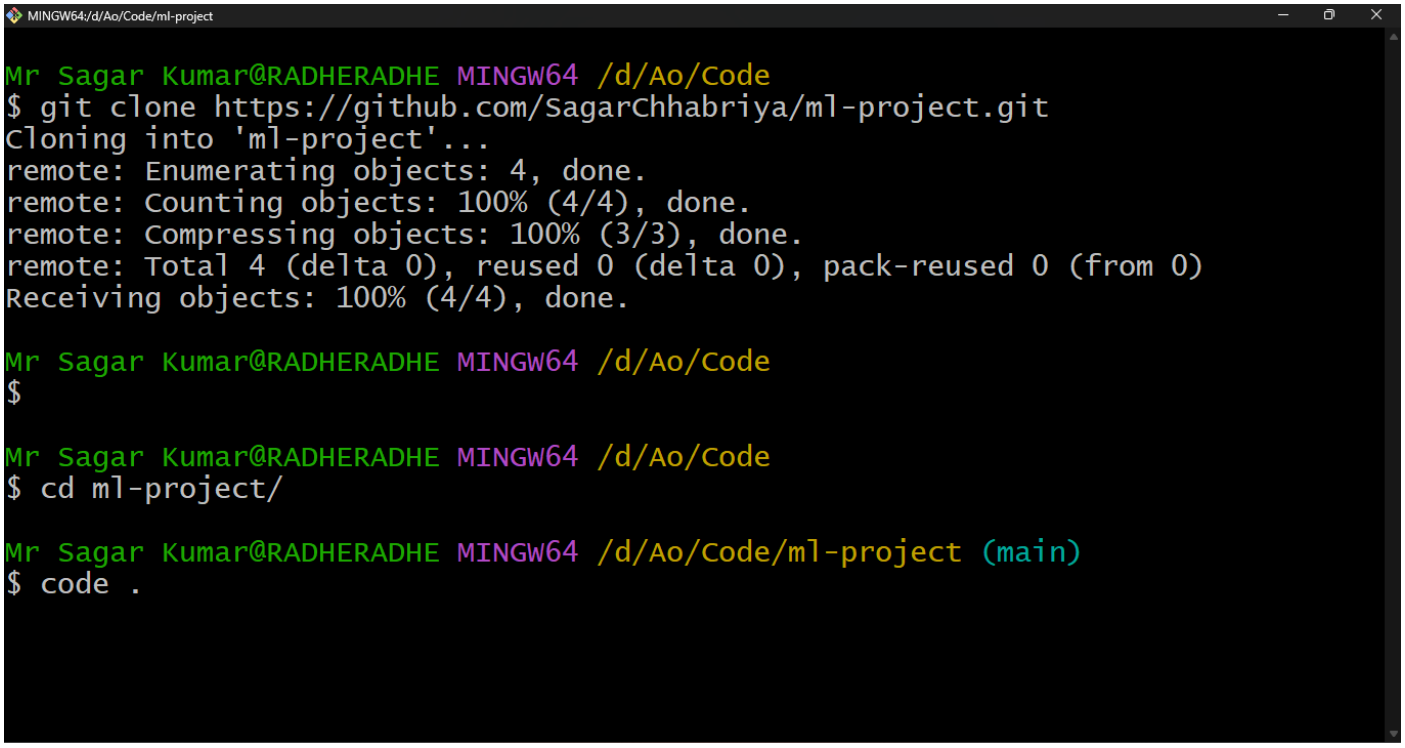
git config --global user.name “yourGitHubUserHandle”

4. Navigate to the cloned repository directory using the terminal: cd ml-project

   Note: If you're note familiar with commands, move to the next page.

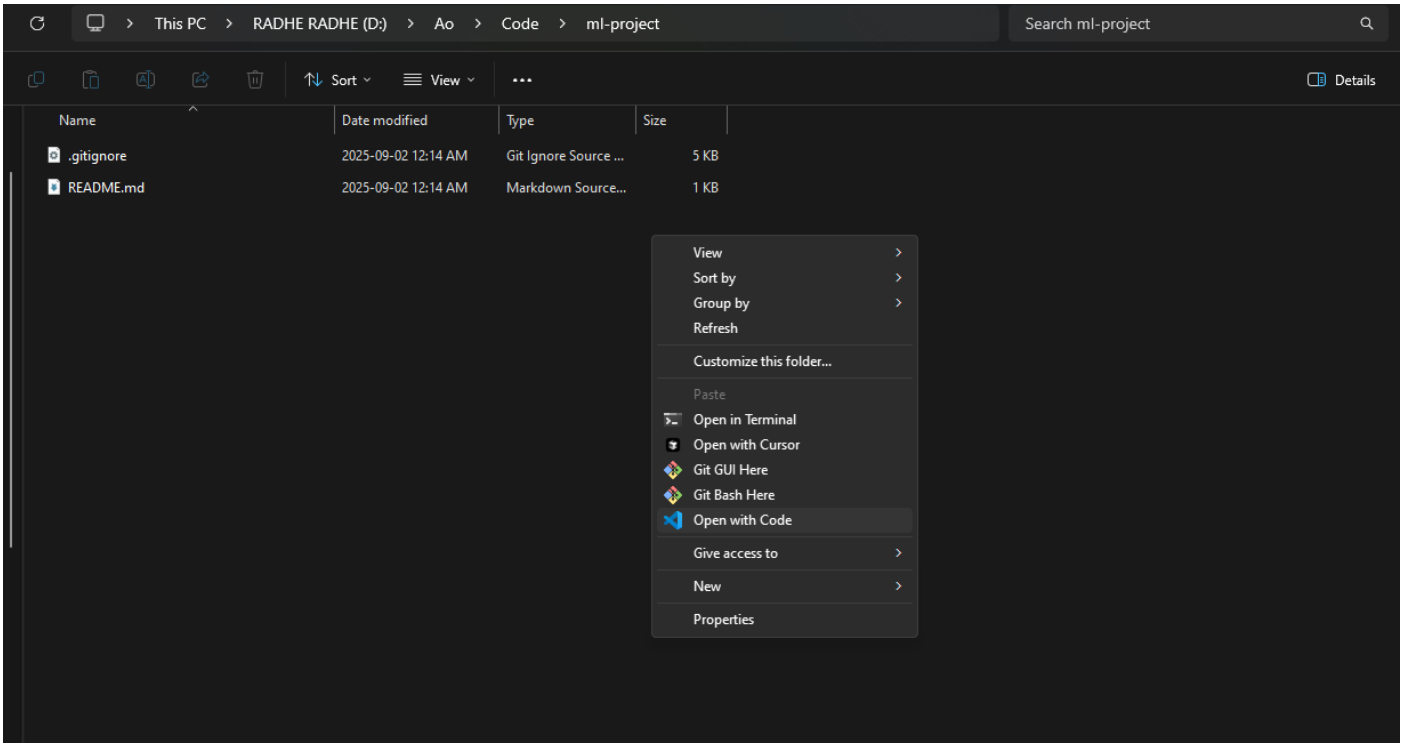



5. Open the repository in VS Code: type code . and hit enter. The VS Code will be opened auto.

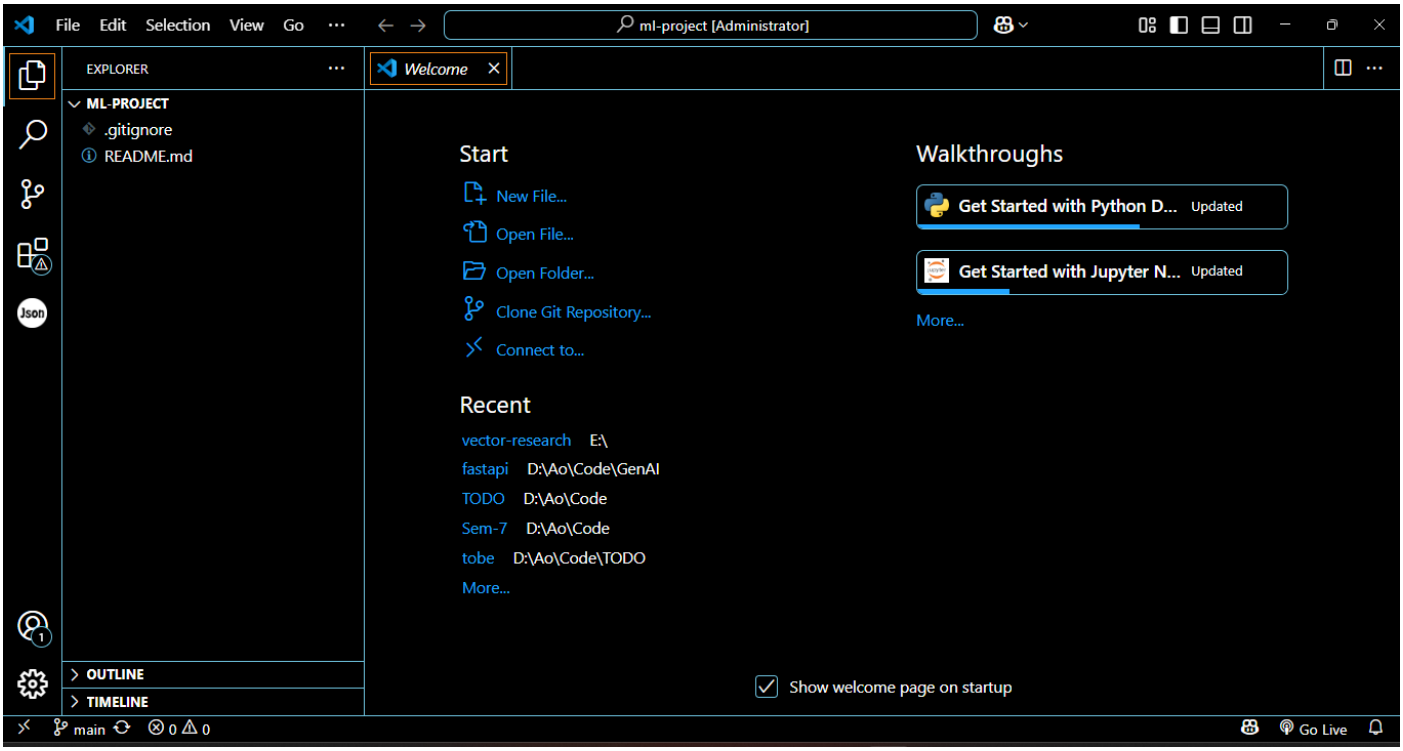


If you're not familiar with the commands, we also have an alternative to steps 4 and 5.

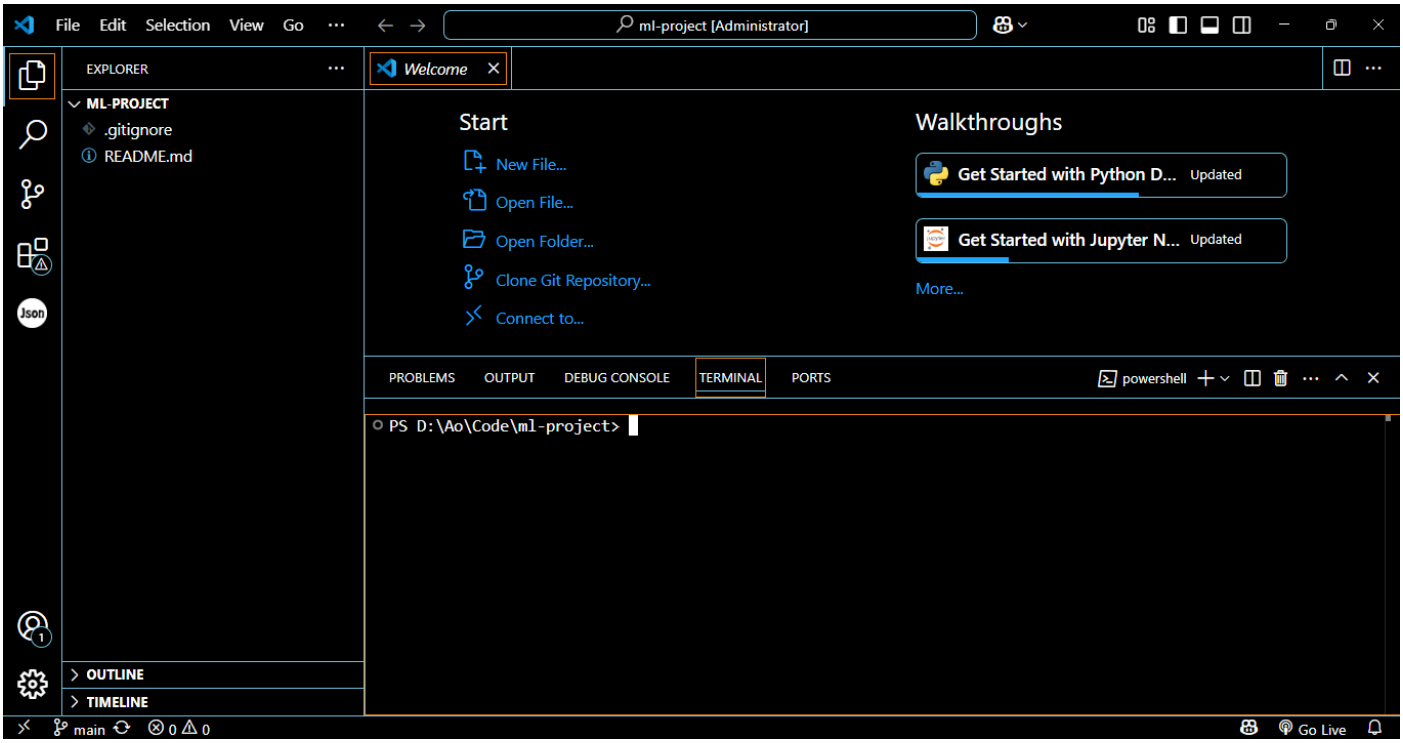
Open the project folder (i.e., directory) > Right Click > open with Code




VS Code Preview after opening in the project directory.



6. In VS Code, open the integrated terminal by pressing Ctrl + J (or use the menu Terminal > New Terminal). And follow the next steps.




## 2.3 Create and Activate a Virtual Environment

1. **Create Virtual Environment**:
   To isolate project dependencies, create a virtual environment (venv) in your project directory:
2. python -m venv my_venv

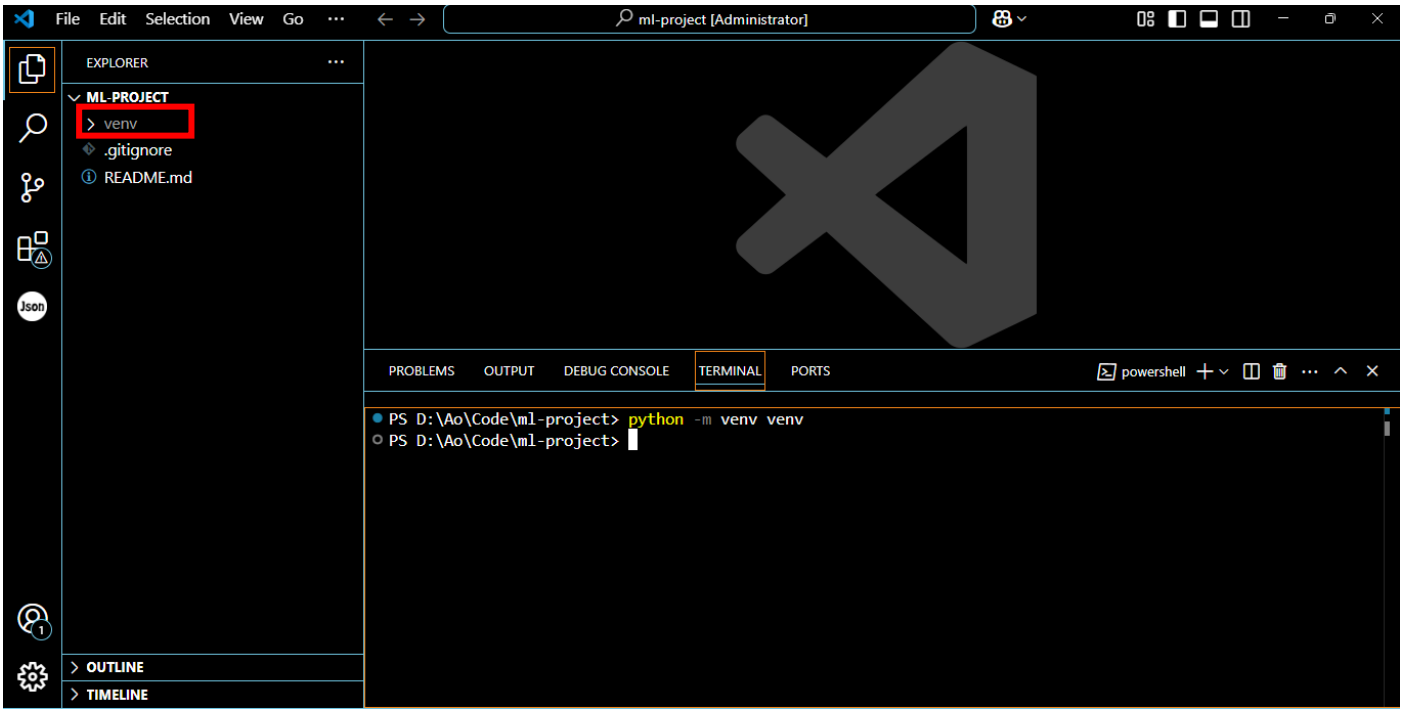
   Replace my_venv with the desired name for your virtual environment. Make sure python is installed and the also the required extensions.

3. **Activate the Virtual Environment**:
    - On Windows:

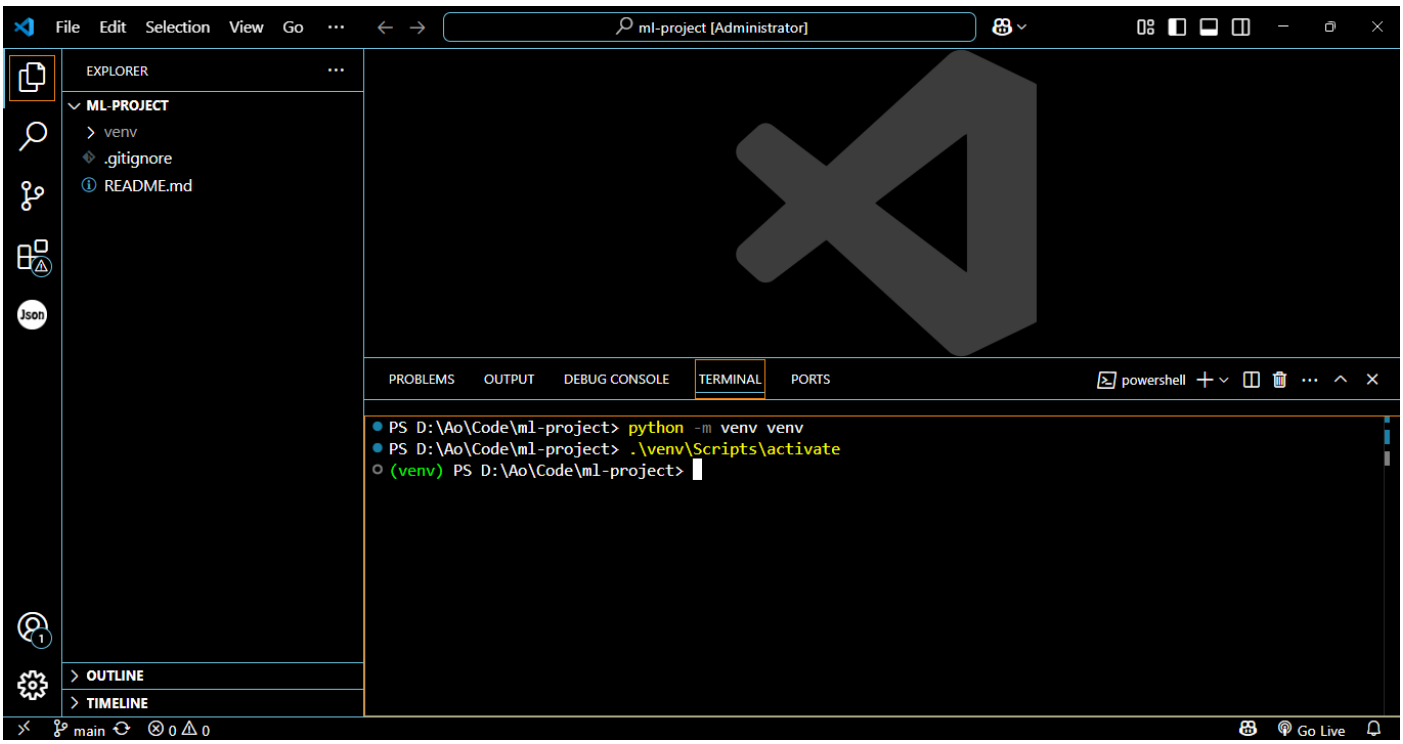
      my_venv\Scripts\activate

    - On macOS/Linux:

      source my_venv/bin/activate




Environment Activated (indicated by the (venv))



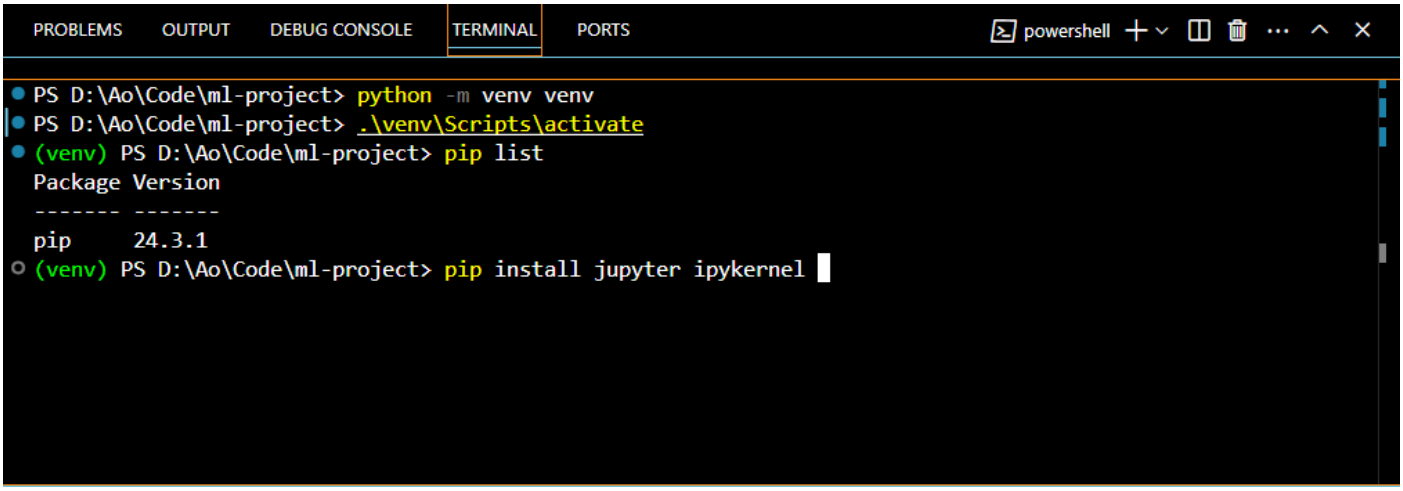
4. If you encounter the error **"not found"** in the PowerShell terminal of VS Code, follow these steps:
    - Run the following command to change the execution policy in PowerShell:
    - Set-ExecutionPolicy RemoteSigned -Scope Process
    - After running the above command, activate the virtual environment again:
    - my_venv\Scripts\activate

## 2.4 Install Required Python Libraries

Once the virtual environment is activated, you can install the required libraries for the project:

***pip list*** *will display the installed packages. In this case only pip is listed that is the default package.*


1. Install the necessary libraries by running:
2. pip install jupyter ipykernel pandas numpy scikit-learn hit enter after listing libraries




3. To install additional libraries, use the following command format:
    - For a single library:
4.
    - pip install <library_name>
    - For multiple libraries in a batch:
    - pip install <library1> <library2> <library3> ...

This completes the setup of your local environment. You are now ready to start working on the project, using the virtual environment to manage project-specific dependencies and tools.

# 3. Model Development

# Step 3. Model Development (model.ipynb)

The model development will be performed within a Jupyter notebook (model.ipynb) to enable interactive experimentation and evaluation. The following tasks outline the steps for building, training, evaluating, and optimizing the machine learning model.

## 3.1 Load Data and Import Libraries

1. **Import Libraries**: Begin by importing the necessary libraries for data manipulation, visualization, and machine learning. Common libraries include pandas, numpy, matplotlib, seaborn, and scikit-learn, among others.
2. **Load Data**: Load the dataset into a pandas DataFrame. Ensure the data is loaded correctly and is in a format suitable for preprocessing.

## 3.2 Data Preprocessing, Exploratory Data Analysis (EDA), and Train-Test Split

1. **Data Preprocessing**: Clean the data by handling missing values, encoding categorical variables, and scaling numerical features. Apply transformations as needed based on the type of data.
2. **Exploratory Data Analysis (EDA)**: Perform an initial analysis of the data to understand distributions, identify relationships, and spot outliers. This may involve generating summary statistics and visualizations.
3. **Train-Test Split**: Split the dataset into training and testing sets. The typical ratio for splitting is 80/20 or 70/30, ensuring that the model can be trained on one portion and evaluated on another.

## 3.3 Model Training

1. **Model Selection**: Choose an appropriate machine learning model based on the task (e.g., classification, regression). Common models include Random Forest, Support Vector Machines, or Linear Regression.
2. **Train the Model**: Fit the model using the training dataset. Ensure that the model is initialized with appropriate hyperparameters and trained on the available features and target variable.

## 3.4 Model Evaluation

1. **Evaluate Performance**: Assess the model's performance using appropriate evaluation metrics. For classification tasks, consider metrics like accuracy, precision, recall, F1-score, and confusion matrix. For regression tasks, evaluate using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), or R-squared.
2. **Further Validation**: If needed, perform additional validation like cross-validation or calculate other relevant metrics to confirm the model’s robustness.

## 3.5 Testing on New Instances (Experiment Mode)

1. **Experiment Mode**: After model training and evaluation, test the model on new data points (instances) in a Jupyter notebook environment. This can be done by passing individual or small batches of new data through the model to generate predictions interactively.

## 3.6 Model Optimization: Hyperparameter Tuning

1. **Hyperparameter Tuning**: To improve model performance, experiment with hyperparameter optimization techniques such as grid search or random search. This step involves selecting the best combination of hyperparameters for the chosen model by evaluating the model on different configurations.

## 3.7 Model Serialization

1. **Model Serialization**: Once the model has been trained and optimized, serialize it using a standard format (e.g., Pickle or Joblib). This step allows the model to be saved to a file, making it available for later use without the need for retraining.

**Model Loading**: Needed in the next step. Load the saved model into memory for inference or further analysis, ensuring that the same version of the model is used consistently.

# 4. Streamlit UI Development

# Step 4. Streamlit UI Development (app.py)

The user interface for the model will be built using **Streamlit**, which allows for quick and easy creation of interactive web applications for data science projects. The following tasks outline the steps to set up the Streamlit UI:

## 4.1 Import Libraries

1. **Import Necessary Libraries**: Begin by importing the required libraries for building the app, including:
   - **Streamlit** for UI components and interaction.
   - Libraries like **joblib** or **pickle** for loading the trained model.

   Example libraries to import:

   - streamlit as st
   - joblib (or pickle for model loading)
   - Any other necessary libraries for handling data preprocessing or visualization.

## 4.2 Set Page Title and Document Title

1. **Page Title**: Use Streamlit's built-in functions to set the title of the app's main page.
2. **Document Title**: Optionally, set the document title for the browser tab, which will appear in the title bar of the user's browser.

   Example:

   - Use st.set_page_config() to define the app's title and other page configurations.

## 4.3 Load Model

1. **Load the Trained Model**: Load the serialized model that was saved earlier (in Step 3: Model Development) using a library like **joblib** or **pickle**. This allows the Streamlit app to use the model for making predictions.

   Example:

   - Use joblib.load() or pickle.load() to load the model from a saved file.

## 4.4 Get Input via Streamlit Input Functions

1. **User Input**: Use Streamlit's input functions to gather user input for making predictions. This may include:
   - **Text Input**: For entering numeric or text-based data.
   - **Selectbox** or **Radio Buttons**: For selecting categorical data.
   - **Sliders**: For selecting numerical values within a range.

   Example:

   - Use st.text_input(), st.selectbox(), st.slider(), or similar functions to collect the required inputs.

## 4.5 Make Predictions and Display Results

1. **Make Predictions**: Pass the collected user input to the loaded model and make predictions based on the input data.
2. **Display Results**: After obtaining the prediction, display the results to the user using Streamlit's built-in functions like st.success(), st.error(), or st.write(). These functions provide simple ways to communicate the results back to the user in a clear and informative way.

   Example:

   - Display success messages using st.success() for correct predictions or st.error() for invalid inputs.

# 5.0 Deployment

# Step 5. Deployment

Once the model and Streamlit application are developed, the next step is to prepare the project for deployment. This section outlines the steps for creating the requirements.txt file and ensuring the correct dependencies for deployment.

## 5.1 Create **requirements.txt**

1. **Generate requirements.txt**:
   The requirements.txt file lists all the necessary Python libraries and their corresponding versions that are required to run the application and make predictions.
2. **Determine Package Versions**:
   Identify the version of each library used in the project, such as numpy, pandas, scikit-learn, streamlit, and any other dependencies required for the app. This can be done by importing each package in a Python script or notebook and calling the .__version__ attribute.

   Example:

   ```
   import numpy as np
   print(np.__version__)  # Returns the version of numpy
   ```

3. **Manually Add to requirements.txt**:
   Based on the version information obtained, manually create the requirements.txt file by specifying each library and its version. For example:

   ```
   numpy==1.21.2
   pandas==1.3.3
   scikit-learn==0.24.2
   streamlit==0.87.0
   joblib==1.0.1
   ```

   **Note**: Avoid using pip freeze > requirements.txt, as it may include unnecessary libraries that were not used during the project. And make sure the filename spell is correct *requirement**s.**txt.*

## 5.2 Prepare for Deployment

1. **Ensure all dependencies are listed**: Verify that all the libraries you have used throughout the project, including those required for model loading, data preprocessing, and the Streamlit UI, are included in the requirements.txt file.
2. **Finalize project structure**: Ensure that the project folder contains all necessary files (such as app.py, model.pkl, and requirements.txt) in an organized manner, ready for deployment.

# 6. Streamlit Community Cloud

# Step 6. Streamlit Community Cloud

The Streamlit Community Cloud provides a platform to deploy your Streamlit app online for public access. This section outlines how to deploy the app on Streamlit Community Cloud using GitHub.

## 6.1 Continue with GitHub

1. **Push Code to GitHub**:
   Before deploying on Streamlit, ensure your project code (including app.py, requirements.txt, and any other necessary files) is pushed to a GitHub repository. If the repository does not already exist, create a new one and upload the project files.
2. **Ensure Public Access**:
   Make sure the repository is **public** or that you have set the correct access permissions to allow Streamlit Community Cloud to access it.

## 6.2 Create Streamlit App

1. **Sign in to Streamlit Community Cloud**:
   Go to Streamlit Community Cloud and sign in using your GitHub account.
2. **Create a New App**:
   - Click on **"New app"**.
   - Select the GitHub repository where your project is stored.
   - Choose the main file for your Streamlit app (e.g., app.py).
   - Set up any other configuration settings as needed.
3. **Deploy the App**:
   Once the repository is linked, Streamlit will automatically deploy your app. You will be provided with a unique URL where your app can be accessed.
4. **Test the Deployment**:
   After deployment, test the app on the provided URL to ensure that the model works correctly, the UI is responsive, and the predictions are accurate.

***End of Project***