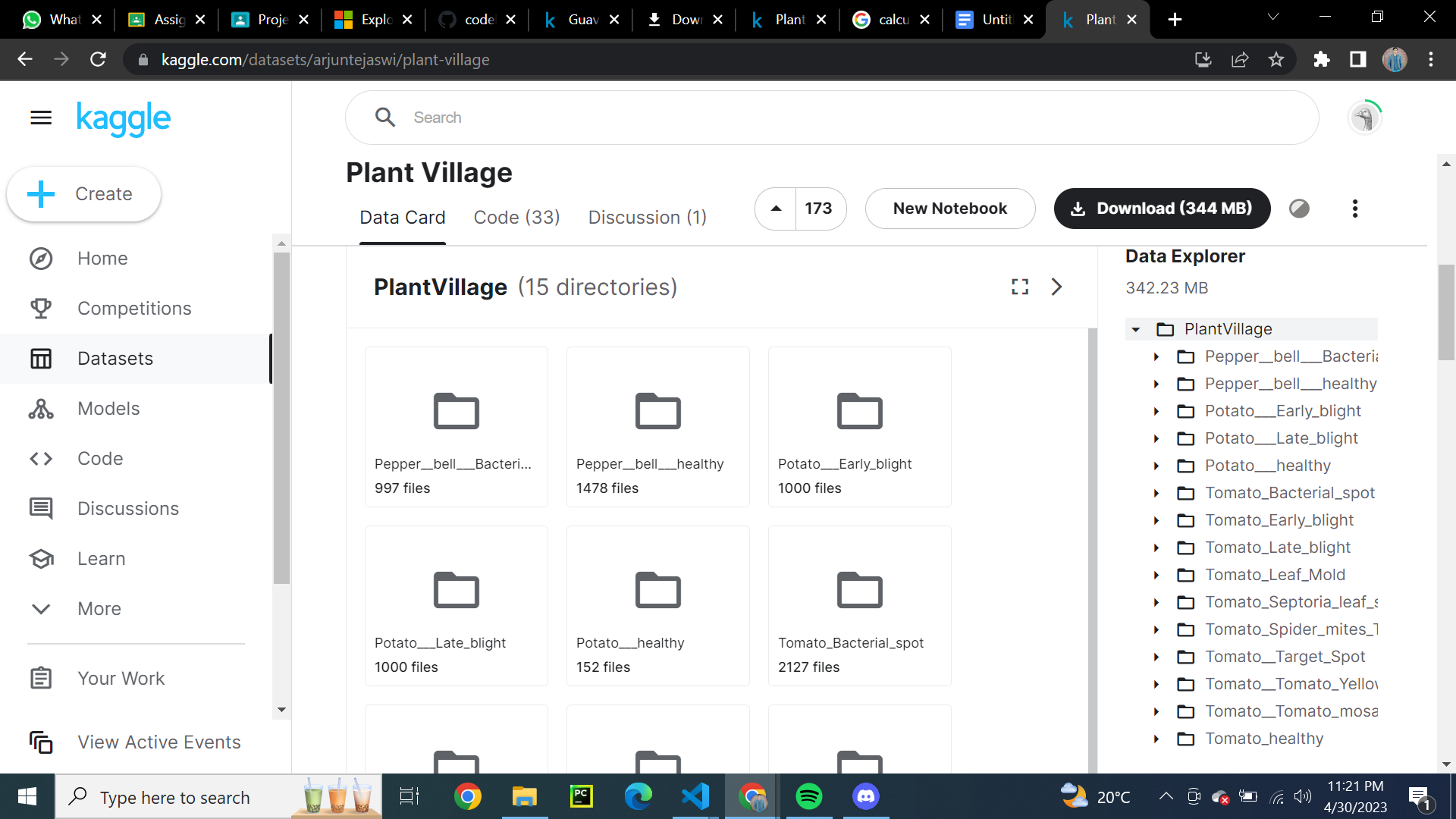
**Digital Image Processing  
Project**

**Talha Rashid**

**19i-1986-A**

**Data Collection:**

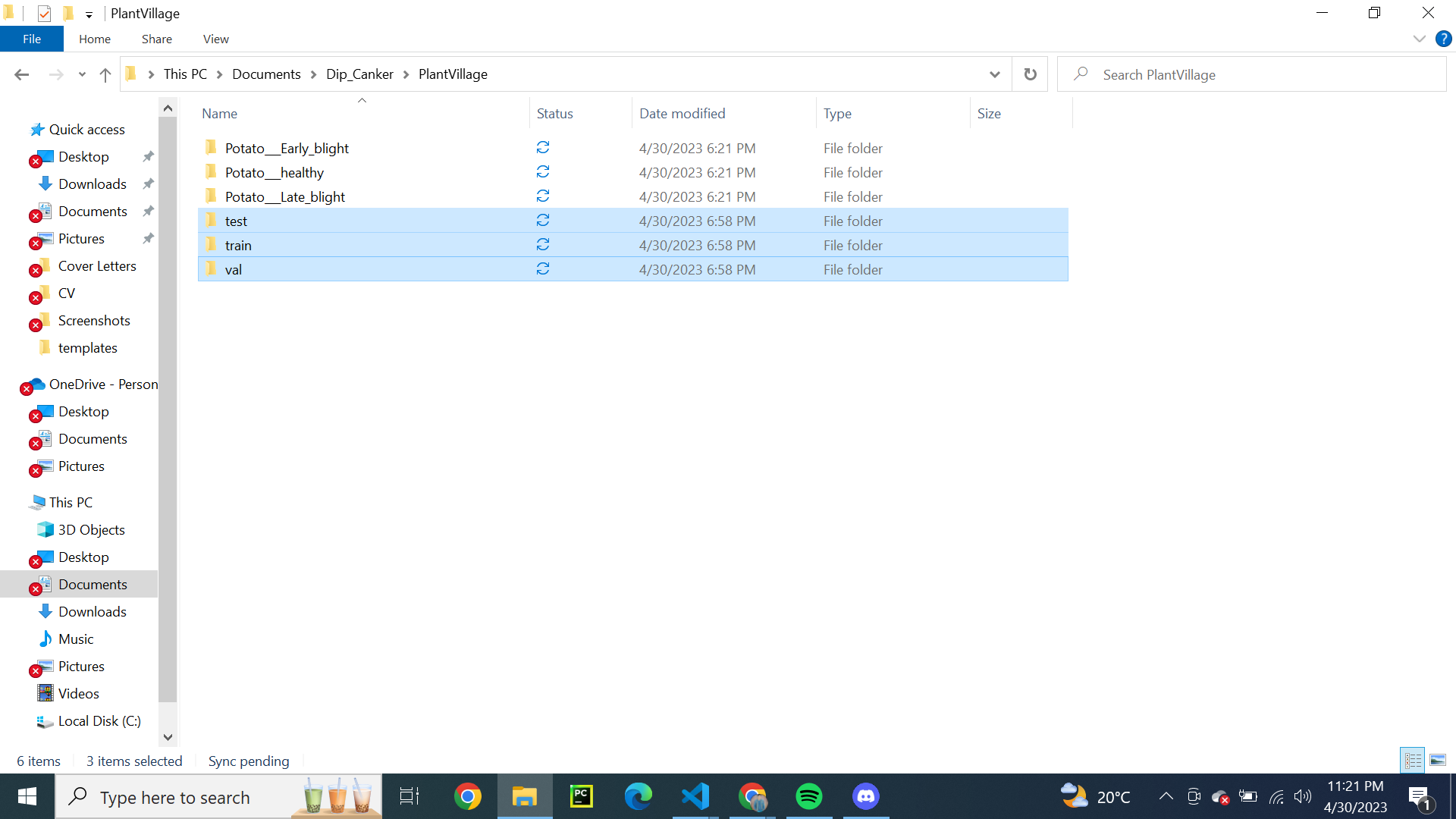
The initial steps involves the collection and selection of data for the "**Plant Village**" dataset available on Kaggle(https://www.kaggle.com/datasets/arjuntejaswi/plant-village). The dataset contains various plant images with different diseases and healthy ones. The code focuses on selecting only three classes, namely *early blight*, *late blight*, and *healthy plants*.



**Data Preprocessing:**

The first step is to import the necessary libraries, including the "*tensorflow*",”*keras*” library for image manipulation and the "*split folders*" library for high-level file operations. The next step is to set the paths for input and output directories. The code then iterates through the directories and selects only the required classes.

This code uses the `*split folders*` library to split a dataset located in the `*input\_folder*` directory into train, validation, and test sets, and save the resulting subsets in the `*output\_folder*` directory. The split is done randomly with a seed value of 42 and a ratio of 80% for the training set, 10% for the validation set, and 10% for the test set.



Overall, this code is useful for preparing a dataset for machine learning by splitting it into appropriate subsets for training, validating, and testing a model.

This code is using the `***ImageDataGenerator***` class from the `*tensorflow.keras.preprocessing.image*` module to generate augmented images for a training dataset.

The `***train\_datagen***` object is created with several parameters that define the types of augmentations to be performed on the images. The `rescale` parameter scales the pixel values of the images to a range of 0 to 1, while `rotation\_range` and `horizontal\_flip` are used to randomly rotate and flip the images during training.

The `***train\_generator***` object is created using the `*flow\_from\_directory*` method, which takes as input the directory path of the training dataset, the target size of the images, the batch size, the class mode (in this case, "sparse" since the labels are integers), and the directory path to which the augmented images will be saved.

The `train\_generator` object can then be used to train a machine learning model using the augmented images generated by the `ImageDataGenerator`. By using augmented images during training, the model can learn to better generalize to new, unseen images and improve its overall accuracy.

Same process of augmentation is repeated for the test and validate dataset.

**Building the model:**

This code defines a sequential model using the Keras API with a specified input shape and number of output classes. The model consists of multiple Conv2D and MaxPooling2D layers, followed by a flatten layer and two dense layers with activation functions of relu and softmax, respectively. The model uses the 'relu' activation function to introduce non-linearity and 'softmax' function to produce probability outputs for each class. The model is intended to be used for classification tasks on images with dimensions specified by IMAGE\_SIZE and CHANNELS.

**Compiling Model:**

Then the `compile()` method is used to configure the learning process of the model. In this code, the model is compiled with:

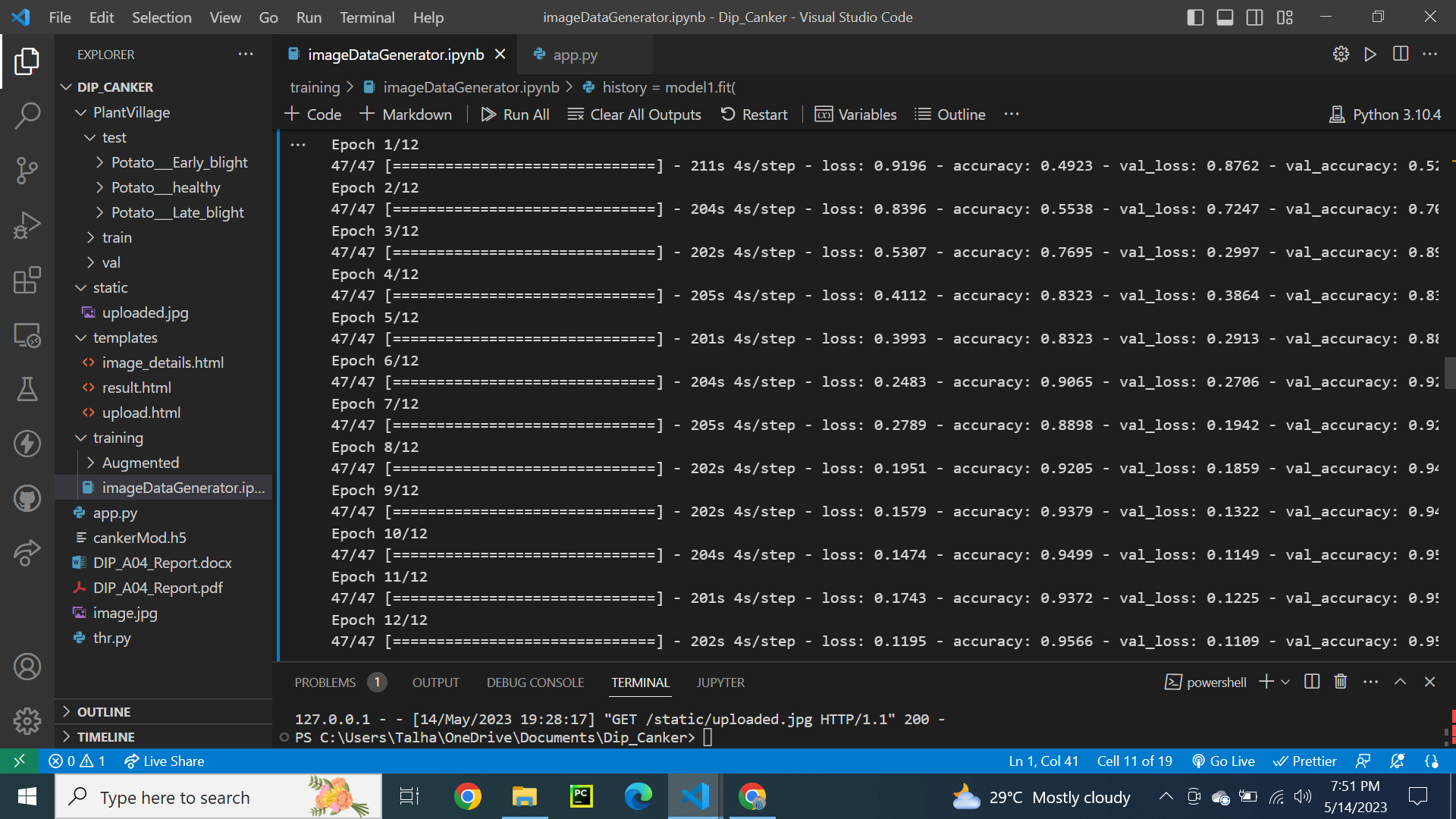
- Optimizer: Adam, which is a popular optimization algorithm that is based on adaptive estimates of lower-order moments.

- Loss function: Sparse categorical crossentropy, which is a common loss function used for multiclass classification problems.

- Metrics: Accuracy, which is a performance metric used to evaluate the performance of the model.

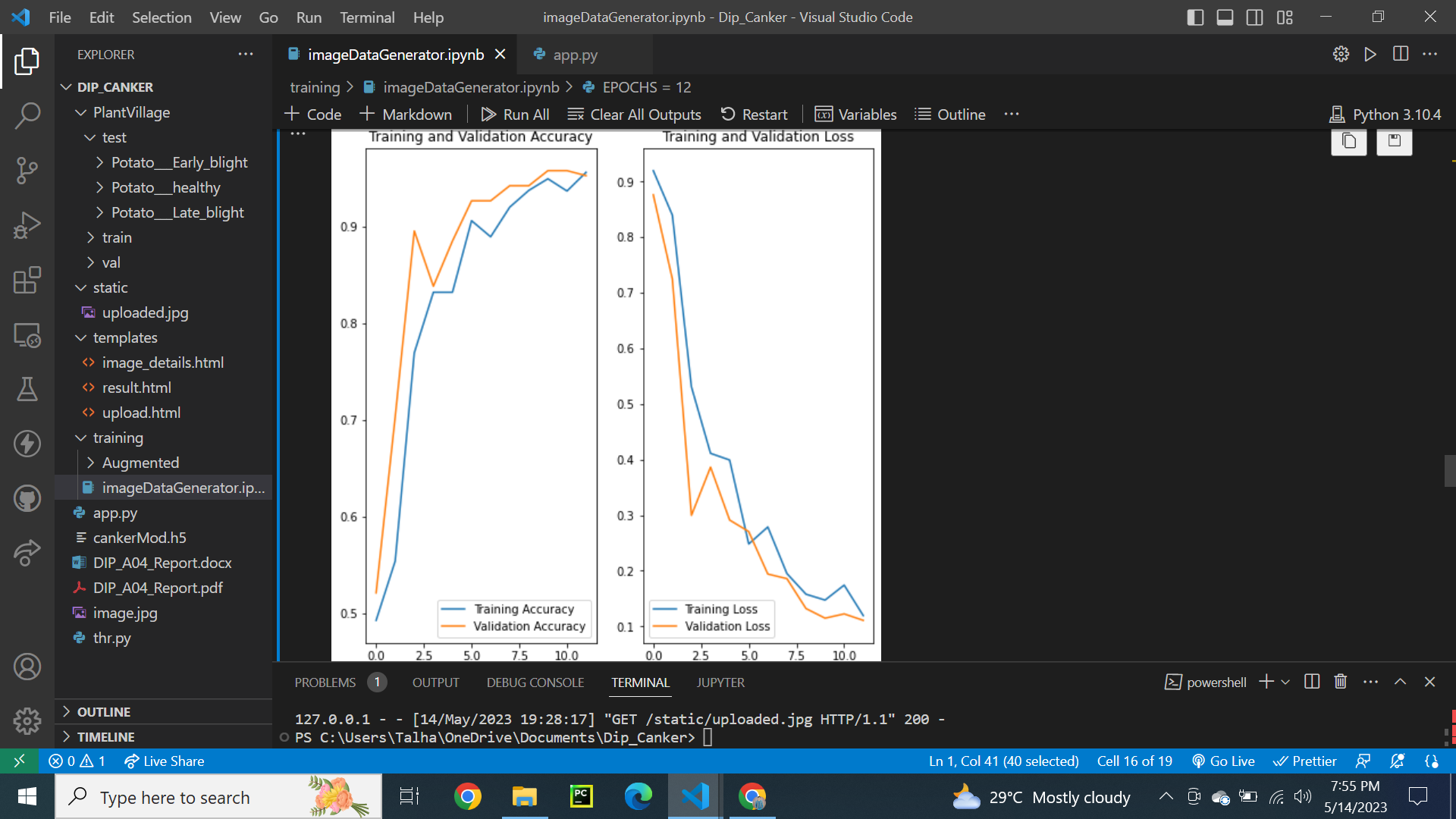
**Model Training:**

The code trains a deep learning model (model1) for detecting canker disease in potato leaves using a dataset of images. The model is trained using the fit() function with 47 steps per epoch, a batch size of 32, and validation data consisting of 6 steps. The training process is verbose, with 12 epochs. The trained model is saved to a file named "cankerMod.h5".



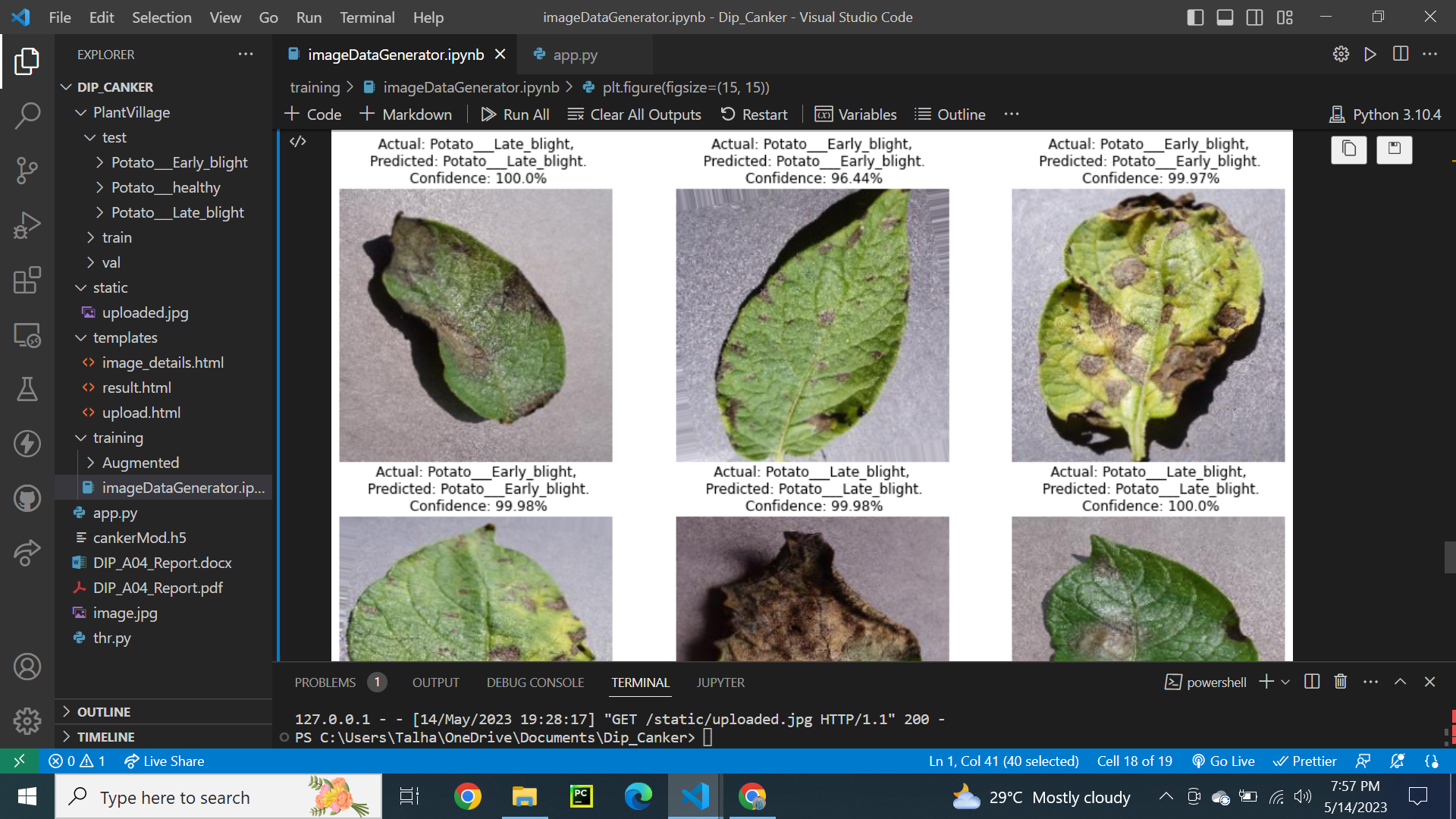
**Model Evaluation:**

The given code evaluates the performance of a trained deep learning model on a test dataset and reports the test loss and accuracy. It then plots the training and validation accuracy as well as the training and validation loss curves over the epochs. The plots help in understanding the model's learning behavior and whether the model is overfitting or not. The plot shows that the training and validation accuracy and loss improve over epochs and there is no overfitting.



**Testing Few Samples:**

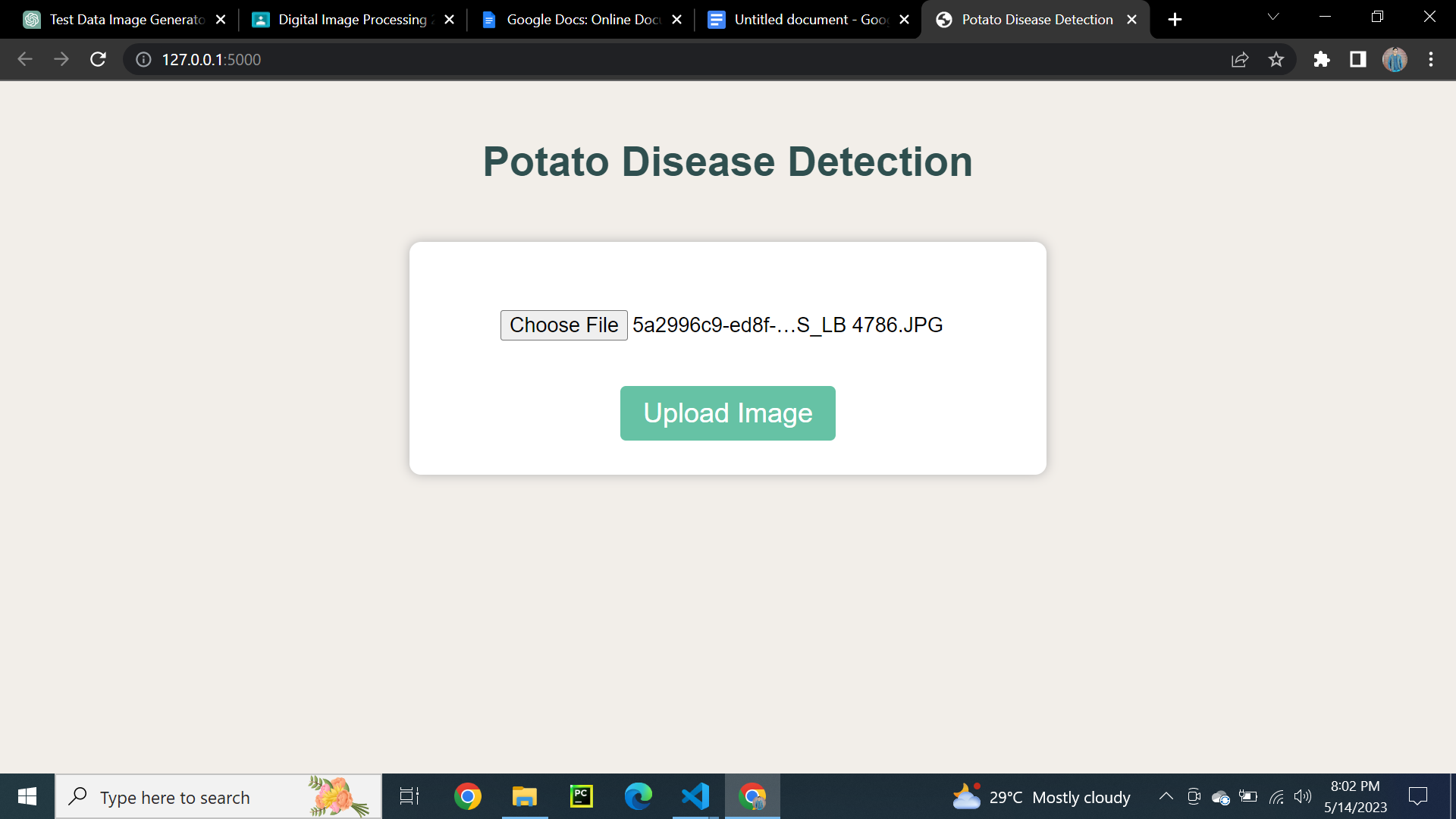
The above code is for generating a visualization of the predictions made by the model on a sample of test images. The code creates a 3x3 grid of test images and displays the actual class, predicted class, and the confidence level of the prediction for each image. The `predict` function takes in a model and an image and returns the predicted class and the confidence level of the prediction. Overall, this code allows for easy visual inspection of the model's performance on a sample of test images. And in the end we are saving the model.



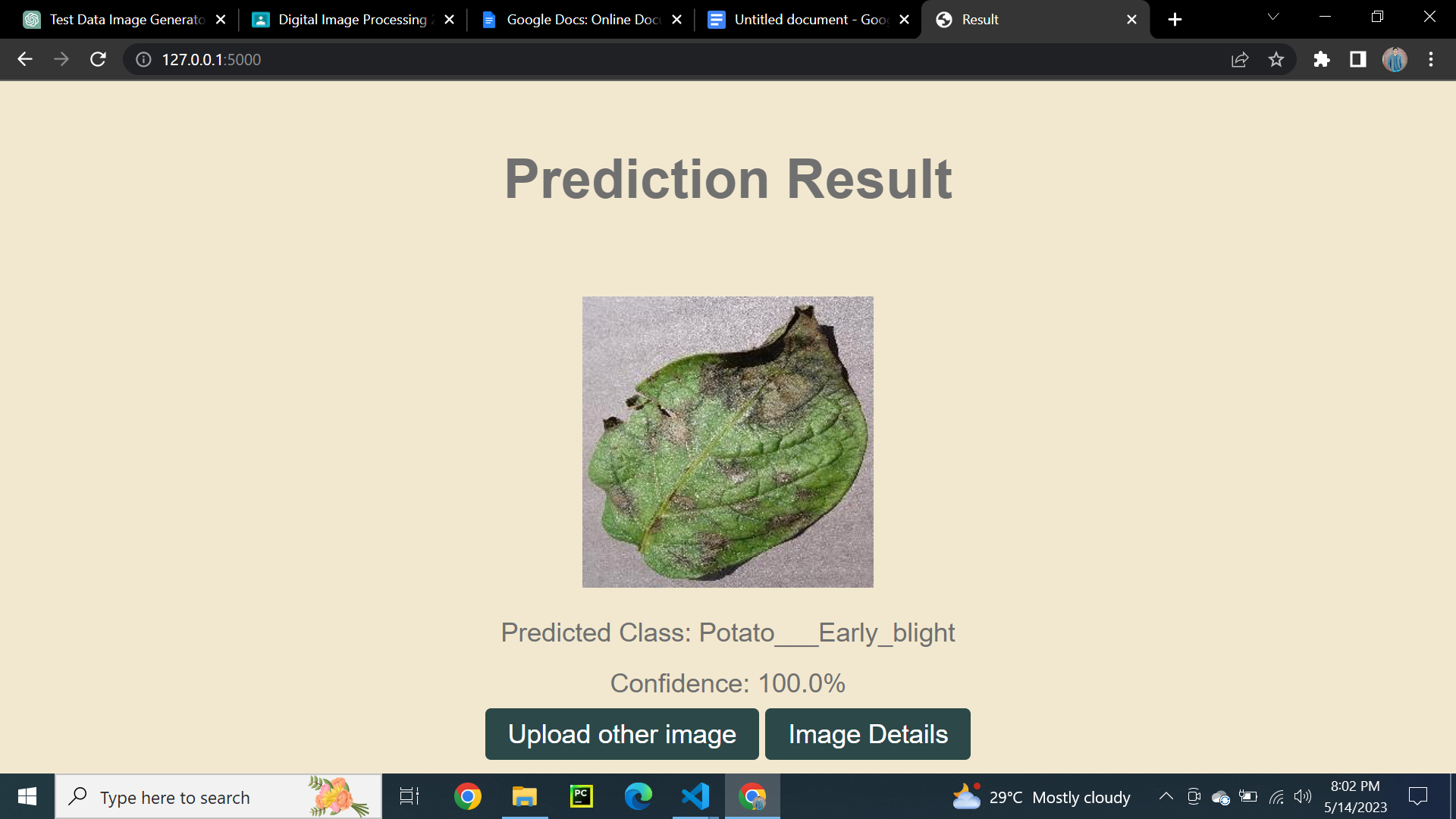
**Flask API:**

Then there is a code for Flask application that allows users to upload an image of a potato leaf and get predictions about whether it is healthy or has early or late blight disease. The code loads a pre-trained Keras model and applies a Laplacian filter to enhance edges, and then sharpens the image to make it clearer for predictions. After preprocessing, the image is fed into the pre-trained model and the predicted class and confidence are calculated. The result is then rendered on a result.html page. The application also has a page to display image details, where the uploaded image is filtered using OpenCV to highlight the safe areas of the leaf. The application can be run locally on the Flask server by running the script.

**Uploading Image:**



**Prediction Results:**



**Analysis(Further Details):**

