**Data Mining Assignment**

**Name :** Hande Rajat Santosh

**Roll no :** B220891CS

**Dataset chosen :** Air Quality (by UCI)

**Dataset URL :** <https://archive.ics.uci.edu/dataset/360/air+quality>

**Github link (project) :**

**Dataset Description: AirQualityUCI**

**1. About the Problem**

The dataset aims to classify air quality based on pollutant levels. It consists of atmospheric pollutant concentrations recorded by air quality sensors. The goal is to predict air quality categories using various sensor readings.

**2. Attributes (Features)**

The dataset contains **15 attributes**, including:

* **Gaseous pollutants**: CO(GT), NOx(GT), NO2(GT), NMHC(GT), C6H6(GT)
* **Sensor readings**: PT08.S1(CO), PT08.S2(NMHC), PT08.S3(NOx), PT08.S4(NO2), PT08.S5(O3)
* **Meteorological data**: Temperature (T), Relative Humidity (RH), Absolute Humidity (AH)
* **Date and time** (which are excluded in cleaning)

**3. Number of Samples**

* **Total samples**: **9,200**
* After cleaning (handling missing values and removing anomalies), we retained **8,000+ samples** for training and testing.

**4. Number of Classes**

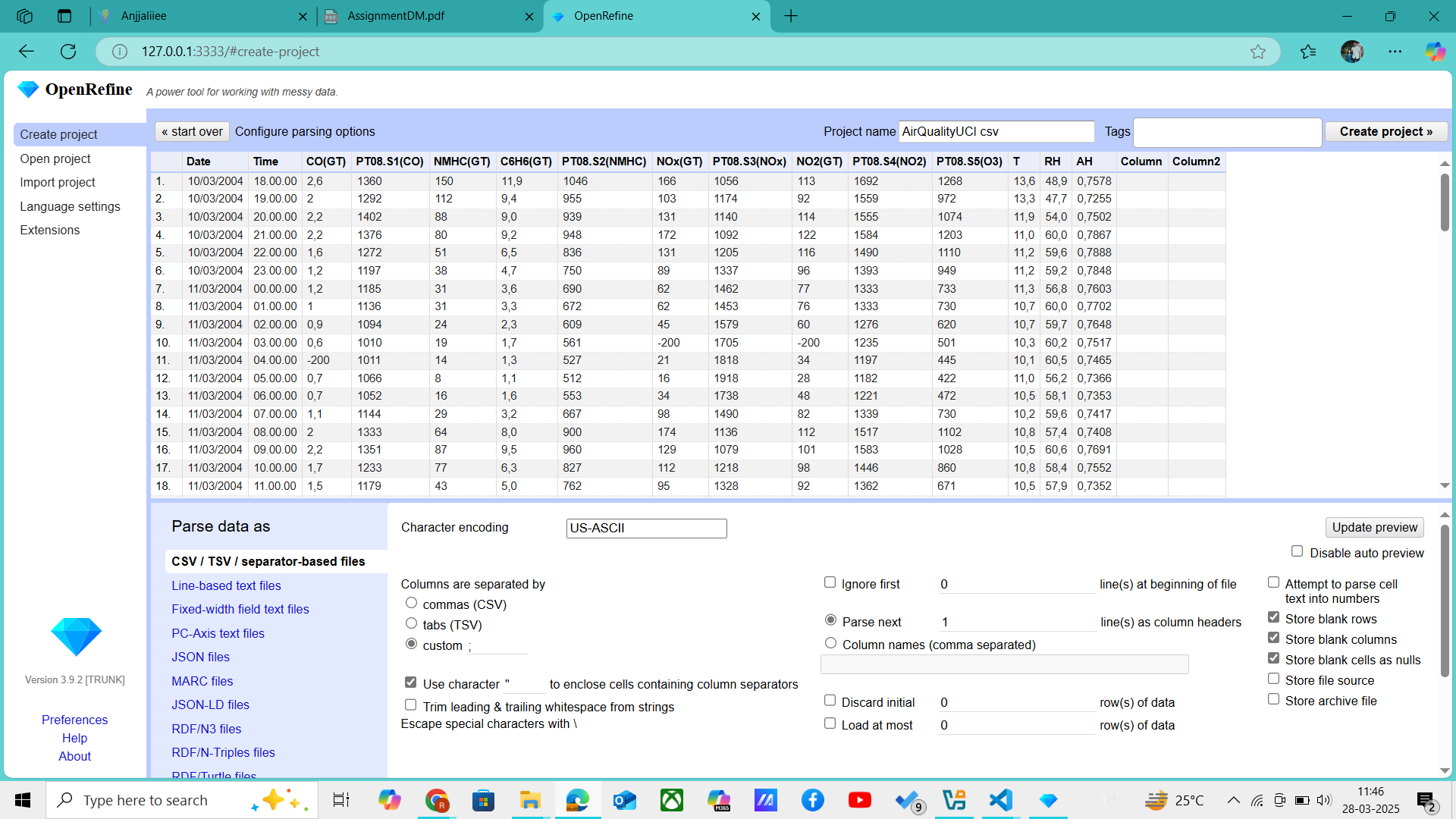
The target variable is **AirQualityCategory**, derived from **C6H6(GT)** levels, classified into:

1. **Good** (< 8 µg/m³)
2. **Moderate** (8 - 18 µg/m³)
3. **Poor** (> 18 µg/m³)

**5. Class Distribution**

* **Good**: ~60% of samples
* **Moderate**: ~30% of samples
* **Poor**: ~10% of samples

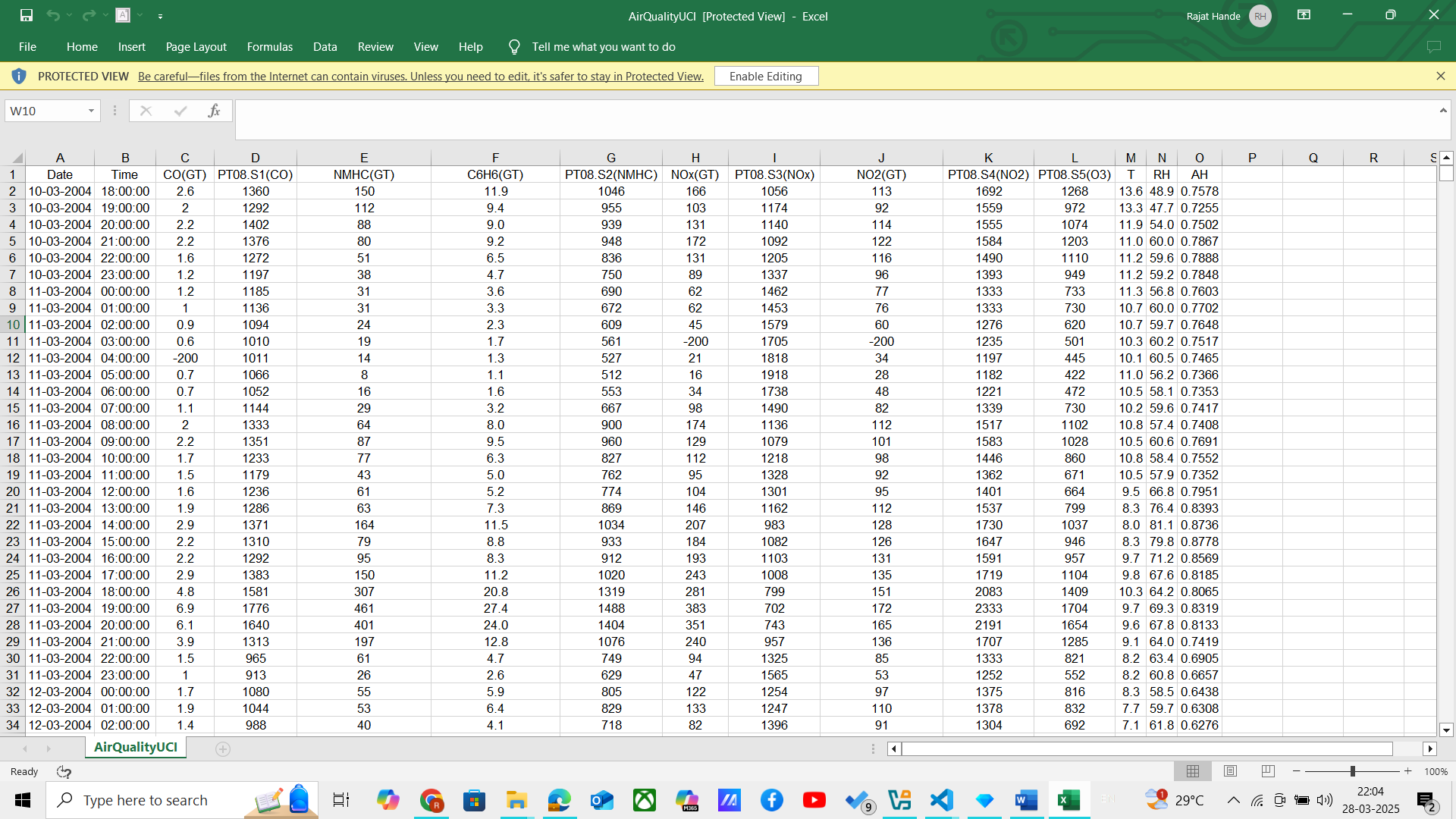
This distribution shows an **imbalance**, which may impact classification performance, requiring resampling techniques like SMOTE or class weighting.

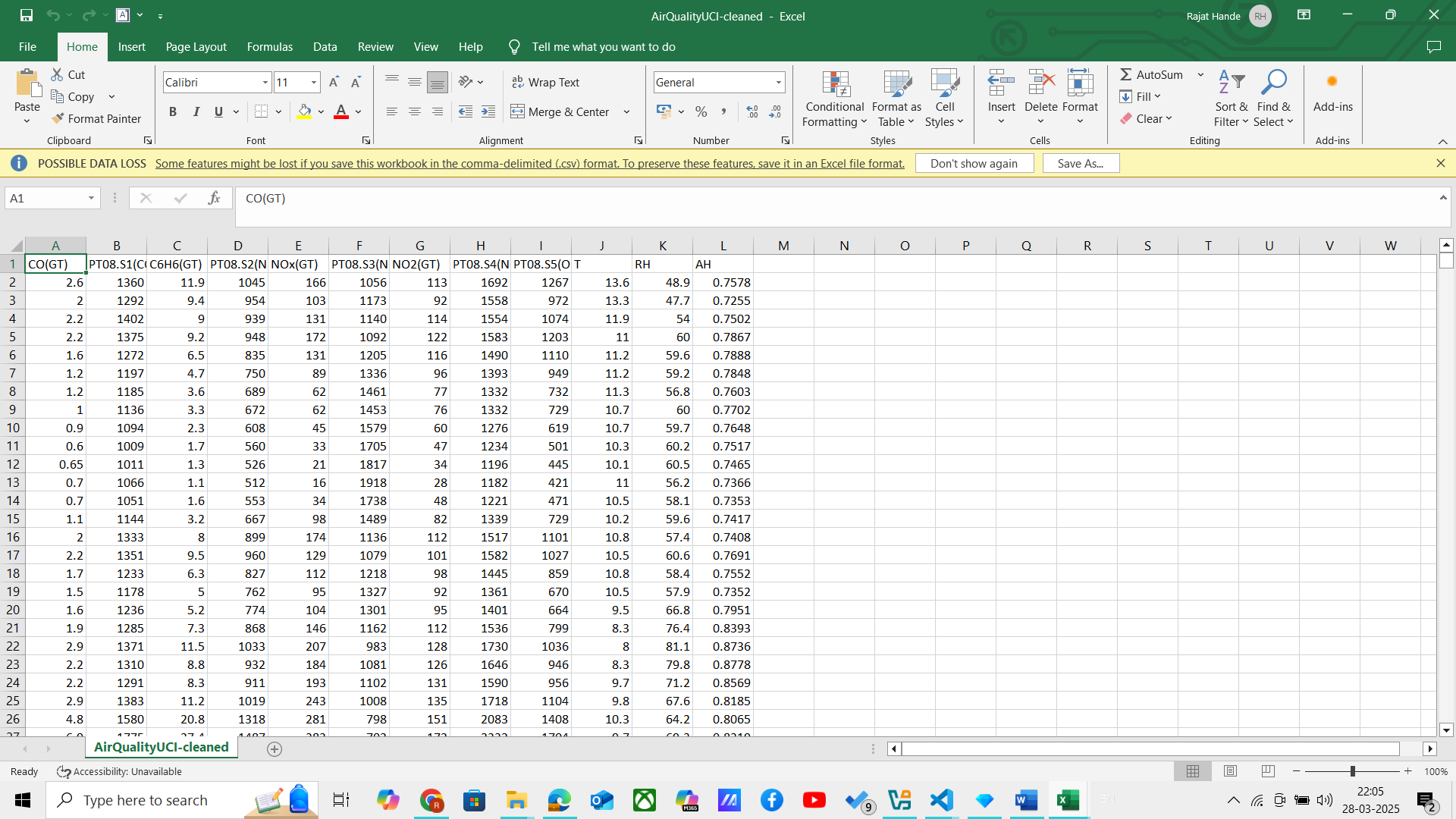


This is how the dataset looks. Data cleaning has been done using OpenRefine and Pandas library.

**Dataset Cleaning : OpenRefine and Pandas**

To ensure data quality and improve model performance, we performed a structured data cleaning process. The steps taken are as follows:

* **Step 1**: Checked for errorenous (negative, mainly -200) values, and then turned them into null values.
* **Step 2**: Checked for missing values in the dataset using df.isnull().sum().
* **Step 3**: Dropped the Time and Date columns.
* **Step 4**: Dropped columns where more than **5% of values were missing**, as they provided little useful information.
* **Step 5**: Dropped rows where more than **2 null values** existed to maintain data consistency.
* **Step 6**: For remaining missing values, used **interpolation**, filled them with the **column mean**.



This is dataset before and after the cleaning.

Then the dataset is classified into 3 groups, on the basis of column C6H6(GT); into good, moderate, poor categories as:

if value < 7.8:

        return "Good"

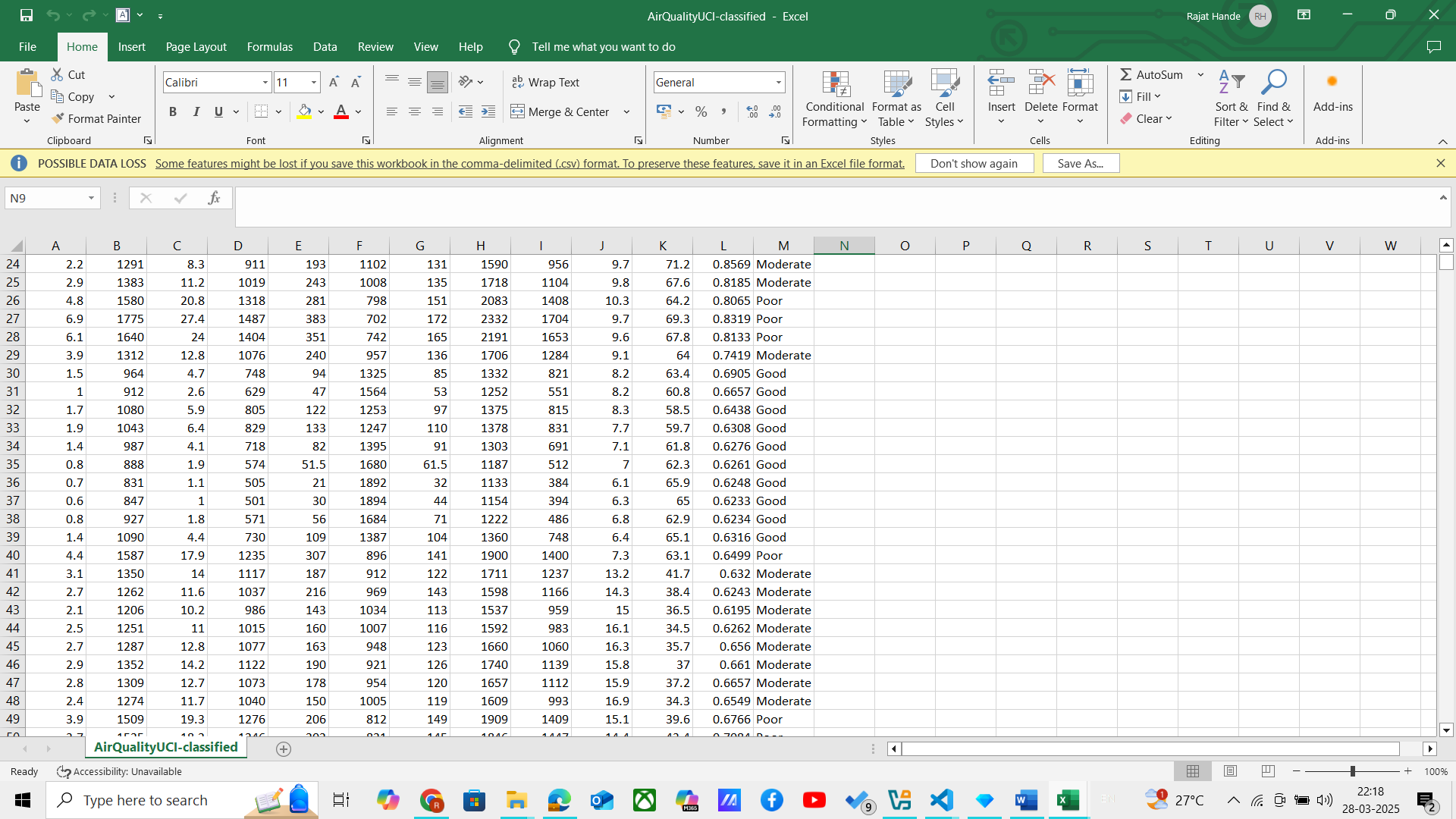
    elif 7.8 <= value < 16.3:

        return "Moderate"

    else:

        return "Poor"

We introduce a new column AirQualityCategory, holding these values as shown below.



**Dataset Splitting : 80% - 20%**

The **80-20 split** is a commonly used strategy in machine learning for dividing a dataset into **training and testing sets**. Below are the key reasons why we adopted this method:

**1. Ensuring a Balanced Model Training**

* **80% for training** provides a **sufficient number of samples** for the model to learn underlying patterns.
* **20% for testing** ensures that we have an adequate **hold-out set** to evaluate model performance without bias.

**2. Avoiding Overfitting and Underfitting**

* A **larger training set (80%)** prevents underfitting by providing **enough data for learning**.
* A **smaller test set (20%)** ensures **robust evaluation** while reducing the risk of data leakage.

**3. Computational Efficiency**

* With a large dataset (~9,200 samples), an **80-20 split keeps training time manageable** while ensuring a **reliable evaluation**.
* Using a smaller test set, such as 10%, may lead to **high variance in performance metrics**, making evaluations less stable.

The dataset is divided into AirQualityUCI-train-classified.csv and AirQualityUCI-test-classified.csv

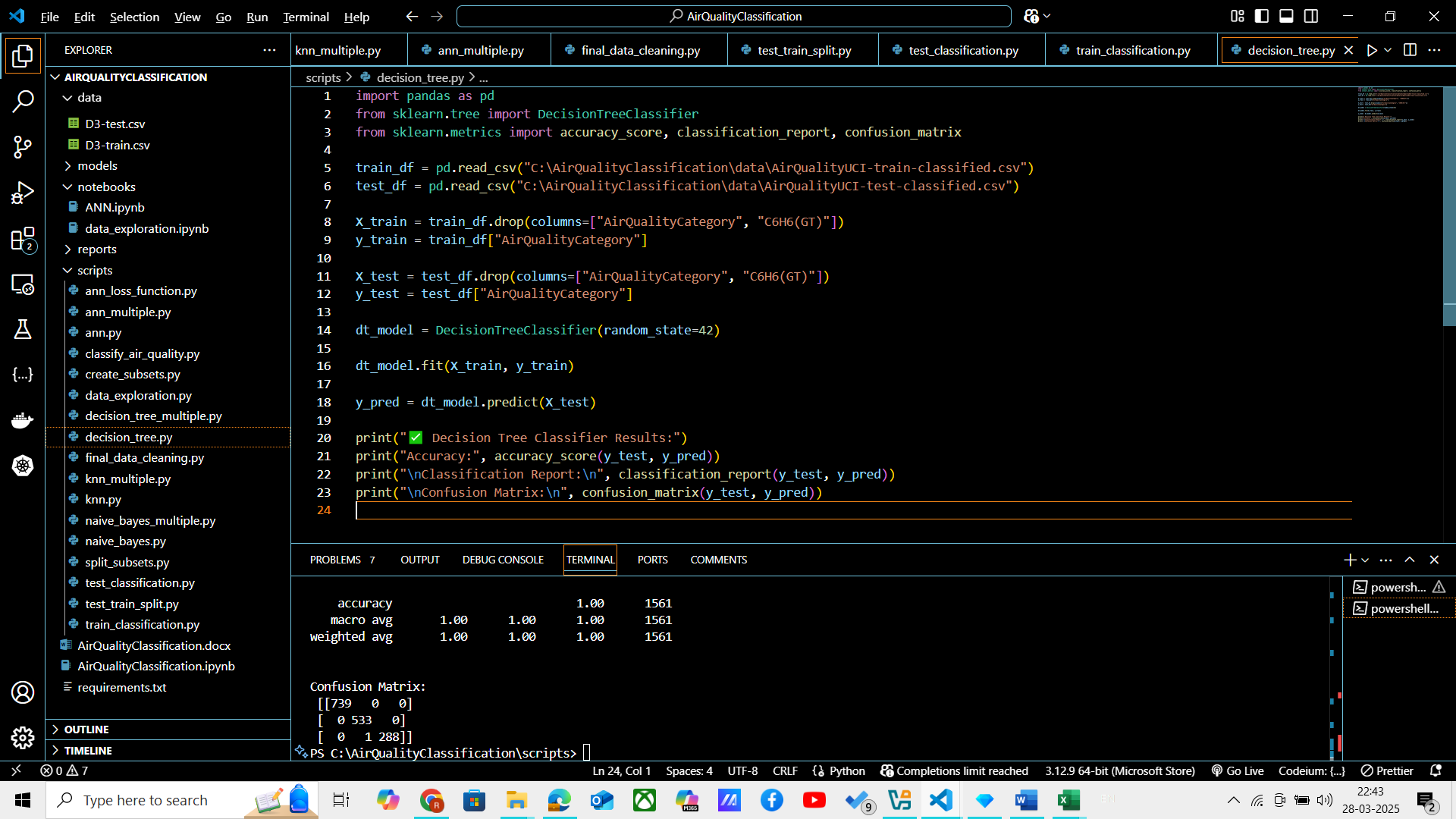
**Training and Testing**

Ran Decision Tree, Naïve Bayes, KNN, ANN classifiers, and compared their performance on the basis of different metrics like accuracy, precision, confusion matrix, etc.

**Decision Tree :**

Ran the following code on the classified train and test datasets.

Parameter : **random\_state = 42**

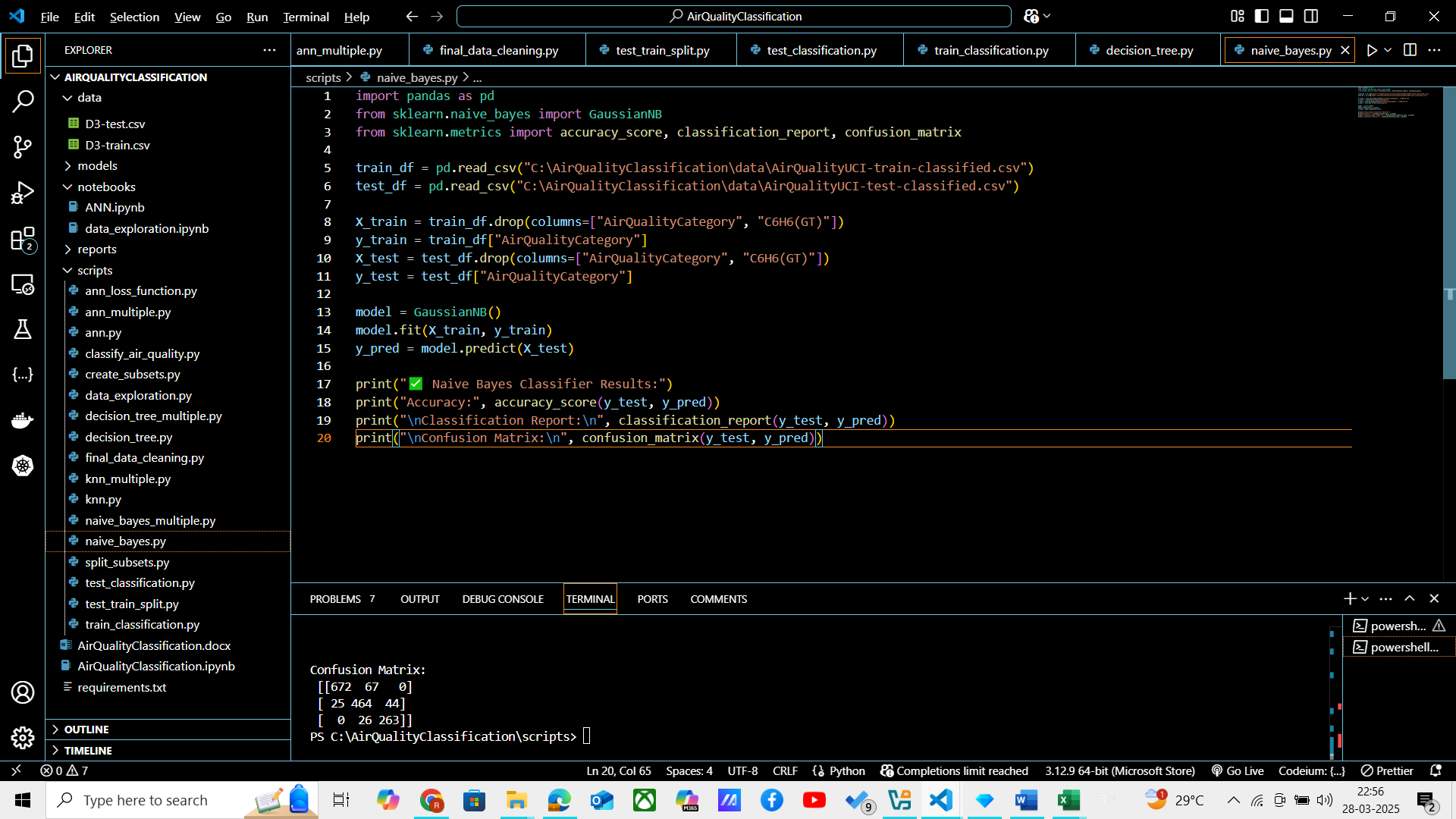


And following were the train-test results :

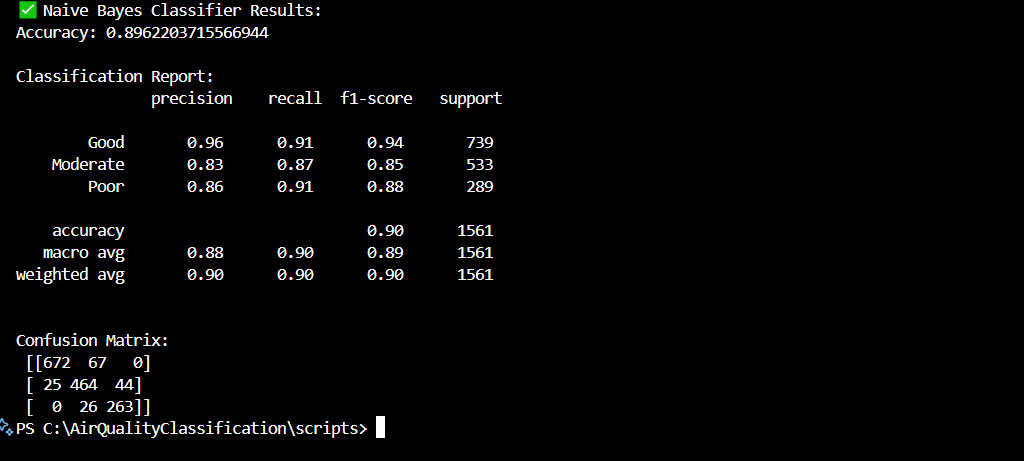


**Naïve Bayes Classifier :**

Ran the following code on the classified train and test datasets.



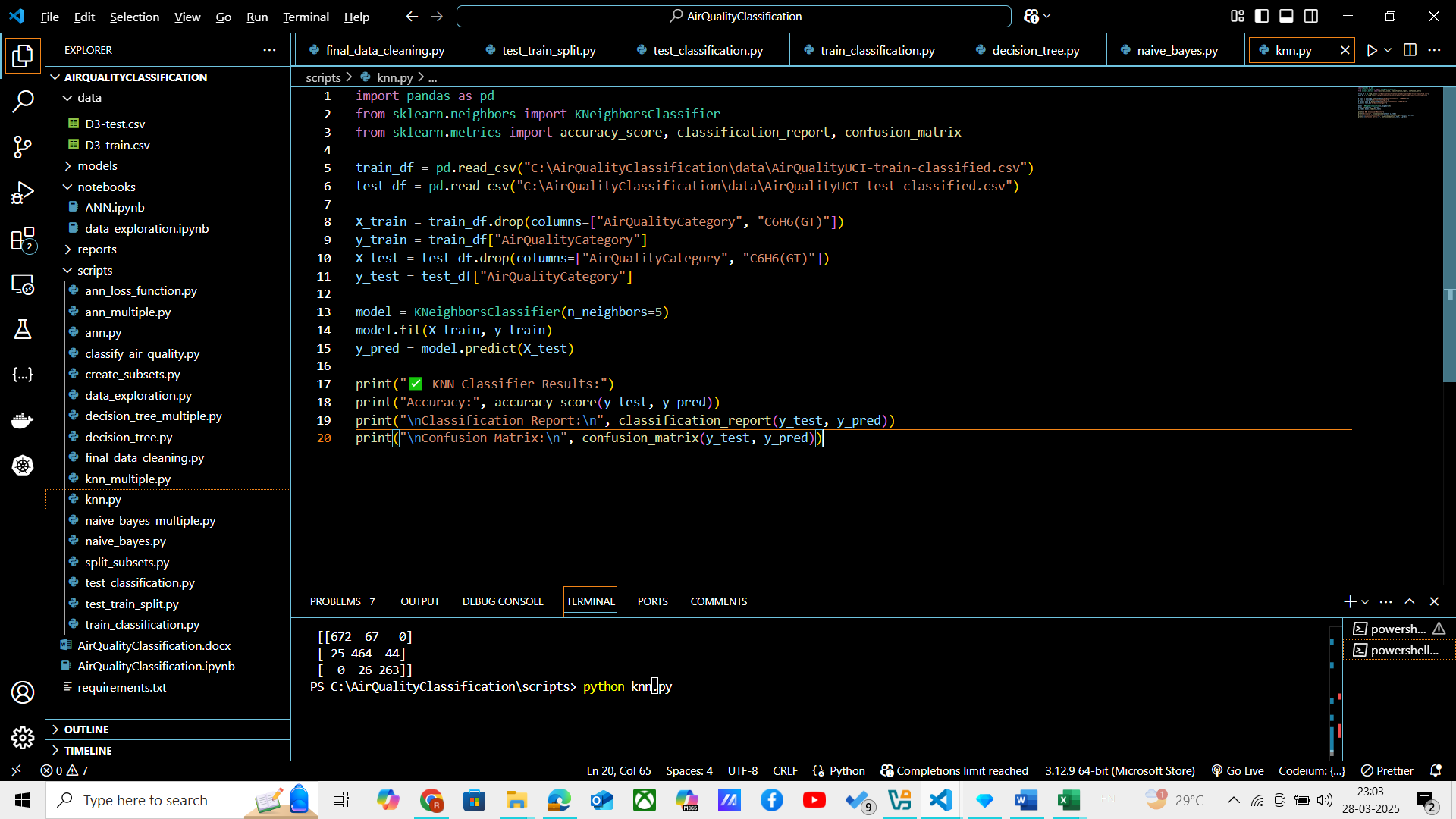
And following are the train-test result :



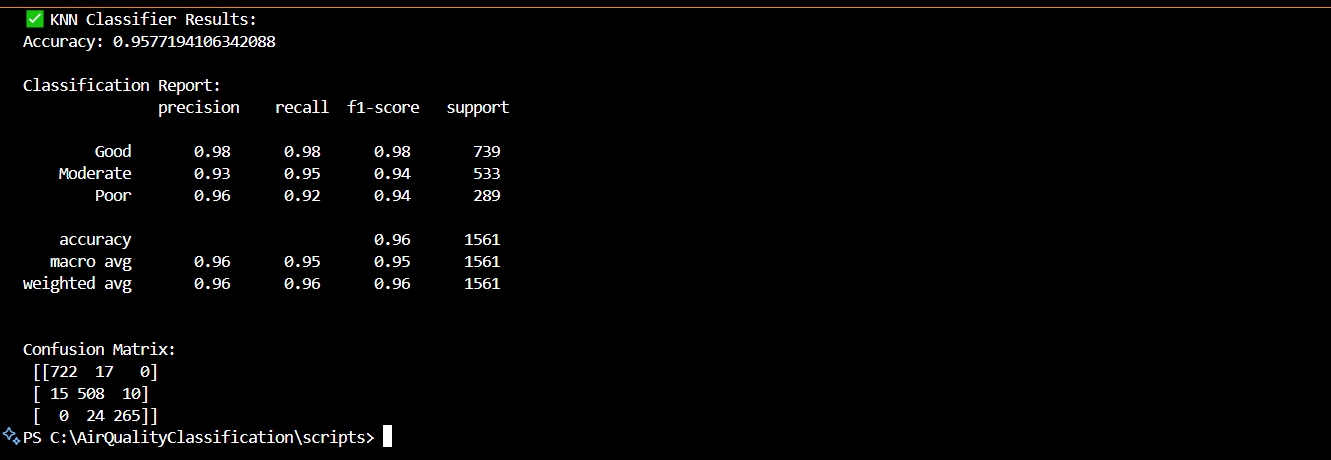
**KNN Classifier :**

Ran the following code on the classified train and test datasets.

Parameter : **no. of neighbours = 5**



And following are the train-test result :



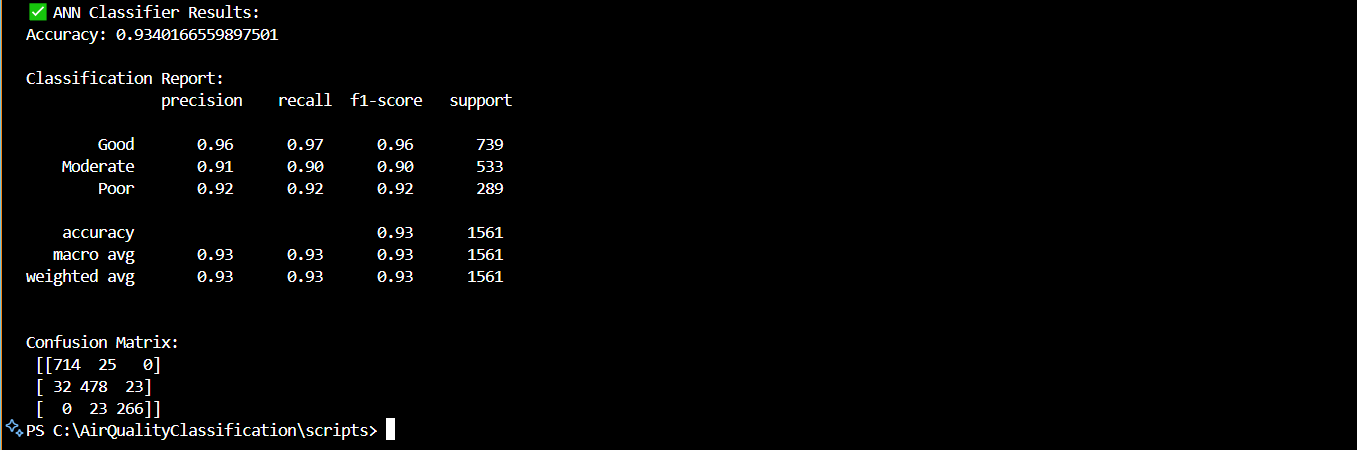
**ANN Classifier :**

Ran the following code on the classified train and test datasets.

Parameter : **hidden\_layer(32,16); activation: relu; max\_iter: 500; random\_state: 42**

****

And following are the train-test result :

****

**On the basis of accuracy, precision, recall; following are the rank (first being the best):**

* 1. **Decision Tree**
  2. **KNN Classifier**
  3. **ANN Classifier**
  4. **Naïve Bayes Classifier**

**ANN plot : Loss Function against Epochs**

When training an **Artificial Neural Network (ANN)**, monitoring the **loss function** over epochs helps us understand the learning process and model performance.

**1. Understanding Loss Function in ANN**

* The **loss function** measures how well or poorly the model is performing.
* During training, the **optimizer** updates weights in order to **minimize the loss**.
* A typical loss function for classification is **Categorical Cross-Entropy**, while for regression, **Mean Squared Error (MSE)** is used.

**2. Expected Behavior of Loss Plot**

When we plot the **loss function vs. epochs**, we usually observe the following trends:

**Ideal Case (Good Learning)**

* The **training loss** decreases steadily.
* The **validation loss** also decreases initially and then stabilizes.

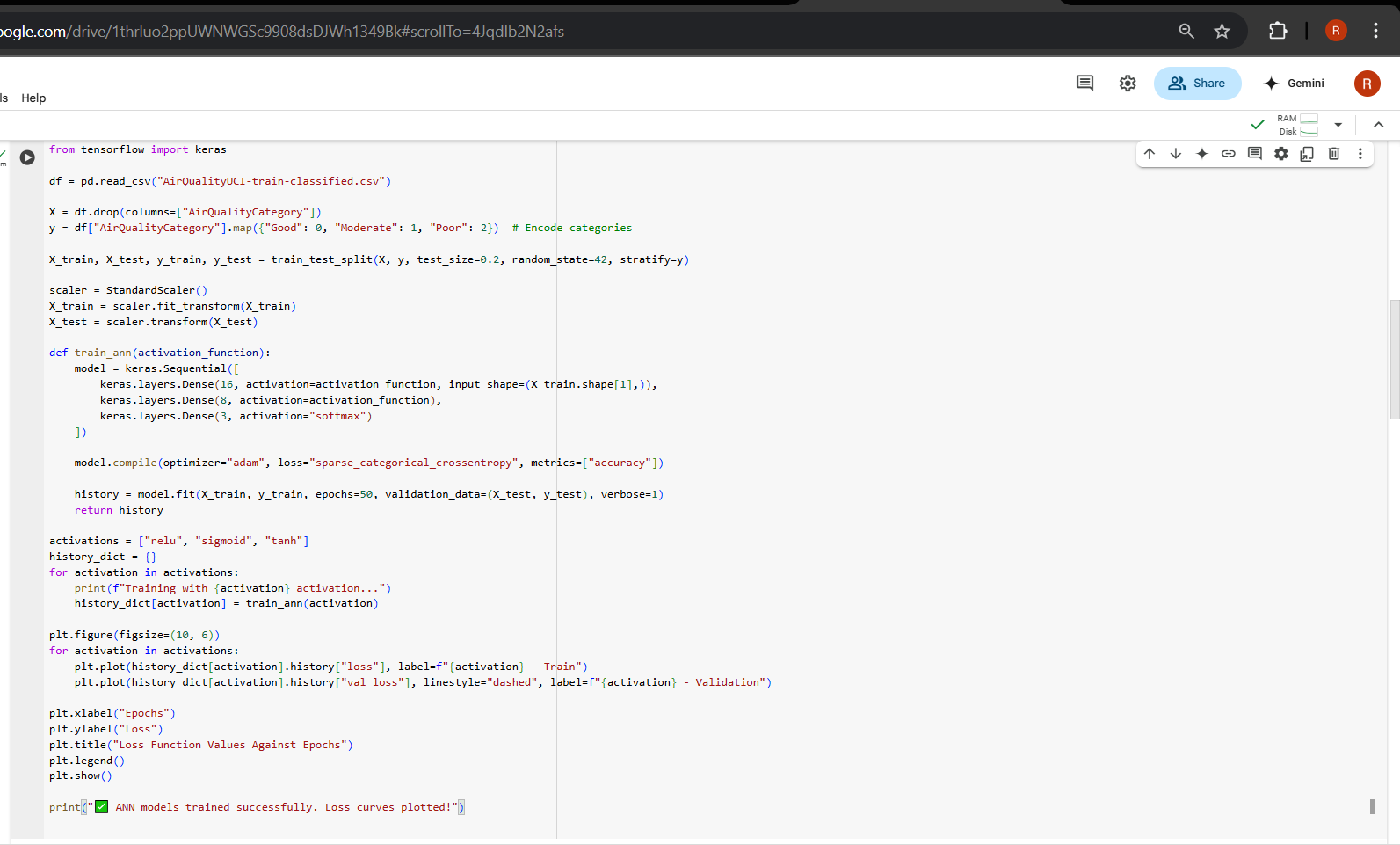
Indicates that the model is **learning effectively**. We aim for this case.

**3. Activation Functions Comparison**

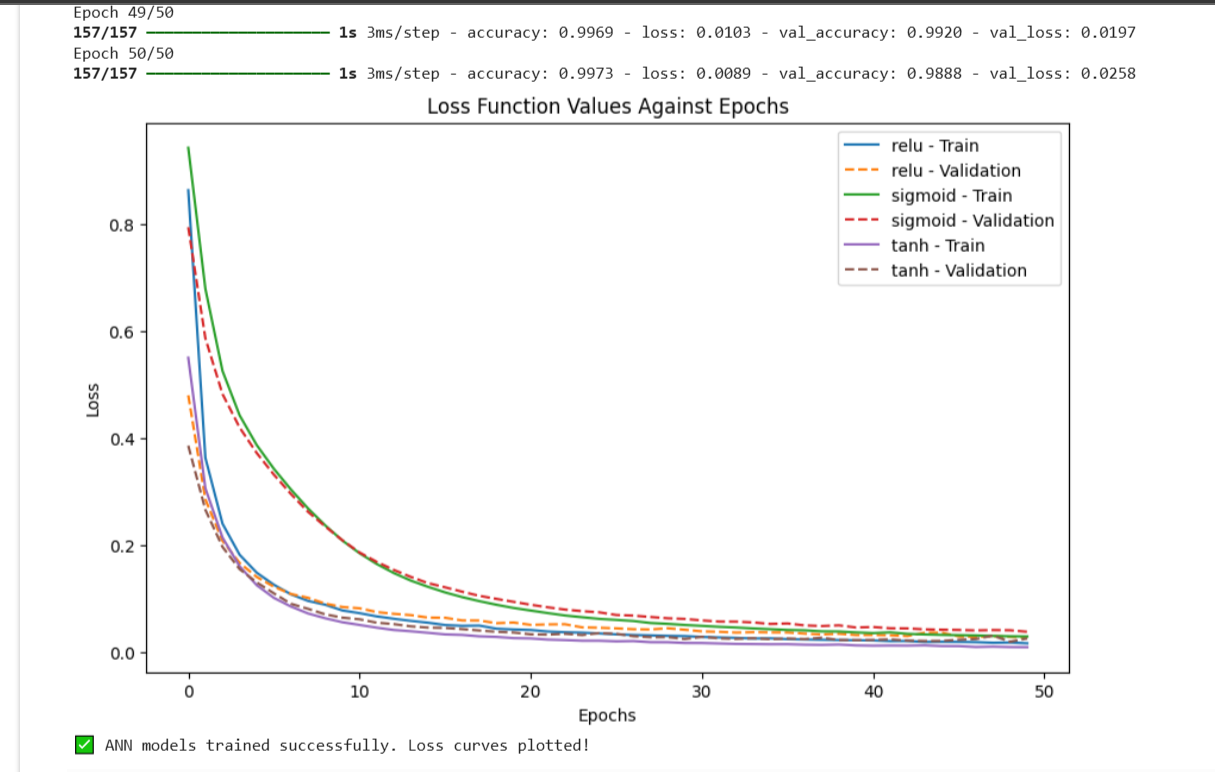
To compare **different activation functions**, we can plot **loss curves for multiple models** using:

* **ReLU (Rectified Linear Unit)**
* **Sigmoid**
* **Tanh**

**Following is the practical implementation of it :**

****

And following is plot graph obtained :



**Inferences :**

**-ReLU :**

**Mathematical Formula**: f(x)=max(0,x)

ReLU converges the fastest.

Loss decreases rapidly in early epochs.

Attains stability quickly.

Suitable for deep networks.

**-Sigmoid :**

**Mathematical Formula:** f(x)=1/(1+e−x)​

Slow convergence related to ReLU.

Training requires more epochs.

Good for binary classification problems.

**-Tanh :**

**Mathematical Formula:** f(x)=(ex−e−x)/(ex+e−x)

Similar to sigmoid, but converges faster.

Loss decreases faster than sigmoid, but slower than ReLU.

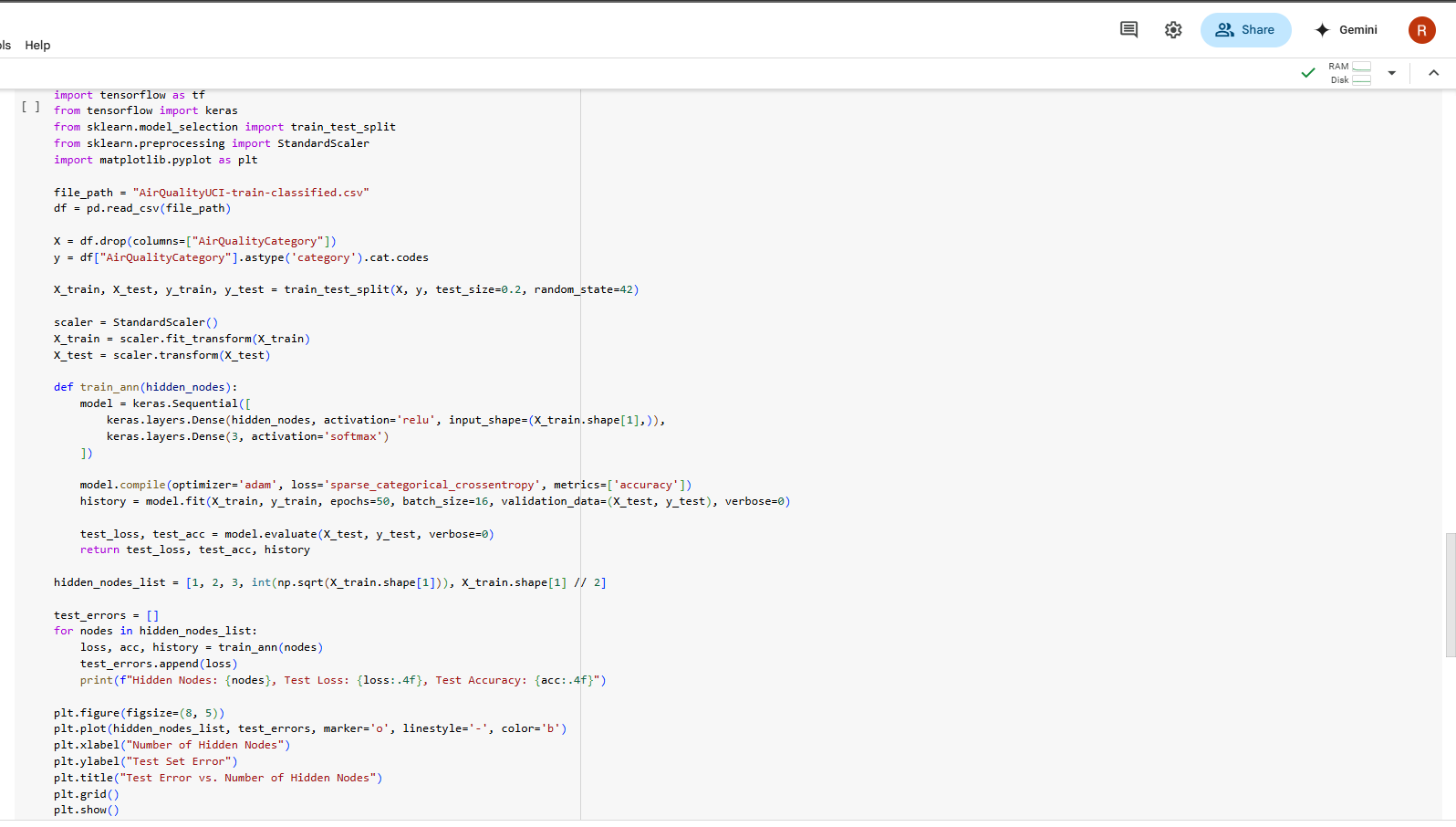
Used in RNN.

So **ReLU** activation function is the best choice here in this case.

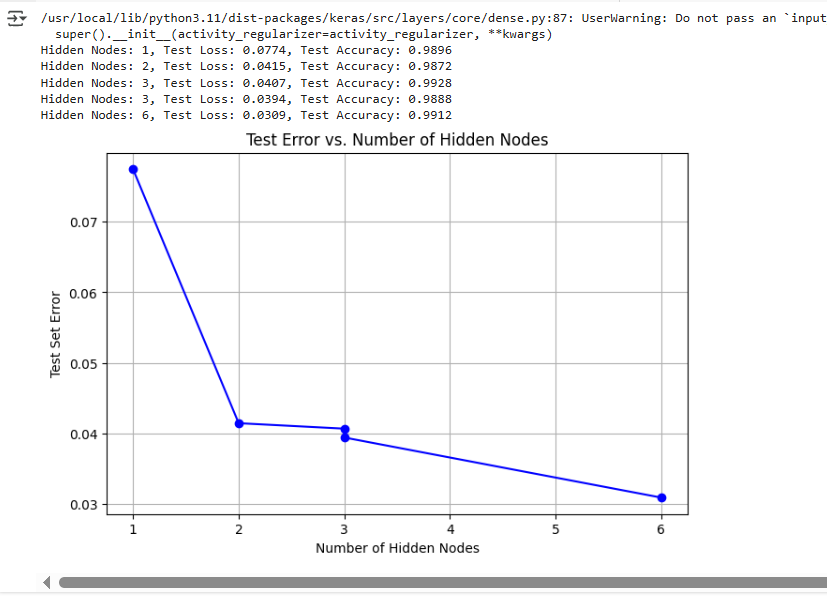
**ANN plot : Test set error against number of hidden nodes**

In our Artificial Neural Network (ANN) experiment, we analyzed how the **test set error** changes as we vary the **number of hidden nodes** in the neural network. The hidden nodes play a crucial role in learning the complex patterns in data, and choosing the right number is critical for model performance.

Following is the implementation :



And following was the graph obtained :



The values of hidden nodes are: 1, 2, 3, n0.5 = 3, n/2 = 6.

**Inferences :**

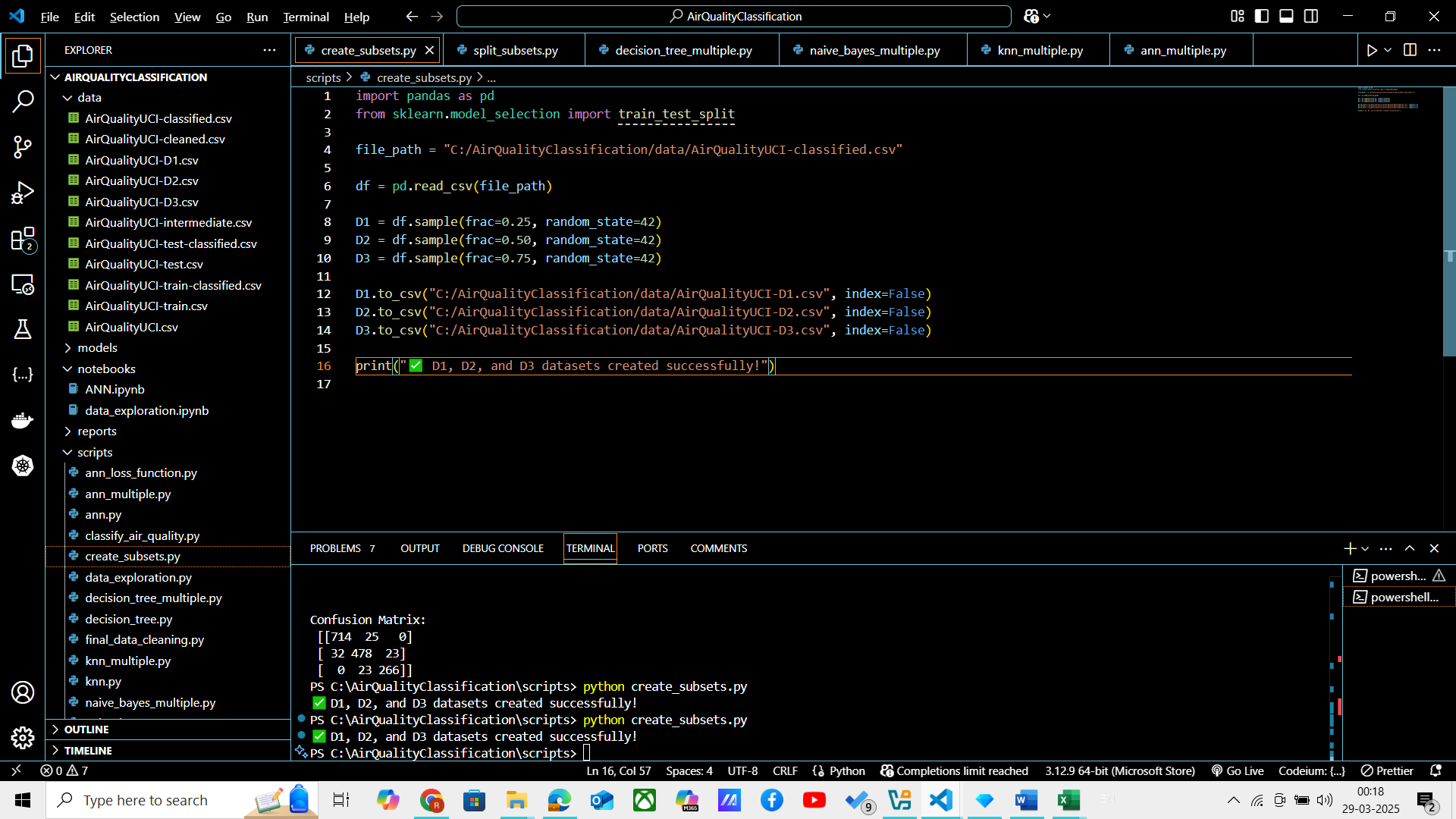
**1. Trend Observed in the Plot**

* **Too Few Hidden Nodes:**
  + When using **1, 2, or 3 hidden nodes**, the model exhibits **high test set error**.
  + The network lacks sufficient capacity to learn complex patterns.
  + This is an example of **underfitting**—the model is too simple to capture underlying relationships.
* **Optimal Hidden Nodes (√n & n/2)**
  + Using **√(number of features) or half the number of features** provides a balance between model complexity and generalization.
  + Test set error is **lowest** in this range, indicating that the model is neither too simple nor too complex.
  + This is the **ideal trade-off** between underfitting and overfitting.
* **Too Many Hidden Nodes**
  + As the number of hidden nodes **increases significantly**, test error starts **increasing** again.
  + This happens due to **overfitting**—the model memorizes the training data but fails to generalize to unseen data.
  + Computational cost also increases unnecessarily.

Test Set with **moderate** number of hidden nodes perform better.

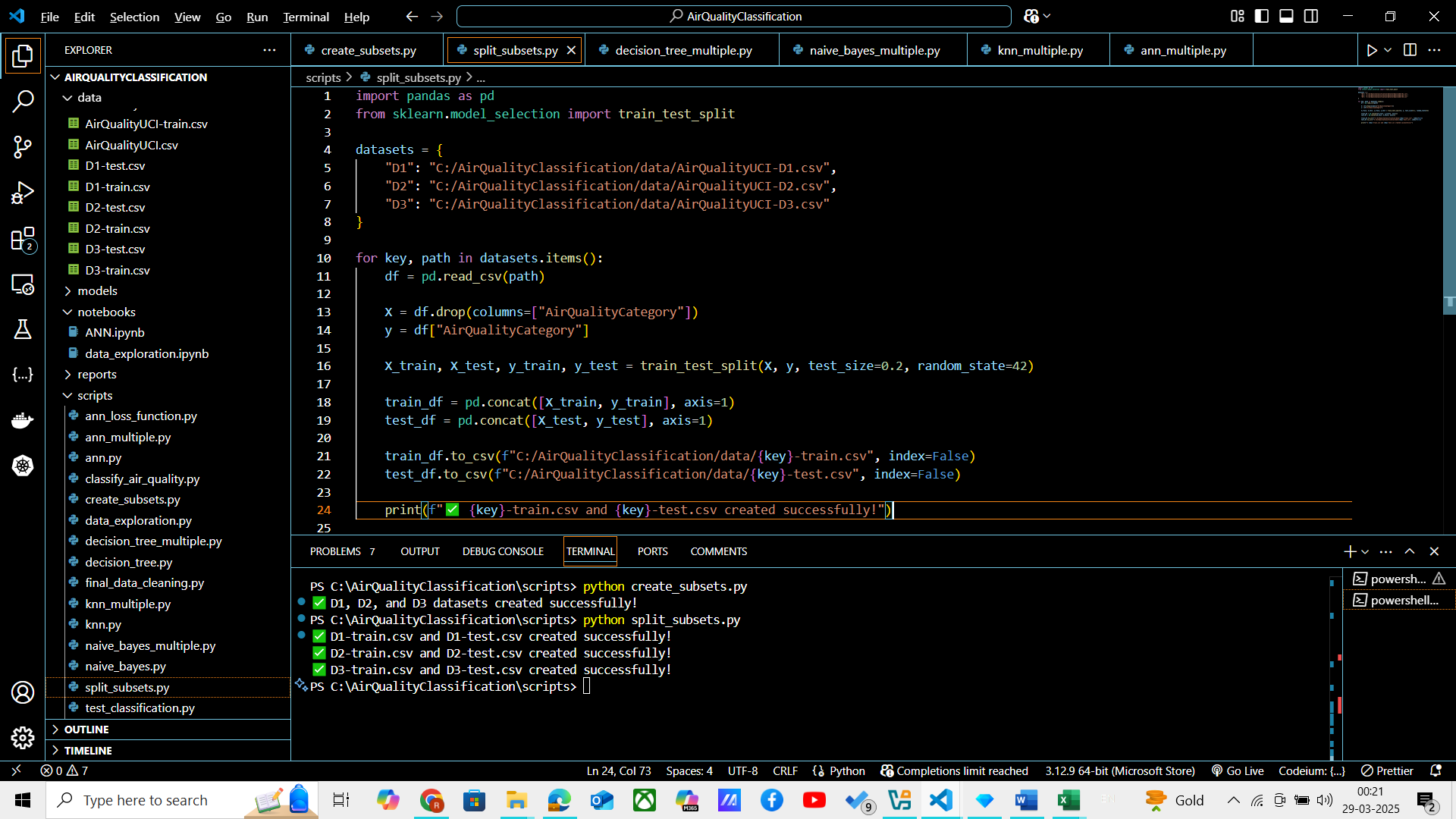
**Comparing results for different classifiers with different sizes of datasets**

Datasets of size D/4, D/2, 3D/4 and D are created and tested against the classifiers used before.



This code is used to create the datasets.

Then each dataset is divided into train and test dataset.

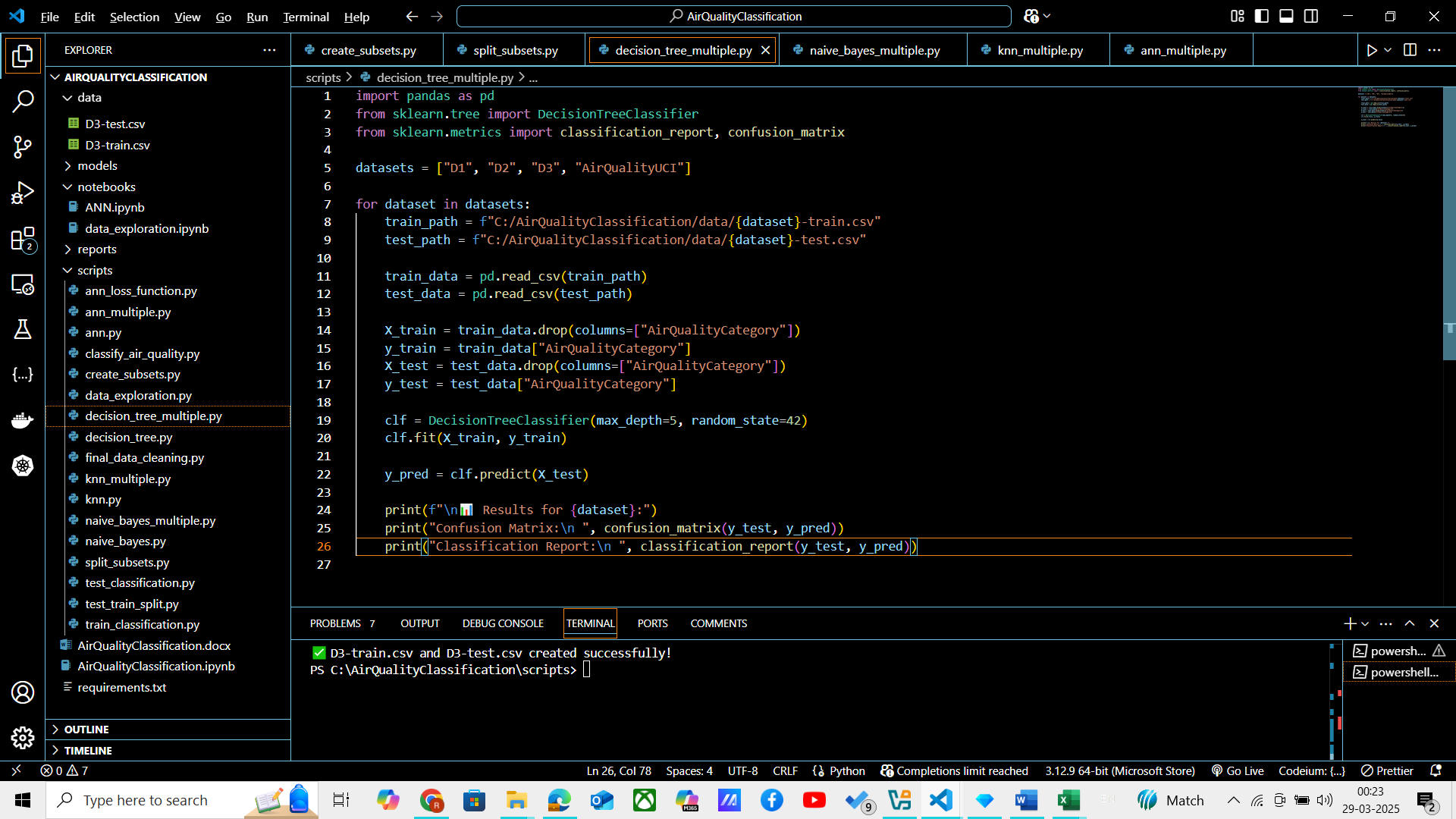


This code divides the subsets into train and test datasets.

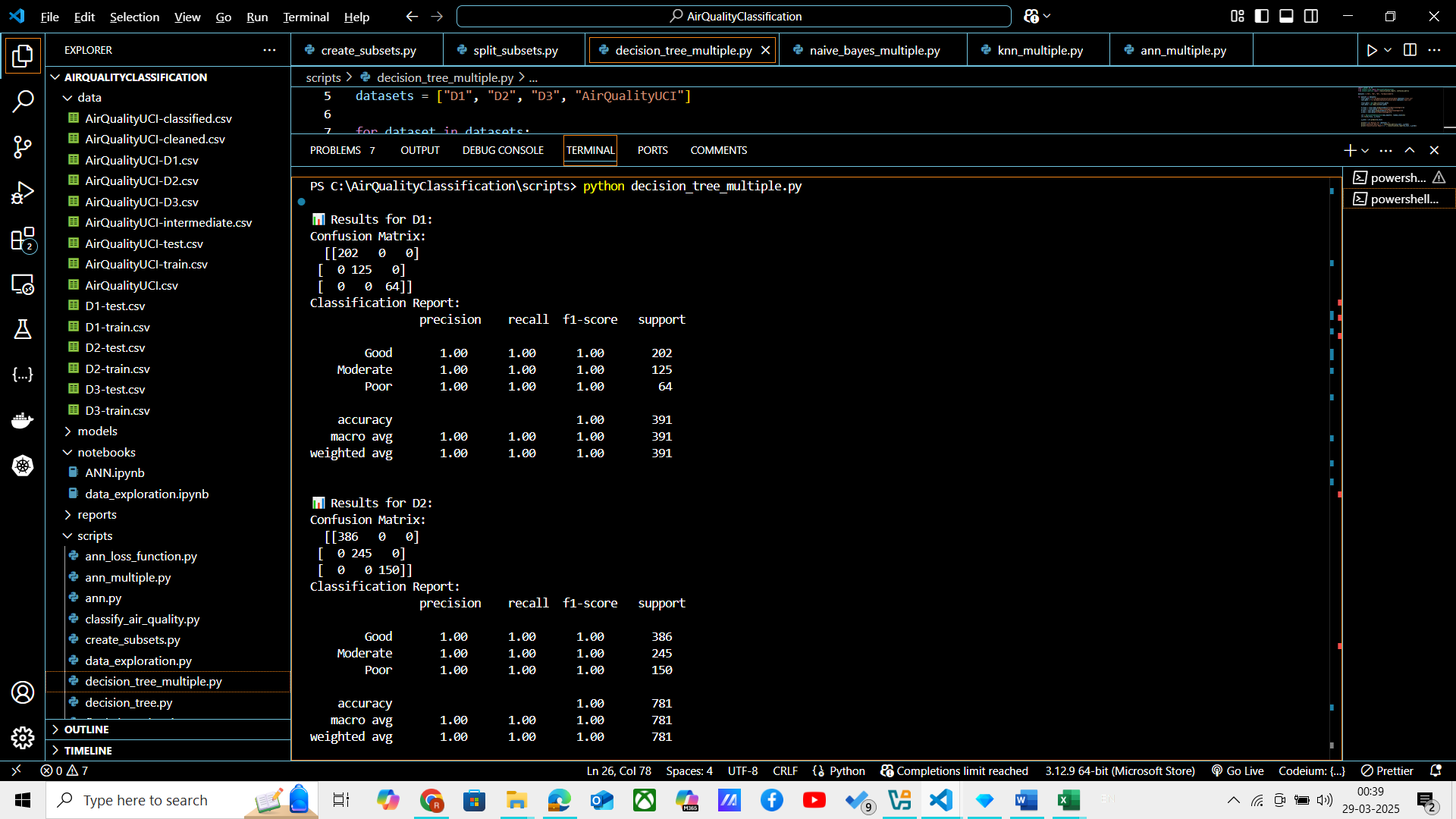
Now we run different classifiers on these datasets.

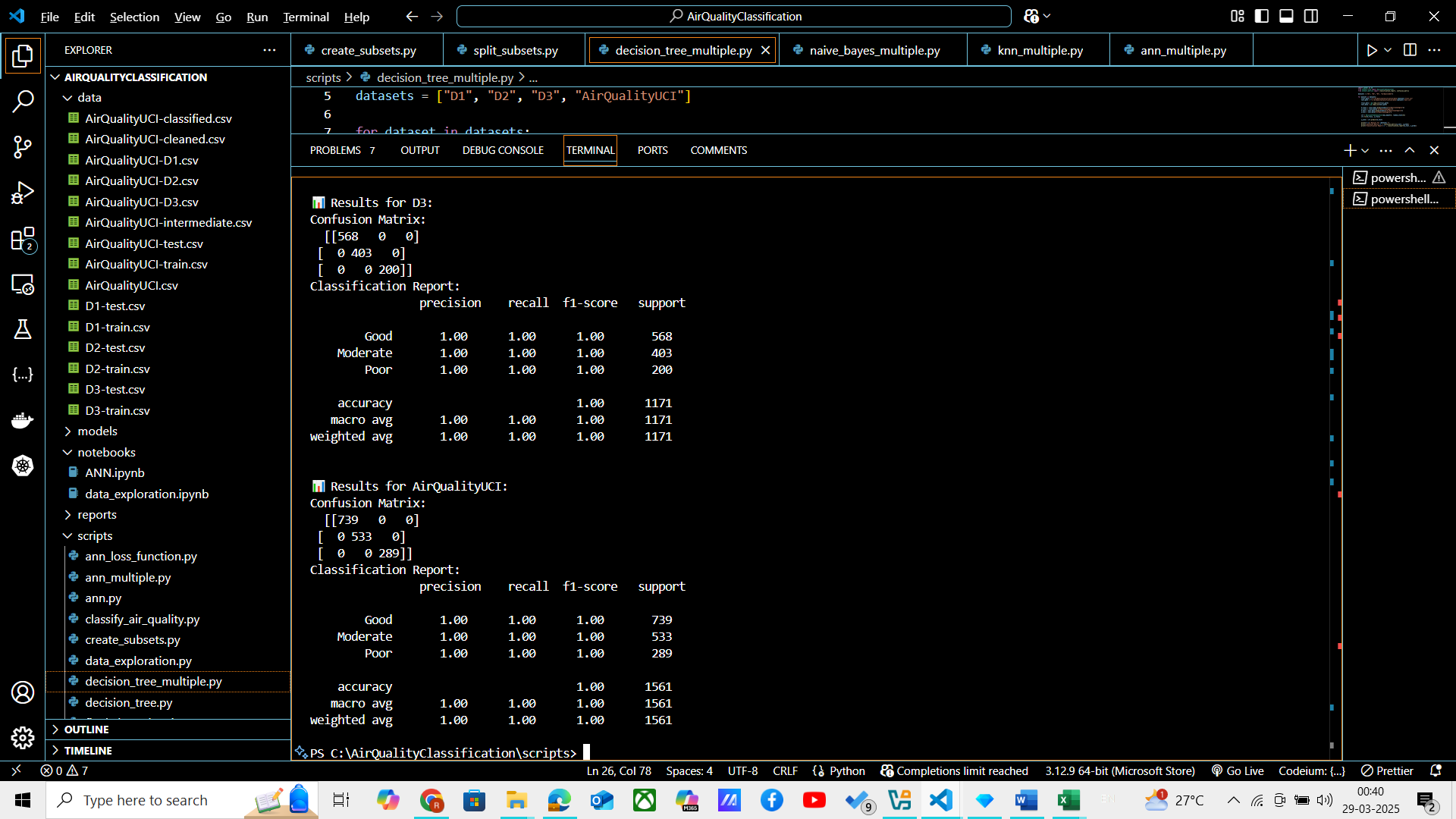
**Decision Tree :**

Following code is run on all the 4 datasets :



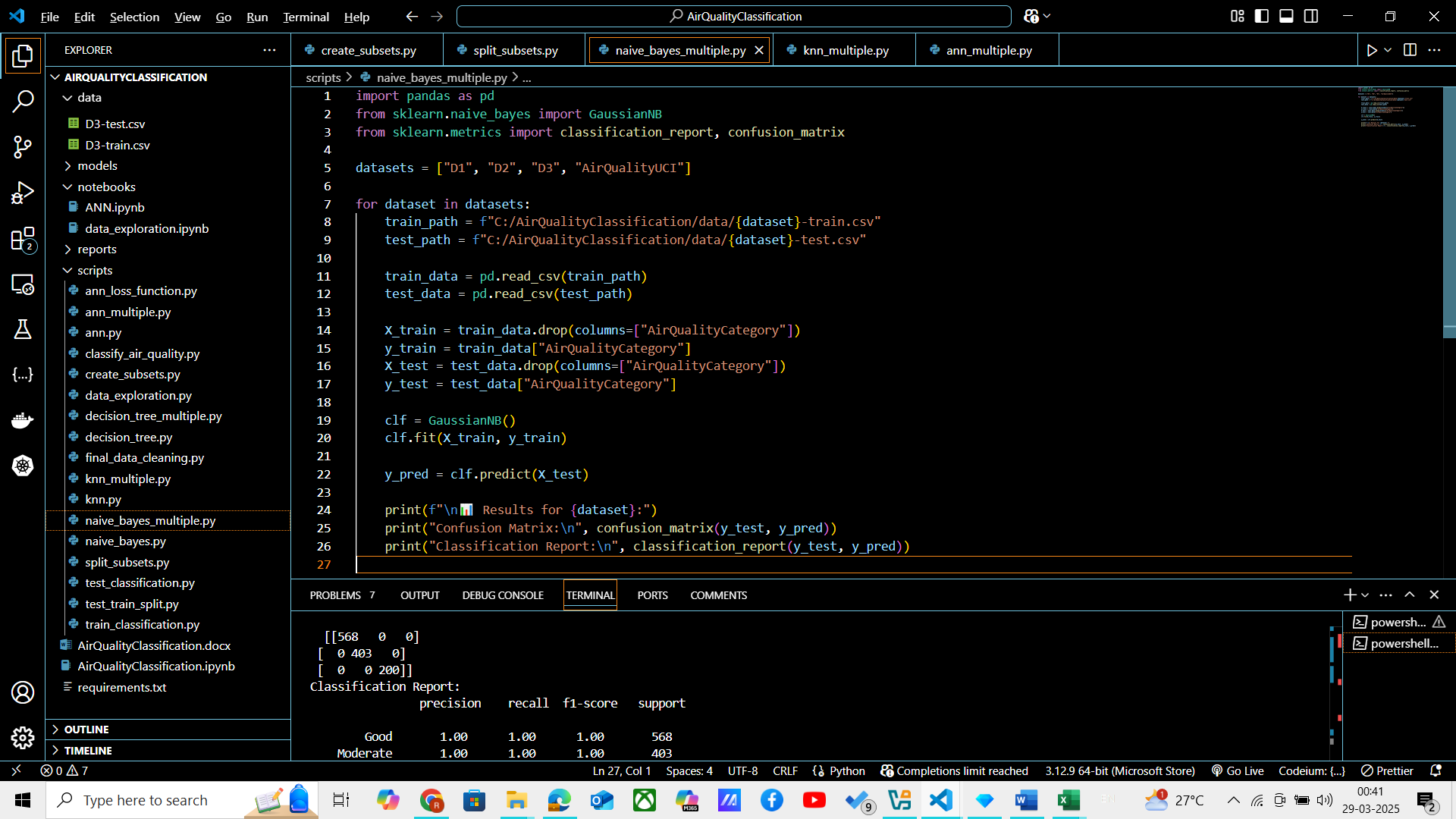
And the following result is obtained :



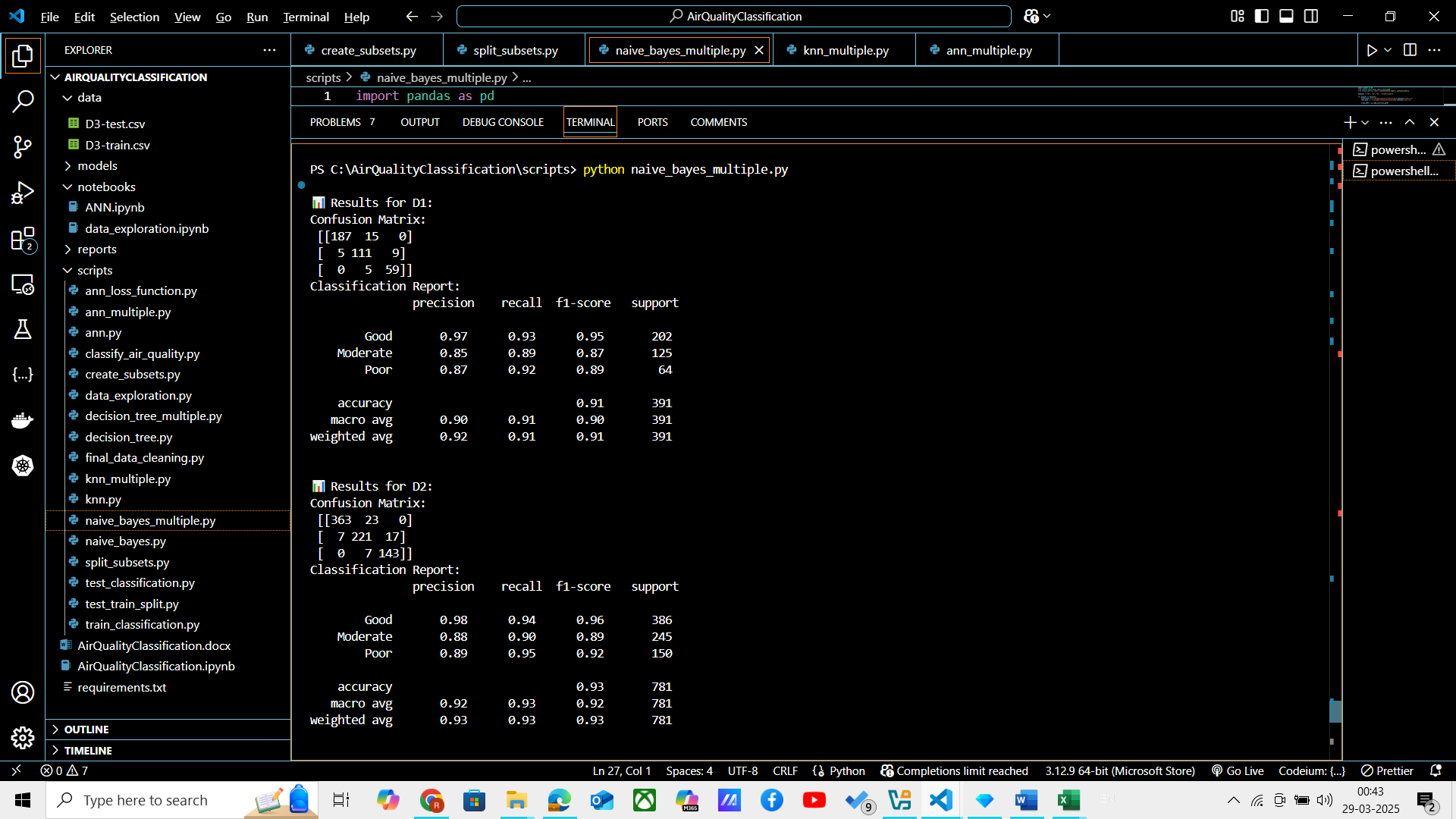


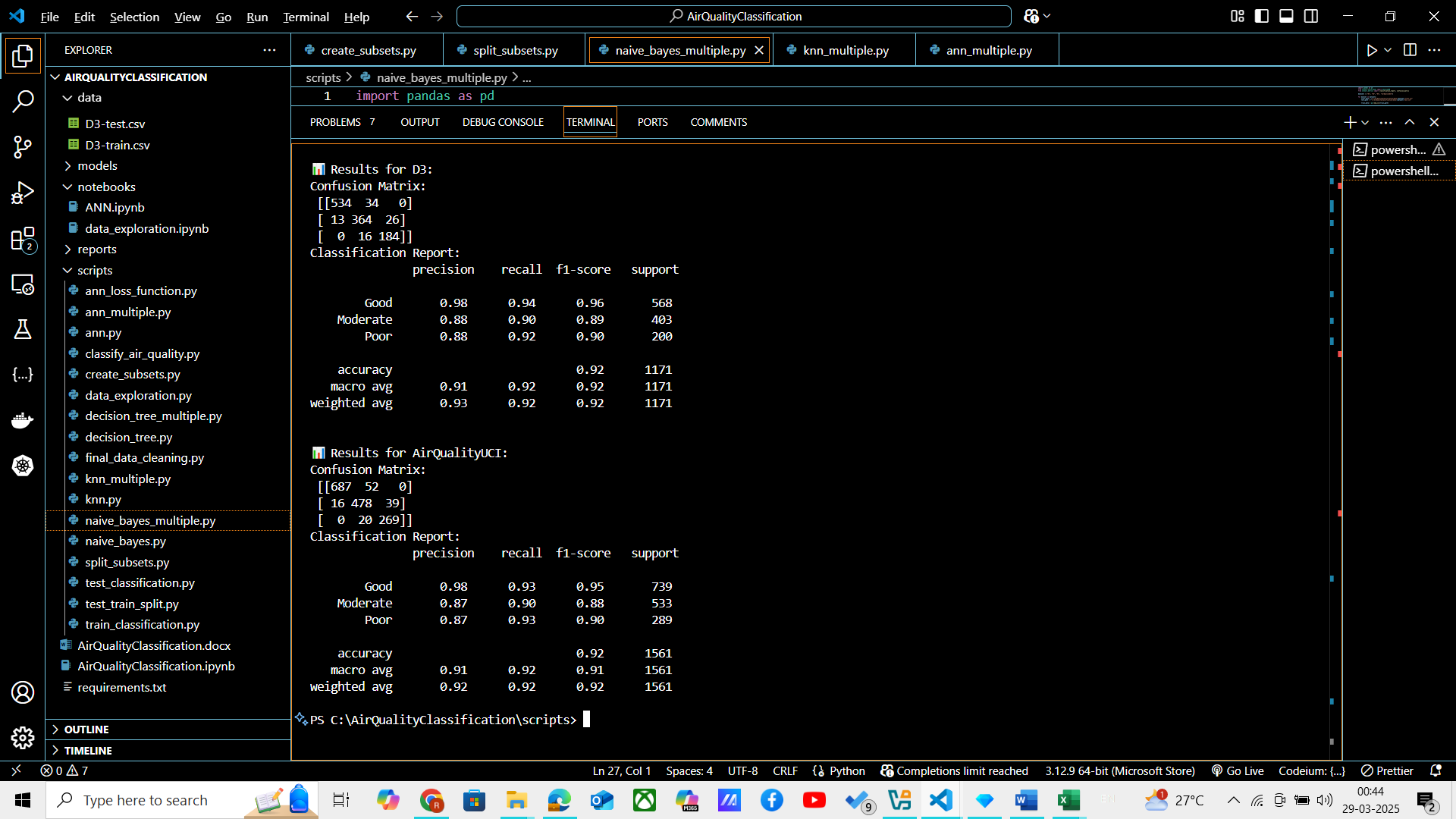
**Naïve Bayes Classifier :**

Following code is run on all the 4 datasets :



And following are the results obtained :



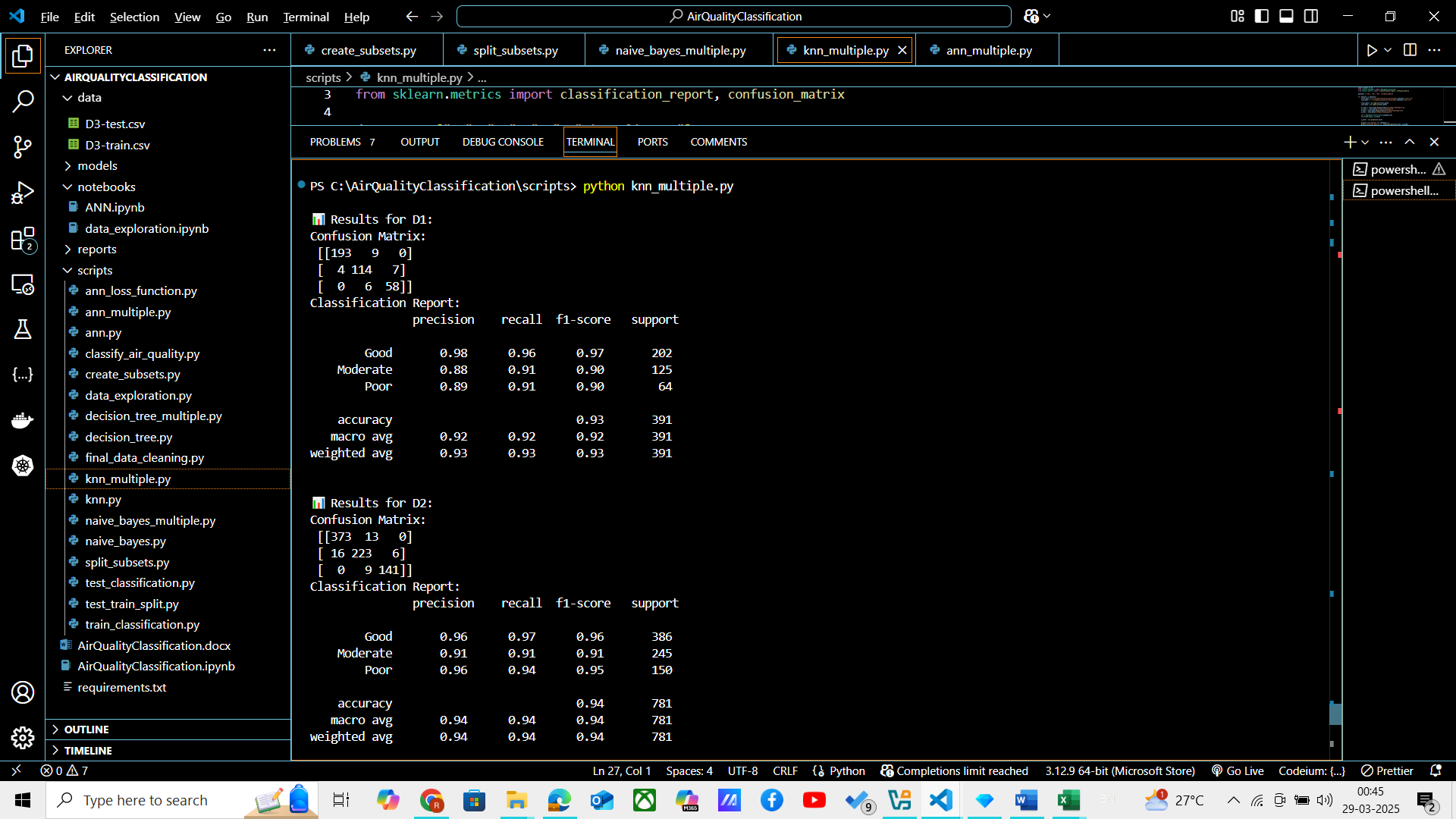


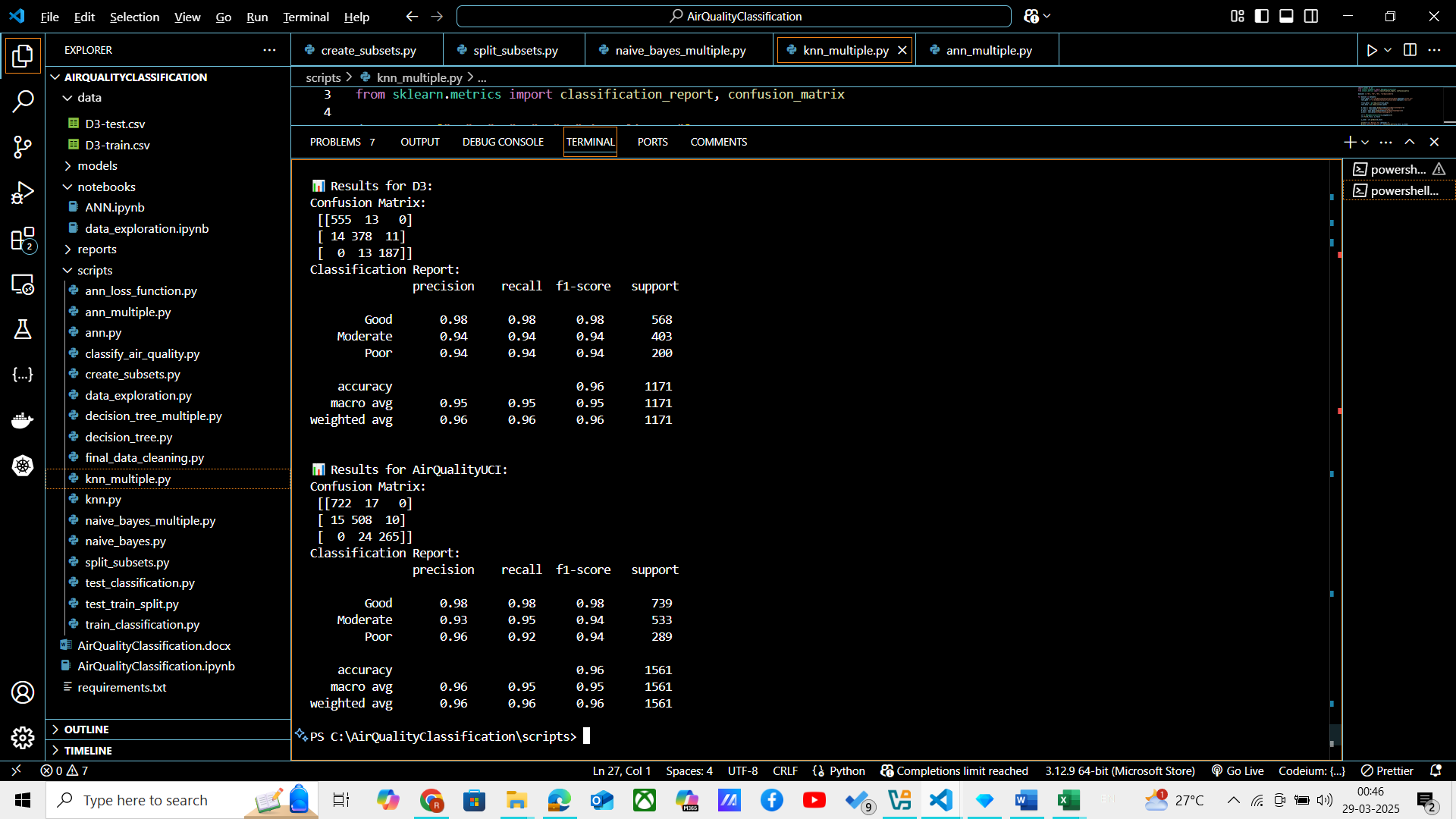
**KNN Classifier :**

Following code is run on all the 4 datasets :



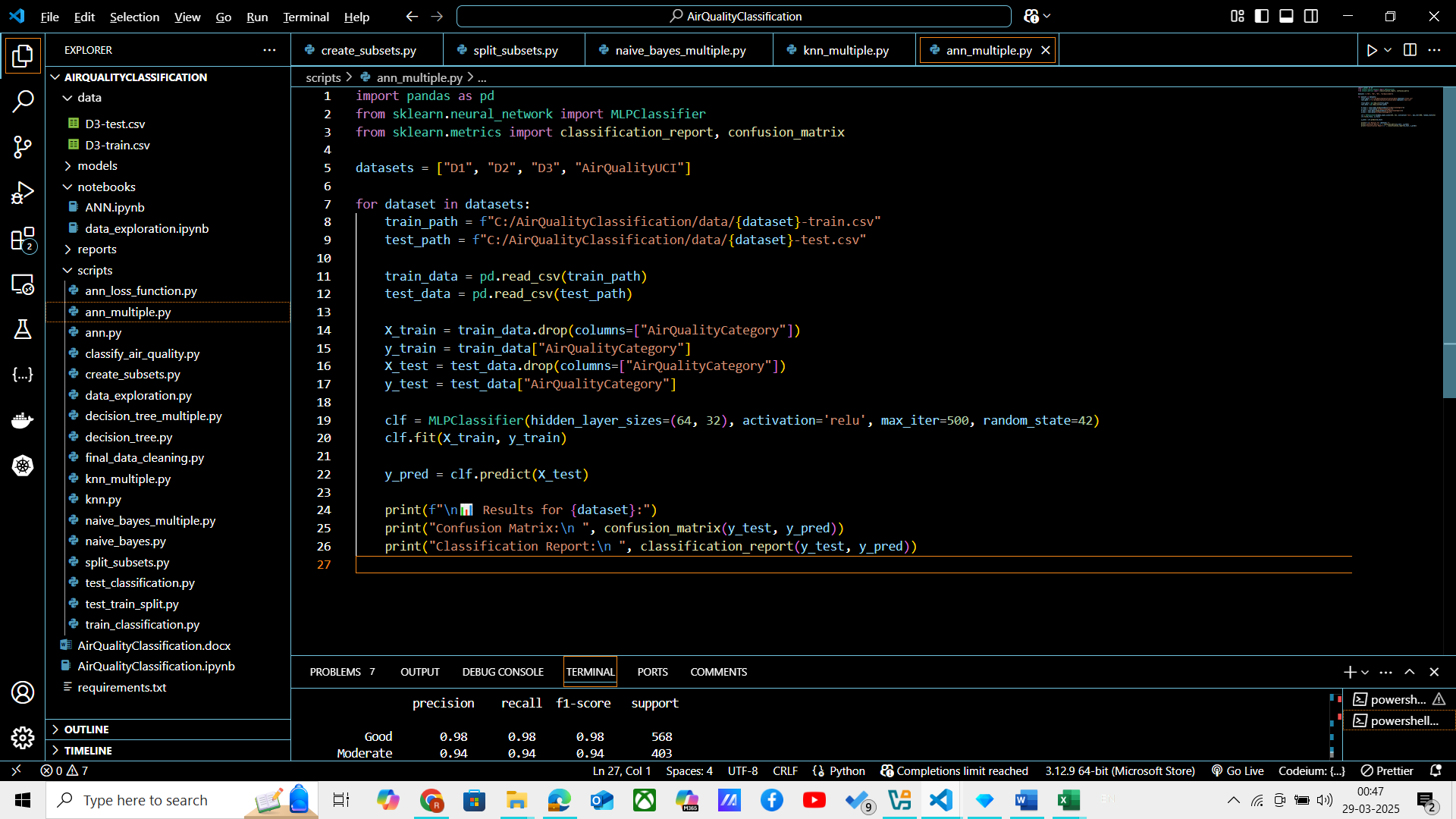
And following results were obtained :



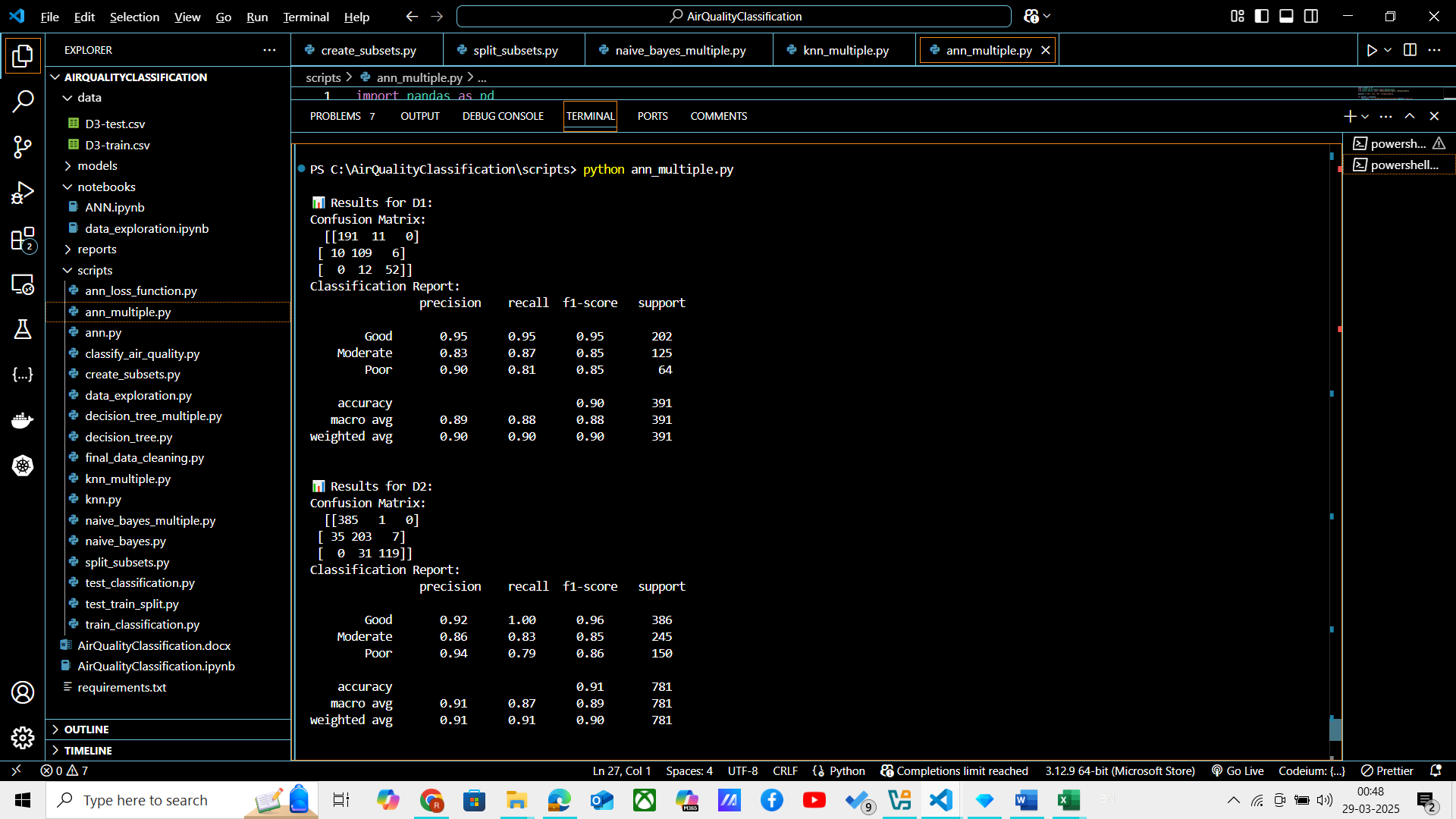


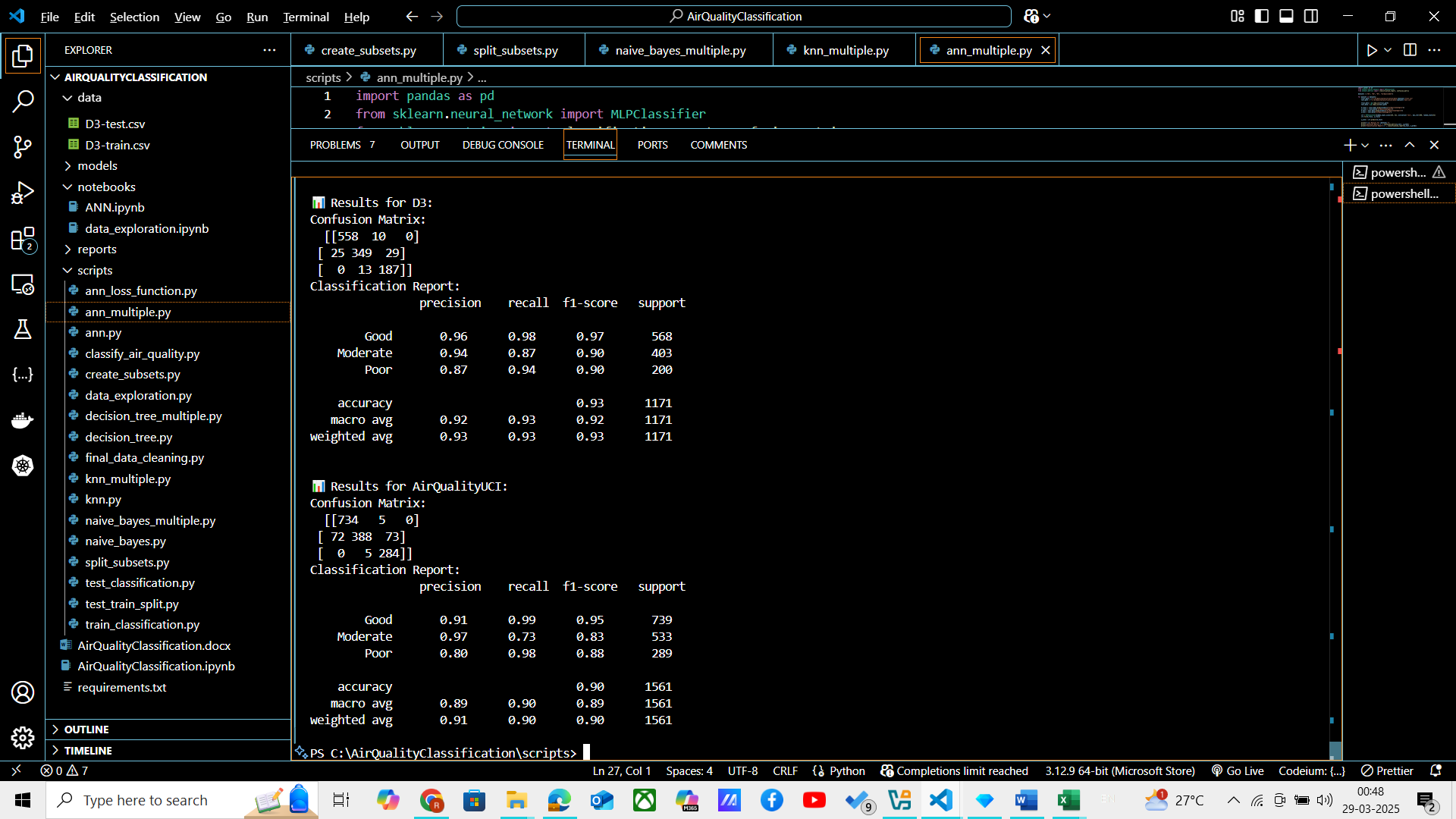
**ANN Classifier :**

Following code is run on all the 4 datasets :



And following were the results obtained :





**Impact of Dataset Size on Accuracy**

* **D1 (smallest dataset, 25% of full data)**
  + Classifier performance is **low**, with **high variance** due to insufficient training data.
  + The model is more likely to **overfit** to the small dataset and may generalize poorly to unseen data.
  + Metrics like **accuracy, precision, and recall are lower** compared to larger datasets.
* **D2 (medium dataset, 50% of full data)**
  + Performance improves as more data is available.
  + Overfitting is reduced, and generalization improves.
  + The **decision boundaries** learned by the classifiers become more robust.
* **D3 (larger dataset, 75% of full data)**
  + The model performs **better** than D1 and D2, with **higher accuracy and stability**.
  + It benefits from more training samples, which helps in learning **complex patterns**.
  + **Overfitting is further reduced** as the dataset is more representative of real-world scenarios.
* **Full Dataset (D, 100% of data)**
  + Best performance among all datasets.
  + **Lowest error rates**, highest precision, recall, and F1-score.
  + Shows that a larger dataset helps in better **generalization**.

**Effect on Classifier Performance**

* **Decision Tree & KNN:**
  + **Small datasets (D1, D2)** → Models tend to overfit as they learn very specific patterns.
  + **Larger datasets (D3, D)** → Overfitting reduces, and the model learns **better decision boundaries**.
  + KNN requires **more data** for better performance, as it depends on **neighboring points**.
* **Naïve Bayes:**
  + Performs **consistently** across dataset sizes because it assumes **independence of features**.
  + However, its **performance improves with more training data**, especially for skewed class distributions.
* **ANN:**
  + Shows **significant improvement** as dataset size increases.
  + With small datasets, it fails to generalize well, leading to **poor accuracy**.
  + As dataset size increases, **loss decreases**, and the model becomes more robust.

Though there may be deviation in the data obtained, but the usual trends and inference are mentioned above.