**Parkinson’s Disease Detection**

**What is Parkinson’s Disease?**

Parkinson's disease is a progressive disorder that affects the nervous system and the parts of the body controlled by the nerves. Symptoms start slowly. The first symptom may be a barely noticeable tremor in just one hand. Tremors are common, but the disorder may also cause stiffness or slowing of movement.

In the early stages of Parkinson's disease, your face may show little or no expression. Your arms may not swing when you walk. Your speech may become soft or slurred. Parkinson's disease symptoms worsen as your condition progresses over time.

Although Parkinson's disease can't be cured, medications might significantly improve your symptoms. Occasionally, your health care provider may suggest surgery to regulate certain regions of your brain and improve your symptoms.

**Dataset**

The Parkinson's Disease Data Set is a comprehensive collection of data that provides clinical, demographic, and behavioural information from Parkinson's disease patients. This data is obtained from diverse sources, including neuroimaging data, clinical assessments, and wearable sensors etc.

With a total of 195 observations and 24 features, this dataset includes variables such as Average vocal fundamental frequency, the health status of the subject, signal fractal scaling exponent etc. Exploratory Data Analysis is carried out on the dataset to identify any potential outliers, as well to get some detailed information about the data. In the next step, the data can be used to train Machine Learning models, which would then make predictions depending on the given input. The Parkinson's Disease Data Set is available for download on Kaggle.

Link to Download:<https://www.kaggle.com/datasets/vikasukani/parkinsons-disease-data-set>

**Tools and Libraries Used**

The project makes use of the following Python libraries:

· NumPy

· Pandas

· Matplotlib

· Seaborn

· Scikit-Learn

**Steps for Detecting Parkinson’s Disease using Machine Learning**

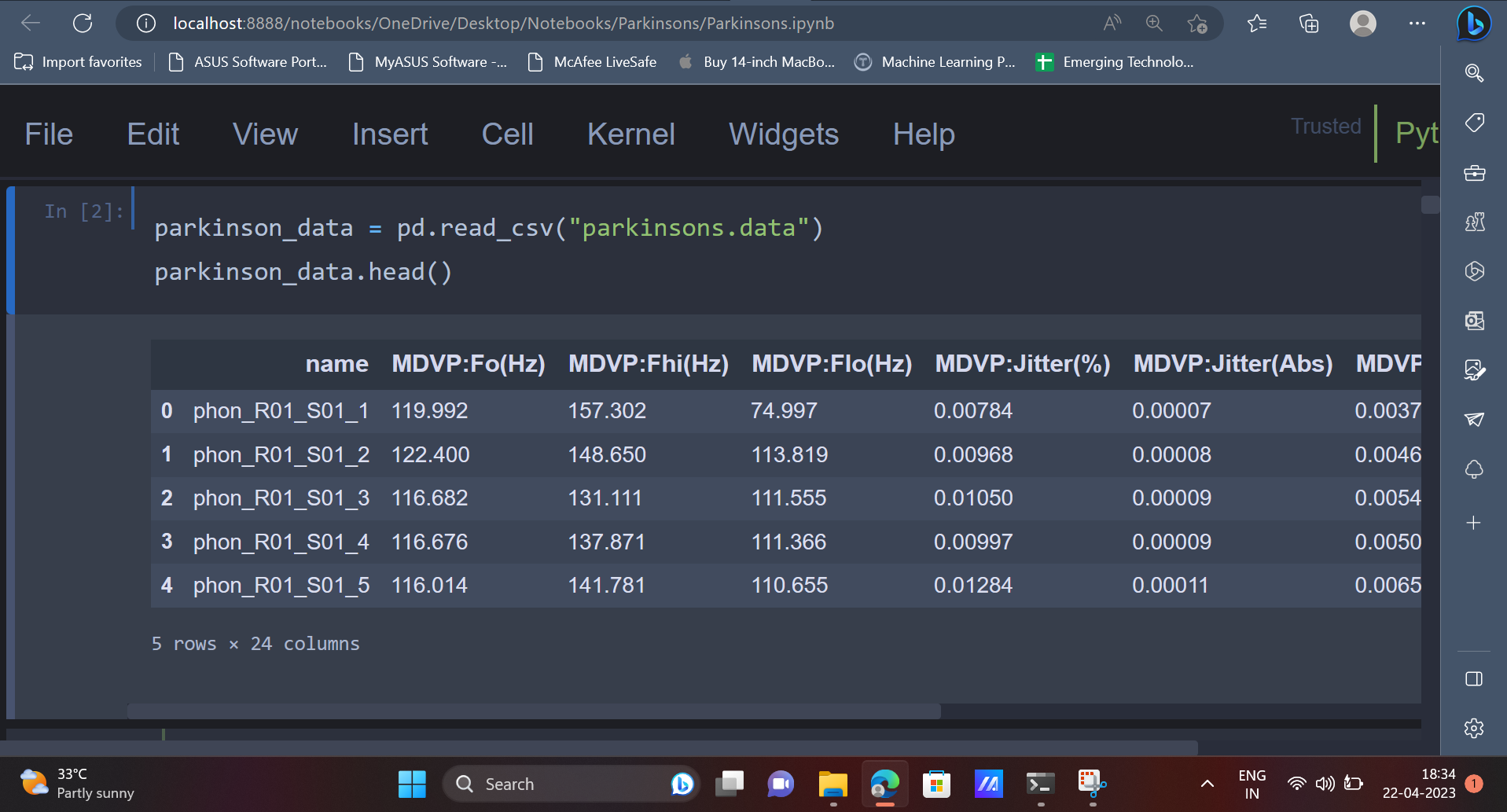
1. Importing the required libraries.

| import numpy as np import pandas as pd import matplotlib.pyplot as plt import seaborn as sns sns.set\_style('darkgrid') |
| --- |

2. Once the necessary libraries have been imported, the next step would be to read the dataset and display the dataset as well. The first 5 rows have been displayed in the screenshot shown below.

| parkinson\_data = pd.read\_csv("parkinsons.data") parkinson\_data.head() |
| --- |

Output:



3. The dataset contains multiple columns which can be seen above. Let’s understand what is the meaning of each attribute/column present in the dataset.

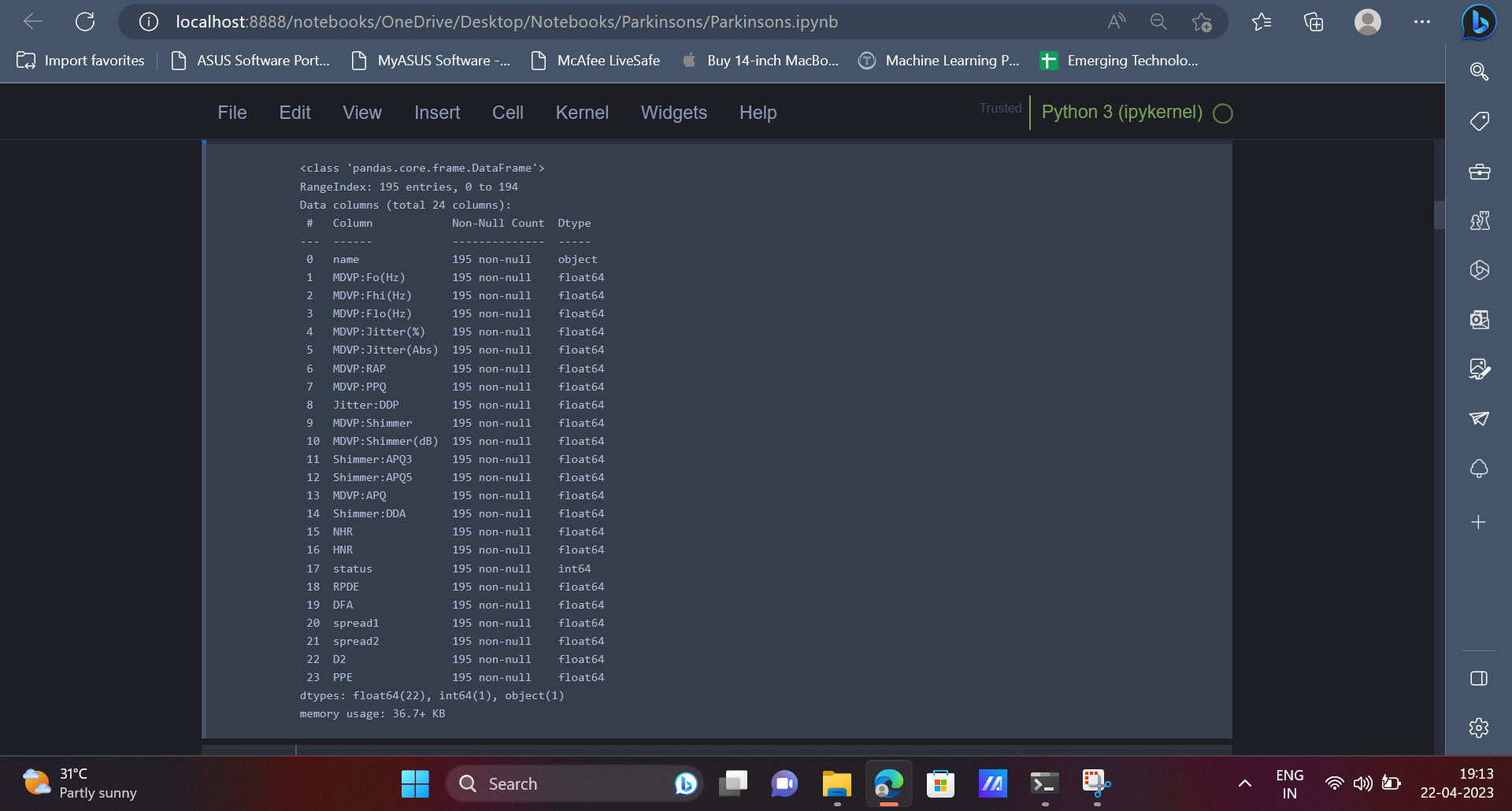
Attributes:

* name - ASCII subject name and recording number
* MDVP:Fo(Hz) - Average vocal fundamental frequency
* MDVP:Fhi(Hz) - Maximum vocal fundamental frequency
* MDVP:Flo(Hz) - Minimum vocal fundamental frequency
* MDVP:Jitter(%), MDVP:Jitter(Abs), MDVP:RAP, MDVP:PPQ, Jitter:DDP - Several measures of variation in fundamental frequency
* MDVP:Shimmer,MDVP:Shimmer(dB),Shimmer:APQ3,Shimmer:APQ5,MDVP:APQ,Shimmer:DDA - Several measures of variation in amplitude
* NHR, HNR - Two measures of the ratio of noise to tonal components in the voice
* status - The health status of the subject (one) - Parkinson's, (zero) - healthy
* RPDE, D2 - Two nonlinear dynamical complexity measures
* DFA - Signal fractal scaling exponent
* spread1,spread2,PPE - Three nonlinear measures of fundamental frequency variation

4. Now that there is clear understanding about the various attributes present in the dataset, the next step would be to use exploratory data analysis to gain more insights as well as discover some hidden patterns or trends present in the dataset.

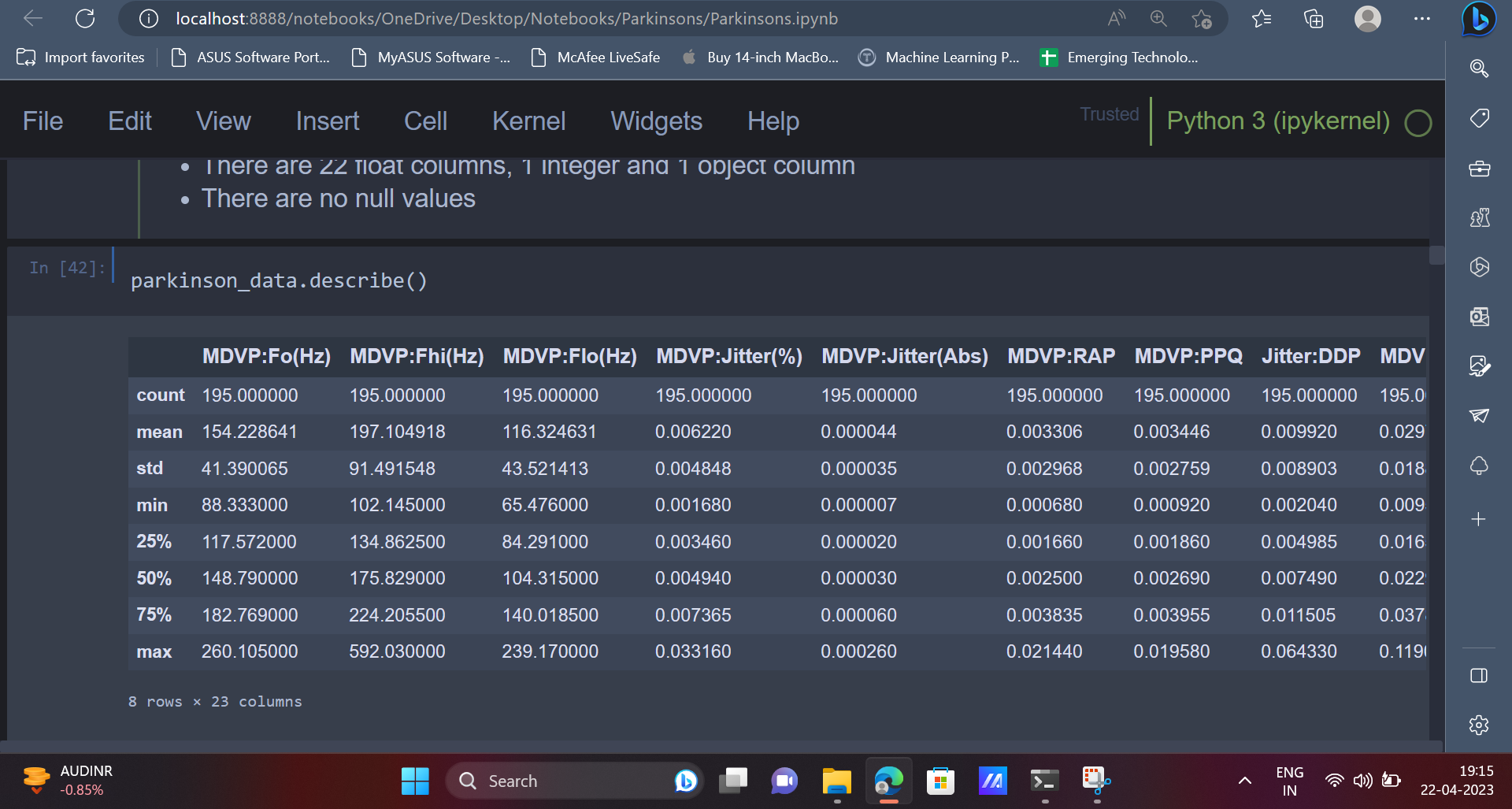
| parkinson\_data.info() |
| --- |

Output:



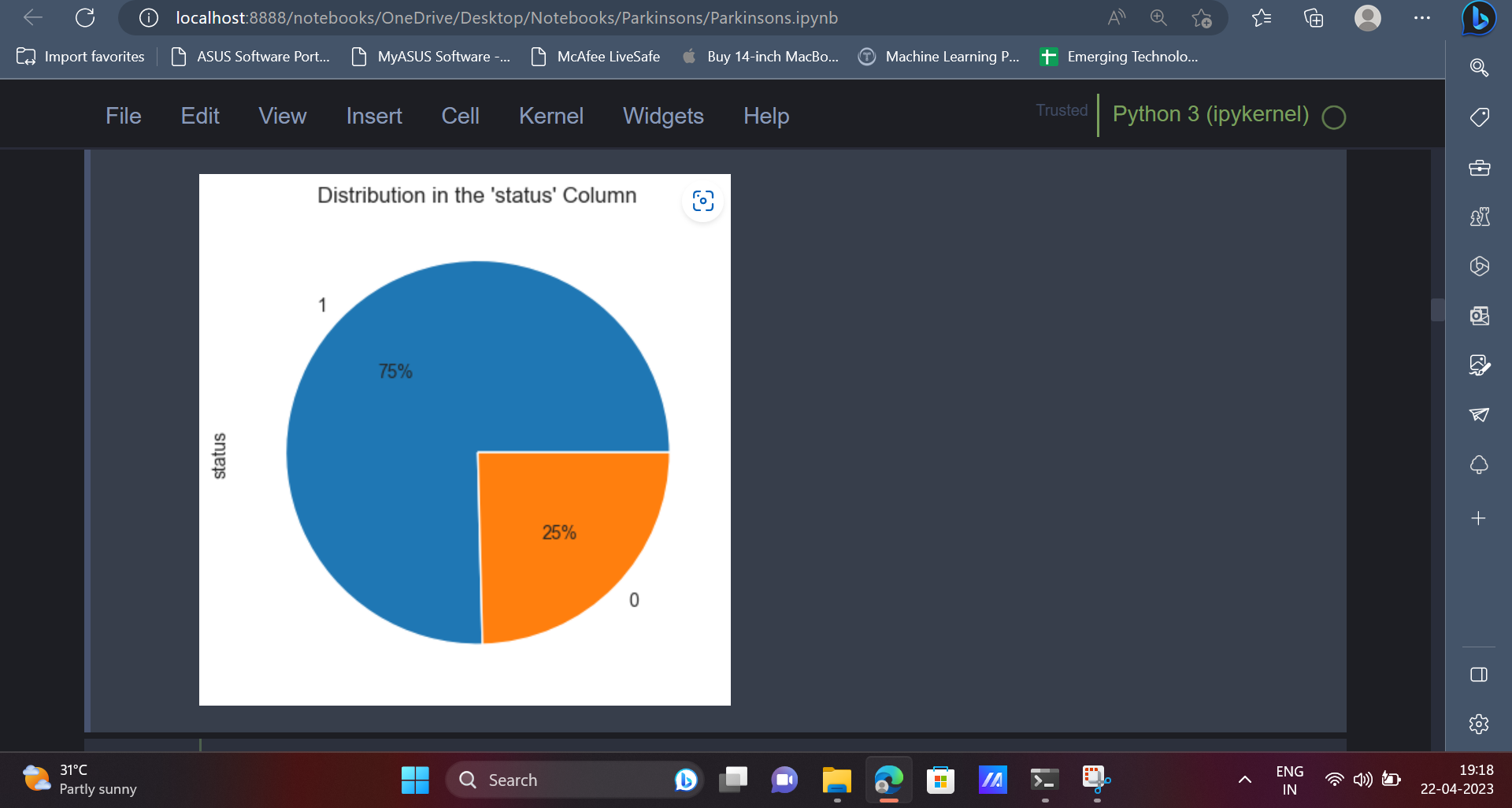
5. Datasets usually contain extreme values, which are usually called outliers. Outliers present in the dataset can be analysed using the below code. There are some outliers as we can see some attributes have huge differences in their 75-percentile value and maximum value, which is evident from the below output.

| parkinson\_data.describe() |
| --- |

Output:  
 6. After exploring the dataset and the various features, one can use data visualisation concepts to present the data and the trends present in the data in a visually appealing format. Data Visualization techniques like pie-charts, dist-plot etc can be used for this purpose.

| parkinson\_data['status'].value\_counts().plot(kind='pie', autopct = "%1.0f%%") plt.title("Distribution in the 'status' Column") plt.show() |
| --- |

Output:

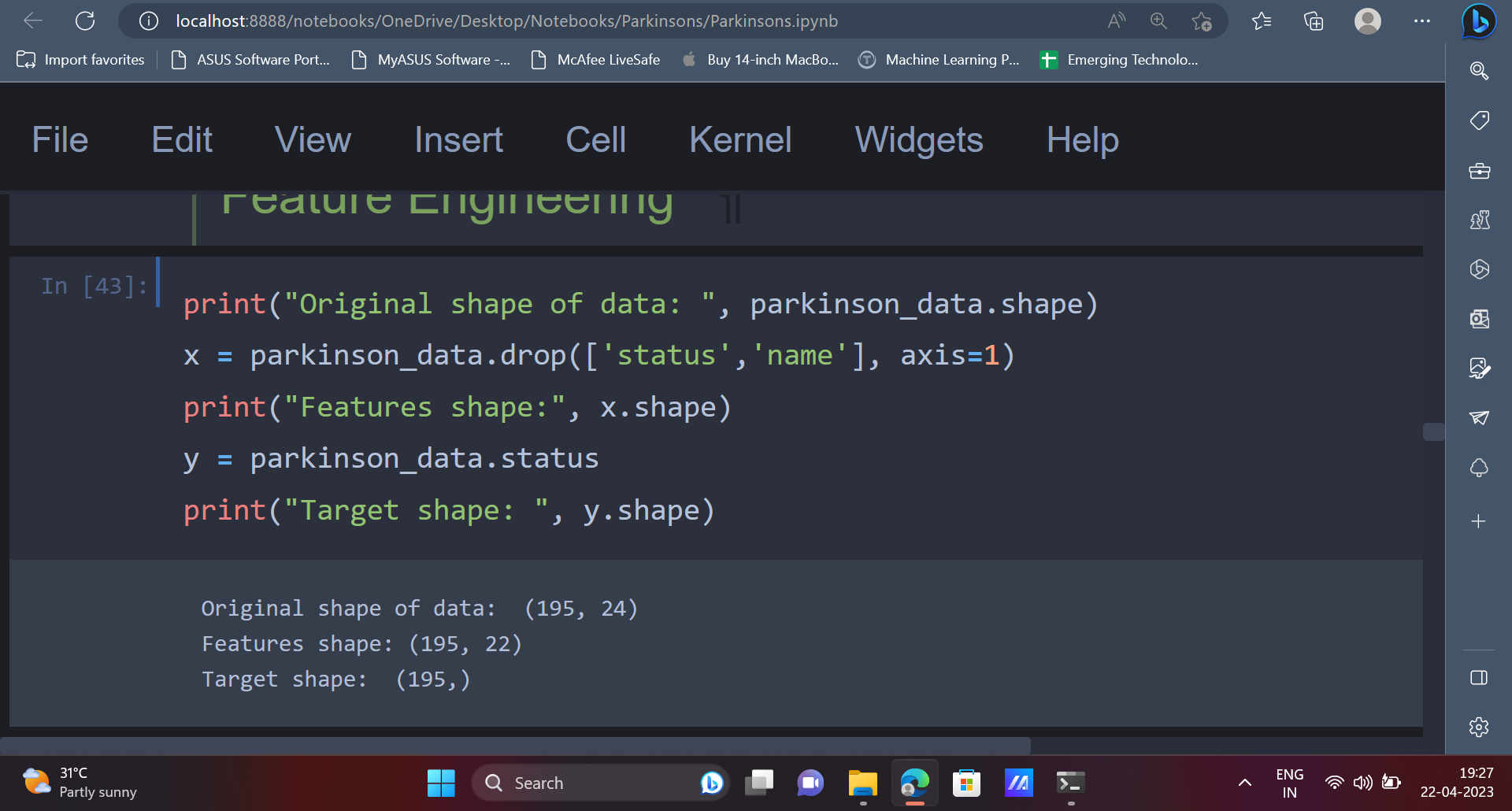


| parkinson\_data.hist(figsize=(15,12)); |  |
| --- | --- |
|  |  |

7. After representing the data visually, it is now time to perform Feature Engineering, to select the required features which will be used to train the Machine Learning model.

| print("Original shape of data: ", parkinson\_data.shape) x = parkinson\_data.drop(['status','name'], axis=1) print("Features shape:", x.shape) y = parkinson\_data.status print("Target shape: ", y.shape) |
| --- |

Output:



8. The dataset contains attributes which belong to a wide range. It would be convenient to have all the values belonging to a similar range or a similar scale which would eventually help the machine learning model understand the data in a better way. The MinMaxScaler can be used for this purpose, as shown below.

| from sklearn.preprocessing import MinMaxScaler scaler = MinMaxScaler((-1, 1)) # fits the data to the MinMaxScaler X = scaler.fit\_transform(x) |
| --- |

The primary purpose of utilising MinMaxScaler is to standardise feature scales and bring them to a similar level, leading to enhanced performance of specific machine learning algorithms. This technique is especially beneficial for algorithms that utilise distance calculations, like support vector machines or k-nearest neighbours.

9. In the next step, the dataset will be split into training data and testing data. The training data will be used to train the model and the testing data will be used to evaluate the model’s performance.

| from sklearn.model\_selection import train\_test\_split x\_train, x\_test, y\_train, y\_test=train\_test\_split(X, y, test\_size=0.2) |
| --- |

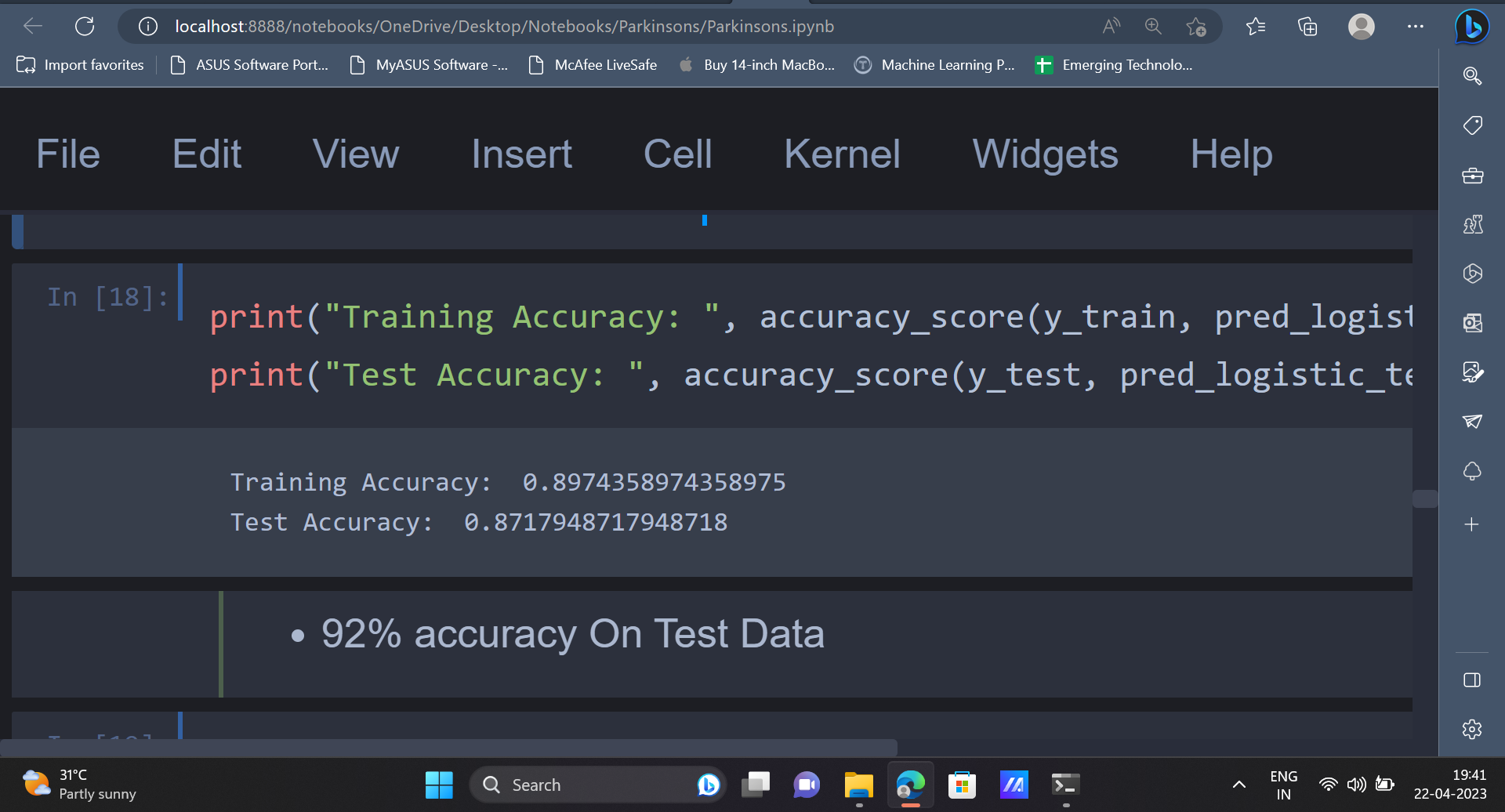
10. Now that the data is split into training data and testing data, various machine learning models can be trained using the training data. We will be using models like Logistic Regression, Decision Tree, Random Forest Classifier etc.

| from sklearn.linear\_model import LogisticRegression # Importing the Logistic Regression and creating an instance of it clf = LogisticRegression() #Train Model clf.fit(x\_train, y\_train) |
| --- |

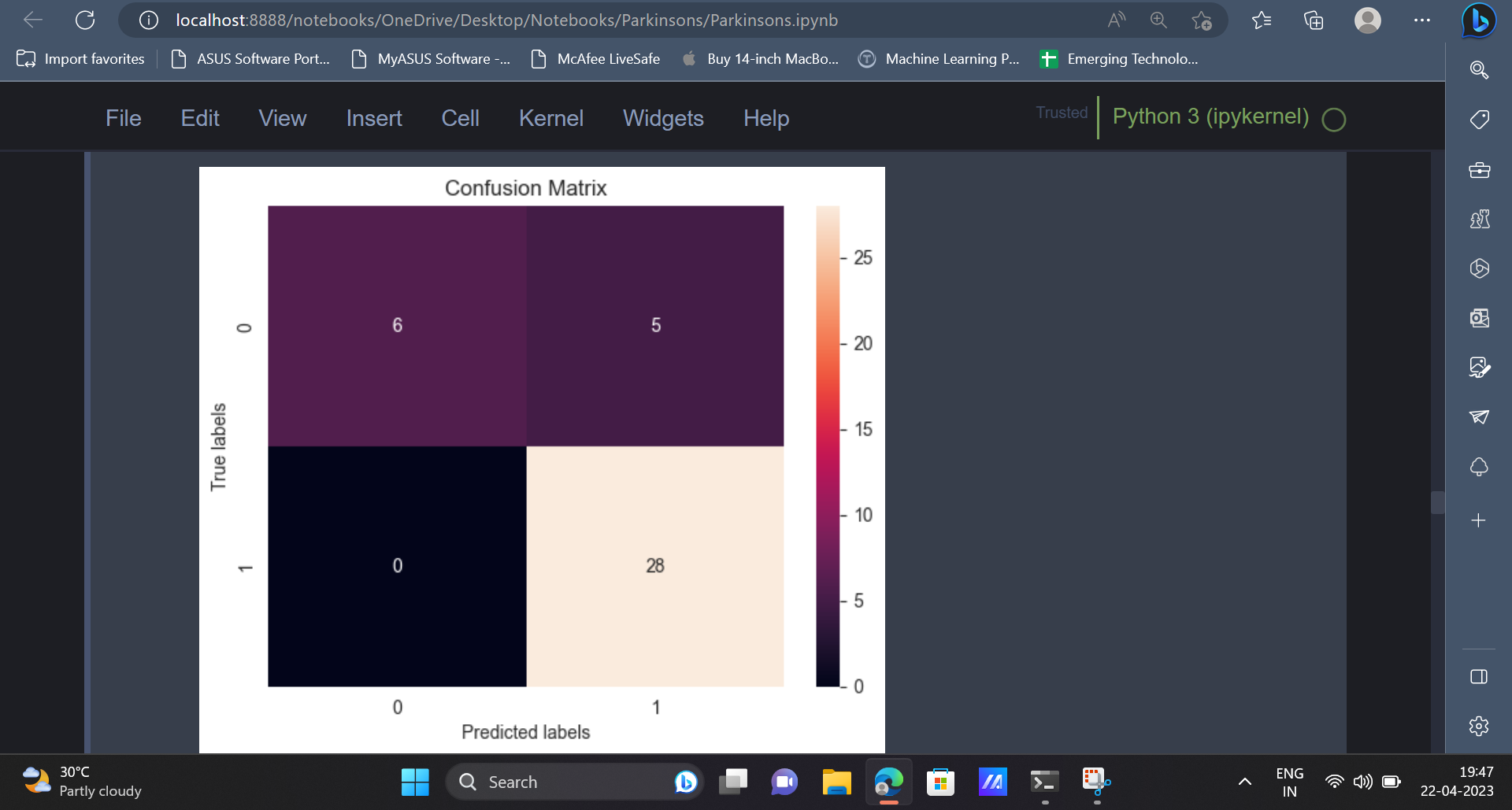
| # Prediction on Test and Train Set  pred\_logistic\_test = clf.predict(x\_test) pred\_logistic\_train = clf.predict(x\_train) |
| --- |

11. Once the model is trained, the model’s performance can be tested using various metrics like confusion matrix, classification report, accuracy score etc.

| from sklearn.metrics import confusion\_matrix,accuracy\_score, classification\_report print("Training Accuracy: ", accuracy\_score(y\_train, pred\_logistic\_train)) print("Test Accuracy: ", accuracy\_score(y\_test, pred\_logistic\_test)) |
| --- |

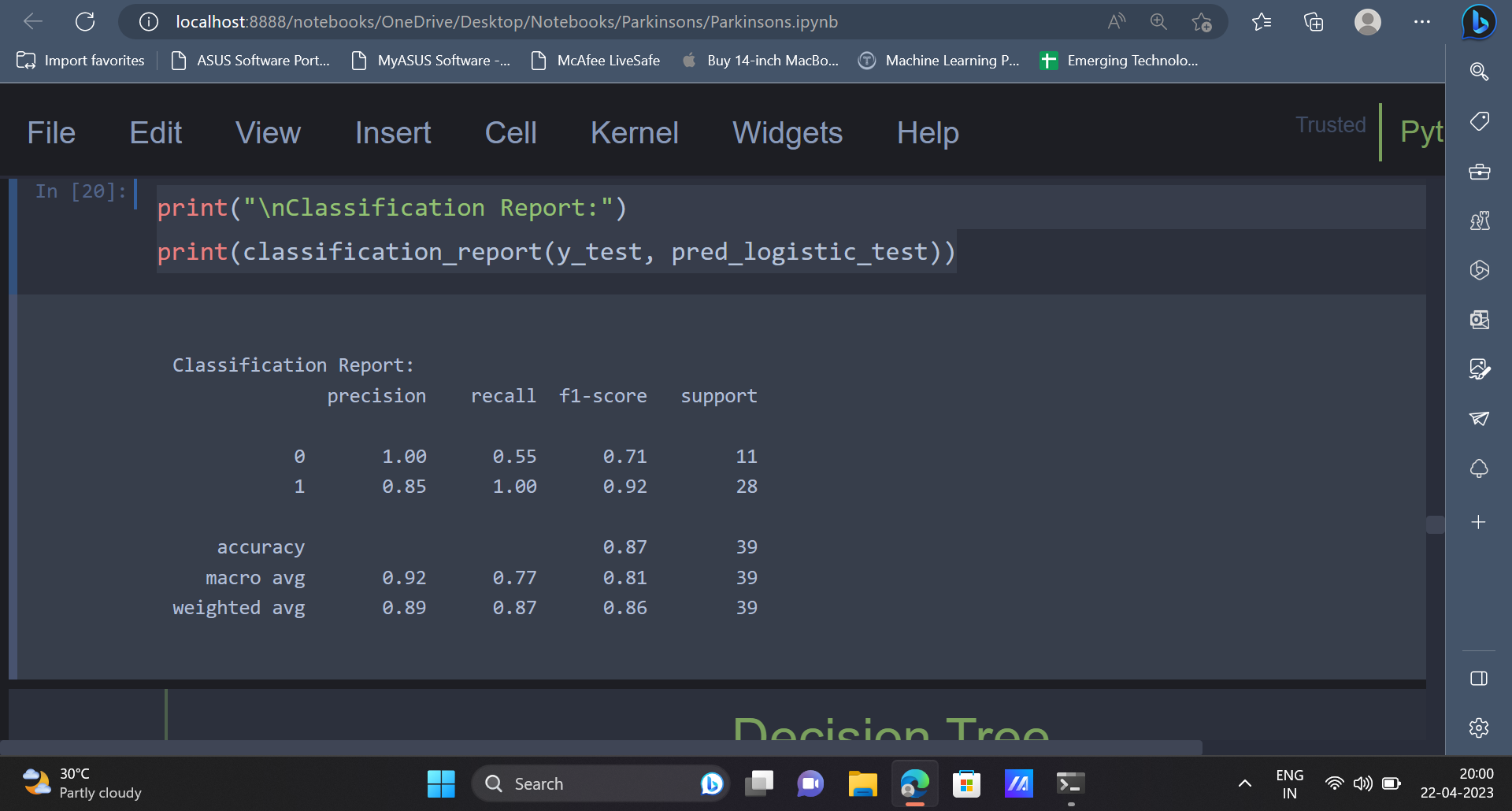
Output

| cm = confusion\_matrix(y\_test, pred\_logistic\_test) ax= plt.subplot() sns.heatmap(cm, annot=True, ax = ax); ax.set\_xlabel('Predicted labels') ax.set\_ylabel('True labels')  ax.set\_title('Confusion Matrix') plt.show() |
| --- |

Output:

| print("\nClassification Report:") print(classification\_report(y\_test, pred\_logistic\_test)) |
| --- |

Output:



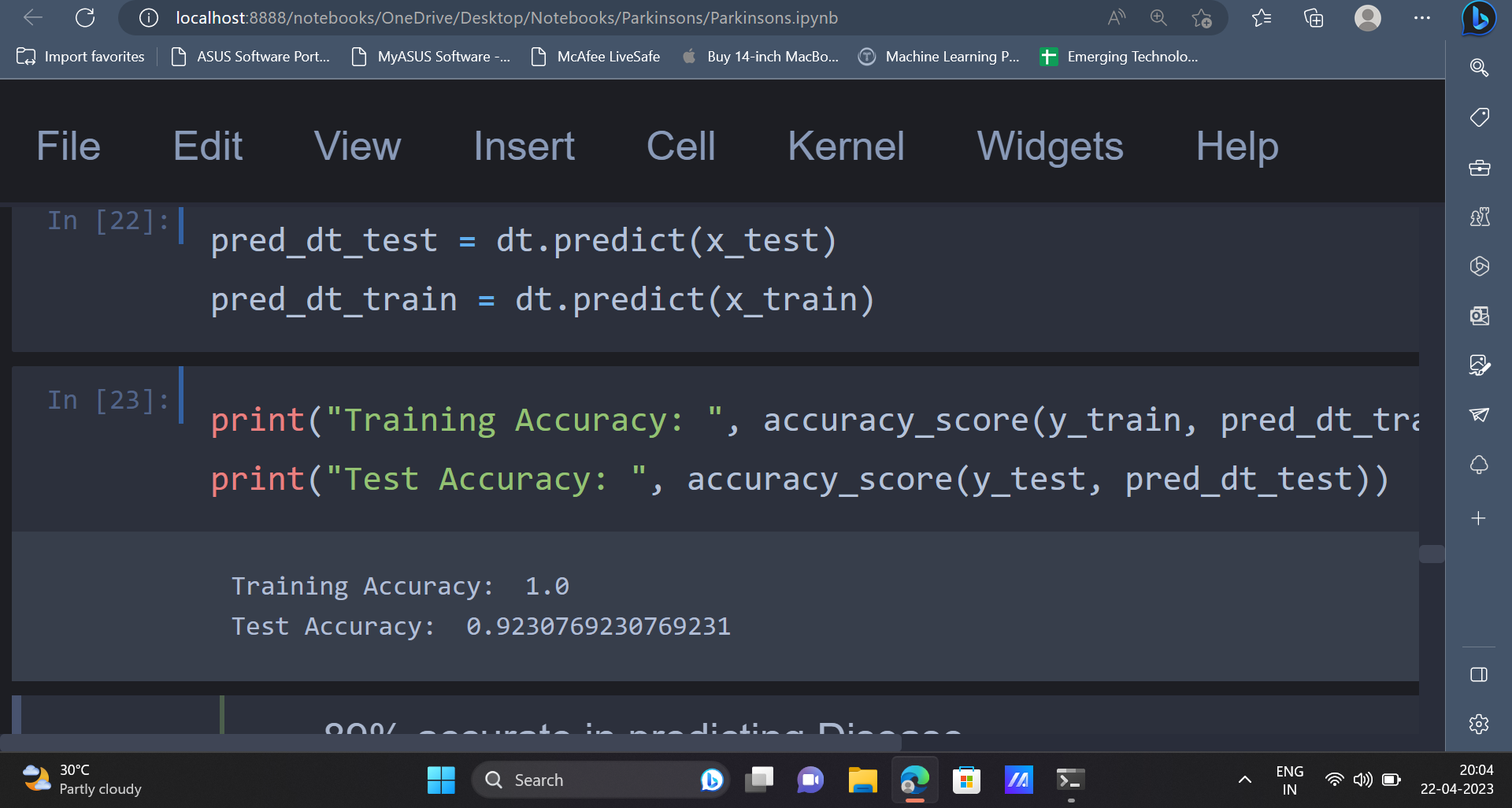
12. The next model, which we will be using is the Decision Tree Classifier.

| from sklearn.tree import DecisionTreeClassifier dt = DecisionTreeClassifier() # Train model  dt.fit(x\_train, y\_train) |
| --- |

| pred\_dt\_test = dt.predict(x\_test) pred\_dt\_train = dt.predict(x\_train)  # Making predictions using the data |
| --- |

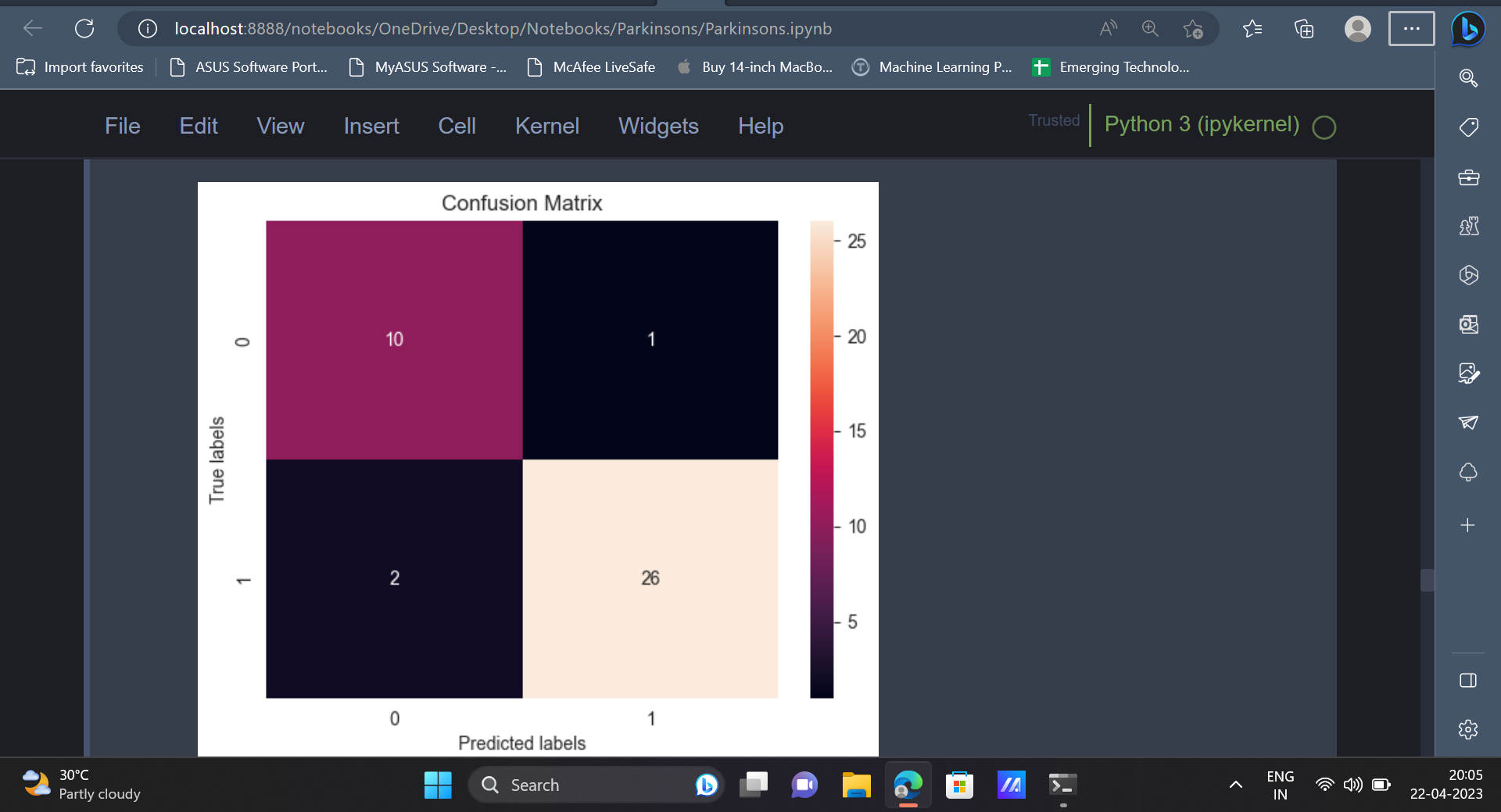
| print("Training Accuracy: ", accuracy\_score(y\_train, pred\_dt\_train)) print("Test Accuracy: ", accuracy\_score(y\_test, pred\_dt\_test)) |
| --- |

Output:

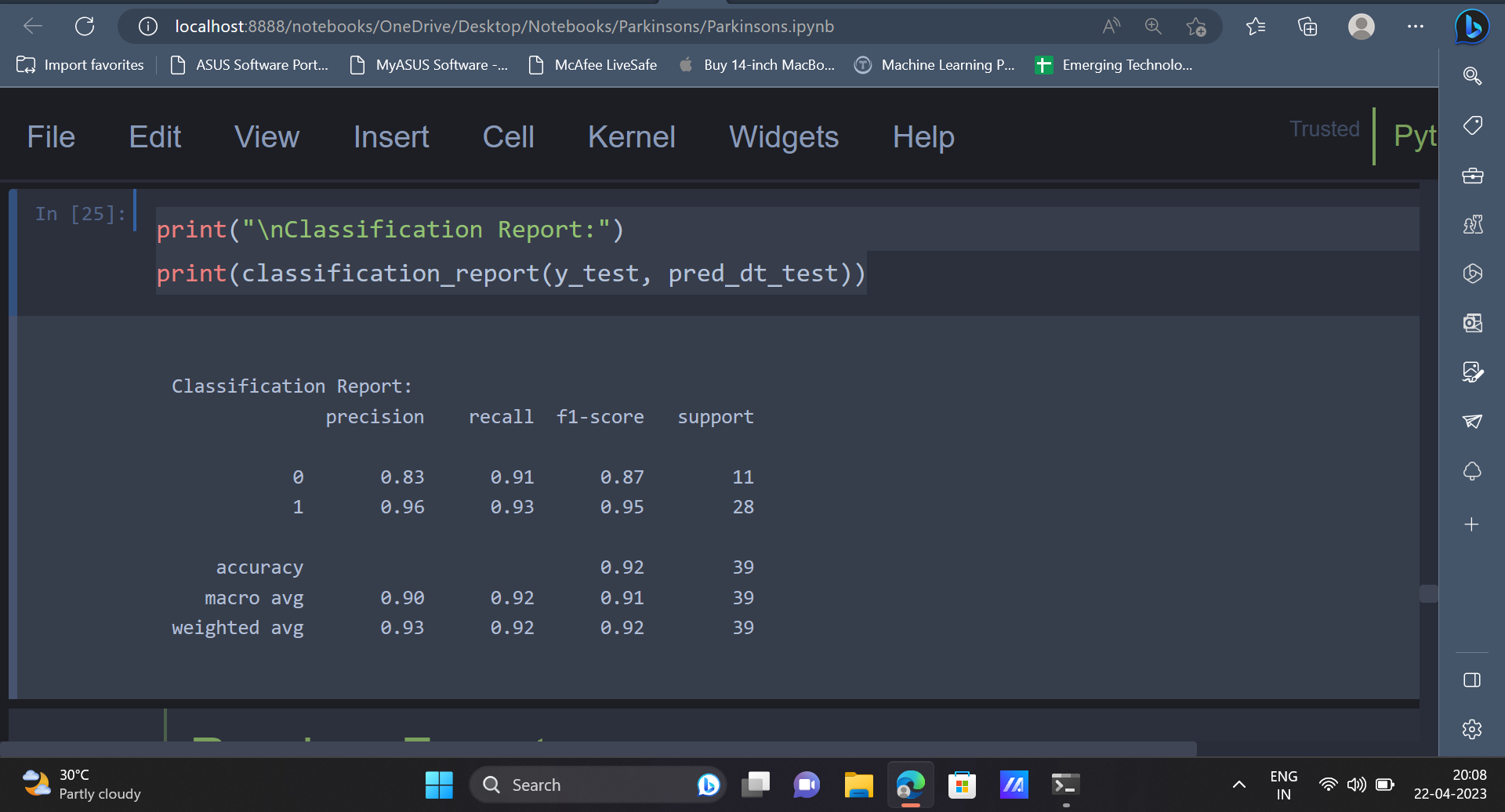


| cm = confusion\_matrix(y\_test, pred\_dt\_test) ax= plt.subplot() sns.heatmap(cm, annot=True, ax = ax); ax.set\_xlabel('Predicted labels') ax.set\_ylabel('True labels')  ax.set\_title('Confusion Matrix') plt.show() |
| --- |

Output:



| print("\nClassification Report:") print(classification\_report(y\_test, pred\_dt\_test)) |
| --- |



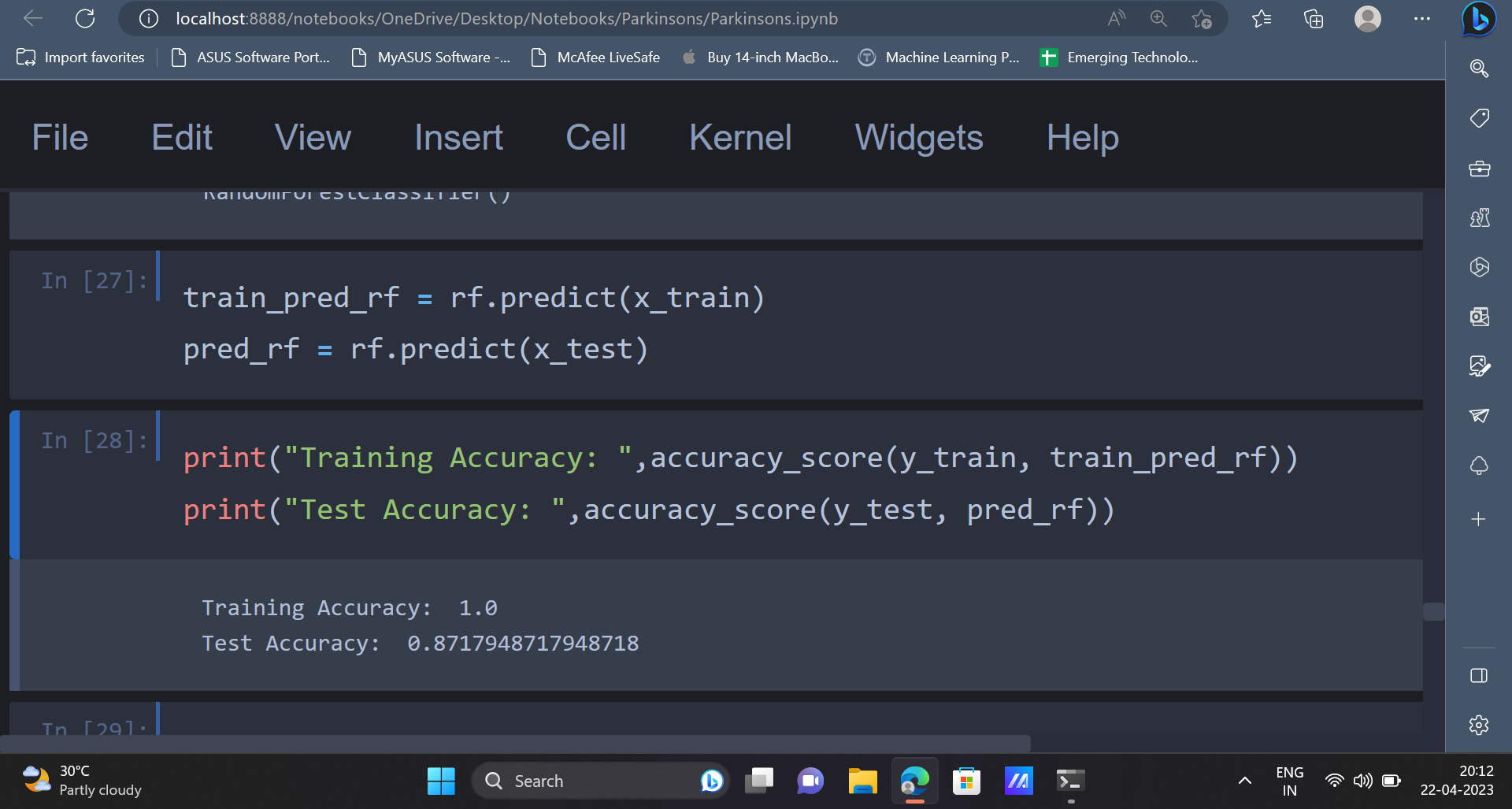
13. After the Decision Tree Classifier, the Random Forest Classifier can be used to predict whether a person will have Parkinson’s disease or not.

| from sklearn.ensemble import RandomForestClassifier rf = RandomForestClassifier() rf.fit(x\_train, y\_train) |
| --- |

| train\_pred\_rf = rf.predict(x\_train) pred\_rf = rf.predict(x\_test) |
| --- |

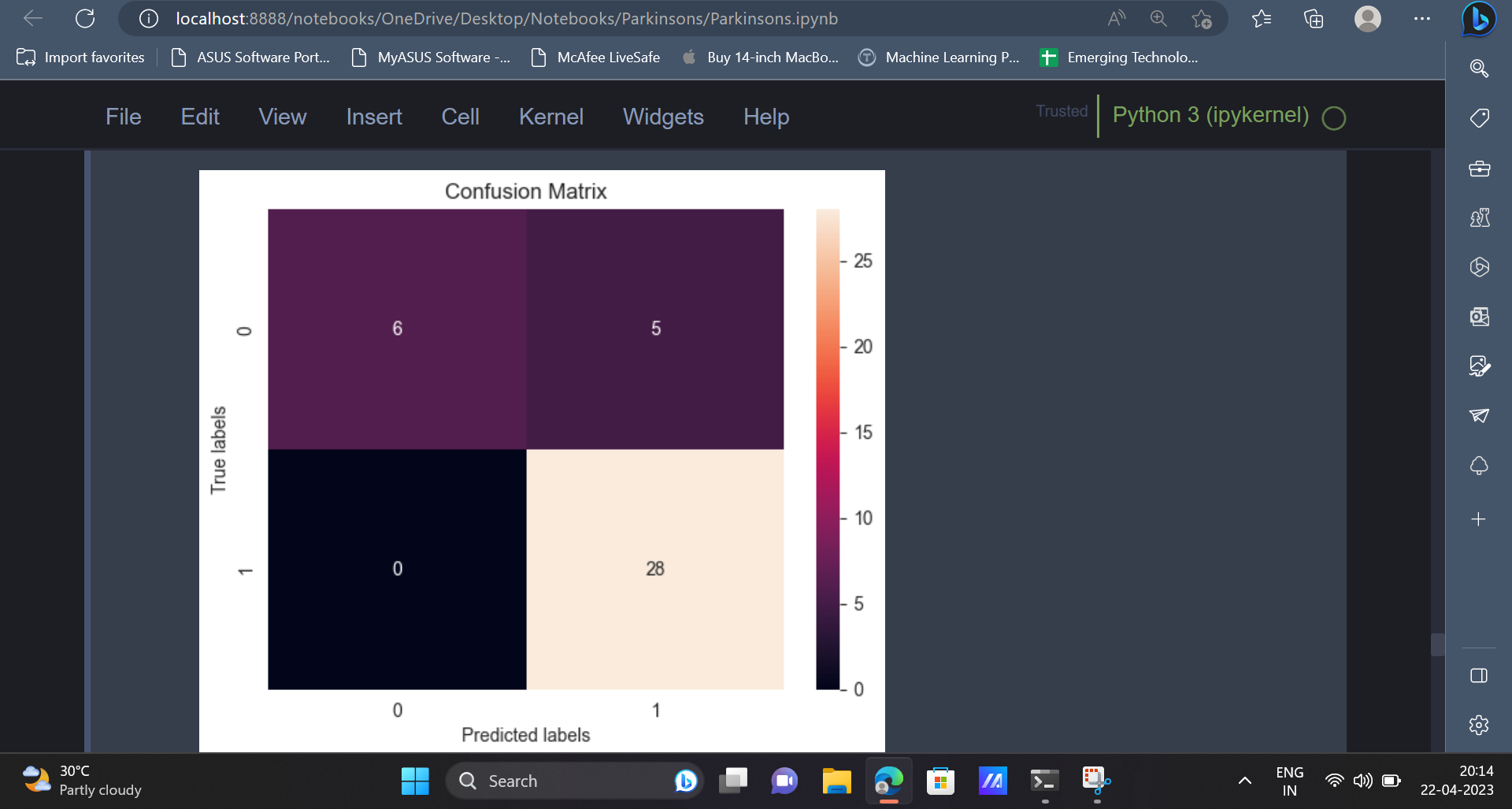
| print("Training Accuracy: ",accuracy\_score(y\_train, train\_pred\_rf)) print("Test Accuracy: ",accuracy\_score(y\_test, pred\_rf)) |
| --- |

Output:



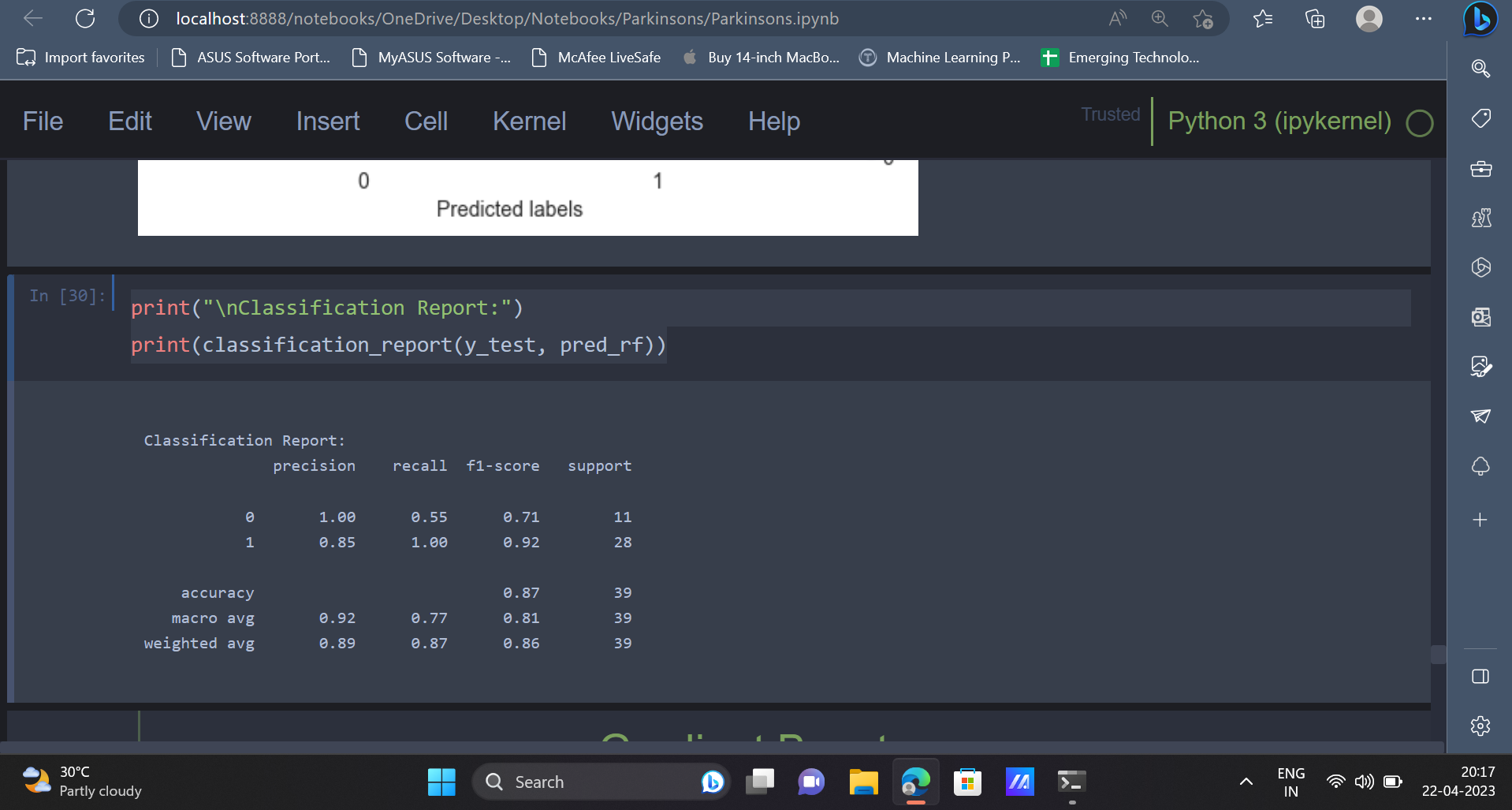
| cm = confusion\_matrix(y\_test, pred\_rf) ax= plt.subplot() sns.heatmap(cm, annot=True, ax = ax); ax.set\_xlabel('Predicted labels') ax.set\_ylabel('True labels')  ax.set\_title('Confusion Matrix') plt.show() |
| --- |

Output:



| print("\nClassification Report:") print(classification\_report(y\_test, pred\_rf)) |
| --- |

Output:

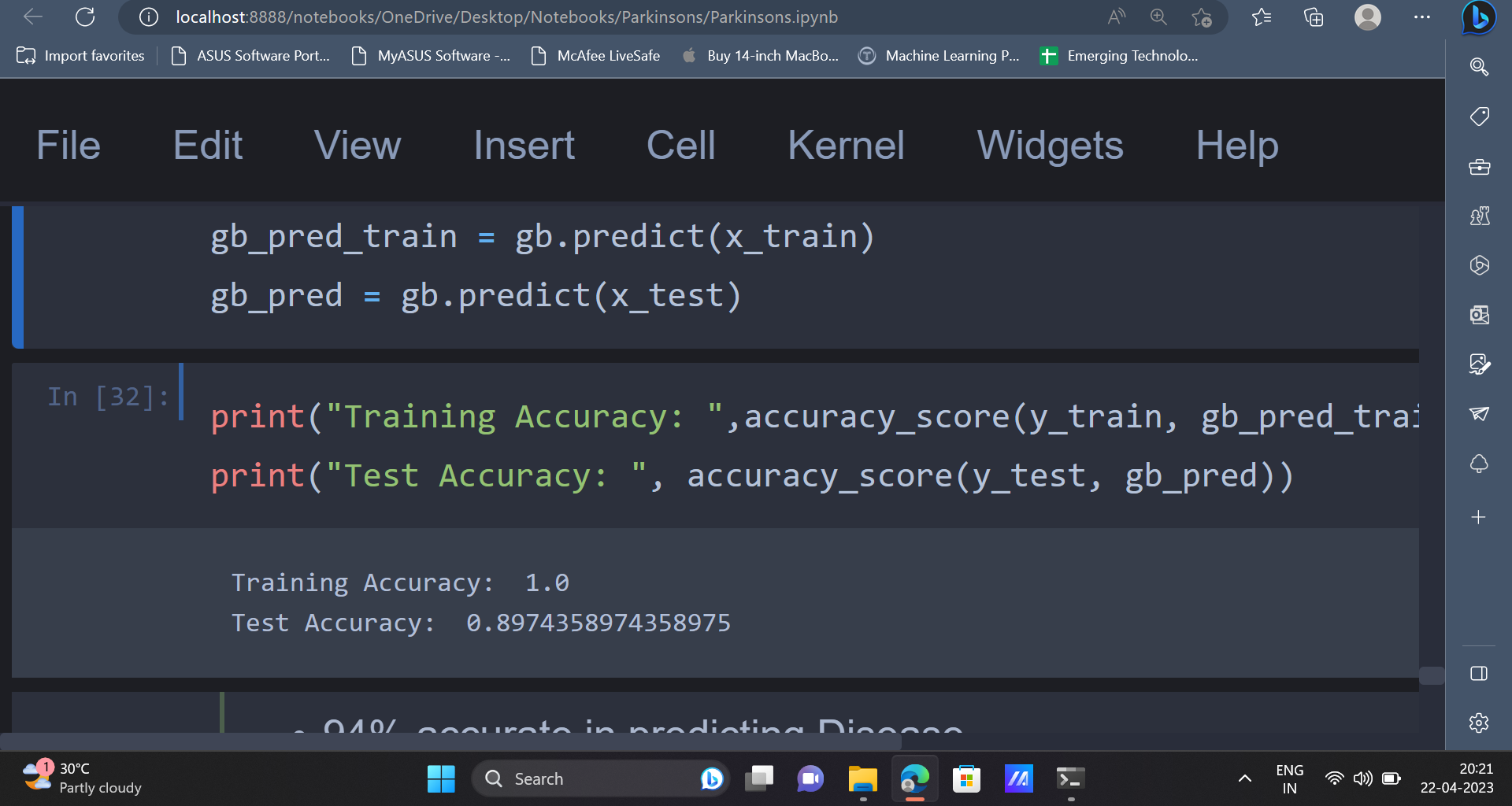


14. The last algorithm which will be used is the Gradient Boosting Algorithm.

| from sklearn.ensemble import GradientBoostingClassifier gb = GradientBoostingClassifier() # Training model gb.fit(x\_train, y\_train)  # Prediction on test and train set gb\_pred\_train = gb.predict(x\_train) gb\_pred = gb.predict(x\_test) |
| --- |

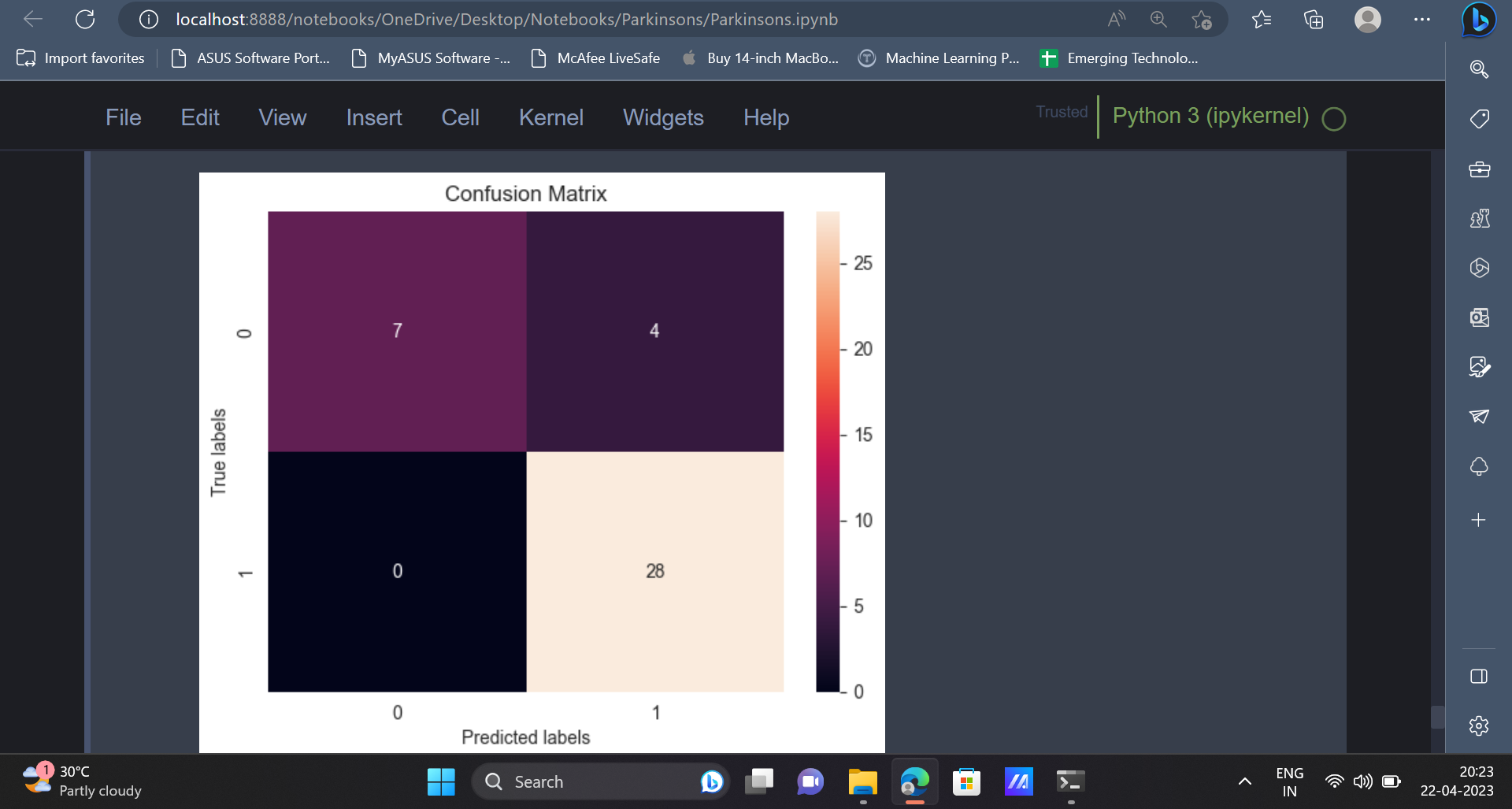
| print("Training Accuracy: ",accuracy\_score(y\_train, gb\_pred\_train)) print("Test Accuracy: ", accuracy\_score(y\_test, gb\_pred)) |
| --- |

Output:



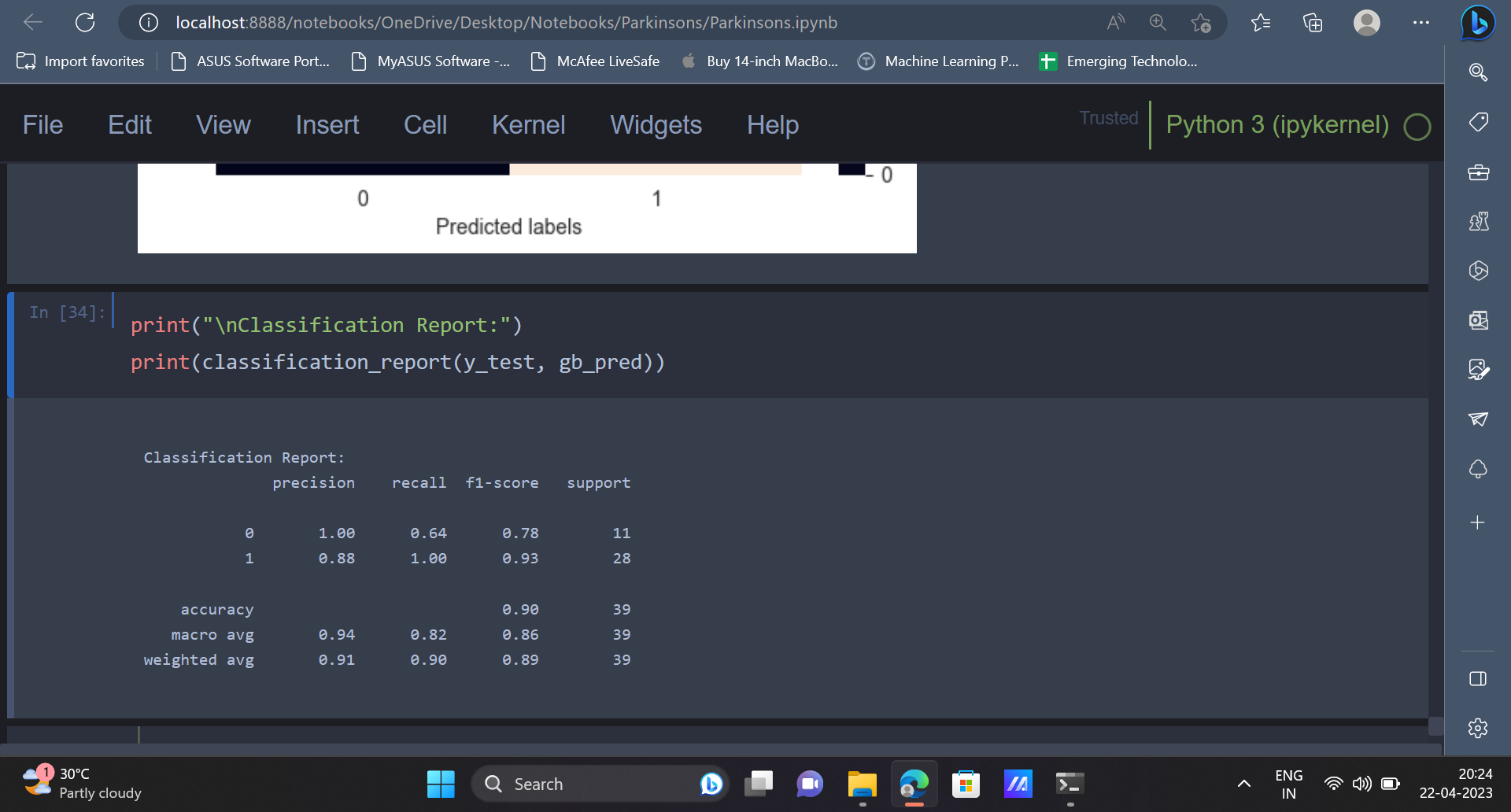
| cm = confusion\_matrix(y\_test, gb\_pred) ax= plt.subplot() sns.heatmap(cm, annot=True, ax = ax); ax.set\_xlabel('Predicted labels') ax.set\_ylabel('True labels')  ax.set\_title('Confusion Matrix') plt.show() |
| --- |

Output:



| print("\nClassification Report:") print(classification\_report(y\_test, gb\_pred)) |
| --- |

Output:



**Conclusion**

The implementation of machine learning algorithms in Parkinson's disease detection is a promising and efficient approach. By analysing the Parkinson's Disease Data Set, we were able to create models that can accurately predict the presence of Parkinson's disease in a patient, with high levels of accuracy and precision.

Nevertheless, it is crucial to emphasise that these models are not a substitute for clinical assessment by trained medical professionals. Therefore, the use of machine learning algorithms for Parkinson's disease detection should be viewed in combination with traditional diagnostic methods. Additionally, further research and development are necessary to refine the accuracy and precision of the models, as well as extend the dataset used to train them.