AI-BASED DIABETES PREDICTION USING MACHINE LEARNING ALGORITHM

Phase 4: Development Part 2

TEAM MEMBERS :

- **GOPINATH N (*TEAM LEADER*)**
- **HARSHA VARTHAN M**
- **ARUN KUMAR M**
- **ARUL SELVAN V**
- **HARI HARAN R**

TOPIC:

continue building the diabetes prediction model by Ml algorithm, training the model and evaluating it's performance.

INTRODUCTION:

Diabetes is a chronic disease that affects how your body turns food into energy. With diabetes, your body either resists the effects of insulin—a hormone that regulates the movement of sugar into your cells—or doesn't produce enough insulin to maintain normal glucose levels.

Early detection and treatment of diabetes can help prevent serious health complications, such as heart disease, stroke, blindness, and kidney disease. Machine learning algorithms can be used to predict diabetes risk based on a variety of factors, including age, gender, family history, medical history, and lifestyle habits.

Selecting a machine learning algorithm

There are many different machine learning algorithms that 543can be used to predict diabetes. Some of the most common algorithms include:

- Logistic regression: A simple but effective algorithm for predicting binary outcomes, such as whether or not someone has diabetes.
- Support vector machines (SVM): A more complex algorithm that can be used to predict both binary and continuous outcomes.
- Random forests: An ensemble learning algorithm that combines the predictions of multiple decision trees to produce a more accurate prediction.
- Deep neural networks: A type of artificial intelligence that can learn complex patterns in data.

The best machine learning algorithm to use for diabetes prediction will depend on the specific dataset being used and the desired performance metrics.

Training the model

Once a machine learning algorithm has been selected, it needs to be trained on a dataset of labeled examples. Labeled examples are data points where the target variable (in this case, whether or not someone has diabetes) is known.

The training process involves feeding the algorithm the labeled examples and allowing it to learn the relationship between the input features (e.g., age, gender, medical history) and the target variable.

Evaluating the performance of the model

Once the model has been trained, it needs to be evaluated on a held-out test set. The test set is a dataset of labeled examples that was not used to train the model.

Evaluating the model on the test set provides an estimate of how well the model will perform on new data. Common performance metrics for diabetes prediction include accuracy, precision, recall, and F1 score.

<u>**Conclusion**</u>

Machine learning algorithms can be used to predict diabetes risk with high accuracy. By predicting diabetes risk, clinicians can identify people who are at high risk and provide them with early intervention and preventive care.

Here are some additional tips for selecting, training, and evaluating a machine learning model for diabetes prediction:

- Use a variety of features. The more features you use to train your model, the more accurate it will be. However, it is important to select features that are relevant to diabetes prediction and to avoid overfitting the model to the training data.
- Use cross-validation. Cross-validation is a technique that allows you to evaluate the performance of your model on multiple datasets. This is important to avoid overfitting the model to the training data.
- Tune the hyperparameters. Hyperparameters are parameters that control the behavior of the machine learning algorithm. Tuning the hyperparameters can improve the performance of the model.
- Use a variety of evaluation metrics. Accuracy is a common evaluation metric, but it is important to also consider other metrics, such as precision, recall, and F1 score. This is because accuracy can

be misleading in some cases, such as when the dataset is imbalanced.

Once you have selected, trained, and evaluated your model, you can use it to predict diabetes risk in new patients. This information can be used to guide clinical decision-making and to provide patients with personalized recommendations for prevention and care.

OVERVIEW OF THE PROCESS:

Overview of the Process

To select a machine learning algorithm for diabetes prediction, you need to consider the following factors:

- Type of data: Is your data structured or unstructured? What are the features of your data?
- Complexity of the problem: How complex is the relationship between the features of your data and the target variable (i.e., whether or not a patient has diabetes)?
- Interpretability: How important is it to be able to interpret how the model makes its predictions?
- Computational resources: How much computational power do you have available?

Once you have considered these factors, you can start to narrow down your choices of machine learning algorithms. Some popular algorithms for diabetes prediction include:

- Logistic regression: A simple but effective algorithm for binary classification tasks.
- Support vector machines (SVMs): A powerful algorithm that can learn complex relationships between features and the target variable.

- Random forests: An ensemble learning algorithm that builds multiple decision trees and averages their predictions to produce a final prediction.
- Gradient boosting machines (GBMs): Another ensemble learning algorithm that builds sequential models to improve the performance of the previous model.
- Deep neural networks (DNNs): A type of machine learning model that can learn complex patterns from data.

Once you have selected a machine learning algorithm, you need to train the model on your data. This involves feeding the model your data and allowing it to learn the relationships between the features and the target variable.

Once the model is trained, you need to evaluate its performance on a held-out test set. This will give you an idea of how well the model will generalize to new data. If the model performs well on the test set, you can deploy it to production.

<u>Step-by-Step Procedure:</u>

To train a machine learning model for diabetes prediction, you can follow these steps:

1. Prepare your data. This includes cleaning the data, handling missing values, and converting categorical variables to numerical variables.
2. Split your data into training and test sets. The training set will be used to train the model, and the test set will be used to evaluate the model's performance.
3. Choose a machine learning algorithm. Consider the factors mentioned above when choosing an algorithm.
4. Train the model. Feed the training data to the model and allow it to learn the relationships between the features and the target variable.

5. Evaluate the model. Evaluate the model's performance on the test set.
6. Deploy the model. If the model performs well on the test set, you can deploy it to production.

   Here is an example of how to train a random forest model for diabetes prediction using the Python programming language:

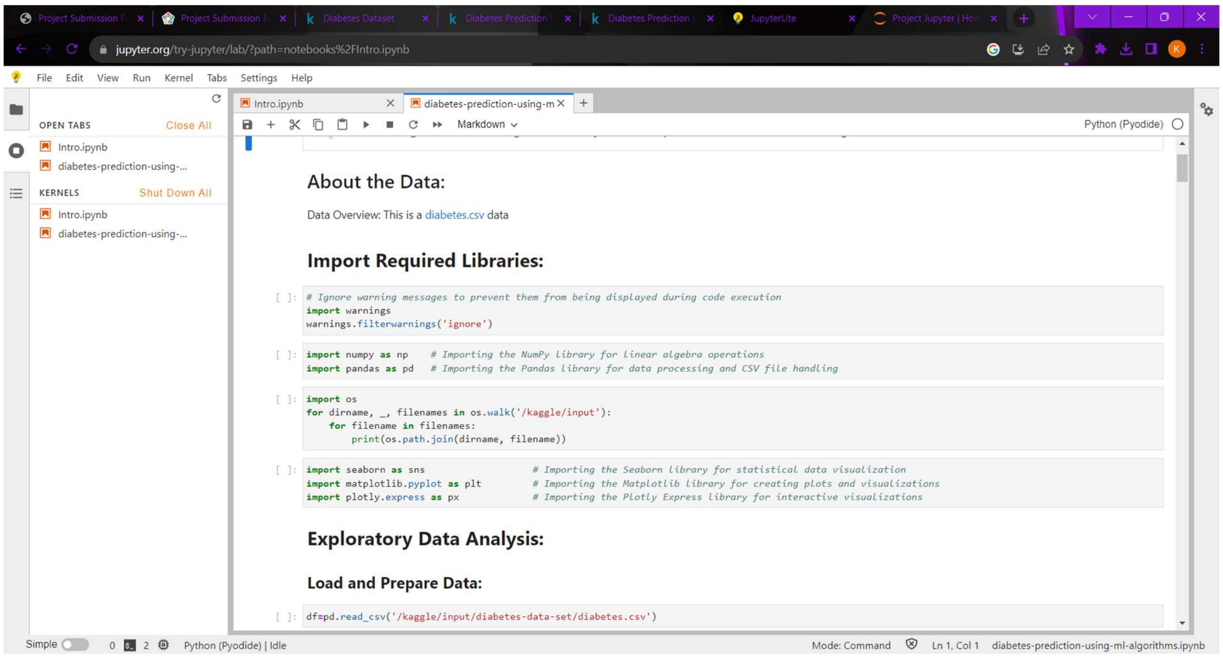
   Python

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# Load the data
data = pd.read_csv("diabetes.csv")

# Split the data into training and test sets
X_train, X_test, y_train, y_test =
train_test_split(data.drop(columns=["Outcome"]), data["Outcome"],
test_size=0.25)

# Create the random forest model
model = RandomForestClassifier()

# Train the model
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
accuracy = np.mean(y_pred == y_test)
print("Accuracy:", accuracy)
```

# Deploy the model
# Save the model to a file or deploy it to a web service

This is just a basic example, and you may need to adjust the parameters of the random forest model or try other algorithms to get the best results.

PROGRAMS:



About the Data:

Data Overview: This is a diabetes.csv data

Import Required Libraries:

```
[ ]: # Ignore warning messages to prevent them from being displayed during code execution
     import warnings
     warnings.filterwarnings('ignore')
```

```
[ ]: import numpy as np    # Importing the NumPy library for linear algebra operations
     import pandas as pd   # Importing the Pandas library for data processing and CSV file handling
```

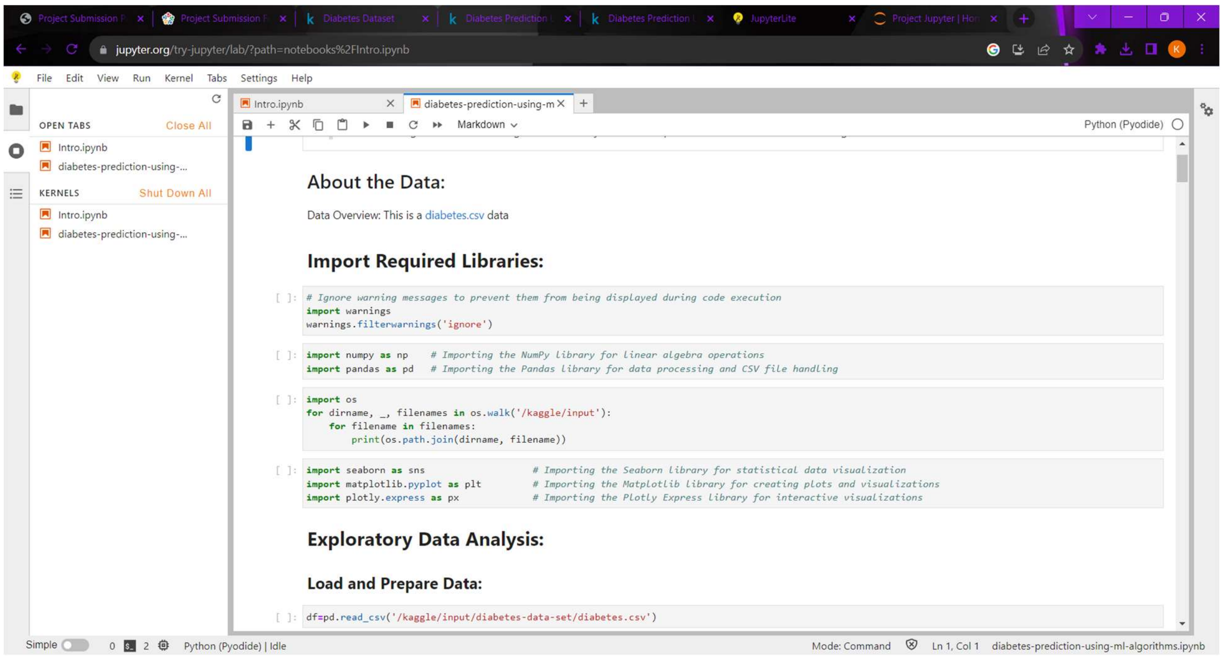
```
[ ]: import os
     for dirname, _, filenames in os.walk('/kaggle/input'):
         for filename in filenames:
             print(os.path.join(dirname, filename))
```

```
[ ]: import seaborn as sns                 # Importing the Seaborn library for statistical data visualization
     import matplotlib.pyplot as plt       # Importing the Matplotlib library for creating plots and visualizations
     import plotly.express as px           # Importing the Plotly Express library for interactive visualizations
```

Exploratory Data Analysis:

Load and Prepare Data:

```
[ ]: df=pd.read_csv('/kaggle/input/diabetes-data-set/diabetes.csv')
```


jupyter.org/try-jupyter/lab/?path=notebooks%2FIntro.ipynb
File Edit View Run Kernel Tabs Settings Help
OPEN TABS Close All
Intro.ipynb
diabetes-prediction-using-...
KERNELS Shut Down All
Intro.ipynb
diabetes-prediction-using-...
Markdown
Python (Pyodide)
Simple 0 2 Python (Pyodide) | Idle
Mode: Command Ln 1, Col 1 diabetes-prediction-using-ml-algorithms.ipynb

About the Data:

Data Overview: This is a diabetes.csv data

Import Required Libraries:

```
[ ]: # Ignore warning messages to prevent them from being displayed during code execution
     import warnings
     warnings.filterwarnings('ignore')
```

```
[ ]: import numpy as np    # Importing the NumPy library for linear algebra operations
     import pandas as pd   # Importing the Pandas library for data processing and CSV file handling
```

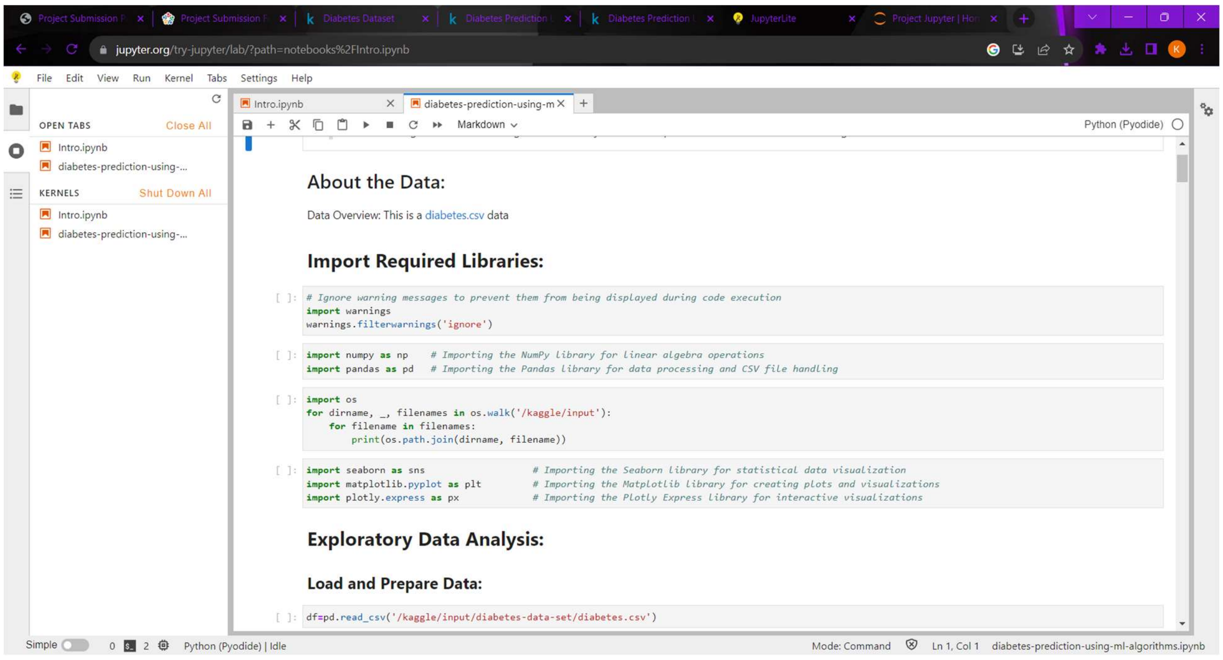
```
[ ]: import os
     for dirname, _, filenames in os.walk('/kaggle/input'):
         for filename in filenames:
             print(os.path.join(dirname, filename))
```

```
[ ]: import seaborn as sns                 # Importing the Seaborn library for statistical data visualization
     import matplotlib.pyplot as plt       # Importing the Matplotlib library for creating plots and visualizations
     import plotly.express as px           # Importing the Plotly Express library for interactive visualizations
```

Exploratory Data Analysis:

Load and Prepare Data:

```
[ ]: df=pd.read_csv('/kaggle/input/diabetes-data-set/diabetes.csv')
```


jupyter.org/try-jupyter/lab/?path=notebooks%2FIntro.ipynb
File Edit View Run Kernel Tabs Settings Help
OPEN TABS Close All
Intro.ipynb
diabetes-prediction-using-...
KERNELS Shut Down All
Intro.ipynb
diabetes-prediction-using-...
Python (Pyodide)
Simple 0 2 Python (Pyodide) | Idle
Mode: Command Ln 1, Col 1 diabetes-prediction-using-ml-algorithms.ipynb

About the Data:

Data Overview: This is a diabetes.csv data

Import Required Libraries:

```
[ ]: # Ignore warning messages to prevent them from being displayed during code execution
     import warnings
     warnings.filterwarnings('ignore')
```

```
[ ]: import numpy as np    # Importing the NumPy library for linear algebra operations
     import pandas as pd   # Importing the Pandas library for data processing and CSV file handling
```

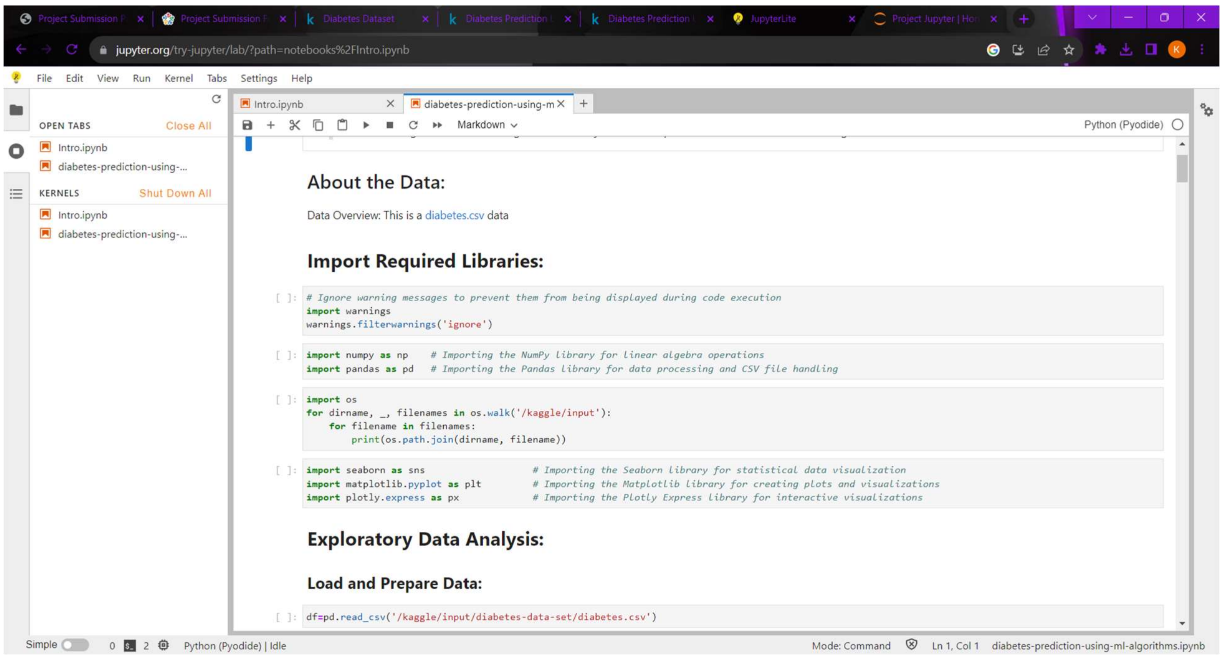
```
[ ]: import os
     for dirname, _, filenames in os.walk('/kaggle/input'):
         for filename in filenames:
             print(os.path.join(dirname, filename))
```

```
[ ]: import seaborn as sns                  # Importing the Seaborn library for statistical data visualization
     import matplotlib.pyplot as plt        # Importing the Matplotlib library for creating plots and visualizations
     import plotly.express as px            # Importing the Plotly Express library for interactive visualizations
```

Exploratory Data Analysis:

Load and Prepare Data:

```
[ ]: df=pd.read_csv('/kaggle/input/diabetes-data-set/diabetes.csv')
```


About the Data:

Data Overview: This is a diabetes.csv data

Import Required Libraries:

[ ]: # Ignore warning messages to prevent them from being displayed during code execution
     import warnings
     warnings.filterwarnings('ignore')

[ ]: import numpy as np   # Importing the NumPy library for linear algebra operations
     import pandas as pd   # Importing the Pandas library for data processing and CSV file handling

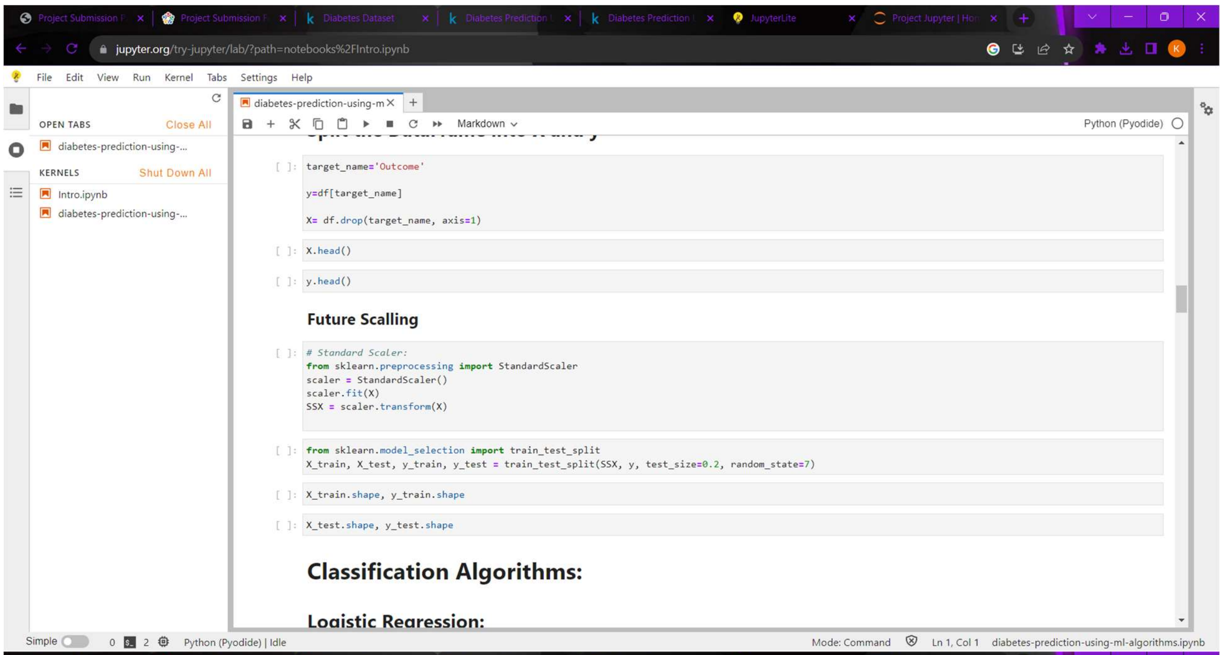
[ ]: import os
     for dirname, _, filenames in os.walk('/kaggle/input'):
         for filename in filenames:
             print(os.path.join(dirname, filename))

[ ]: import seaborn as sns   # Importing the Seaborn library for statistical data visualization
     import matplotlib.pyplot as plt   # Importing the Matplotlib library for creating plots and visualizations
     import plotly.express as px   # Importing the Plotly Express library for interactive visualizations

Exploratory Data Analysis:

Load and Prepare Data:

[ ]: df=pd.read_csv('/kaggle/input/diabetes-data-set/diabetes.csv')


jupyter.org/try-jupyter/lab/?path=notebooks%2FIntro.ipynb
File Edit View Run Kernel Tabs Settings Help
OPEN TABS Close All
diabetes-prediction-using-...
KERNELS Shut Down All
Intro.ipynb
diabetes-prediction-using-...
diabetes-prediction-using-m
Markdown
Python (Pyodide)
Simple 0 2 Python (Pyodide) | Idle
Mode: Command Ln 1, Col 1 diabetes-prediction-using-ml-algorithms.ipynb

```
[ ]: target_name='Outcome'

     y=df[target_name]

     X= df.drop(target_name, axis=1)
```

```
[ ]: X.head()
```

```
[ ]: y.head()
```

Future Scalling

```
[ ]: # Standard Scaler:
     from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()
     scaler.fit(X)
     SSX = scaler.transform(X)
```

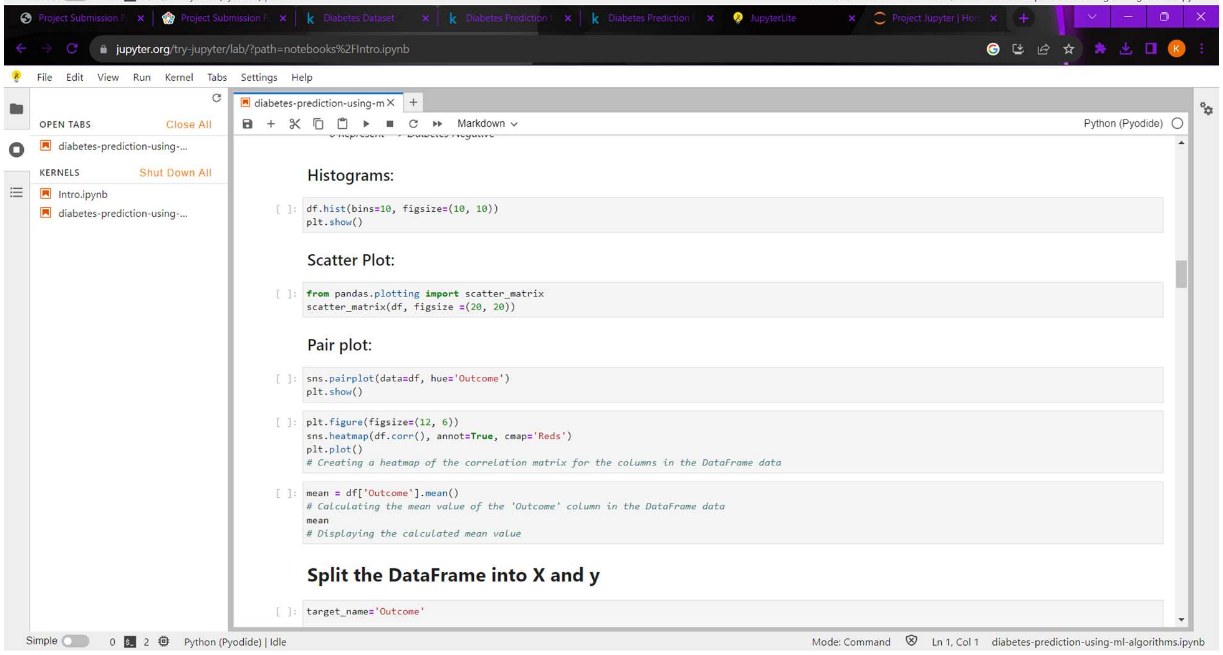
```
[ ]: from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(SSX, y, test_size=0.2, random_state=7)
```

```
[ ]: X_train.shape, y_train.shape
```

```
[ ]: X_test.shape, y_test.shape
```

Classification Algorithms:

Logistic Regression:


jupyter.org/try-jupyter/lab/?path=notebooks%2FIntro.ipynb
File Edit View Run Kernel Tabs Settings Help
OPEN TABS Close All
diabetes-prediction-using-...
KERNELS Shut Down All
Intro.ipynb
diabetes-prediction-using-...
diabetes-prediction-using-m
Markdown
Python (Pyodide)
Simple 0 2 Python (Pyodide) | Idle
Mode: Command Ln 1, Col 1 diabetes-prediction-using-ml-algorithms.ipynb

Histograms:

```
[ ]: df.hist(bins=10, figsize=(10, 10))
     plt.show()
```

Scatter Plot:

```
[ ]: from pandas.plotting import scatter_matrix
     scatter_matrix(df, figsize =(20, 20))
```

Pair plot:

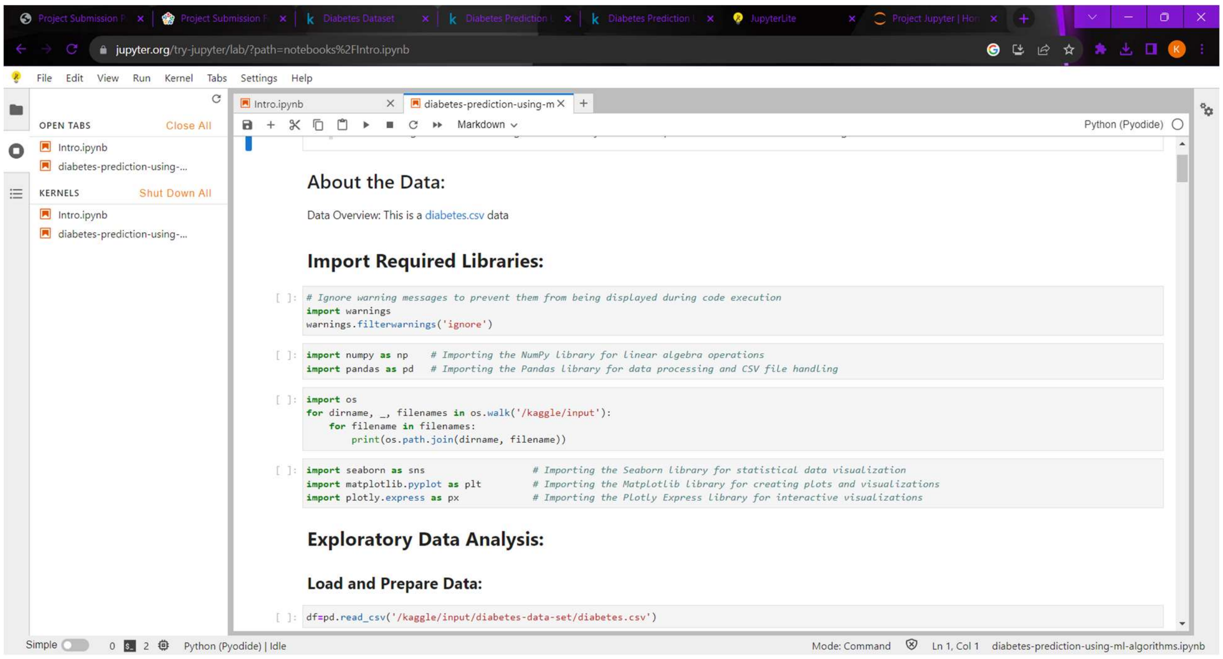
```
[ ]: sns.pairplot(data=df, hue='Outcome')
     plt.show()
```

```
[ ]: plt.figure(figsize=(12, 6))
     sns.heatmap(df.corr(), annot=True, cmap='Reds')
     plt.plot()
     # Creating a heatmap of the correlation matrix for the columns in the DataFrame data
```

```
[ ]: mean = df['Outcome'].mean()
     # Calculating the mean value of the 'Outcome' column in the DataFrame data
     mean
     # Displaying the calculated mean value
```

Split the DataFrame into X and y

```
[ ]: target_name='Outcome'
```



About the Data:

Data Overview: This is a diabetes.csv data

Import Required Libraries:

```
# Ignore warning messages to prevent them from being displayed during code execution
import warnings
warnings.filterwarnings('ignore')
```

```
import numpy as np    # Importing the NumPy library for linear algebra operations
import pandas as pd   # Importing the Pandas library for data processing and CSV file handling
```

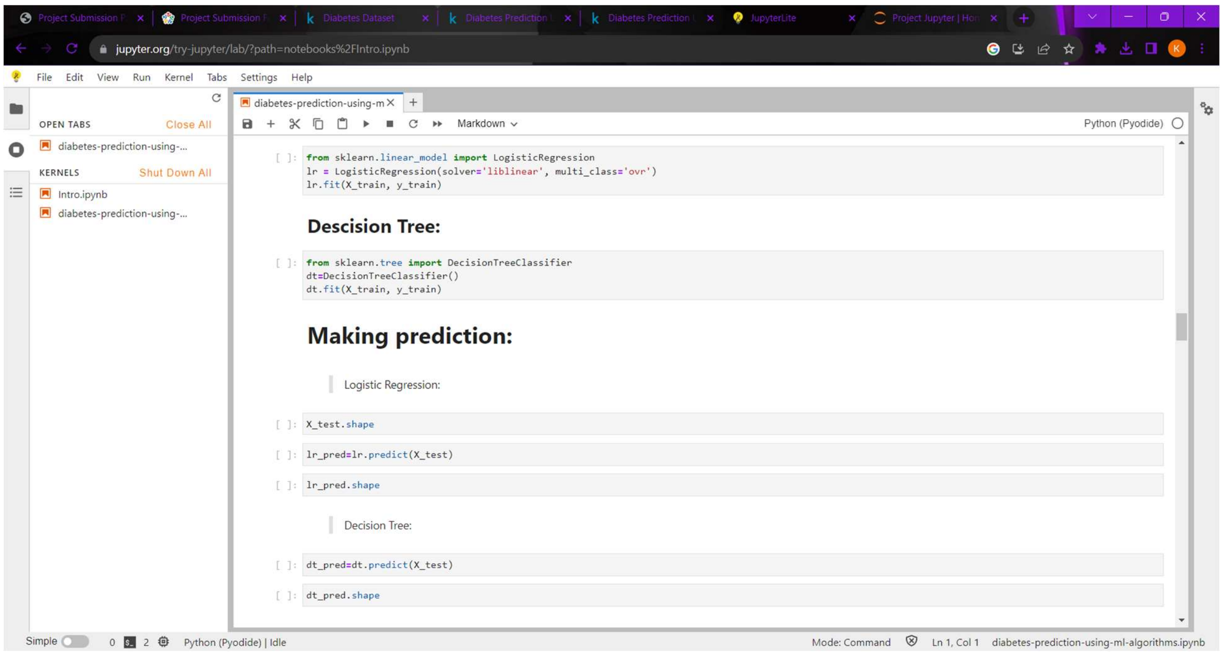
```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
import seaborn as sns                    # Importing the Seaborn library for statistical data visualization
import matplotlib.pyplot as plt          # Importing the Matplotlib library for creating plots and visualizations
import plotly.express as px              # Importing the Plotly Express library for interactive visualizations
```

Exploratory Data Analysis:

Load and Prepare Data:

```
df=pd.read_csv('/kaggle/input/diabetes-data-set/diabetes.csv')
```



```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='liblinear', multi_class='ovr')
lr.fit(X_train, y_train)
```

Descision Tree:

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(X_train, y_train)
```

Making prediction:

> Logistic Regression:

```
X_test.shape
```

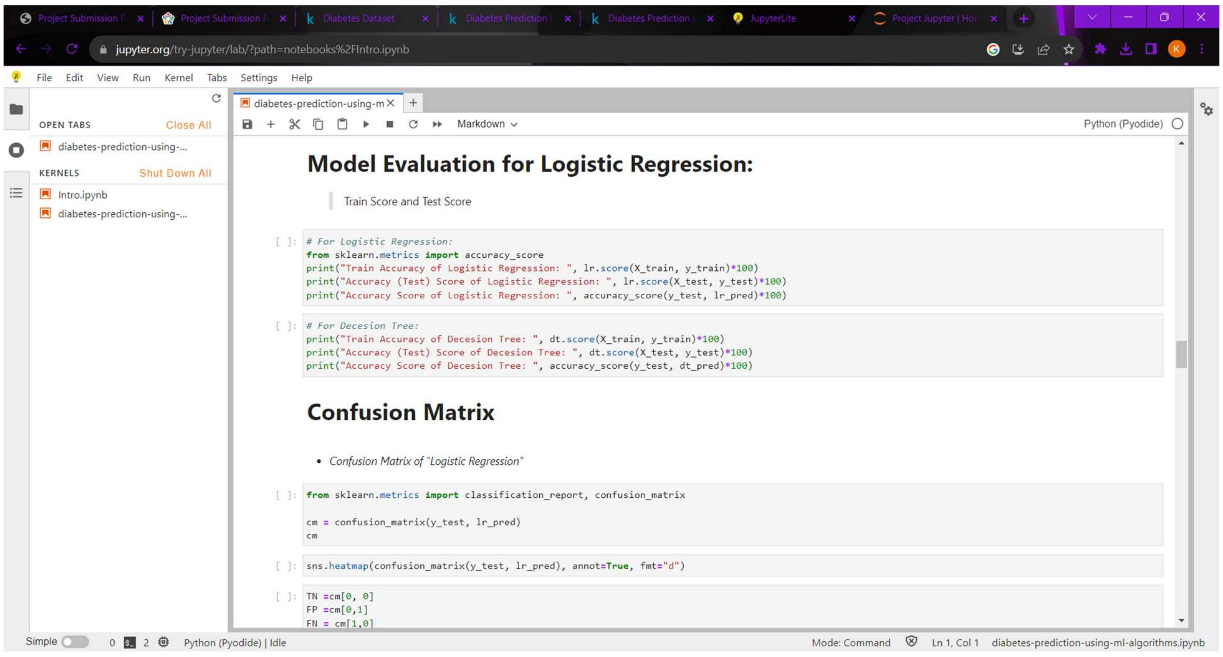
```
lr_pred=lr.predict(X_test)
```

```
lr_pred.shape
```

> Decision Tree:

```
dt_pred=dt.predict(X_test)
```

```
dt_pred.shape
```


Model Evaluation for Logistic Regression:

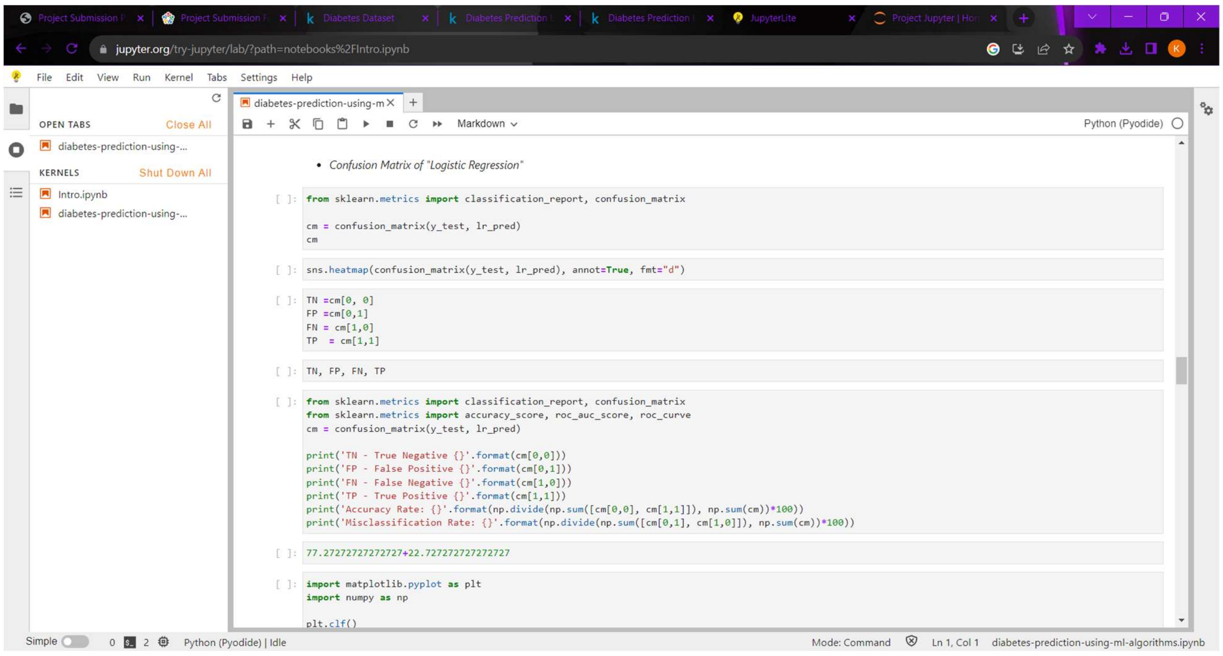
> Train Score and Test Score

```
[ ]: # For Logistic Regression:
     from sklearn.metrics import accuracy_score
     print("Train Accuracy of Logistic Regression: ", lr.score(X_train, y_train)*100)
     print("Accuracy (Test) Score of Logistic Regression: ", lr.score(X_test, y_test)*100)
     print("Accuracy Score of Logistic Regression: ", accuracy_score(y_test, lr_pred)*100)

[ ]: # For Decesion Tree:
     print("Train Accuracy of Decesion Tree: ", dt.score(X_train, y_train)*100)
     print("Accuracy (Test) Score of Decesion Tree: ", dt.score(X_test, y_test)*100)
     print("Accuracy Score of Decesion Tree: ", accuracy_score(y_test, dt_pred)*100)
```

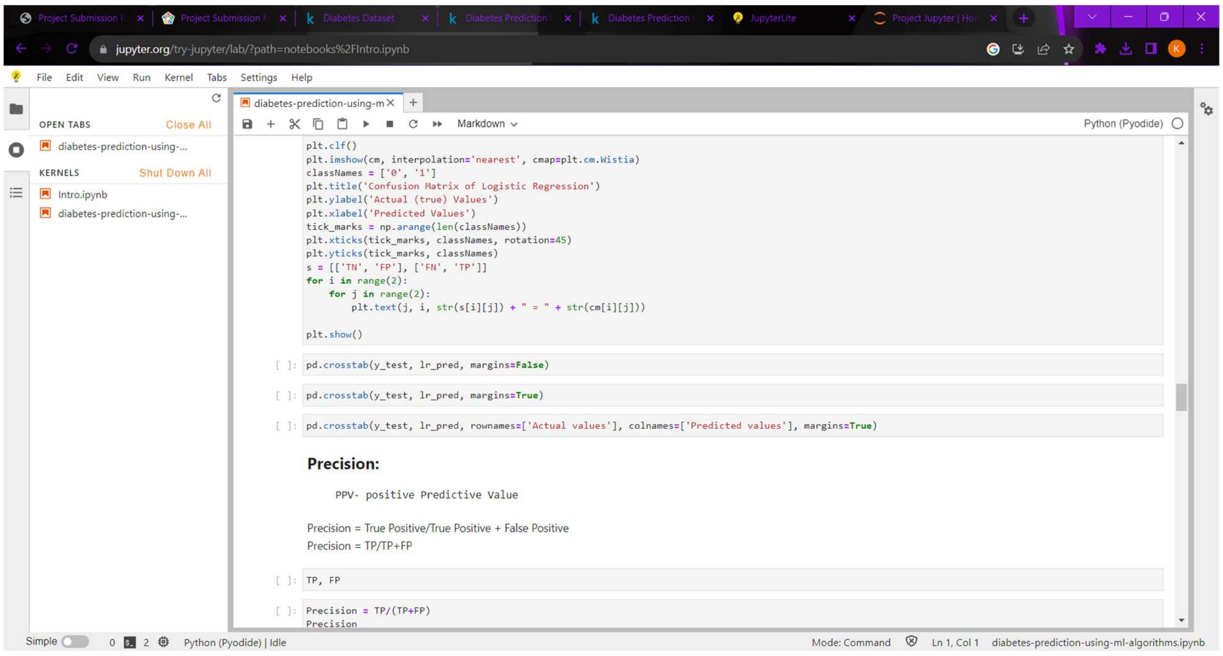
Confusion Matrix

- *Confusion Matrix of "Logistic Regression"*

```
[ ]: from sklearn.metrics import classification_report, confusion_matrix

     cm = confusion_matrix(y_test, lr_pred)
     cm

[ ]: sns.heatmap(confusion_matrix(y_test, lr_pred), annot=True, fmt="d")

[ ]: TN =cm[0, 0]
     FP =cm[0,1]
     FN = cm[1,0]
```


- *Confusion Matrix of "Logistic Regression"*

```
[ ]: from sklearn.metrics import classification_report, confusion_matrix

     cm = confusion_matrix(y_test, lr_pred)
     cm

[ ]: sns.heatmap(confusion_matrix(y_test, lr_pred), annot=True, fmt="d")

[ ]: TN =cm[0, 0]
     FP =cm[0,1]
     FN = cm[1,0]
     TP  = cm[1,1]

[ ]: TN, FP, FN, TP

[ ]: from sklearn.metrics import classification_report, confusion_matrix
     from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
     cm = confusion_matrix(y_test, lr_pred)

     print('TN - True Negative {}'.format(cm[0,0]))
     print('FP - False Positive {}'.format(cm[0,1]))
     print('FN - False Negative {}'.format(cm[1,0]))
     print('TP - True Positive {}'.format(cm[1,1]))
     print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0], cm[1,1]]), np.sum(cm))*100))
     print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1], cm[1,0]]), np.sum(cm))*100))

[ ]: 77.27272727272727+22.727272727272727

[ ]: import matplotlib.pyplot as plt
     import numpy as np

     plt.clf()
```



```
plt.clf()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Confusion Matrix of Logistic Regression')
plt.ylabel('Actual (true) Values')
plt.xlabel('Predicted Values')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN', 'FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j, i, str(s[i][j]) + " = " + str(cm[i][j]))

plt.show()
```

```
[ ]: pd.crosstab(y_test, lr_pred, margins=False)
```

```
[ ]: pd.crosstab(y_test, lr_pred, margins=True)
```

```
[ ]: pd.crosstab(y_test, lr_pred, rownames=['Actual values'], colnames=['Predicted values'], margins=True)
```

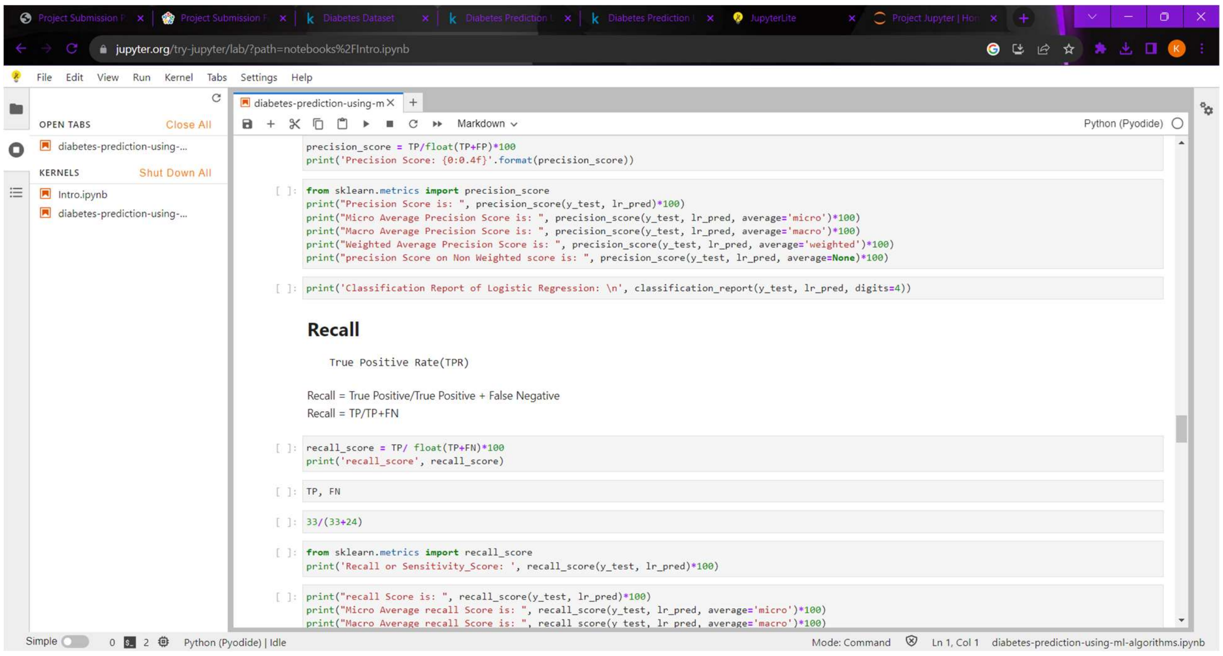
Precision:

```
    PPV- positive Predictive Value
```

Precision = True Positive/True Positive + False Positive
Precision = TP/TP+FP

```
[ ]: TP, FP
```

```
[ ]: Precision = TP/(TP+FP)
     Precision
```



```
precision_score = TP/float(TP+FP)*100
print('Precision Score: {0:0.4f}'.format(precision_score))
```

```
[ ]: from sklearn.metrics import precision_score
     print("Precision Score is: ", precision_score(y_test, lr_pred)*100)
     print("Micro Average Precision Score is: ", precision_score(y_test, lr_pred, average='micro')*100)
     print("Macro Average Precision Score is: ", precision_score(y_test, lr_pred, average='macro')*100)
     print("Weighted Average Precision Score is: ", precision_score(y_test, lr_pred, average='weighted')*100)
     print("precision Score on Non Weighted score is: ", precision_score(y_test, lr_pred, average=None)*100)
```

```
[ ]: print('Classification Report of Logistic Regression: \n', classification_report(y_test, lr_pred, digits=4))
```

Recall

```
    True Positive Rate(TPR)
```

Recall = True Positive/True Positive + False Negative
Recall = TP/TP+FN

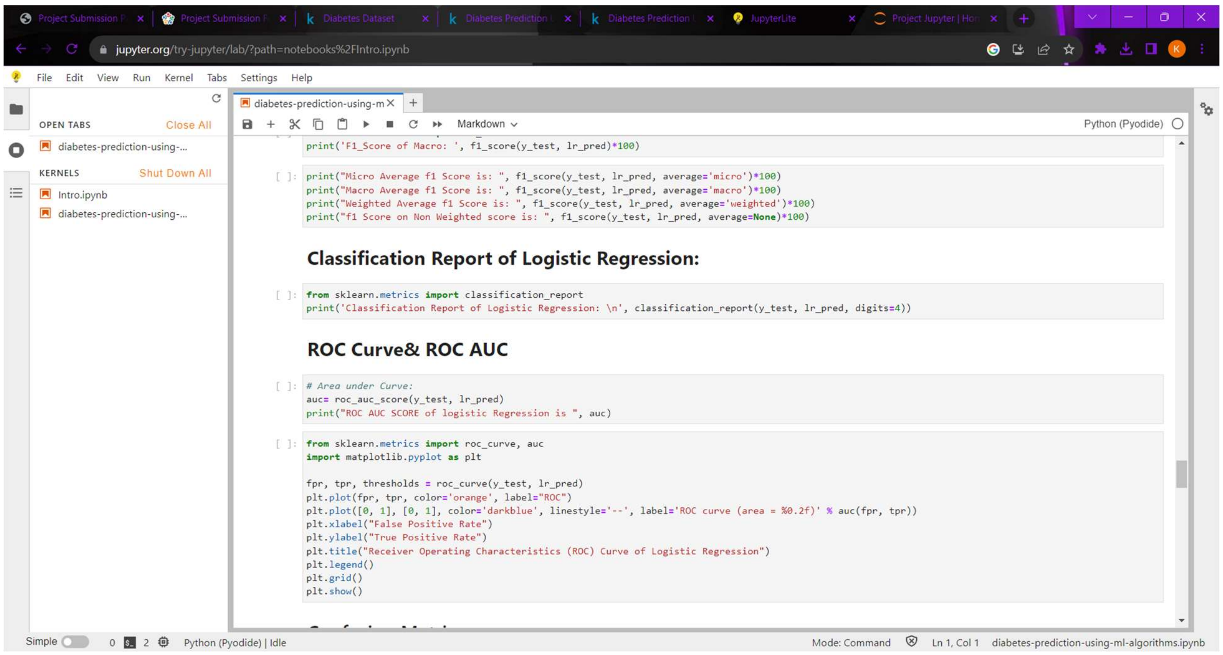
```
[ ]: recall_score = TP/ float(TP+FN)*100
     print('recall_score', recall_score)
```

```
[ ]: TP, FN
```

```
[ ]: 33/(33+24)
```

```
[ ]: from sklearn.metrics import recall_score
     print('Recall or Sensitivity_Score: ', recall_score(y_test, lr_pred)*100)
```

```
[ ]: print("recall Score is: ", recall_score(y_test, lr_pred)*100)
     print("Micro Average recall Score is: ", recall_score(y_test, lr_pred, average='micro')*100)
     print("Macro Average recall Score is: ", recall_score(y_test, lr_pred, average='macro')*100)
```



```
print('F1_Score of Macro: ', f1_score(y_test, lr_pred)*100)
```

```
[ ]: print("Micro Average f1 Score is: ", f1_score(y_test, lr_pred, average='micro')*100)
     print("Macro Average f1 Score is: ", f1_score(y_test, lr_pred, average='macro')*100)
     print("Weighted Average f1 Score is: ", f1_score(y_test, lr_pred, average='weighted')*100)
     print("f1 Score on Non Weighted score is: ", f1_score(y_test, lr_pred, average=None)*100)
```

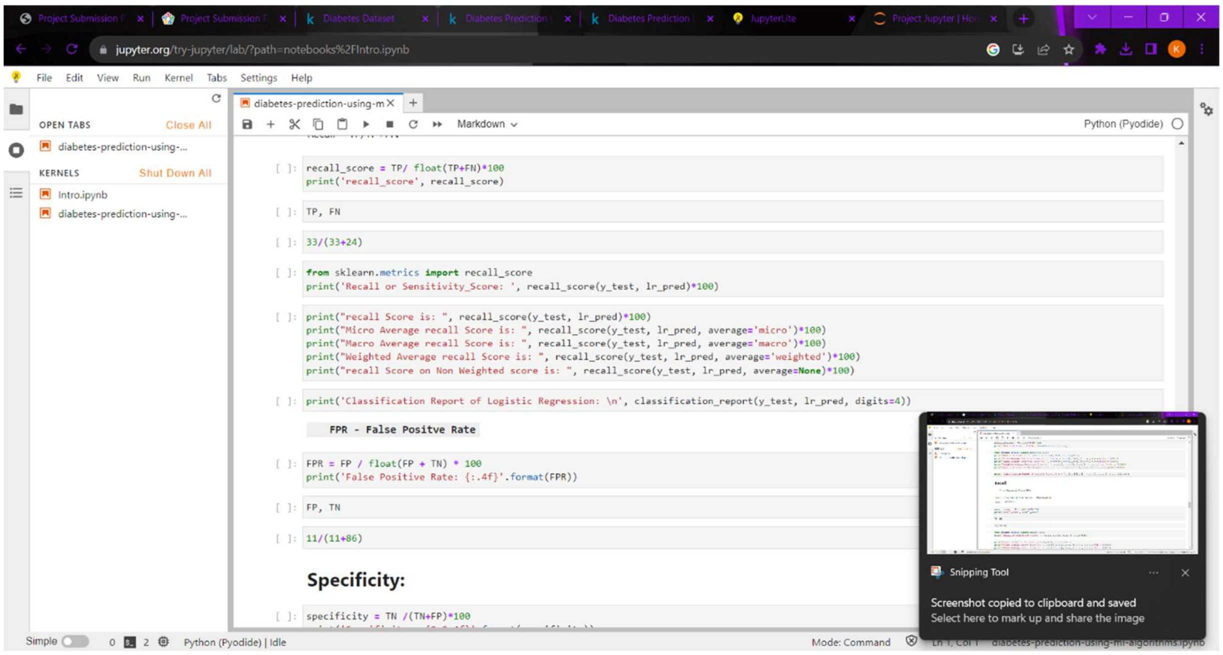
Classification Report of Logistic Regression:

```
[ ]: from sklearn.metrics import classification_report
     print('Classification Report of Logistic Regression: \n', classification_report(y_test, lr_pred, digits=4))
```

ROC Curve& ROC AUC

```
[ ]: # Area under Curve:
     auc= roc_auc_score(y_test, lr_pred)
     print("ROC AUC SCORE of logistic Regression is ", auc)
```

```
[ ]: from sklearn.metrics import roc_curve, auc
     import matplotlib.pyplot as plt

     fpr, tpr, thresholds = roc_curve(y_test, lr_pred)
     plt.plot(fpr, tpr, color='orange', label="ROC")
     plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='ROC curve (area = %0.2f)' % auc(fpr, tpr))
     plt.xlabel("False Positive Rate")
     plt.ylabel("True Positive Rate")
     plt.title("Receiver Operating Characteristics (ROC) Curve of Logistic Regression")
     plt.legend()
     plt.grid()
     plt.show()
```



```
[ ]: recall_score = TP/ float(TP+FN)*100
     print('recall_score', recall_score)
```

```
[ ]: TP, FN
```

```
[ ]: 33/(33+24)
```

```
[ ]: from sklearn.metrics import recall_score
     print('Recall or Sensitivity_Score: ', recall_score(y_test, lr_pred)*100)
```

```
[ ]: print("recall Score is: ", recall_score(y_test, lr_pred)*100)
     print("Micro Average recall Score is: ", recall_score(y_test, lr_pred, average='micro')*100)
     print("Macro Average recall Score is: ", recall_score(y_test, lr_pred, average='macro')*100)
     print("Weighted Average recall Score is: ", recall_score(y_test, lr_pred, average='weighted')*100)
     print("recall Score on Non Weighted score is: ", recall_score(y_test, lr_pred, average=None)*100)
```

```
[ ]: print('Classification Report of Logistic Regression: \n', classification_report(y_test, lr_pred, digits=4))
```

```
    FPR - False Positve Rate
```

```
[ ]: FPR = FP / float(FP + TN) * 100
     print('False Positive Rate: {:.4f}'.format(FPR))
```

```
[ ]: FP, TN
```

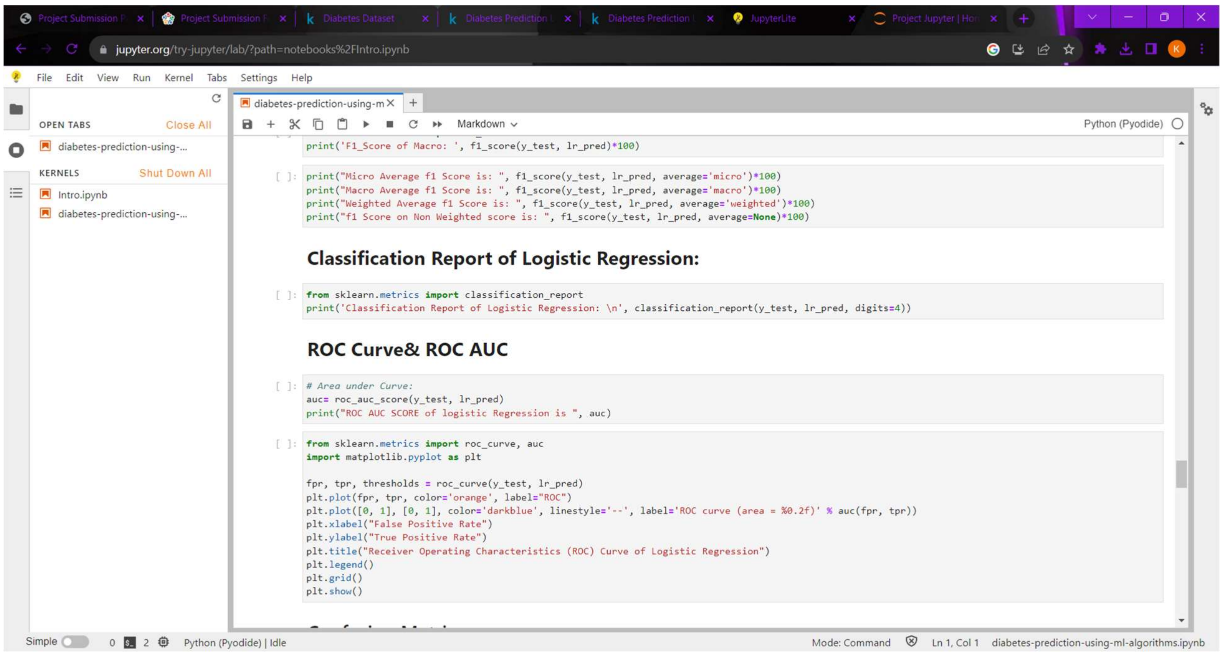
```
[ ]: 11/(11+86)
```

Specificity:

```
[ ]: specificity = TN /(TN+FP)*100
```

Snipping Tool

Screenshot copied to clipboard and saved
Select here to mark up and share the image



```
print('F1_Score of Macro: ', f1_score(y_test, lr_pred)*100)
```

```
[ ]: print("Micro Average f1 Score is: ", f1_score(y_test, lr_pred, average='micro')*100)
     print("Macro Average f1 Score is: ", f1_score(y_test, lr_pred, average='macro')*100)
     print("Weighted Average f1 Score is: ", f1_score(y_test, lr_pred, average='weighted')*100)
     print("f1 Score on Non Weighted score is: ", f1_score(y_test, lr_pred, average=None)*100)
```

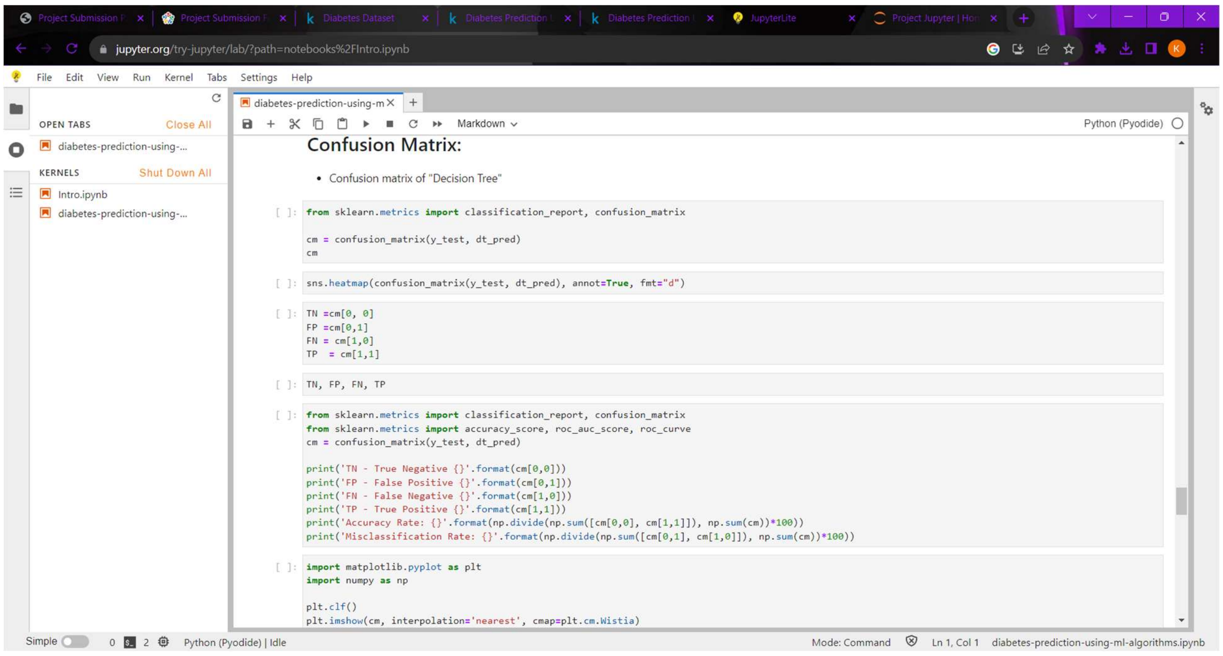
Classification Report of Logistic Regression:

```
[ ]: from sklearn.metrics import classification_report
     print('Classification Report of Logistic Regression: \n', classification_report(y_test, lr_pred, digits=4))
```

ROC Curve& ROC AUC

```
[ ]: # Area under Curve:
     auc= roc_auc_score(y_test, lr_pred)
     print("ROC AUC SCORE of logistic Regression is ", auc)
```

```
[ ]: from sklearn.metrics import roc_curve, auc
     import matplotlib.pyplot as plt

     fpr, tpr, thresholds = roc_curve(y_test, lr_pred)
     plt.plot(fpr, tpr, color='orange', label="ROC")
     plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='ROC curve (area = %0.2f)' % auc(fpr, tpr))
     plt.xlabel("False Positive Rate")
     plt.ylabel("True Positive Rate")
     plt.title("Receiver Operating Characteristics (ROC) Curve of Logistic Regression")
     plt.legend()
     plt.grid()
     plt.show()
```



Confusion Matrix:

- Confusion matrix of "Decision Tree"

```
[ ]: from sklearn.metrics import classification_report, confusion_matrix

     cm = confusion_matrix(y_test, dt_pred)
     cm
```

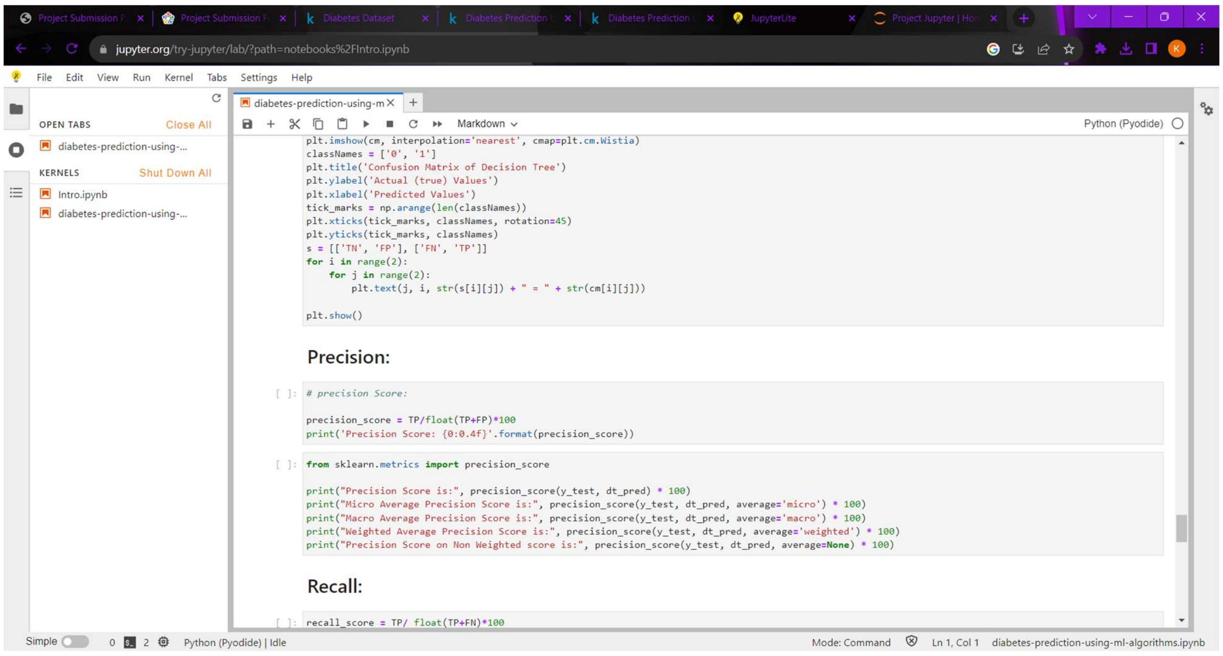
```
[ ]: sns.heatmap(confusion_matrix(y_test, dt_pred), annot=True, fmt="d")
```

```
[ ]: TN =cm[0, 0]
     FP =cm[0,1]
     FN = cm[1,0]
     TP  = cm[1,1]
```

```
[ ]: TN, FP, FN, TP
```

```
[ ]: from sklearn.metrics import classification_report, confusion_matrix
     from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
     cm = confusion_matrix(y_test, dt_pred)

     print('TN - True Negative {}'.format(cm[0,0]))
     print('FP - False Positive {}'.format(cm[0,1]))
     print('FN - False Negative {}'.format(cm[1,0]))
     print('TP - True Positive {}'.format(cm[1,1]))
     print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0], cm[1,1]]), np.sum(cm))*100))
     print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1], cm[1,0]]), np.sum(cm))*100))
```

```
[ ]: import matplotlib.pyplot as plt
     import numpy as np

     plt.clf()
     plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
```



```
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Confusion Matrix of Decision Tree')
plt.ylabel('Actual (true) Values')
plt.xlabel('Predicted Values')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN', 'FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j, i, str(s[i][j]) + " = " + str(cm[i][j]))

plt.show()
```

Precision:

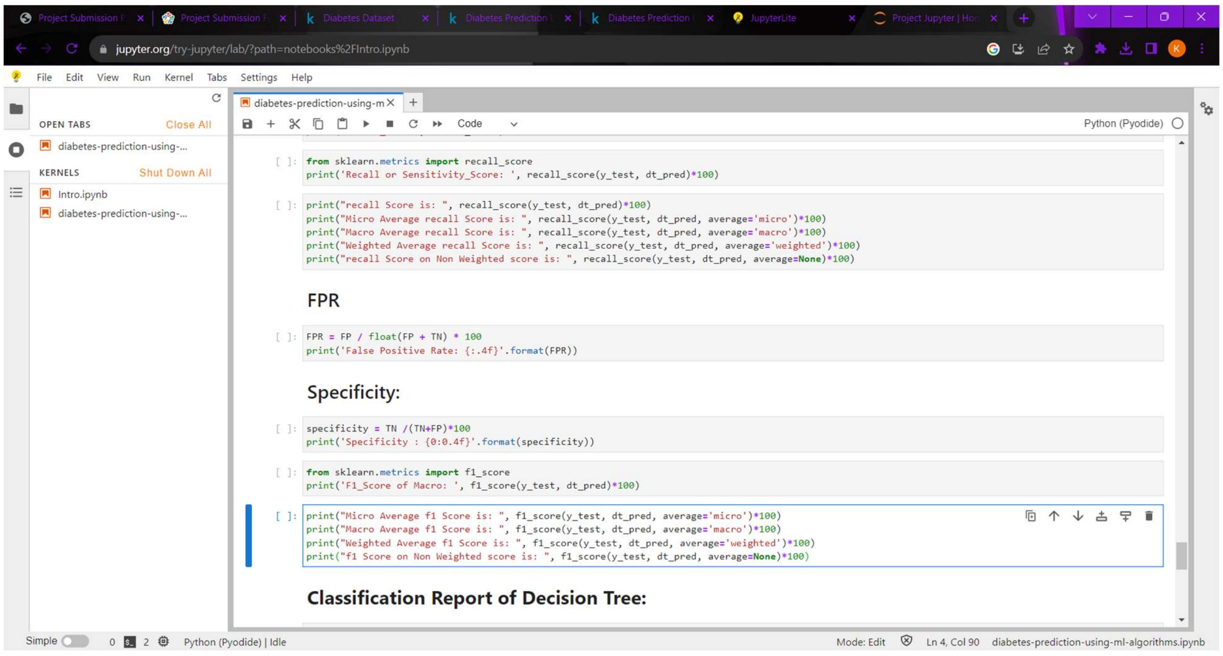
```
[ ]: # precision Score:

     precision_score = TP/float(TP+FP)*100
     print('Precision Score: {0:0.4f}'.format(precision_score))
```

```
[ ]: from sklearn.metrics import precision_score

     print("Precision Score is:", precision_score(y_test, dt_pred) * 100)
     print("Micro Average Precision Score is:", precision_score(y_test, dt_pred, average='micro') * 100)
     print("Macro Average Precision Score is:", precision_score(y_test, dt_pred, average='macro') * 100)
     print("Weighted Average Precision Score is:", precision_score(y_test, dt_pred, average='weighted') * 100)
     print("Precision Score on Non Weighted score is:", precision_score(y_test, dt_pred, average=None) * 100)
```

Recall:

```
[ ]: recall_score = TP/ float(TP+FN)*100
```

Python (Pyodide) | Idle   Mode: Command   Ln 1, Col 1   diabetes-prediction-using-ml-algorithms.ipynb



```
[ ]: from sklearn.metrics import recall_score
     print('Recall or Sensitivity_Score: ', recall_score(y_test, dt_pred)*100)
```

```
[ ]: print("recall Score is: ", recall_score(y_test, dt_pred)*100)
     print("Micro Average recall Score is: ", recall_score(y_test, dt_pred, average='micro')*100)
     print("Macro Average recall Score is: ", recall_score(y_test, dt_pred, average='macro')*100)
     print("Weighted Average recall Score is: ", recall_score(y_test, dt_pred, average='weighted')*100)
     print("recall Score on Non Weighted score is: ", recall_score(y_test, dt_pred, average=None)*100)
```

FPR

```
[ ]: FPR = FP / float(FP + TN) * 100
     print('False Positive Rate: {:.4f}'.format(FPR))
```

Specificity:

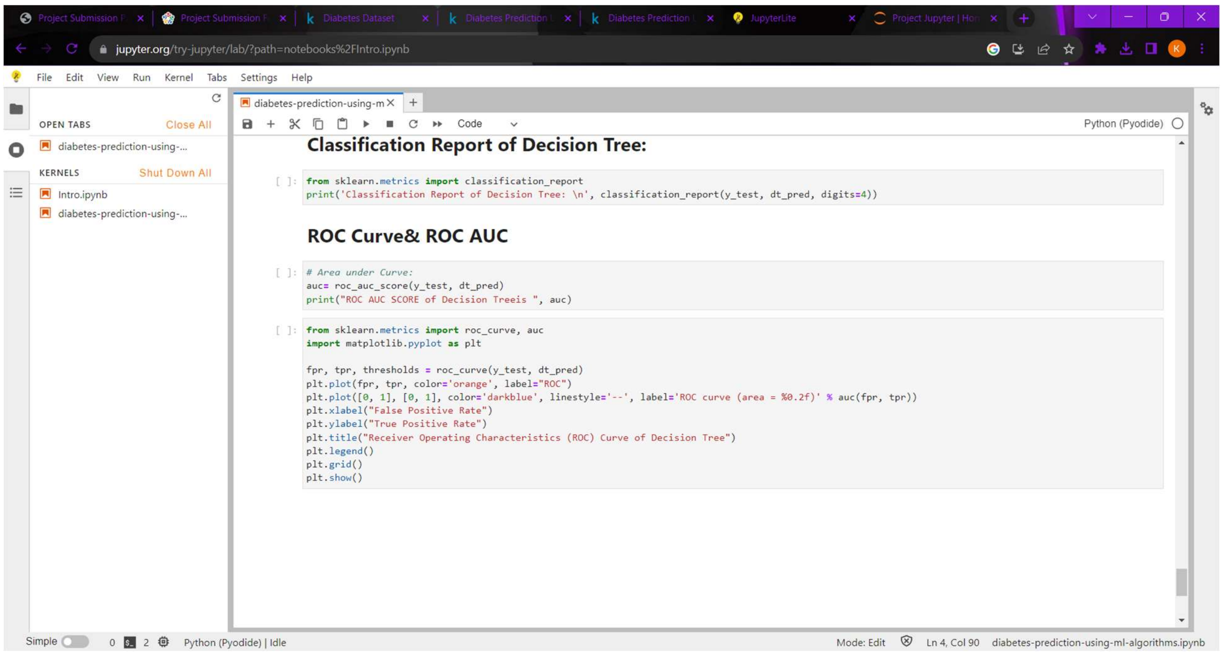
```
[ ]: specificity = TN /(TN+FP)*100
     print('Specificity : {0:0.4f}'.format(specificity))
```

```
[ ]: from sklearn.metrics import f1_score
     print('F1_Score of Macro: ', f1_score(y_test, dt_pred)*100)
```

```
[ ]: print("Micro Average f1 Score is: ", f1_score(y_test, dt_pred, average='micro')*100)
     print("Macro Average f1 Score is: ", f1_score(y_test, dt_pred, average='macro')*100)
     print("Weighted Average f1 Score is: ", f1_score(y_test, dt_pred, average='weighted')*100)
     print("f1 Score on Non Weighted score is: ", f1_score(y_test, dt_pred, average=None)*100)
```

Classification Report of Decision Tree:

Python (Pyodide) | Idle   Mode: Edit   Ln 4, Col 90   diabetes-prediction-using-ml-algorithms.ipynb



Classification Report of Decision Tree:

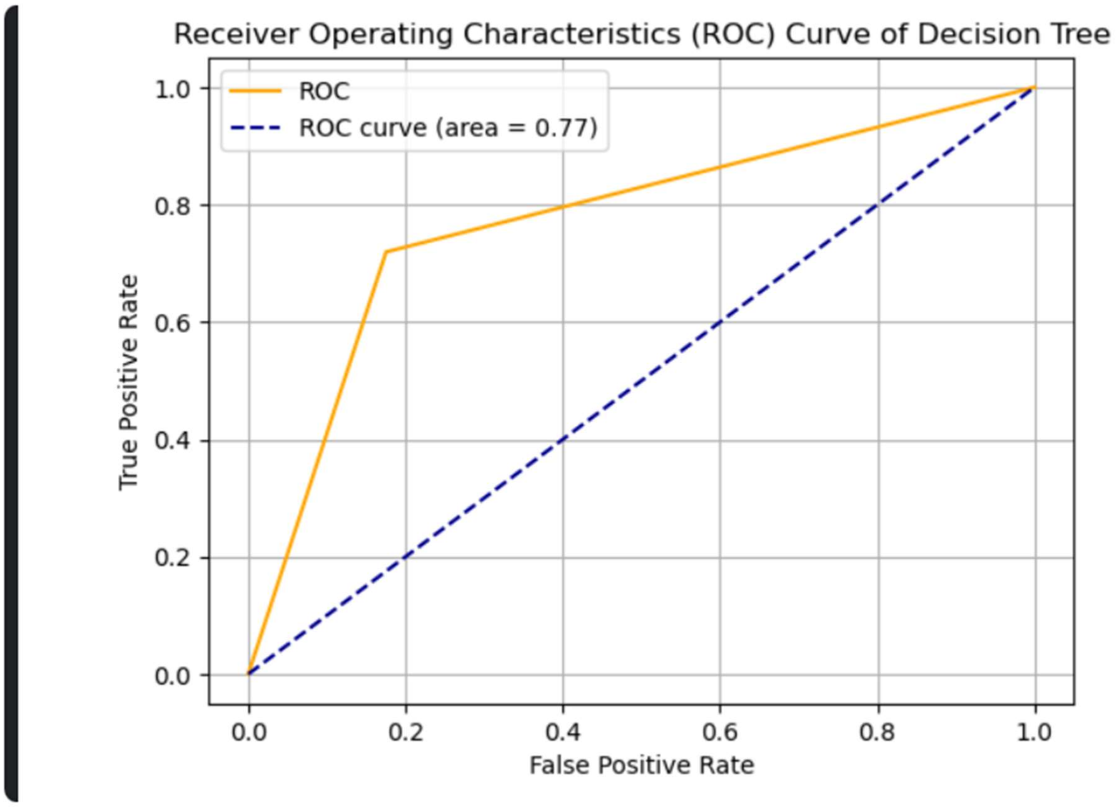
```
[ ]: from sklearn.metrics import classification_report
     print('Classification Report of Decision Tree: \n', classification_report(y_test, dt_pred, digits=4))
```

ROC Curve& ROC AUC

```
[ ]: # Area under Curve:
     auc= roc_auc_score(y_test, dt_pred)
     print("ROC AUC SCORE of Decision Treeis ", auc)
```

```
[ ]: from sklearn.metrics import roc_curve, auc
     import matplotlib.pyplot as plt

     fpr, tpr, thresholds = roc_curve(y_test, dt_pred)
     plt.plot(fpr, tpr, color='orange', label="ROC")
     plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='ROC curve (area = %0.2f)' % auc(fpr, tpr))
     plt.xlabel("False Positive Rate")
     plt.ylabel("True Positive Rate")
     plt.title("Receiver Operating Characteristics (ROC) Curve of Decision Tree")
     plt.legend()
     plt.grid()
     plt.show()
```

ACCURACY WITH GRAPH PLOTTED:



```
Precision Score: 70.6897
```

```
Micro Average Precision Score is: 78.57142857142857
Macro Average Precision Score is: 77.01149425287358
Weighted Average Precision Score is: 78.65353037766832
```

CONCLUSION:

We have typed and executed the code using jupyterlite notebook run time of 76.8s by using the machine learning algorithm.