AI BASED DIABETES PREDICTION SYSTEM USING AI TECHNOLOGIES

PHASE – 5 FINAL SUBMISSION

TEAM MEMBERS :

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

INTRODUCTION:

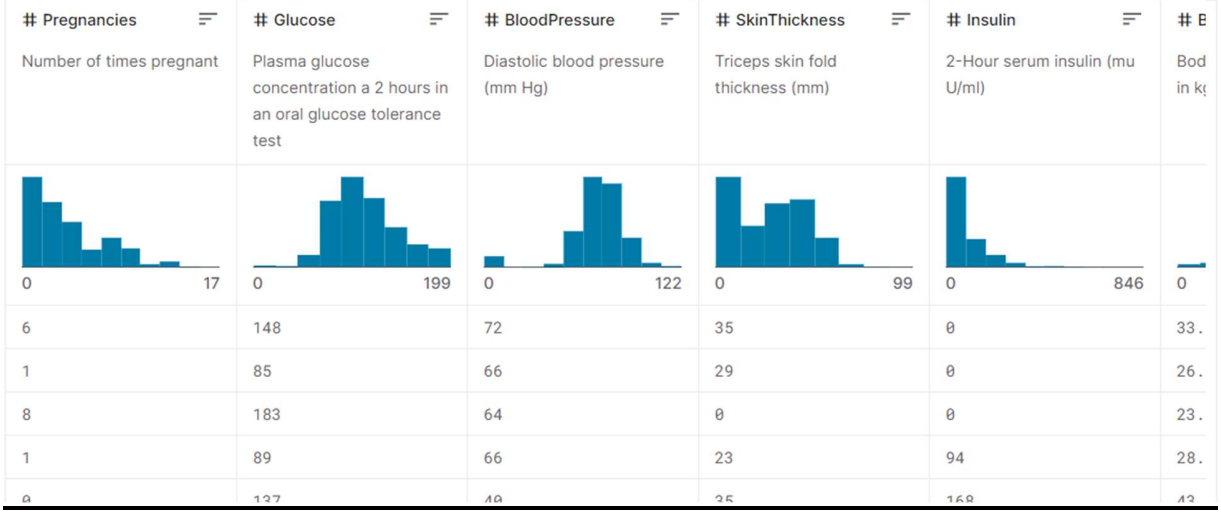
Artificial Intelligence (AI) has revolutionized healthcare by offering innovative solutions to predict and manage various medical conditions, and one area where AI has made a significant impact is diabetes prediction. Diabetes is a chronic and widespread metabolic disorder that affects millions of people worldwide. AI-based diabetes prediction leverages advanced algorithms, machine learning techniques, and big data analysis to assist in the early detection and management of diabetes. By analyzing a range of patient data, including medical history, genetic information, lifestyle factors, and biomarkers, AI can generate accurate predictions about an individual's risk of developing diabetes.

These predictive models not only help in identifying high-risk individuals but also offer personalized recommendations for lifestyle modifications and preventive measures, ultimately reducing the burden of diabetes-related complications and healthcare costs. AI-based diabetes prediction systems are a prime example of how technology can empower healthcare professionals and individuals to proactively address a global health challenge, fostering a more proactive and personalized approach to diabetes management.

GIVEN DATASET LINK:

https://www.kaggle.com/datasets/mathchi/diabetes-data-set

- Insulin: 2-Hour serum insulin (mu U/ml)
- BMI: Body mass index (weight in kg/(height in m)^2)
- DiabetesPedigreeFunction: Diabetes pedigree function
- Age: Age (years)
- Outcome: Class variable (0 or 1)



# Pregnancies	# Glucose	# BloodPressure	# SkinThickness	# Insulin	# B
Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)	2-Hour serum insulin (mu U/ml)	Bod in k
0 17	0 199	0 122	0 99	0 846	0
6	148	72	35	0	33.
1	85	66	29	0	26.
8	183	64	0	0	23.
1	89	66	23	94	28.
0	137	40	35	168	43

HERE THE LIST OF TOOLS THAT ARE COMMONLY USED IN THE PROGRESS:

The process of AI-based diabetes prediction involves several steps, and there are various tools commonly used at each stage to facilitate the analysis and prediction. Here is a step-by-step list of these tools:

1. Data Collection and Preprocessing:

Python:

Python is a popular programming language for data analysis and machine learning.

Pandas:

A Python library for data manipulation and analysis.

NumPy:

Used for numerical operations on data arrays.

SciKit-Learn:

Provides tools for machine learning, including preprocessing functions.

SQL or NoSQL Databases:

For storing and managing large datasets.

2. Feature Engineering:

Jupyter Notebook:

An interactive environment for data exploration and feature engineering.

Feature selection libraries:

Like scikit-learn's `SelectKBest` or `SelectFromModel` for selecting relevant features.

3. Model Development:

Scikit-Learn:

Offers various machine learning algorithms for classification, regression, and model evaluation.

TensorFlow and PyTorch:

For building deep learning models.

AutoML Tools:

Such as Google AutoML or H2O.ai, which automate the model selection and tuning process.

4. Model Evaluation:

Scikit-Learn:

For metrics like accuracy, precision, recall, F1-score, and ROC curves.

Confusion Matrix Visualization Tools:

Like Seaborn and Matplotlib.

Cross-Validation Libraries:

For estimating model performance more robustly.

5. Deployment and Integration:

Flask or Django:

Python web frameworks for building a web-based interface for the model.

Docker:

For containerization of the AI model.

Cloud Services:

Such as AWS, Azure, or Google Cloud for scalable deployment.

6. Data Monitoring and Maintenance:

Monitoring Tools:

To keep an eye on model performance and data quality.

Scheduled Jobs:

For periodic model updates and retraining.

Database Management Systems:

To store and manage prediction results and user data.

7. User Interface (Optional):

HTML/CSS/JavaScript:

For creating user-friendly web interfaces.

React, Angular, or Vue.js: Frameworks for building interactive front-end applications.

8. Privacy and Security (Critical for healthcare applications):

HIPAA-Compliant Tools and Practices:

Ensuring compliance with healthcare data security regulations.

Data Encryption Tools:

To secure data at rest and in transit.

Access Control and Authentication Systems:

To control who can access patient data.

Remember that the choice of tools may vary based on the specific requirements and constraints of the project. Additionally, it's important to ensure that the handling of medical data is in compliance with relevant healthcare regulations and standards to protect patient privacy and data security.

1.DESIGN THINKING AND PRESENT IN FORM OF DOCUMENT

Design thinking is a user-centered, iterative approach to problem-solving and innovation. To present a design thinking document for an AI-based diabetes prediction system, you can use a structured format that includes the key stages of design thinking: empathize, define, ideate, prototype, and test. Here's an example document outlining these stages:

1. Empathize

Goal:

Understand the needs and concerns of individuals at risk of diabetes, healthcare providers, and stakeholders.

Activities:

- Conduct user interviews and surveys to gather insights.

- Analyze existing healthcare data and statistics.

- Engage with healthcare professionals for expert opinions.

Key Insights:

- Many individuals lack awareness of their diabetes risk.

- Healthcare providers need accurate and accessible predictive tools.

- Data privacy and security are significant concerns.

2. Define

Goal:

Clearly define the problem and establish the project's scope.

Activities:

- Create personas for end-users.

- Define the project's goals, objectives, and constraints.

- Develop a problem statement.

Problem Statement:

"Individuals at risk of diabetes lack a user-friendly and accurate tool for early prediction and personalized risk management, and healthcare providers need an efficient system to assist in diabetes prevention."

3. Ideate

Goal:

Generate creative solutions to the defined problem.

Activities:

- Conduct brainstorming sessions with the project team.

- Seek inspiration from AI and healthcare experts.

- Encourage "out-of-the-box" thinking.

Ideas:

- Develop a mobile app for users to assess their diabetes risk.

- Utilize AI to create personalized diabetes risk profiles.

- Implement a web-based platform for healthcare providers to access patient data.

4. Prototype

Goal:

Create tangible representations of the proposed solutions.

Activities:

- Develop wireframes for the mobile app.

- Design mockups of the AI prediction interface.

- Create a prototype of the web-based platform.

5. Test

Goal:

Gather feedback and refine the prototypes.

Activities:

- Conduct usability tests with potential users.

- Collect feedback from healthcare professionals.

- Analyze the usability and accuracy of the AI prediction model.

Feedback:

- Users find the mobile app easy to use but want more educational content.

- Healthcare providers value the platform's data visualization but suggest real-time data integration.

6. Refine and Iterate

Goal:

Use the feedback to make necessary improvements to the prototypes.

Activities:

- Refine the mobile app with additional educational content.

- Enhance the web-based platform for real-time data integration.

- Fine-tune the AI prediction model for increased accuracy.

7. Final Solution

Goal:

Present the final AI-based diabetes prediction solution.

Components:

- Mobile app for users to assess diabetes risk.

- AI model for personalized risk prediction.

- Web-based platform for healthcare providers.

8. Implementation Plan

Goal:

Outline the plan for implementing the solution.

Activities:

- Development of the mobile app, AI model, and web platform.
- Data integration and privacy measures.
- User training and onboarding.

9. Evaluation and Continuous Improvement

Goal:

Establish a system for ongoing evaluation and improvement.

Activities:

- Regularly gather user feedback.
- Monitor the performance and accuracy of the AI model.
- Address data privacy and security concerns.

By following this design thinking process, we have created a user-centric AI-based diabetes prediction system that not only addresses the needs of individuals and healthcare providers but also prioritizes data privacy and security. This approach ensures that the solution is effective, user-friendly, and adaptable for future improvements.

2:DESIGN AND INNOVATION:

Step-by-step guide to designing an innovation for AI-based diabetes prediction:

1. Identify a problem to solve. What are the limitations of current AI-based diabetes prediction systems? What are the unmet needs of patients and caregivers?

2. Brainstorm possible solutions. How can AI be used to improve diabetes prediction in new and innovative ways? Consider different approaches, such as

using new data sources, developing new machine learning algorithms, or integrating AI with other technologies.

3. Evaluate potential solutions. Assess the feasibility, cost-effectiveness, and potential impact of each solution.

4. Develop a prototype. Create a working prototype of your solution to test its feasibility and gather feedback from users.

5. Test and refine the prototype. Conduct user testing and gather feedback to iterate on and improve your prototype.

6. Deploy the solution. Once the prototype is refined and tested, deploy it to a wider audience.

Here are some specific examples of innovations that could be developed for AI-based diabetes prediction:

- Develop a more accurate and personalized risk assessment tool. Current AI-based diabetes prediction models are still relatively inaccurate, and they do not take into account all of the individual risk factors for diabetes. Developing a more accurate and personalized risk assessment tool would help to identify individuals at high risk of developing diabetes earlier, so that they can take preventive measures.

- Develop a system that can predict the onset of diabetes complications. Current AI-based diabetes prediction models can only predict the risk of developing diabetes itself. Developing a system that can predict the onset of diabetes complications would help patients and caregivers to better manage their disease and reduce their risk of serious complications.

- Develop a system that can integrate AI-based diabetes prediction with other technologies, such as wearable devices or electronic health records. This would allow for more continuous and personalized monitoring of a patient's risk of diabetes and diabetes complications.

Overall, the goal of designing an innovation for AI-based diabetes prediction is to improve the accuracy, personalization, and accessibility of diabetes prediction. This can be done by developing new machine learning algorithms, using new data sources, and integrating AI with other technologies.
Here are some additional considerations for designing an AI-based diabetes prediction system:

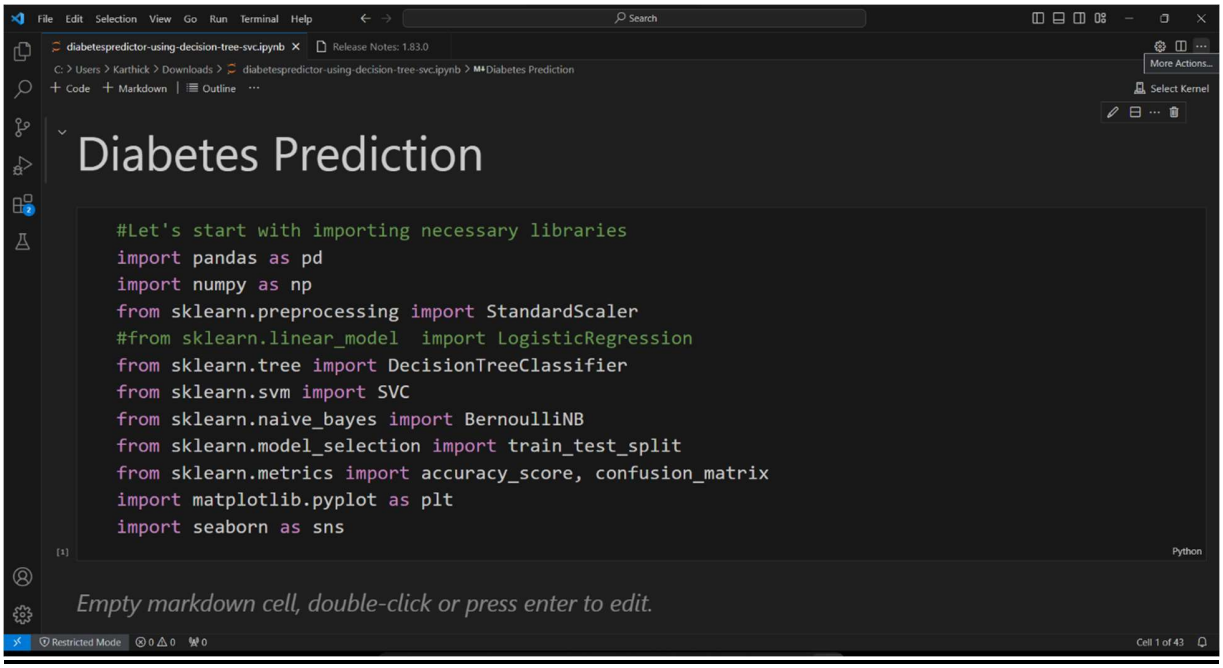
- Data privacy and security: AI-based diabetes prediction systems collect and process sensitive patient data. It is important to ensure that this data is protected from unauthorized access and use.

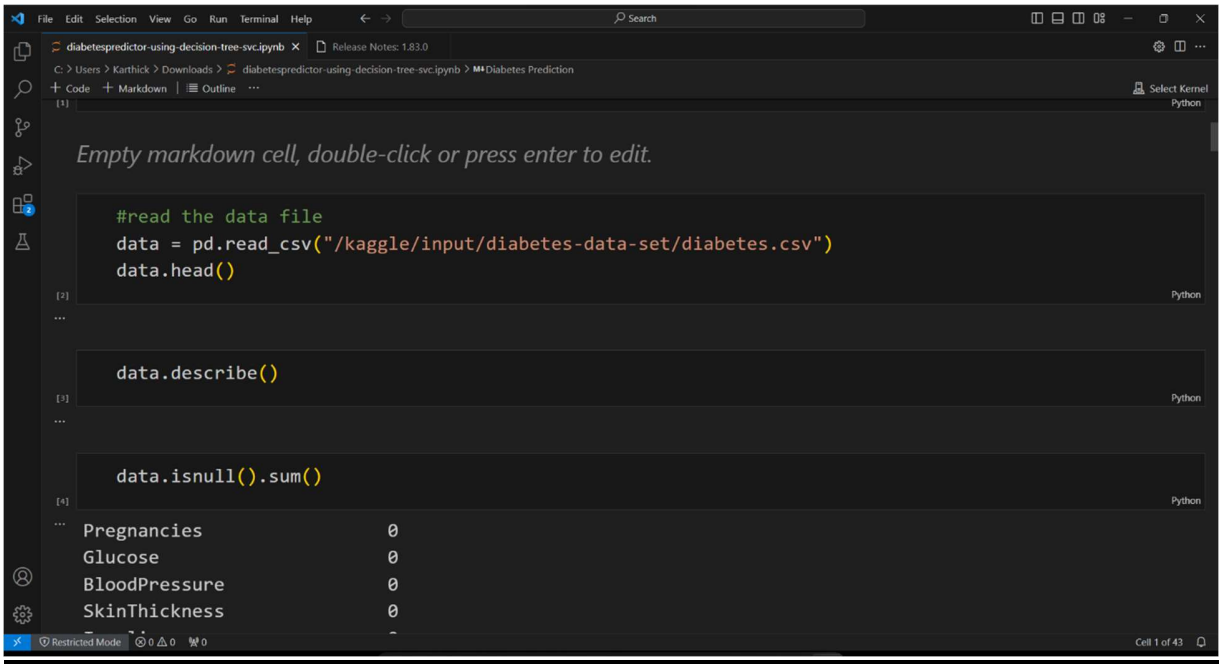
- Transparency and accountability: AI-based diabetes prediction systems should be transparent about how they work and how they make decisions. This is important for building trust with patients and caregivers.

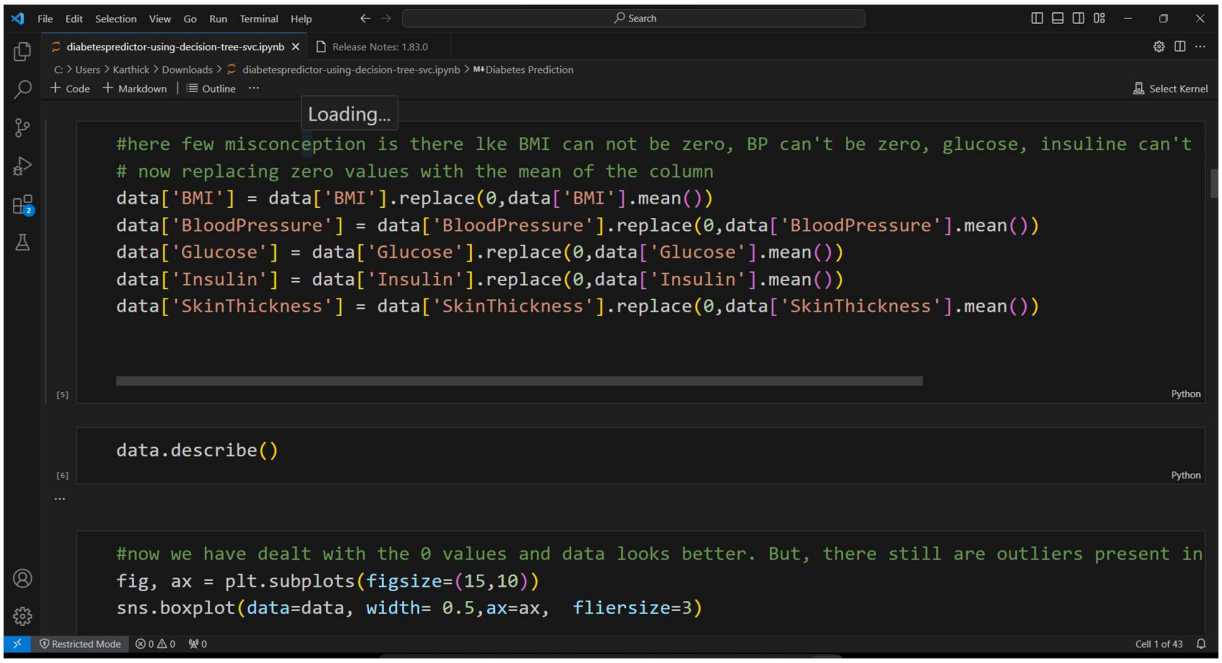
- Equity and inclusion: AI-based diabetes prediction systems should be designed and evaluated in a way that ensures that they are fair and equitable for all patients, regardless of race, ethnicity, socioeconomic status, or other factors.

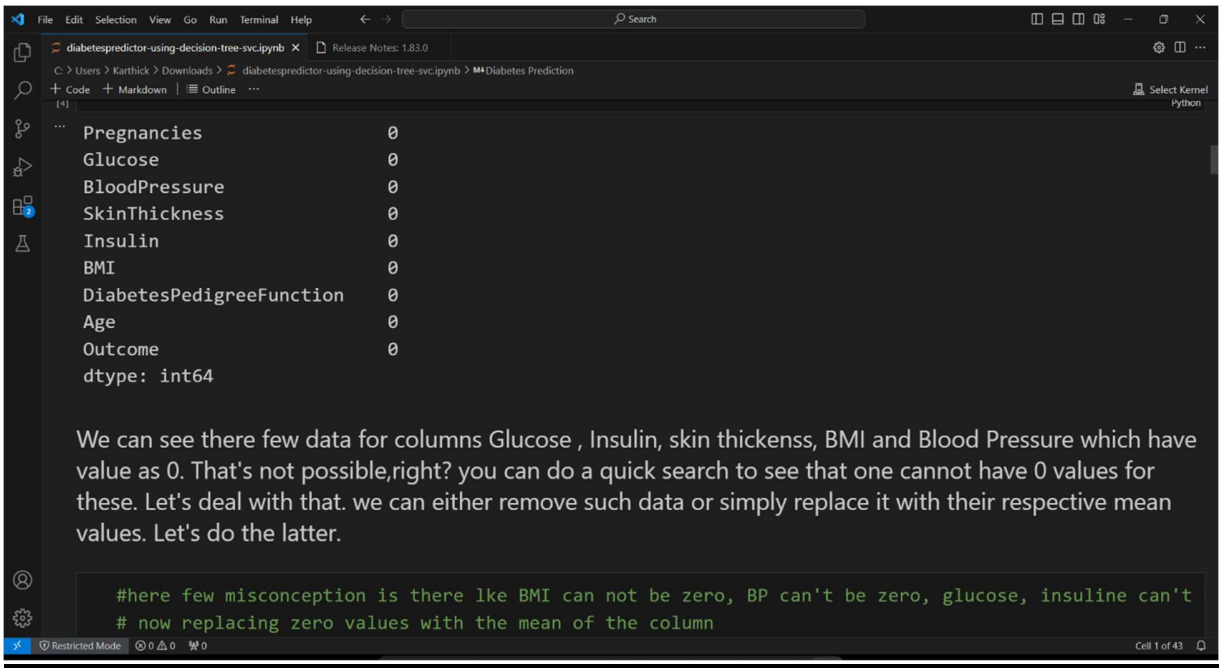
By carefully considering these factors, it is possible to design AI-based diabetes prediction systems that are safe, effective, and beneficial for patients and caregivers.

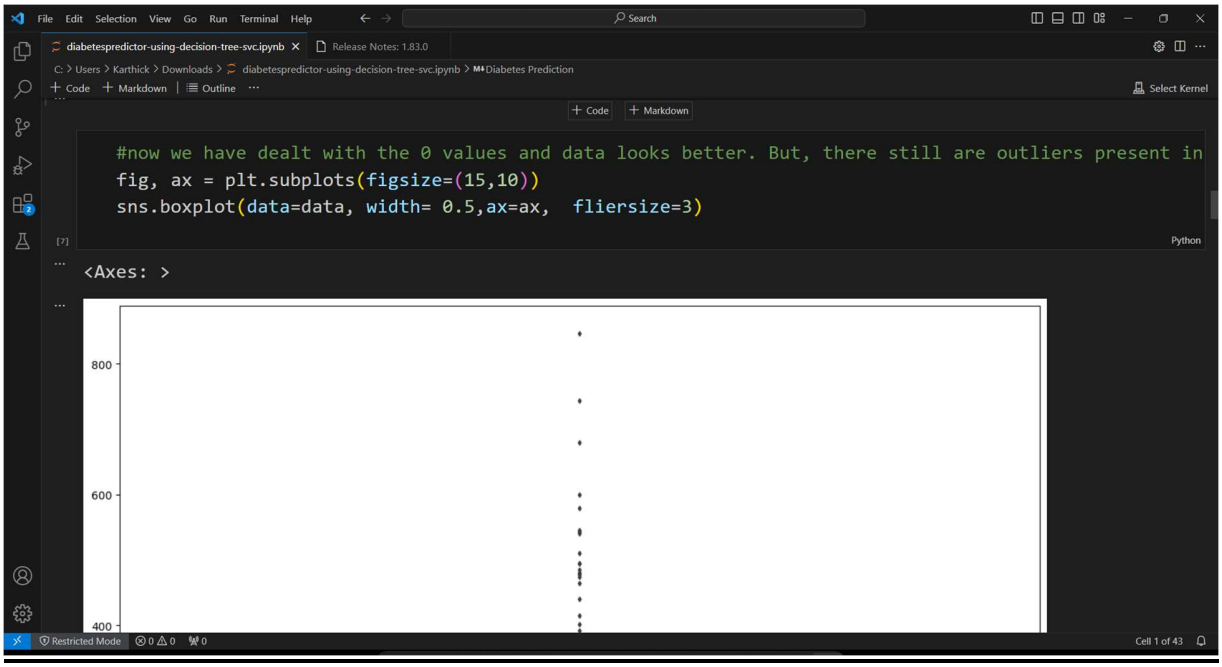
DIABETES PREDICTION USING DECISION TREE TECHNIQUE:

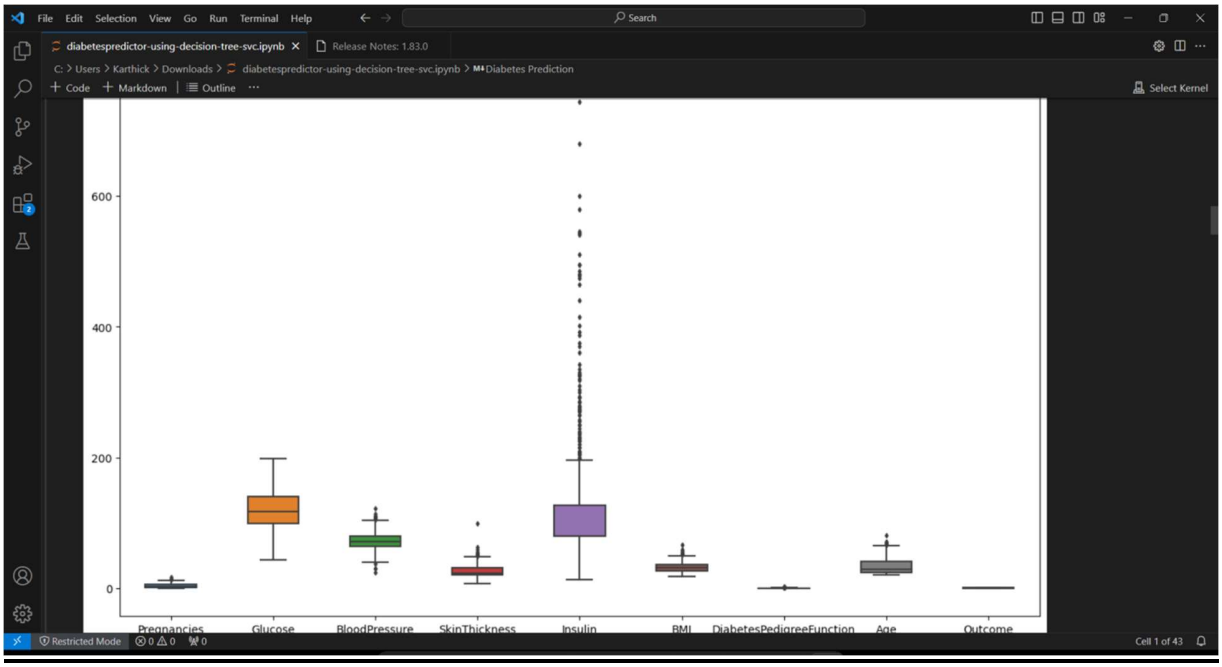

File Edit Selection View Go Run Terminal Help
Search
diabetespredictor-using-decision-tree-svc.ipynb
Release Notes: 1.83.0
C: > Users > Karthick > Downloads > diabetespredictor-using-decision-tree-svc.ipynb > Diabetes Prediction
Code Markdown Outline
More Actions...
Select Kernel
Diabetes Prediction
#Let's start with importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
#from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
[1]
Python
Empty markdown cell, double-click or press enter to edit.
Restricted Mode
Cell 1 of 43

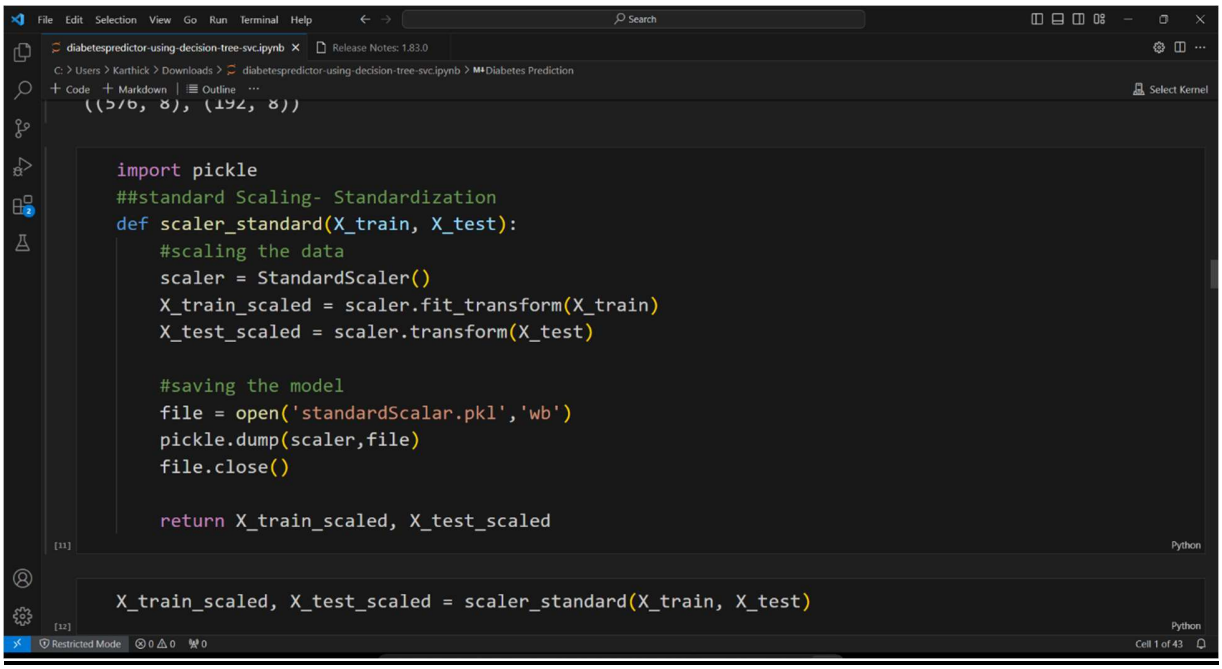

File Edit Selection View Go Run Terminal Help
Search
diabetespredictor-using-decision-tree-svc.ipynb
Release Notes: 1.83.0
C: > Users > Karthick > Downloads > diabetespredictor-using-decision-tree-svc.ipynb > Diabetes Prediction
Code Markdown Outline
Select Kernel
[1]
Python
Empty markdown cell, double-click or press enter to edit.
#read the data file
data = pd.read_csv("/kaggle/input/diabetes-data-set/diabetes.csv")
data.head()
[2]
Python
data.describe()
[3]
Python
data.isnull().sum()
[4]
Python
Pregnancies 0
Glucose 0
BloodPressure 0
SkinThickness 0
Restricted Mode
Cell 1 of 43

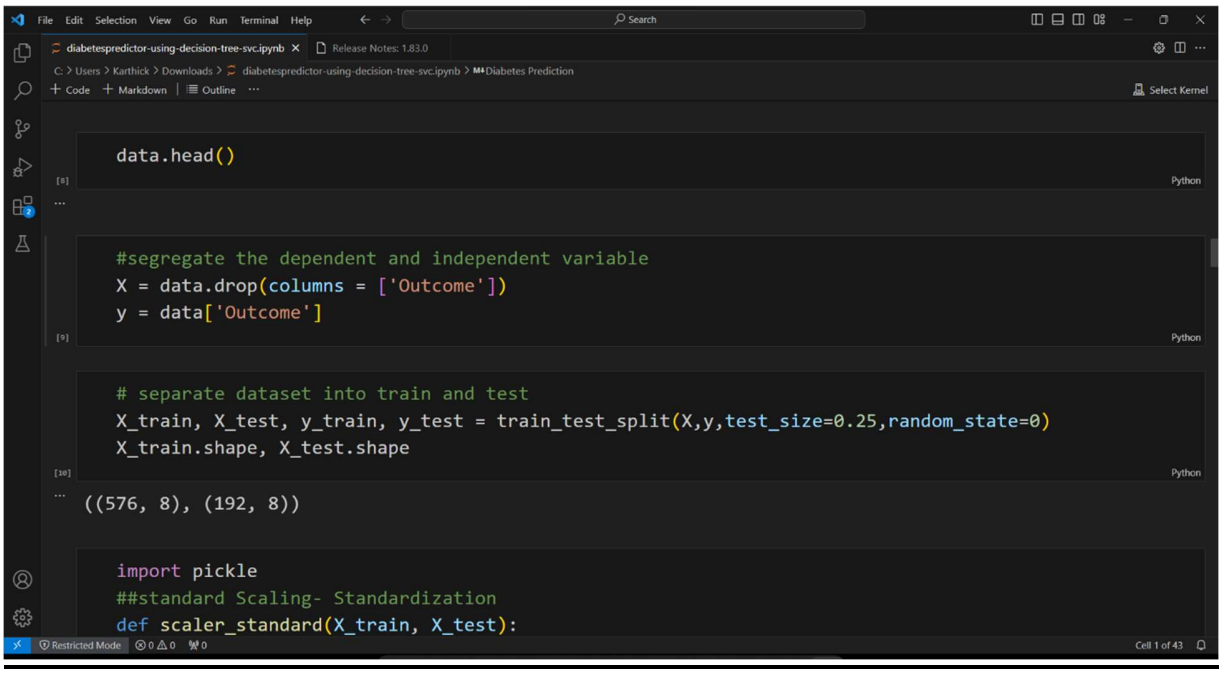

Loading...
#here few misconception is there lke BMI can not be zero, BP can't be zero, glucose, insuline can't
now replacing zero values with the mean of the column
data['BMI'] = data['BMI'].replace(0,data['BMI'].mean())
data['BloodPressure'] = data['BloodPressure'].replace(0,data['BloodPressure'].mean())
data['Glucose'] = data['Glucose'].replace(0,data['Glucose'].mean())
data['Insulin'] = data['Insulin'].replace(0,data['Insulin'].mean())
data['SkinThickness'] = data['SkinThickness'].replace(0,data['SkinThickness'].mean())
data.describe()
#now we have dealt with the 0 values and data looks better. But, there still are outliers present in
fig, ax = plt.subplots(figsize=(15,10))
sns.boxplot(data=data, width= 0.5,ax=ax, fliersize=3)

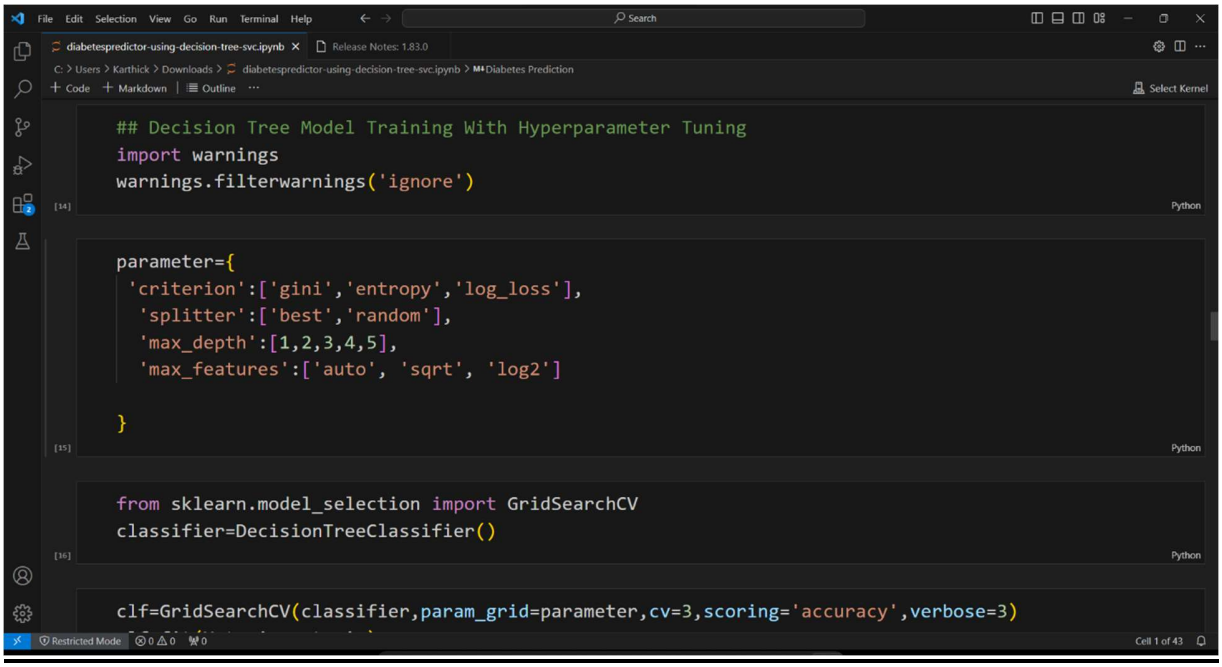

Pregnancies 0
Glucose 0
BloodPressure 0
SkinThickness 0
Insulin 0
BMI 0
DiabetesPedigreeFunction 0
Age 0
Outcome 0
dtype: int64
We can see there few data for columns Glucose , Insulin, skin thickenss, BMI and Blood Pressure which have value as 0. That's not possible,right? you can do a quick search to see that one cannot have 0 values for these. Let's deal with that. we can either remove such data or simply replace it with their respective mean values. Let's do the latter.
#here few misconception is there lke BMI can not be zero, BP can't be zero, glucose, insuline can't
now replacing zero values with the mean of the column


File Edit Selection View Go Run Terminal Help
Search
diabetespredictor-using-decision-tree-svc.ipynb
Release Notes: 1.83.0
C: > Users > Karthick > Downloads > diabetespredictor-using-decision-tree-svc.ipynb > Diabetes Prediction
Code Markdown Outline
Select Kernel
Code Markdown
#now we have dealt with the 0 values and data looks better. But, there still are outliers present in
fig, ax = plt.subplots(figsize=(15,10))
sns.boxplot(data=data, width= 0.5,ax=ax, fliersize=3)
Python
<Axes: >
800
600
400
Restricted Mode
Cell 1 of 43


File Edit Selection View Go Run Terminal Help
Search
diabetespredictor-using-decision-tree-svc.ipynb
Release Notes: 1.83.0
C: > Users > Karthick > Downloads > diabetespredictor-using-decision-tree-svc.ipynb > Diabetes Prediction
Code Markdown Outline
Select Kernel
600
400
200
0
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
Restricted Mode
Cell 1 of 43


((576, 8), (192, 8))
import pickle
##standard Scaling- Standardization
def scaler_standard(X_train, X_test):
#scaling the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
#saving the model
file = open('standardScalar.pkl','wb')
pickle.dump(scaler,file)
file.close()
return X_train_scaled, X_test_scaled
[11]
Python
X_train_scaled, X_test_scaled = scaler_standard(X_train, X_test)
[12]
Python


data.head()
[8]
Python
#segregate the dependent and independent variable
X = data.drop(columns = ['Outcome'])
y = data['Outcome']
[9]
Python
separate dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=0)
X_train.shape, X_test.shape
[10]
Python
((576, 8), (192, 8))
import pickle
##standard Scaling- Standardization
def scaler_standard(X_train, X_test):



```
## Decision Tree Model Training With Hyperparameter Tuning
import warnings
warnings.filterwarnings('ignore')
```

[14] Python

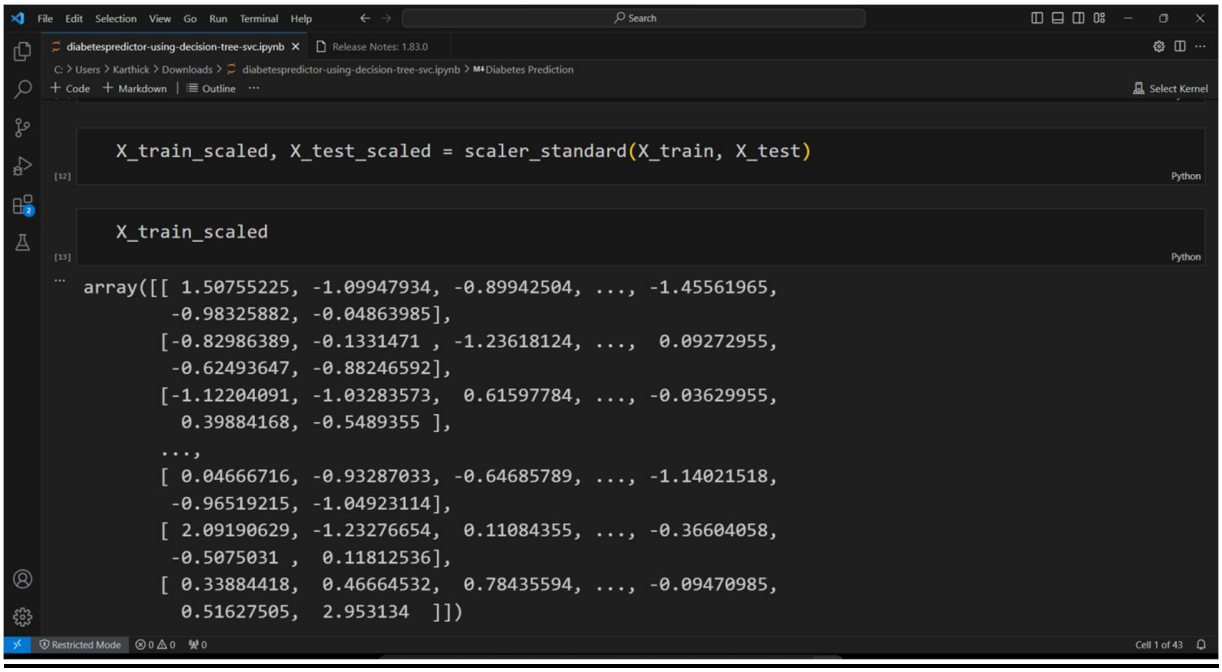
```
parameter={
 'criterion':['gini','entropy','log_loss'],
  'splitter':['best','random'],
  'max_depth':[1,2,3,4,5],
  'max_features':['auto', 'sqrt', 'log2']

}
```

[15] Python

```
from sklearn.model_selection import GridSearchCV
classifier=DecisionTreeClassifier()
```

[16] Python

```
clf=GridSearchCV(classifier,param_grid=parameter,cv=3,scoring='accuracy',verbose=3)
```



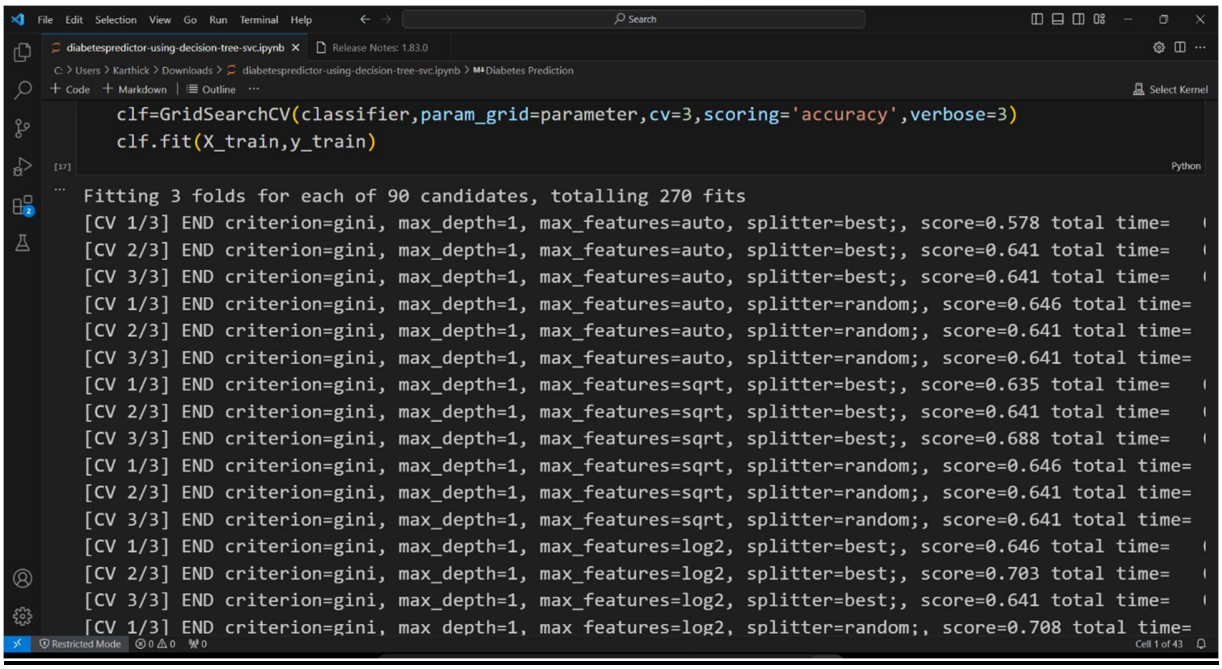
```
X_train_scaled, X_test_scaled = scaler_standard(X_train, X_test)
```

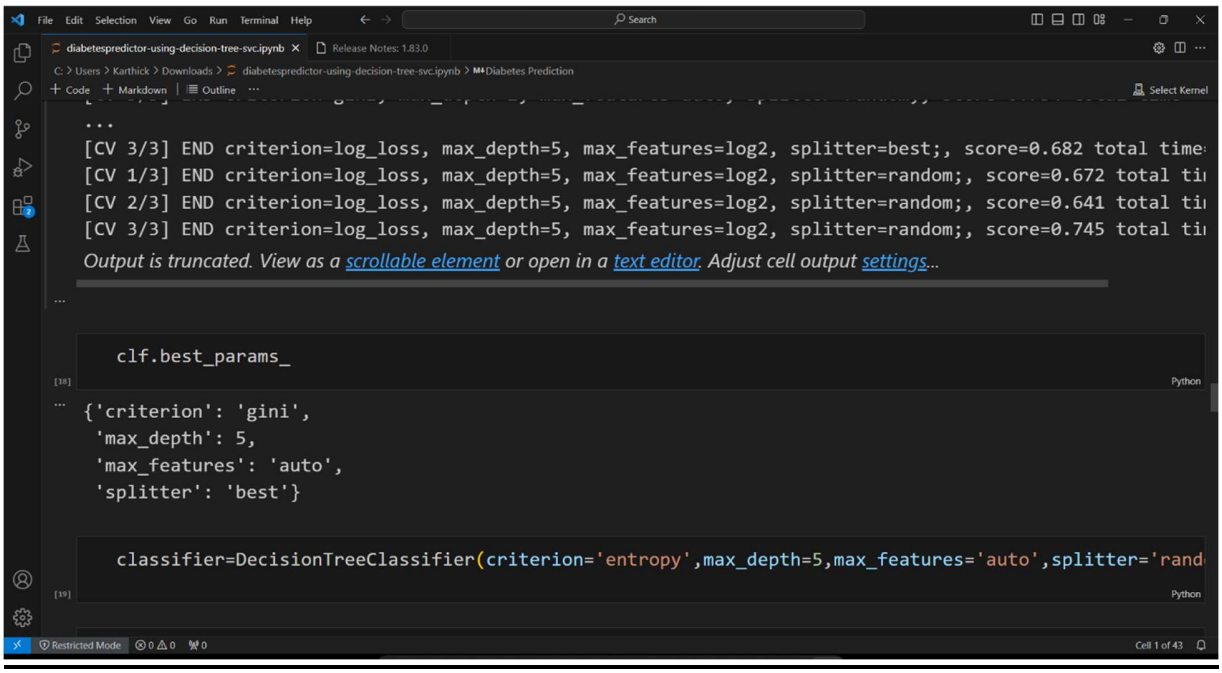
[12] Python

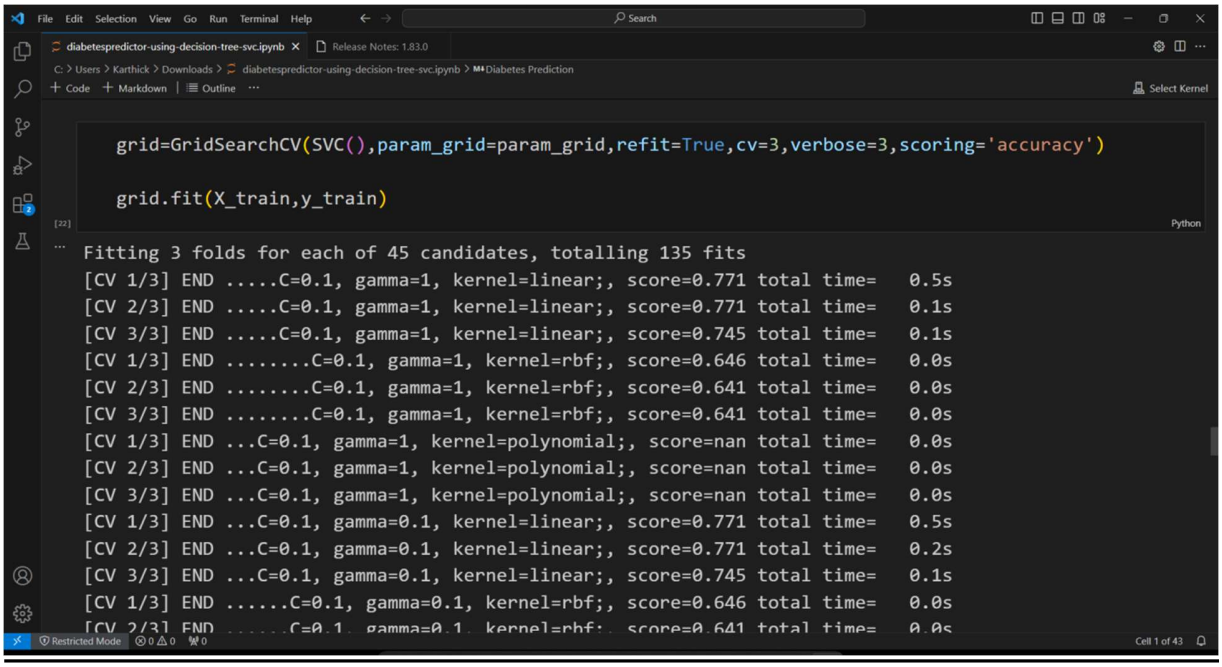
```
X_train_scaled
```

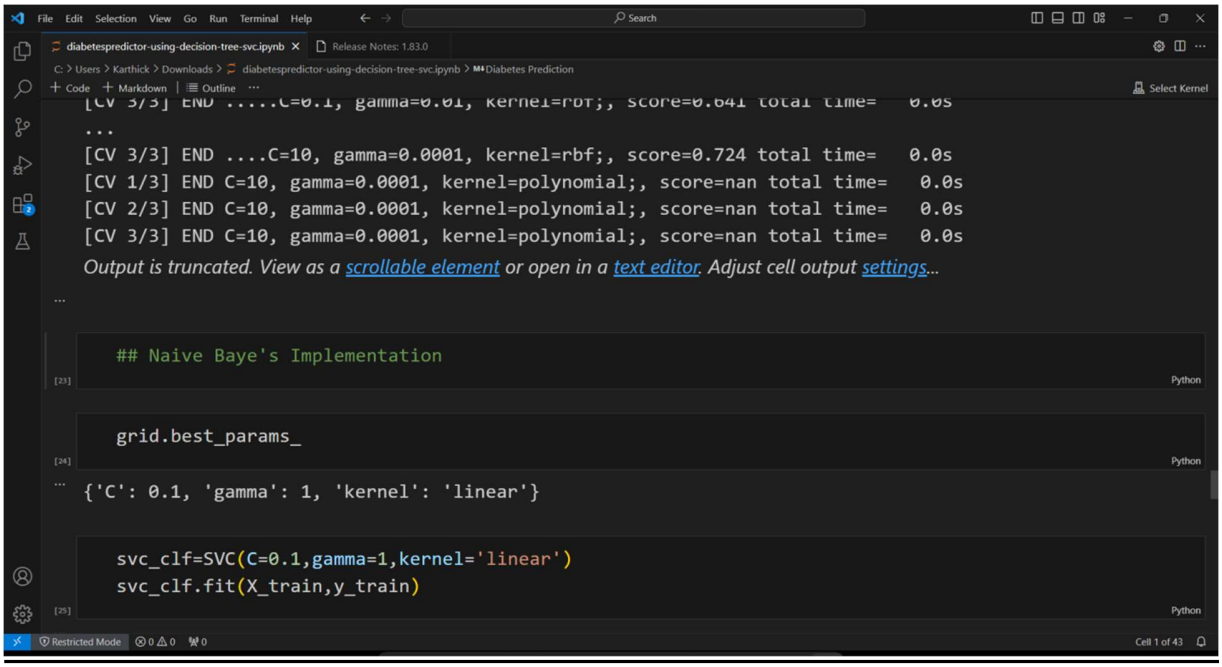
[13] Python

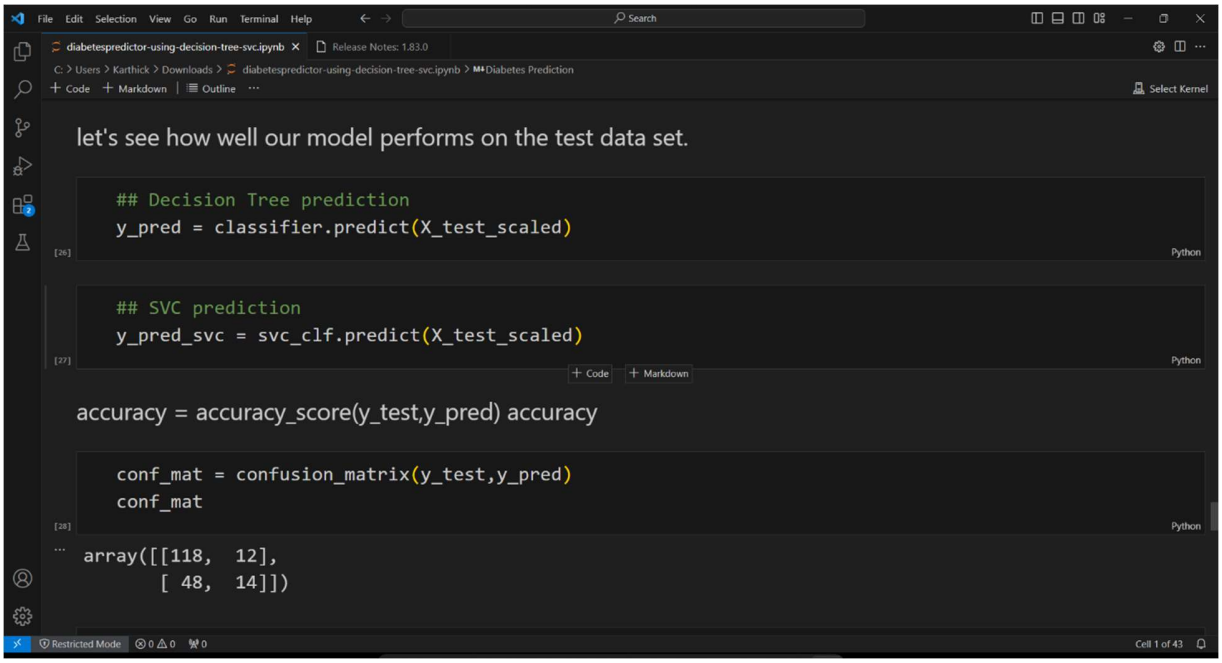
```
array([[ 1.50755225, -1.09947934, -0.89942504, ..., -1.45561965,
        -0.98325882, -0.04863985],
       [-0.82986389, -0.1331471 , -1.23618124, ...,  0.09272955,
        -0.62493647, -0.88246592],
       [-1.12204091, -1.03283573,  0.61597784, ..., -0.03629955,
         0.39884168, -0.5489355 ],
       ...,
       [ 0.04666716, -0.93287033, -0.64685789, ..., -1.14021518,
        -0.96519215, -1.04923114],
       [ 2.09190629, -1.23276654,  0.11084355, ..., -0.36604058,
        -0.5075031 ,  0.11812536],
       [ 0.33884418,  0.46664532,  0.78435594, ..., -0.09470985,
         0.51627505,  2.953134  ]])
```

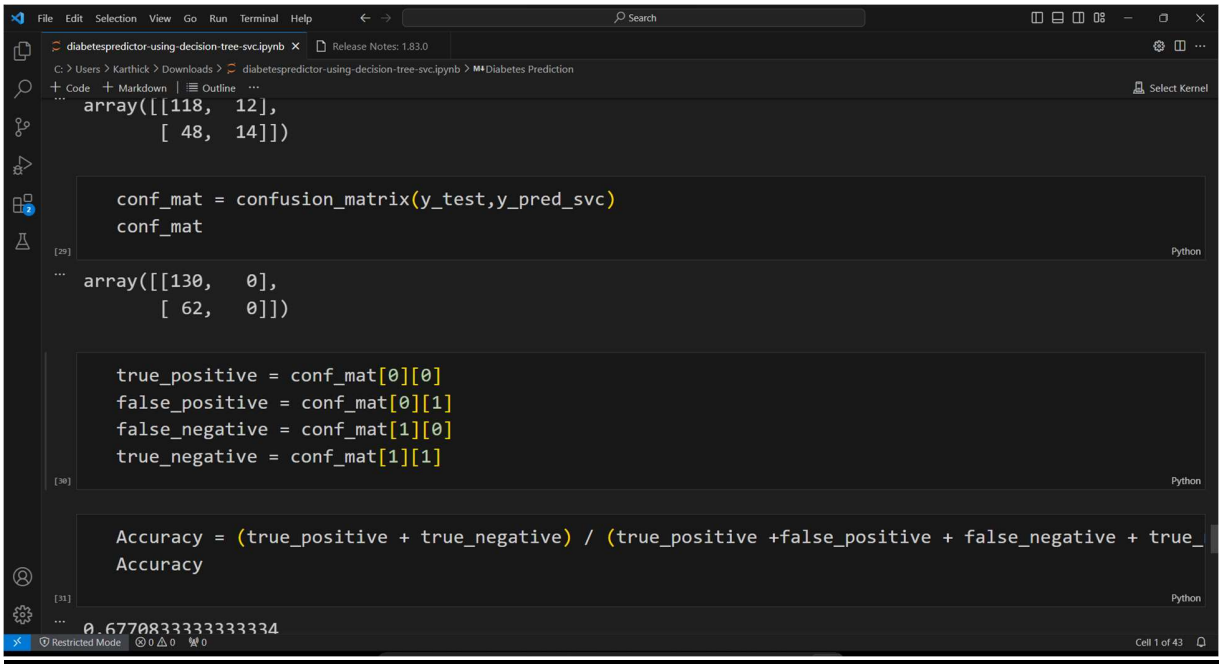

File Edit Selection View Go Run Terminal Help
Search
diabetespredictor-using-decision-tree-svc.ipynb
Release Notes: 1.83.0
C: > Users > Karthick > Downloads > diabetespredictor-using-decision-tree-svc.ipynb > Diabetes Prediction
+ Code + Markdown | Outline
Select Kernel
clf=GridSearchCV(classifier,param_grid=parameter,cv=3,scoring='accuracy',verbose=3)
clf.fit(X_train,y_train)
[17]
Python
Fitting 3 folds for each of 90 candidates, totalling 270 fits
[CV 1/3] END criterion=gini, max_depth=1, max_features=auto, splitter=best;, score=0.578 total time=
[CV 2/3] END criterion=gini, max_depth=1, max_features=auto, splitter=best;, score=0.641 total time=
[CV 3/3] END criterion=gini, max_depth=1, max_features=auto, splitter=best;, score=0.641 total time=
[CV 1/3] END criterion=gini, max_depth=1, max_features=auto, splitter=random;, score=0.646 total time=
[CV 2/3] END criterion=gini, max_depth=1, max_features=auto, splitter=random;, score=0.641 total time=
[CV 3/3] END criterion=gini, max_depth=1, max_features=auto, splitter=random;, score=0.641 total time=
[CV 1/3] END criterion=gini, max_depth=1, max_features=sqrt, splitter=best;, score=0.635 total time=
[CV 2/3] END criterion=gini, max_depth=1, max_features=sqrt, splitter=best;, score=0.641 total time=
[CV 3/3] END criterion=gini, max_depth=1, max_features=sqrt, splitter=best;, score=0.688 total time=
[CV 1/3] END criterion=gini, max_depth=1, max_features=sqrt, splitter=random;, score=0.646 total time=
[CV 2/3] END criterion=gini, max_depth=1, max_features=sqrt, splitter=random;, score=0.641 total time=
[CV 3/3] END criterion=gini, max_depth=1, max_features=sqrt, splitter=random;, score=0.641 total time=
[CV 1/3] END criterion=gini, max_depth=1, max_features=log2, splitter=best;, score=0.646 total time=
[CV 2/3] END criterion=gini, max_depth=1, max_features=log2, splitter=best;, score=0.703 total time=
[CV 3/3] END criterion=gini, max_depth=1, max_features=log2, splitter=best;, score=0.641 total time=
[CV 1/3] END criterion=gini, max_depth=1, max_features=log2, splitter=random;, score=0.708 total time=
Restricted Mode
Cell 1 of 43

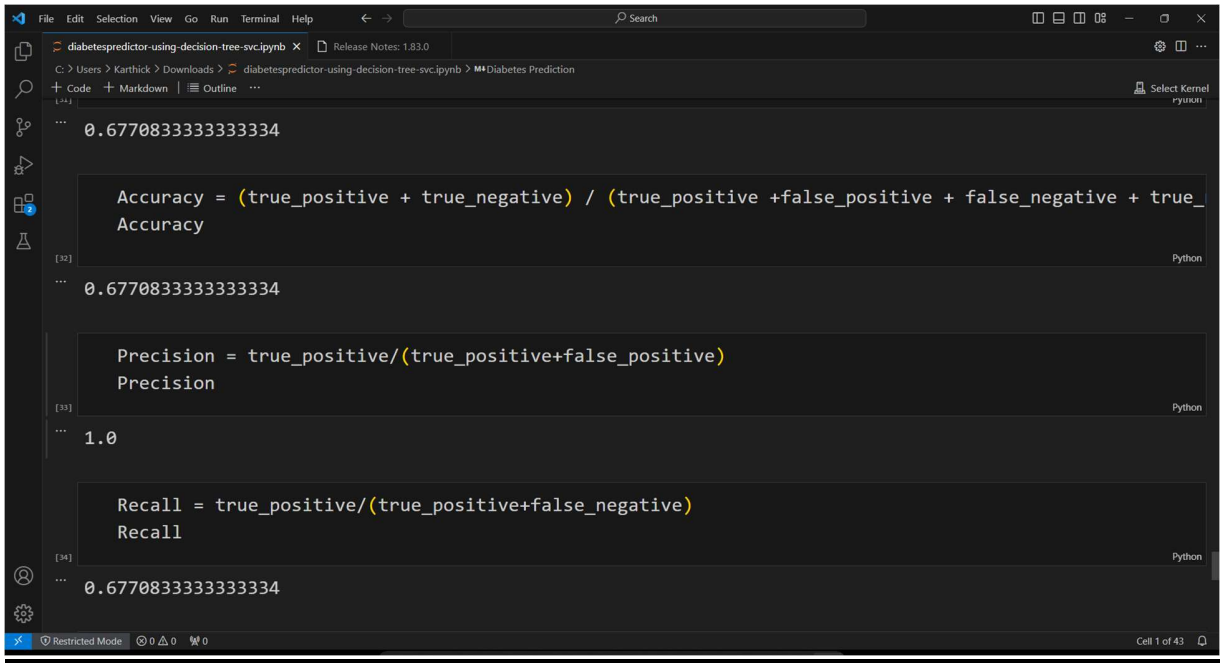

File Edit Selection View Go Run Terminal Help
Search
diabetespredictor-using-decision-tree-svc.ipynb
Release Notes: 1.83.0
C: > Users > Karthick > Downloads > diabetespredictor-using-decision-tree-svc.ipynb > Diabetes Prediction
+ Code + Markdown | Outline
Select Kernel
...
[CV 3/3] END criterion=log_loss, max_depth=5, max_features=log2, splitter=best;, score=0.682 total time=
[CV 1/3] END criterion=log_loss, max_depth=5, max_features=log2, splitter=random;, score=0.672 total ti
[CV 2/3] END criterion=log_loss, max_depth=5, max_features=log2, splitter=random;, score=0.641 total ti
[CV 3/3] END criterion=log_loss, max_depth=5, max_features=log2, splitter=random;, score=0.745 total ti
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
clf.best_params_
[18]
Python
{'criterion': 'gini',
'max_depth': 5,
'max_features': 'auto',
'splitter': 'best'}
classifier=DecisionTreeClassifier(criterion='entropy',max_depth=5,max_features='auto',splitter='rand
[19]
Python
Restricted Mode
Cell 1 of 43

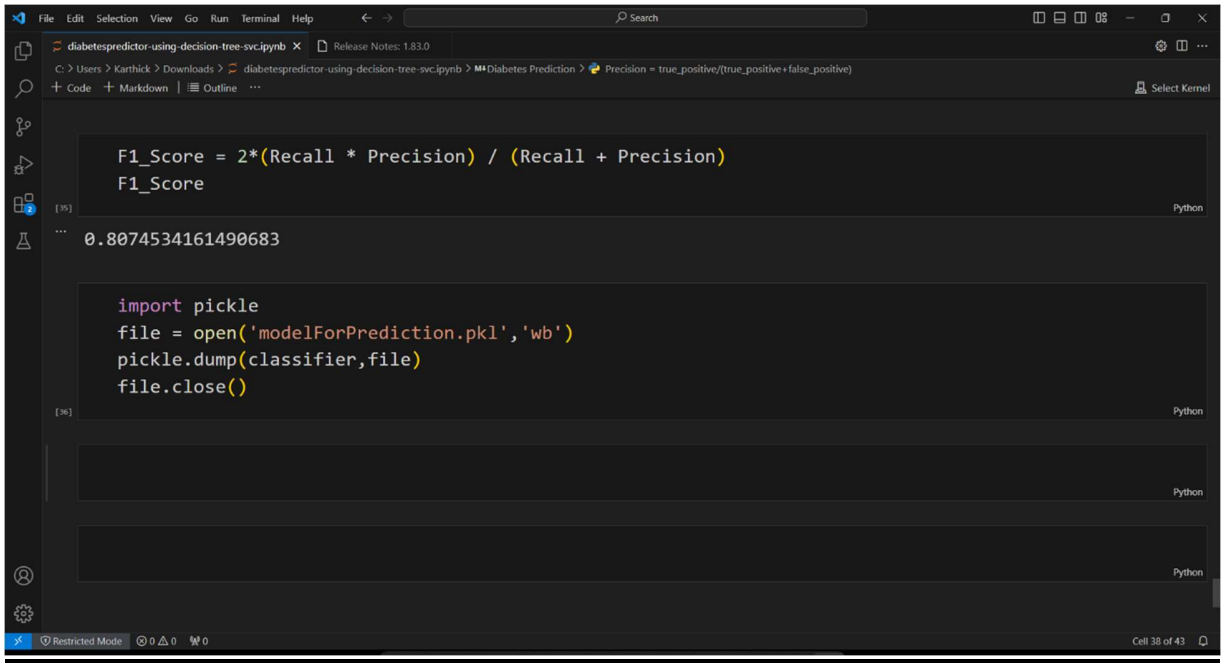

File Edit Selection View Go Run Terminal Help
Search
diabetespredictor-using-decision-tree-svc.ipynb
Release Notes: 1.83.0
C: > Users > Karthick > Downloads > diabetespredictor-using-decision-tree-svc.ipynb > Diabetes Prediction
+ Code + Markdown | Outline
Select Kernel
grid=GridSearchCV(SVC(),param_grid=param_grid,refit=True,cv=3,verbose=3,scoring='accuracy')
grid.fit(X_train,y_train)
[22]
Python
Fitting 3 folds for each of 45 candidates, totalling 135 fits
[CV 1/3] END .....C=0.1, gamma=1, kernel=linear;, score=0.771 total time= 0.5s
[CV 2/3] END .....C=0.1, gamma=1, kernel=linear;, score=0.771 total time= 0.1s
[CV 3/3] END .....C=0.1, gamma=1, kernel=linear;, score=0.745 total time= 0.1s
[CV 1/3] END .....C=0.1, gamma=1, kernel=rbf;, score=0.646 total time= 0.0s
[CV 2/3] END .....C=0.1, gamma=1, kernel=rbf;, score=0.641 total time= 0.0s
[CV 3/3] END .....C=0.1, gamma=1, kernel=rbf;, score=0.641 total time= 0.0s
[CV 1/3] END ...C=0.1, gamma=1, kernel=polynomial;, score=nan total time= 0.0s
[CV 2/3] END ...C=0.1, gamma=1, kernel=polynomial;, score=nan total time= 0.0s
[CV 3/3] END ...C=0.1, gamma=1, kernel=polynomial;, score=nan total time= 0.0s
[CV 1/3] END ...C=0.1, gamma=0.1, kernel=linear;, score=0.771 total time= 0.5s
[CV 2/3] END ...C=0.1, gamma=0.1, kernel=linear;, score=0.771 total time= 0.2s
[CV 3/3] END ...C=0.1, gamma=0.1, kernel=linear;, score=0.745 total time= 0.1s
[CV 1/3] END .....C=0.1, gamma=0.1, kernel=rbf;, score=0.646 total time= 0.0s
Restricted Mode
Cell 1 of 43

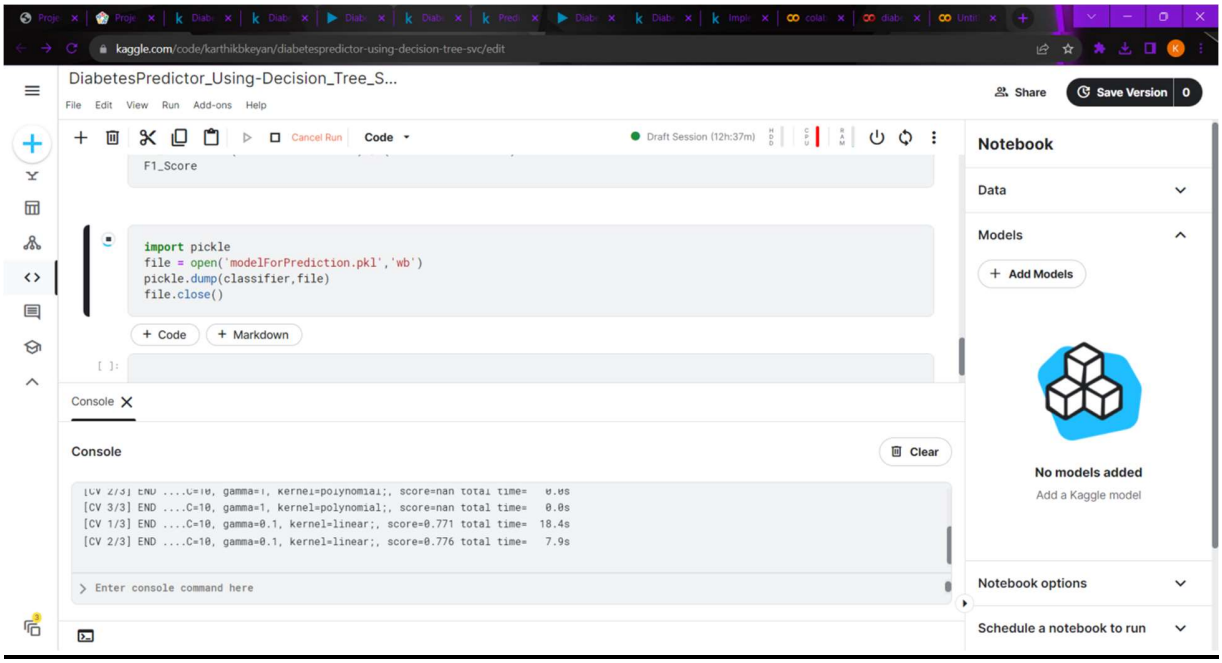

File Edit Selection View Go Run Terminal Help
Search
diabetespredictor-using-decision-tree-svc.ipynb
Release Notes: 1.83.0
C: > Users > Karthick > Downloads > diabetespredictor-using-decision-tree-svc.ipynb > Diabetes Prediction
+ Code + Markdown | Outline
Select Kernel
...
[CV 3/3] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.724 total time= 0.0s
[CV 1/3] END C=10, gamma=0.0001, kernel=polynomial;, score=nan total time= 0.0s
[CV 2/3] END C=10, gamma=0.0001, kernel=polynomial;, score=nan total time= 0.0s
[CV 3/3] END C=10, gamma=0.0001, kernel=polynomial;, score=nan total time= 0.0s
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
Naive Baye's Implementation
[23]
Python
grid.best_params_
[24]
Python
{'C': 0.1, 'gamma': 1, 'kernel': 'linear'}
svc_clf=SVC(C=0.1,gamma=1,kernel='linear')
svc_clf.fit(X_train,y_train)
[25]
Python
Restricted Mode
Cell 1 of 43


let's see how well our model performs on the test data set.
Decision Tree prediction
y_pred = classifier.predict(X_test_scaled)
SVC prediction
y_pred_svc = svc_clf.predict(X_test_scaled)
accuracy = accuracy_score(y_test,y_pred) accuracy
conf_mat = confusion_matrix(y_test,y_pred)
conf_mat
array([[118, 12],
[48, 14]])


array([[118, 12],
[48, 14]])
conf_mat = confusion_matrix(y_test,y_pred_svc)
conf_mat
array([[130, 0],
[62, 0]])
true_positive = conf_mat[0][0]
false_positive = conf_mat[0][1]
false_negative = conf_mat[1][0]
true_negative = conf_mat[1][1]
Accuracy = (true_positive + true_negative) / (true_positive +false_positive + false_negative + true_
Accuracy
0.6770833333333334


File Edit Selection View Go Run Terminal Help
Search
diabetespredictor-using-decision-tree-svc.ipynb
Release Notes: 1.83.0
C: > Users > Karthick > Downloads > diabetespredictor-using-decision-tree-svc.ipynb > Diabetes Prediction
Code Markdown Outline
Select Kernel
0.6770833333333334
Accuracy = (true_positive + true_negative) / (true_positive +false_positive + false_negative + true_
Accuracy
[32]
Python
0.6770833333333334
Precision = true_positive/(true_positive+false_positive)
Precision
[33]
Python
1.0
Recall = true_positive/(true_positive+false_negative)
Recall
[34]
Python
0.6770833333333334
Restricted Mode
Cell 1 of 43

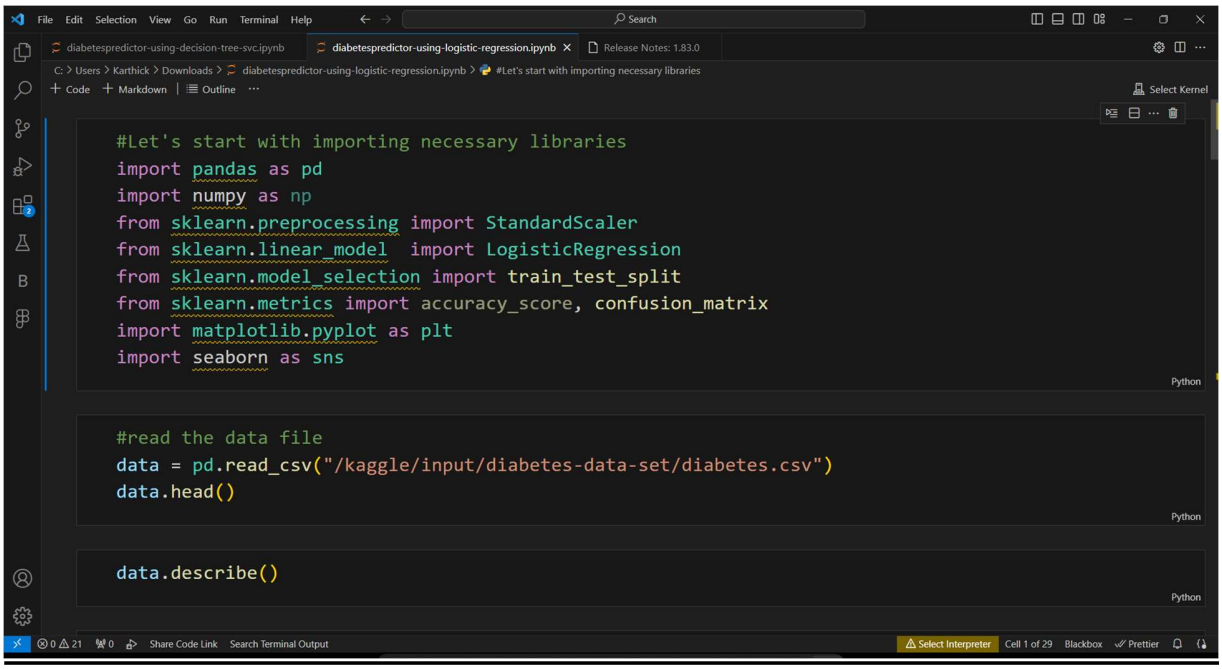

File Edit Selection View Go Run Terminal Help
Search
diabetespredictor-using-decision-tree-svc.ipynb
Release Notes: 1.83.0
C: > Users > Karthick > Downloads > diabetespredictor-using-decision-tree-svc.ipynb > Diabetes Prediction > Precision = true_positive/(true_positive+false_positive)
Code Markdown Outline
Select Kernel
F1_Score = 2*(Recall * Precision) / (Recall + Precision)
F1_Score
[35]
Python
0.8074534161490683
import pickle
file = open('modelForPrediction.pkl','wb')
pickle.dump(classifier,file)
file.close()
[36]
Python
Python
Python
Restricted Mode
Cell 38 of 43

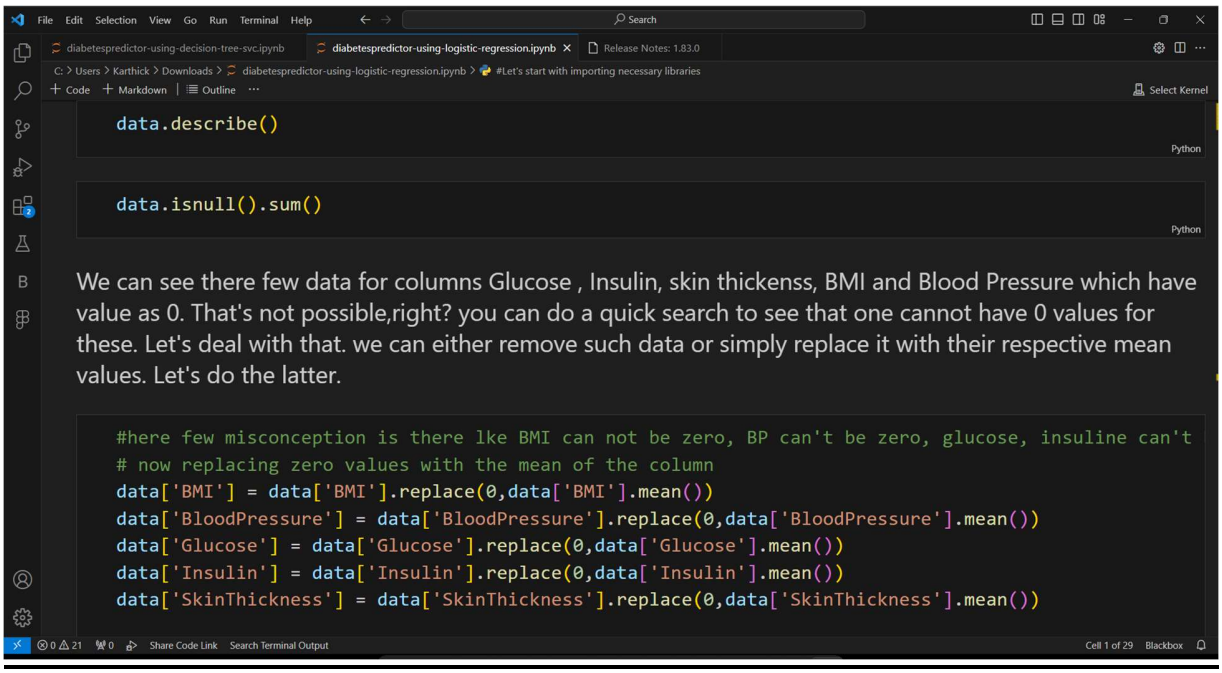


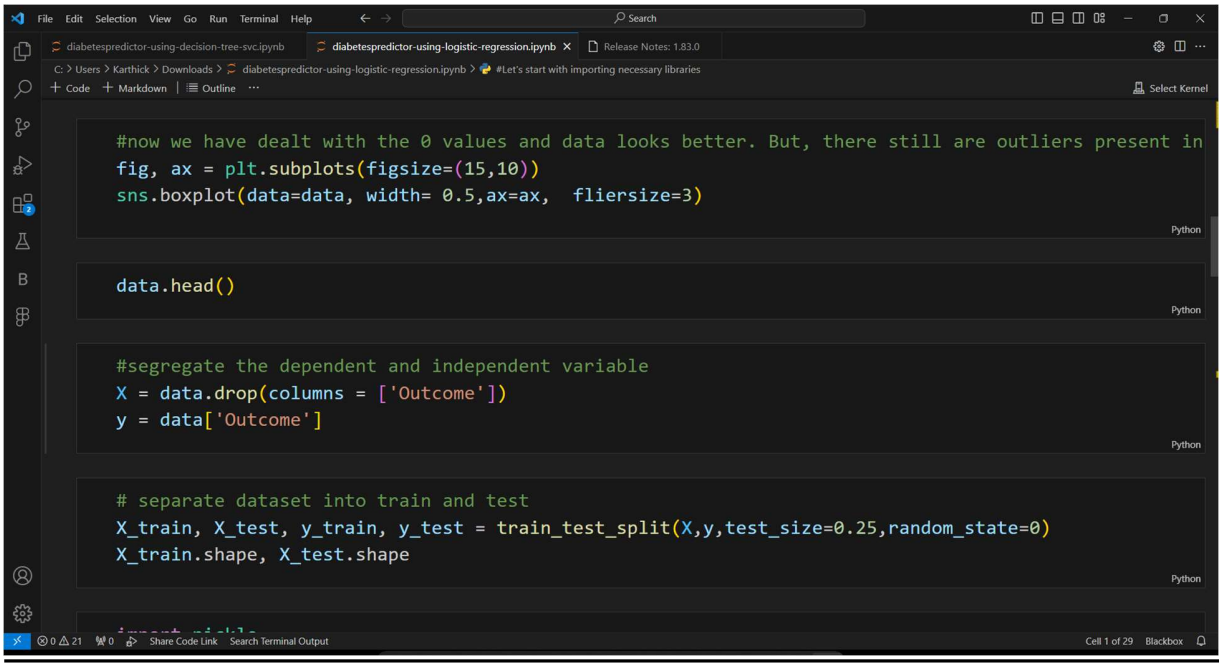
OUTPUT:

```
[CV 2/3] END .....C=10, gamma=1, kernel=linear;, score=0.776 total time=   7.9s
[CV 3/3] END .....C=10, gamma=1, kernel=linear;, score=0.740 total time=   5.0s
[CV 1/3] END .....C=10, gamma=1, kernel=rbf;, score=0.646 total time=   0.0s
[CV 2/3] END .....C=10, gamma=1, kernel=rbf;, score=0.641 total time=   0.0s
[CV 3/3] END .....C=10, gamma=1, kernel=rbf;, score=0.641 total time=   0.0s
```

DIABETES PREDICTION USING LOGISTIC REGRESSION:





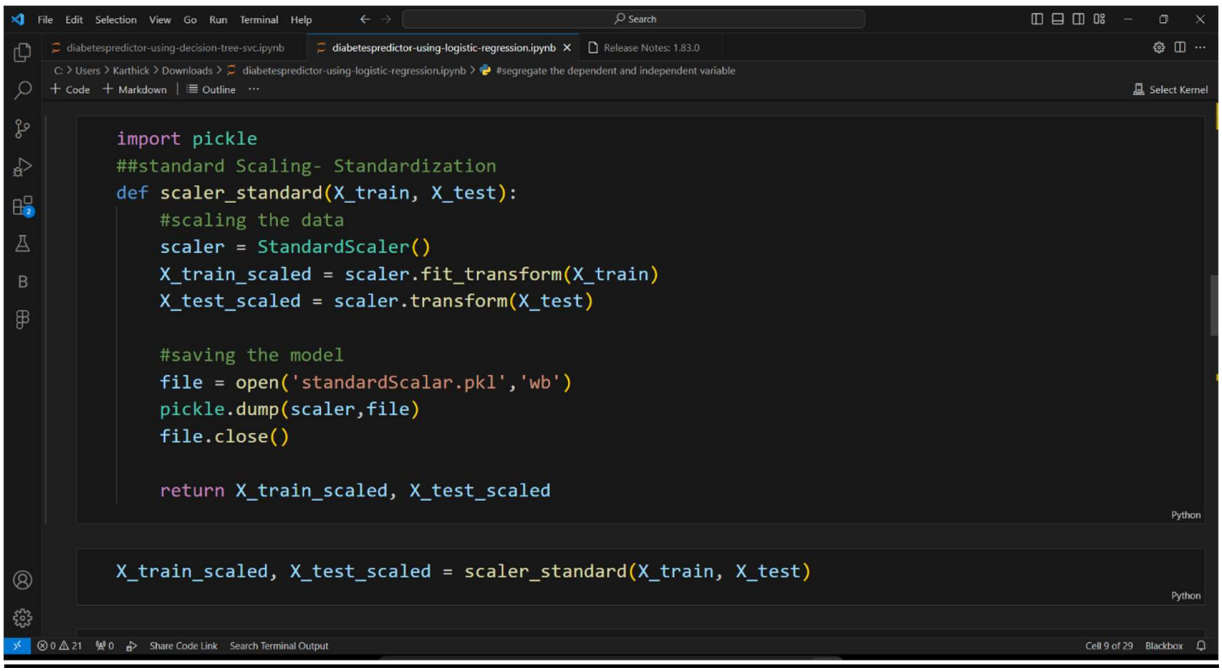


```python
#now we have dealt with the 0 values and data looks better. But, there still are outliers present in
fig, ax = plt.subplots(figsize=(15,10))
sns.boxplot(data=data, width= 0.5,ax=ax,  fliersize=3)
```

```python
data.head()
```

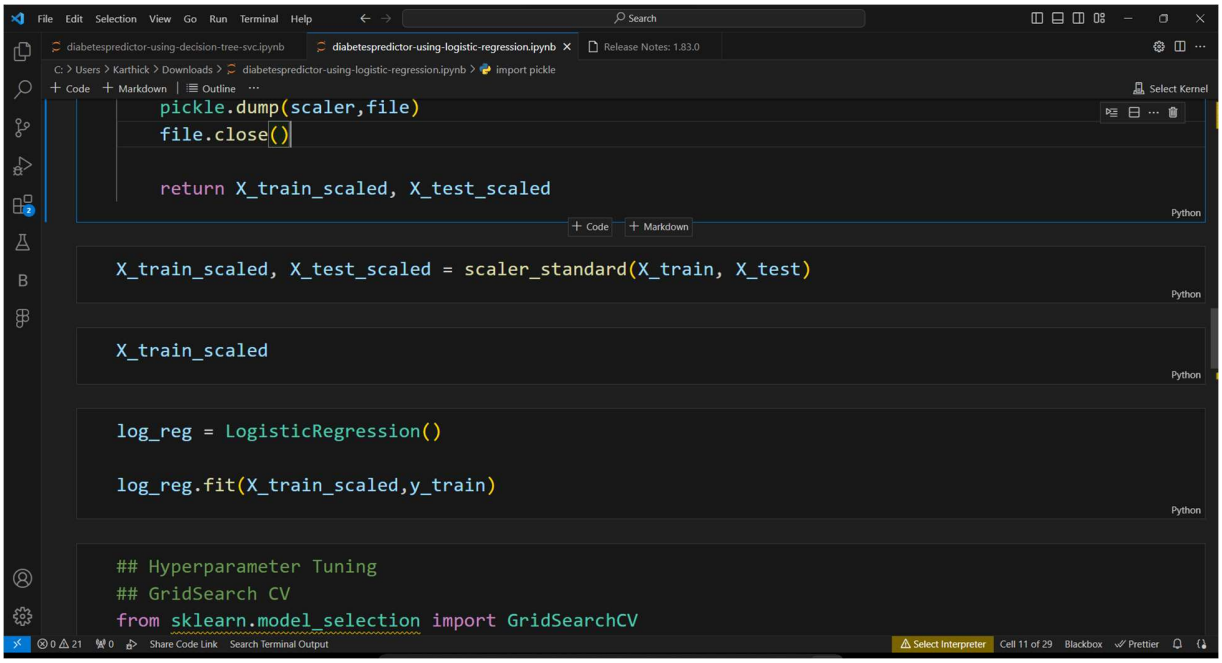
```python
#segregate the dependent and independent variable
X = data.drop(columns = ['Outcome'])
y = data['Outcome']
```

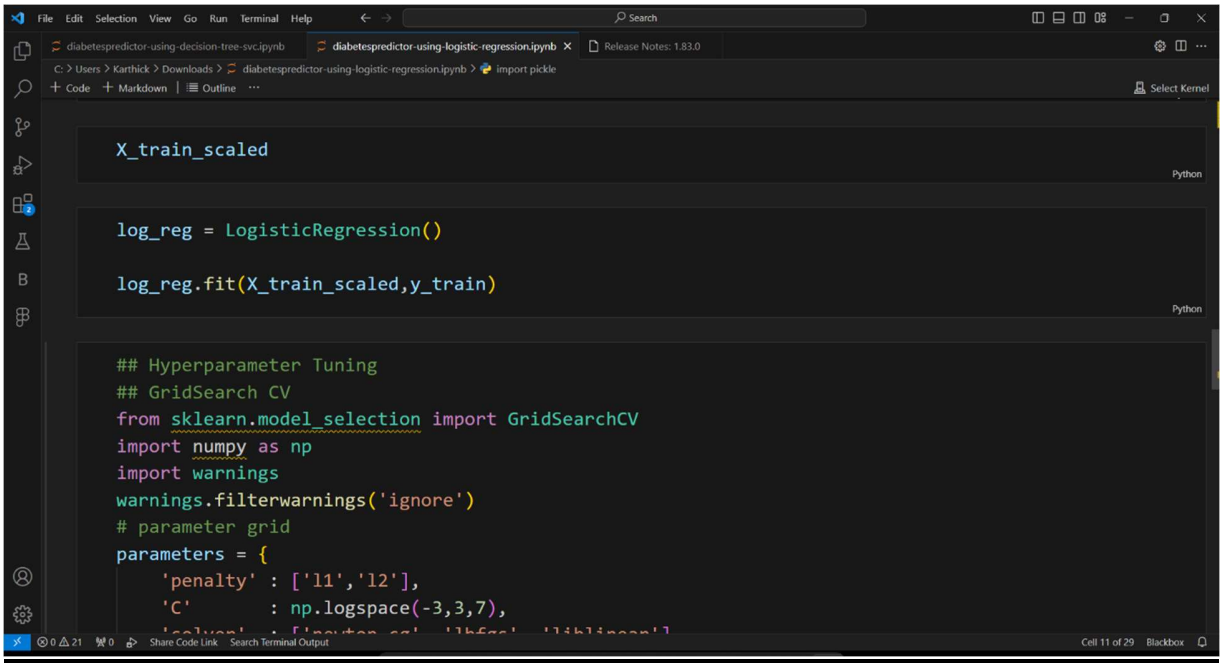
```python
# separate dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=0)
X_train.shape, X_test.shape
```

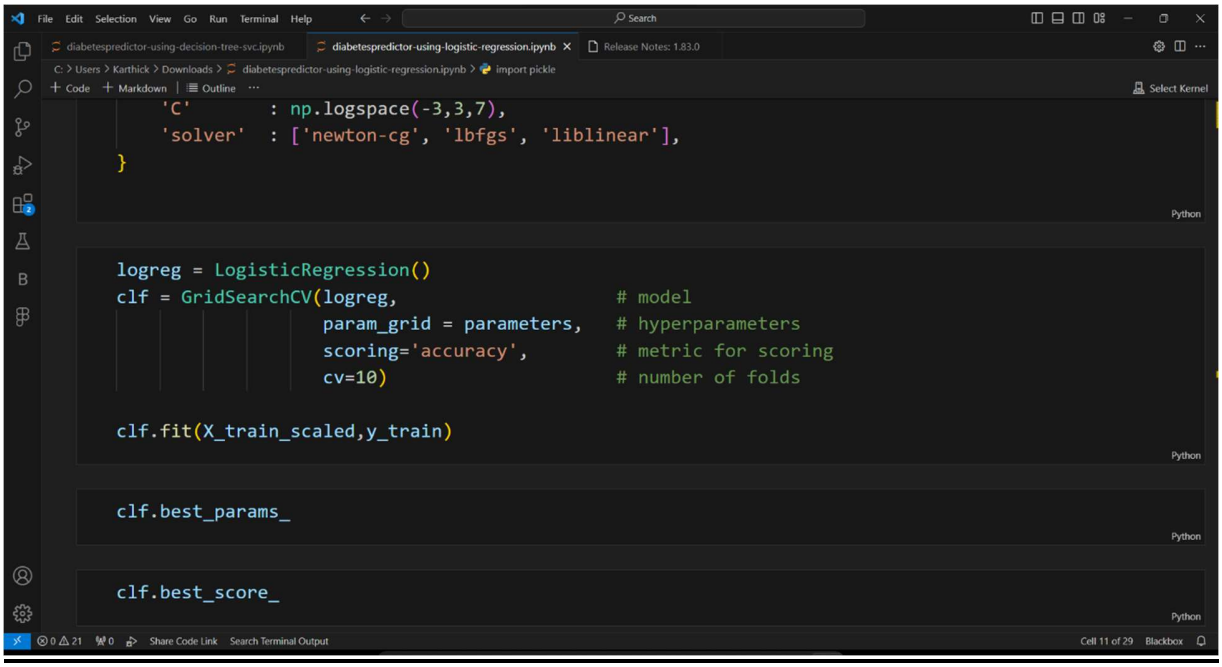


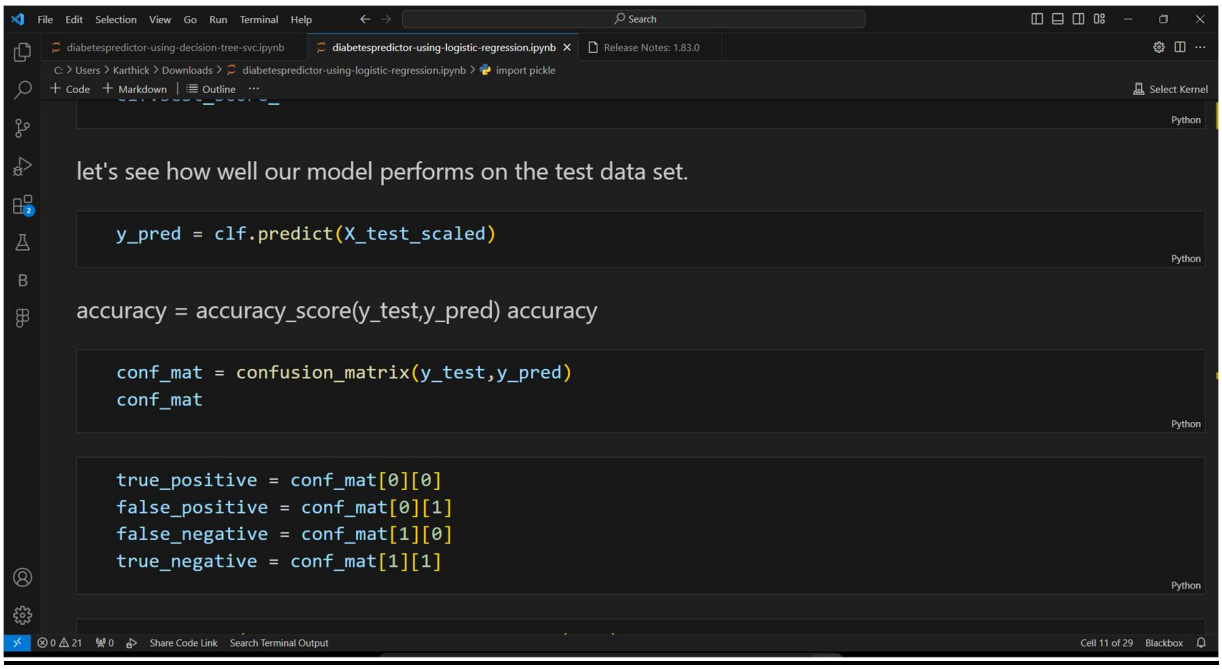
```python
import pickle
##standard Scaling- Standardization
def scaler_standard(X_train, X_test):
    #scaling the data
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    #saving the model
    file = open('standardScalar.pkl','wb')
    pickle.dump(scaler,file)
    file.close()

    return X_train_scaled, X_test_scaled
```

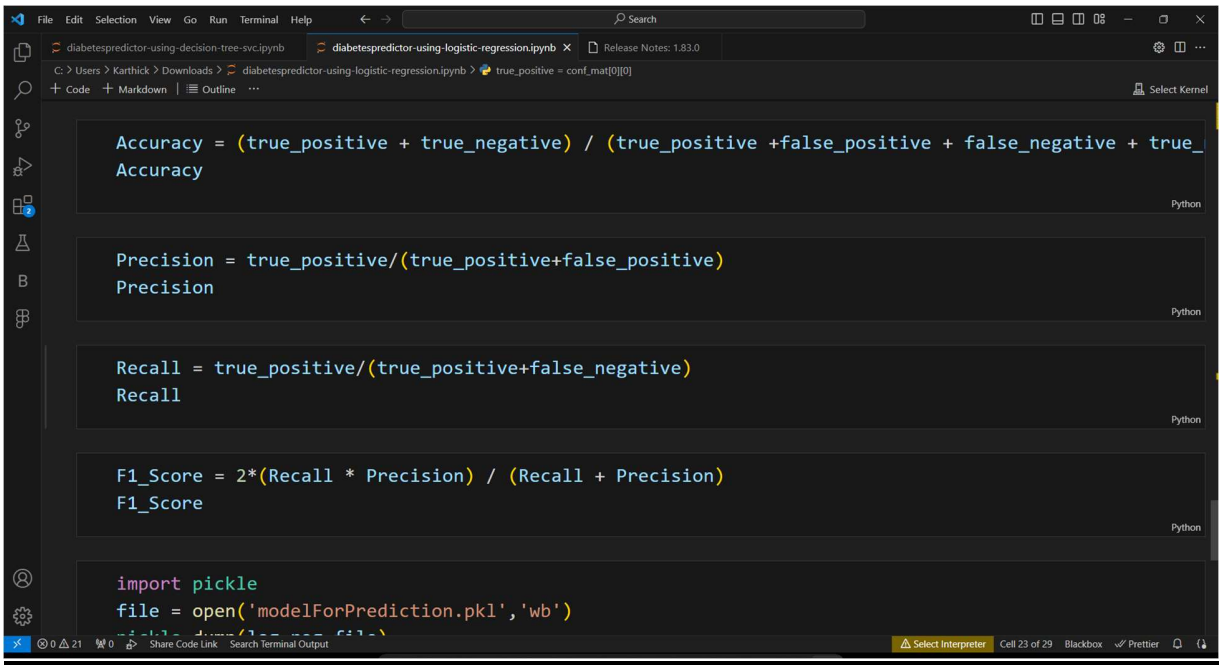
```python
X_train_scaled, X_test_scaled = scaler_standard(X_train, X_test)
```

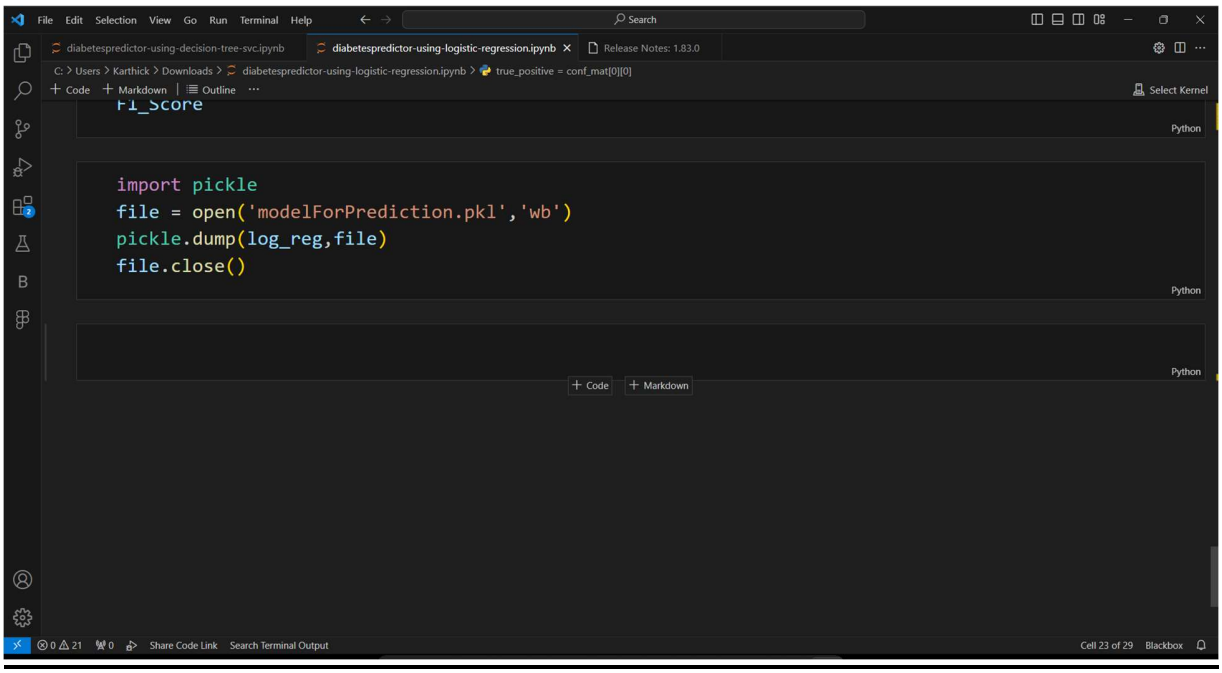

pickle.dump(scaler,file)
file.close()
return X_train_scaled, X_test_scaled
X_train_scaled, X_test_scaled = scaler_standard(X_train, X_test)
X_train_scaled
log_reg = LogisticRegression()
log_reg.fit(X_train_scaled,y_train)
Hyperparameter Tuning
GridSearch CV
from sklearn.model_selection import GridSearchCV

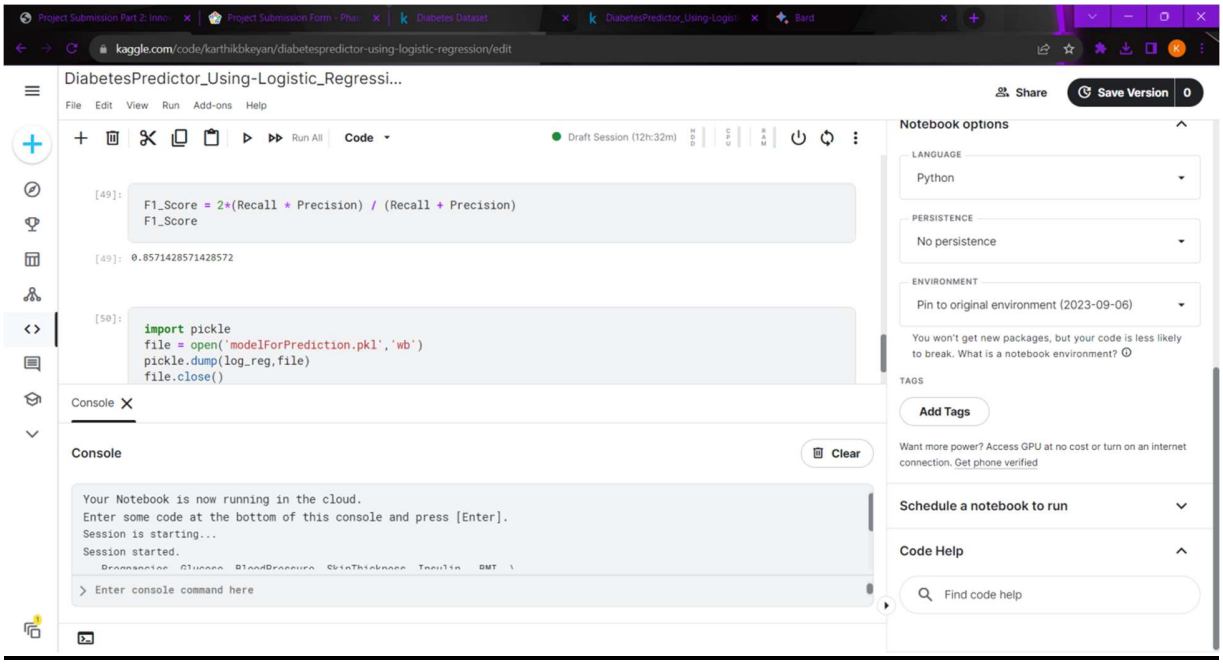

X_train_scaled
log_reg = LogisticRegression()
log_reg.fit(X_train_scaled,y_train)
Hyperparameter Tuning
GridSearch CV
from sklearn.model_selection import GridSearchCV
import numpy as np
import warnings
warnings.filterwarnings('ignore')
# parameter grid
parameters = {
'penalty' : ['l1','l2'],
'C' : np.logspace(-3,3,7),


'C' : np.logspace(-3,3,7),
'solver' : ['newton-cg', 'lbfgs', 'liblinear'],
}
logreg = LogisticRegression()
clf = GridSearchCV(logreg, # model
param_grid = parameters, # hyperparameters
scoring='accuracy', # metric for scoring
cv=10) # number of folds
clf.fit(X_train_scaled,y_train)
clf.best_params_
clf.best_score_


let's see how well our model performs on the test data set.
y_pred = clf.predict(X_test_scaled)
accuracy = accuracy_score(y_test,y_pred) accuracy
conf_mat = confusion_matrix(y_test,y_pred)
conf_mat
true_positive = conf_mat[0][0]
false_positive = conf_mat[0][1]
false_negative = conf_mat[1][0]
true_negative = conf_mat[1][1]


File Edit Selection View Go Run Terminal Help
Search
diabetespredictor-using-decision-tree-svc.ipynb
diabetespredictor-using-logistic-regression.ipynb
Release Notes: 1.83.0
C: > Users > Karthick > Downloads > diabetespredictor-using-logistic-regression.ipynb > true_positive = conf_mat[0][0]
+ Code + Markdown | Outline ...
Select Kernel
Accuracy = (true_positive + true_negative) / (true_positive +false_positive + false_negative + true_
Accuracy
Python
Precision = true_positive/(true_positive+false_positive)
Precision
Python
Recall = true_positive/(true_positive+false_negative)
Recall
Python
F1_Score = 2*(Recall * Precision) / (Recall + Precision)
F1_Score
Python
import pickle
file = open('modelForPrediction.pkl','wb')
Share Code Link
Search Terminal Output
Select Interpreter
Cell 23 of 29
Blackbox
Prettier


File Edit Selection View Go Run Terminal Help
Search
diabetespredictor-using-decision-tree-svc.ipynb
diabetespredictor-using-logistic-regression.ipynb
Release Notes: 1.83.0
C: > Users > Karthick > Downloads > diabetespredictor-using-logistic-regression.ipynb > true_positive = conf_mat[0][0]
+ Code + Markdown | Outline ...
Select Kernel
F1_Score
Python
import pickle
file = open('modelForPrediction.pkl','wb')
pickle.dump(log_reg,file)
file.close()
Python
Python
+ Code + Markdown
Share Code Link
Search Terminal Output
Cell 23 of 29
Blackbox



OUTPUT:

0.796875
0.9
0.8181818181818182
0.857142857142857

3.BUILD LOADING AND PREPROCESSING THE DATASET

To build, load, and preprocess the dataset for AI-based diabetes prediction, you can follow these steps:

1. Collect the dataset.

The first step is to collect a dataset of patient data that includes both features that are predictive of diabetes and the target variable, which is whether or not the patient has diabetes. There are a number of publicly available datasets that can be used for AI-based diabetes prediction, such as the Pima Indians Diabetes Database.

2. Load the dataset into a programming language.

Once you have collected the dataset, you need to load it into a programming language such as Python or R. This will allow you to perform data preprocessing and machine learning tasks.

3. Preprocess the dataset.

Before you can train an AI model, you need to preprocess the dataset. This may involve cleaning the data, removing outliers, and encoding categorical features.

Here are some specific data preprocessing steps that you may need to perform:

- **Handle missing values:**

  Missing values are a common problem in real-world datasets. You can handle missing values by removing them, imputing them with a mean or median value, or using a more sophisticated technique such as multiple imputation.

- **Remove outliers:**

  Outliers are data points that are significantly different from the rest of the data. Outliers can skew the results of machine learning models, so it is important to remove them before training a model. You can identify outliers using statistical methods or by visualizing the data.

- **Encode categorical features:**

  Machine learning models can only work with numerical data. If your dataset contains categorical features, you need to encode them into numerical values. There are a number of different ways to encode categorical features, such as one-hot encoding and label encoding.

4. Split the dataset into training and testing sets.

Once you have preprocessed the dataset, you need to split it into training and testing sets. The training set will be used to train the AI model, and the testing set will be used to evaluate the model's performance.

You can split the dataset using a variety of methods, such as random sampling or stratified sampling. Stratified sampling ensures that the training and testing sets have the same distribution of target values.

5. Build and train the AI model.

Once you have split the dataset into training and testing sets, you can build and train the AI model. There are a variety of different machine learning algorithms that you can use for AI-based diabetes prediction, such as logistic regression, support vector machines, and random forests.

The specific algorithm that you choose will depend on your specific needs and the characteristics of your dataset.

6. Evaluate the model's performance.

Once you have trained the AI model, you need to evaluate its performance on the testing set. This will give you an idea of how well the model will generalize to new data.

You can evaluate the model's performance using a variety of metrics, such as accuracy, precision, recall, and F1 score.

Once you are satisfied with the model's performance, you can deploy it to production. This may involve integrating the model into a healthcare information system or developing a mobile app that uses the model to predict diabetes risk.

By following these steps, you can build a robust and effective AI-based diabetes prediction system.

PROGRAMS

DATA CLEANING:

```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#read dataset
df=pd.read_csv('../input/diabetes
```

-data-set/diabetes.csv')

EDA

```
df.head()
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
df.columns
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
df.shape
```

```
(768, 9)
```

In [7]:

linkcode

```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.(
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.2
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.7
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.0(
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.0(
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.0(
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.0(
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.0(

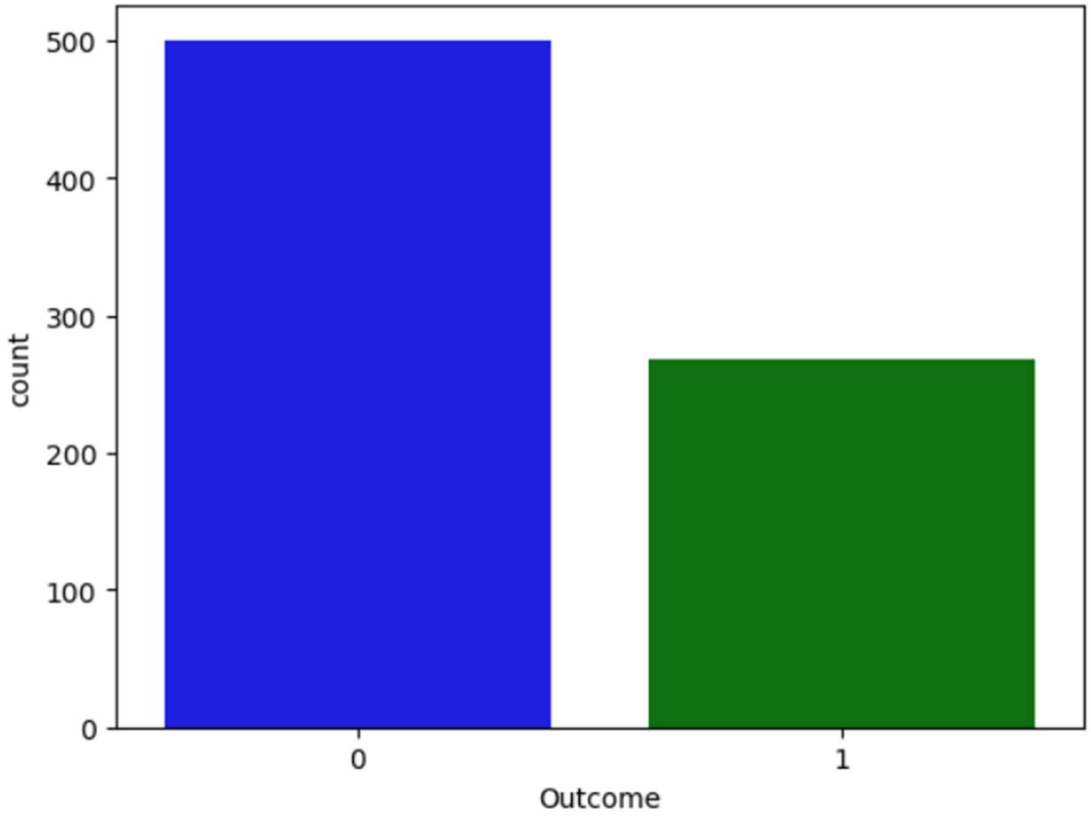
```
#checking null values
df.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
```

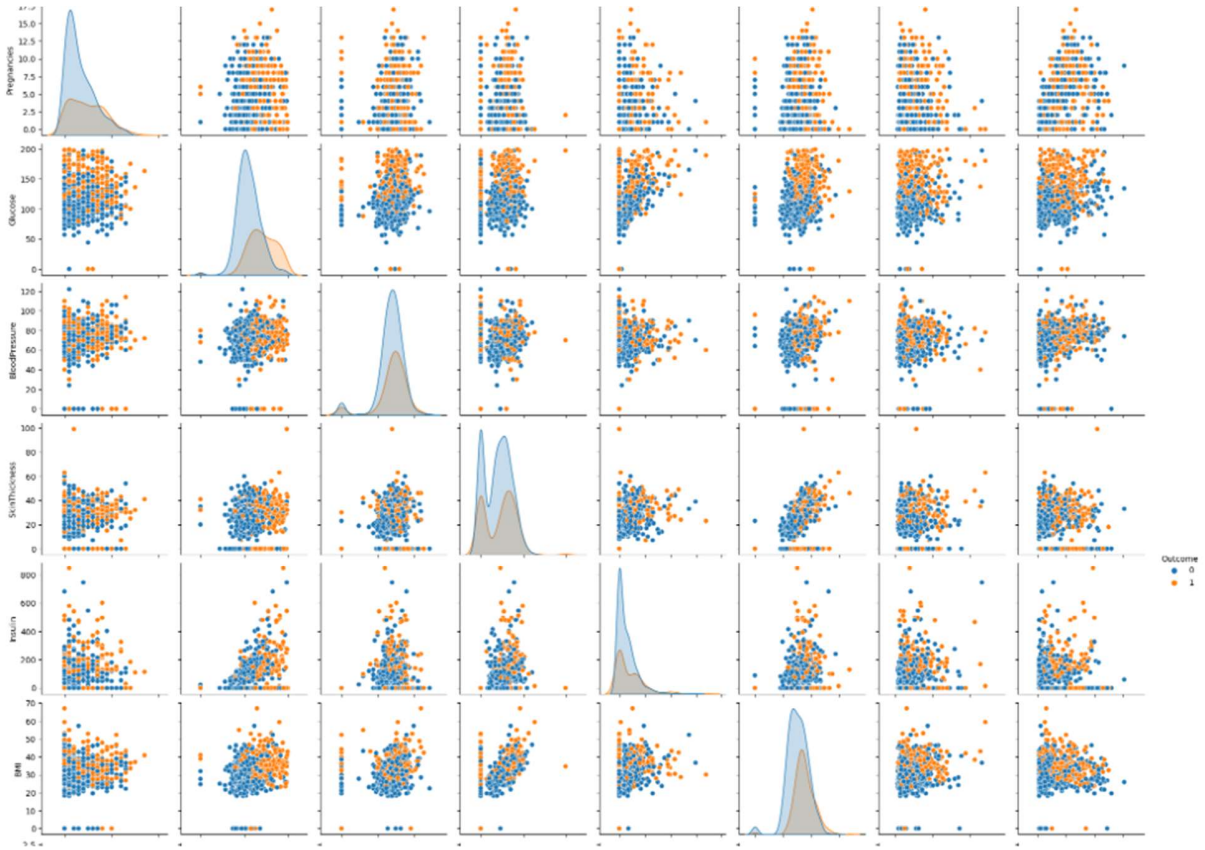
```
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
#countplot
sns.countplot(x='Outcome',data=df,palette=['b','g'])
```

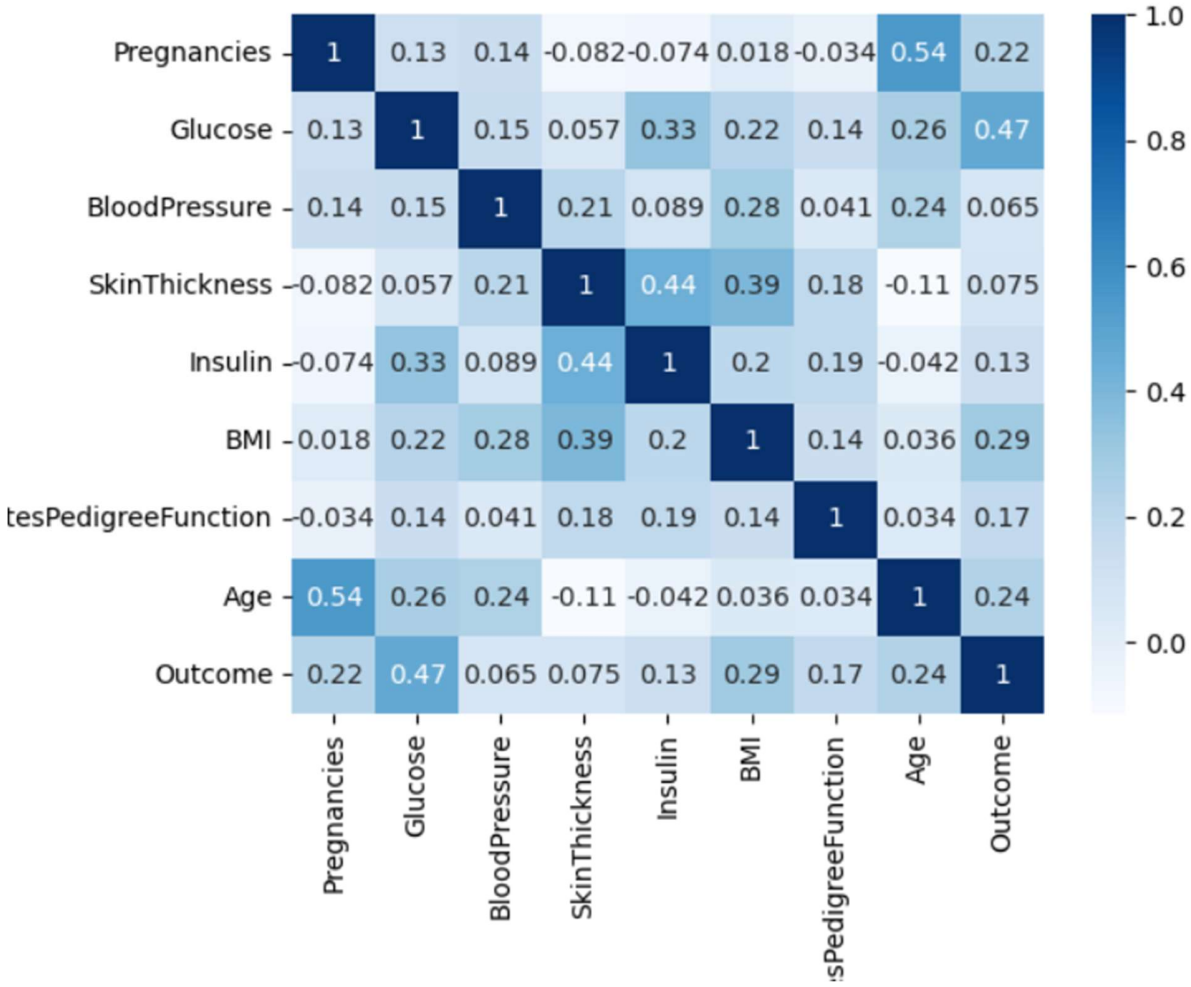
```
<AxesSubplot:xlabel='Outcome', ylabel='count'>
```




```
#pairplot
sns.pairplot(data=df,hue='Outcome')
plt.show()
```




```
#correlation heatmap
sns.heatmap(df.corr(),annot=True,cmap='Blues')
plt.show()
```




```
#replacing zero value with NaN
df_new=df
df_new[["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]] = df_ne
w[["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]].replace(0, np
.NaN)
```

In [13]:

```
#count of Nan
df_new.isnull().sum()
```

Out[13]:

```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
```

```
BMI                       11
DiabetesPedigreeFunction   0
Age                        0
Outcome                    0
dtype: int64
#replacing NaN with mean values
df_new["Glucose"].fillna(df_new["Glucose"].mean(), inplace = True)
df_new["BloodPressure"].fillna(df_new["BloodPressure"].mean(), inplace = True)
df_new["SkinThickness"].fillna(df_new["SkinThickness"].mean(), inplace = True)
df_new["Insulin"].fillna(df_new["Insulin"].mean(), inplace = True)
df_new["BMI"].fillna(df_new["BMI"].mean(), inplace = True)
```

In [15]:

```
#checking null values
df_new.isnull().sum()
```

Out[15]:

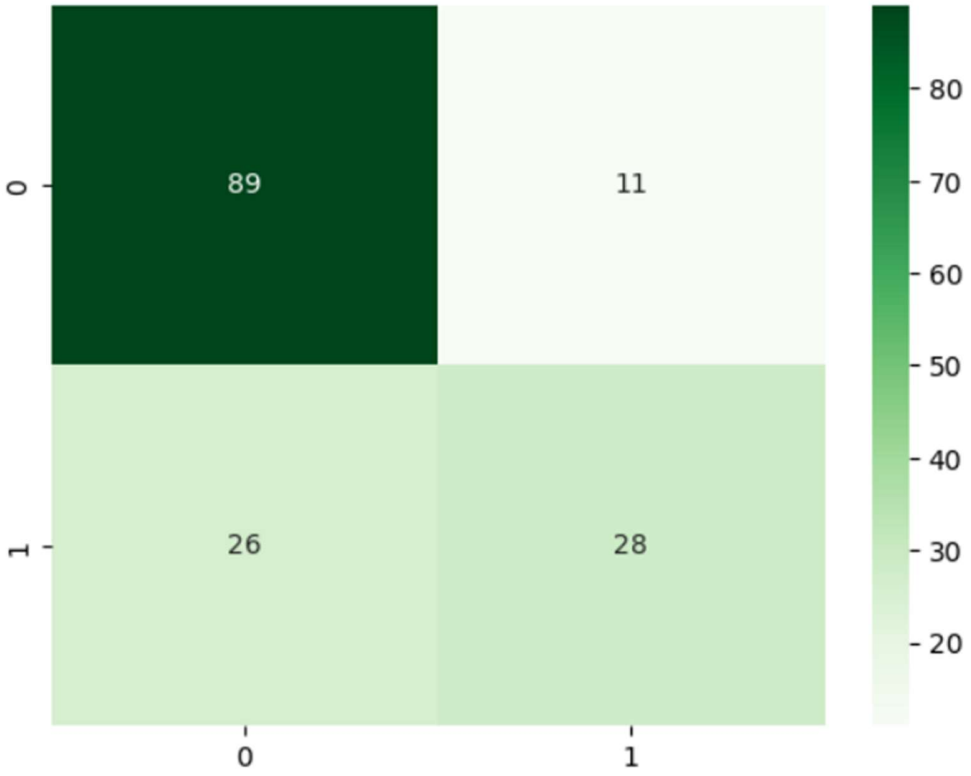
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
y=df_new['Outcome']
X=df_new.drop('Outcome',axis=1)
```

In [17]:

```
linkcode
#spliting X and y
from sklearn.model_selection import train_test_split

X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.20,random_state=0,
stratify=df_new['Outcome'])
from sklearn.linear_model import LogisticRegression

model=LogisticRegression()
model.fit(X_train,Y_train)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:818: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
LogisticRegression()
In [19]:
#get prediction
y_predict=model.predict(X_test)
y_predict
Out[19]:
array([0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0])
In [20]:
linkcode
#confusion matrix
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(Y_test,y_predict)
cm
Out[20]:
array([[89, 11],
       [26, 28]])
In [21]:
linkcode
#heatmap of confusion matrix
sns.heatmap(pd.DataFrame(cm),annot=True, cmap="Greens")
```

```
<AxesSubplot:>
```




```
#accuracy score
from sklearn.metrics import accuracy_score

accuracy=accuracy_score(Y_test,y_predict)
print("Accuracy : ",round(accuracy,2)*100,'%')
```

ACCURACY:

76.0%

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 can 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.

Conclusion

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:

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.

Step-by-Step Procedure:

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.

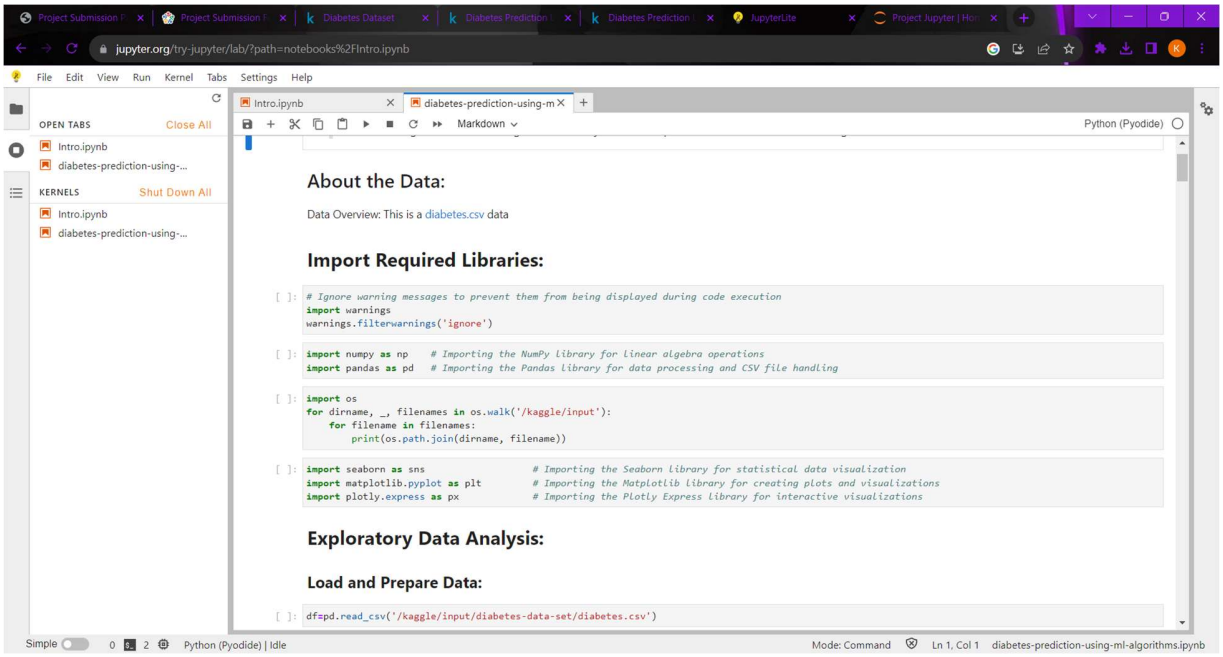
Here is an example of how to train a random forest model for diabetes prediction using the Python programming language:

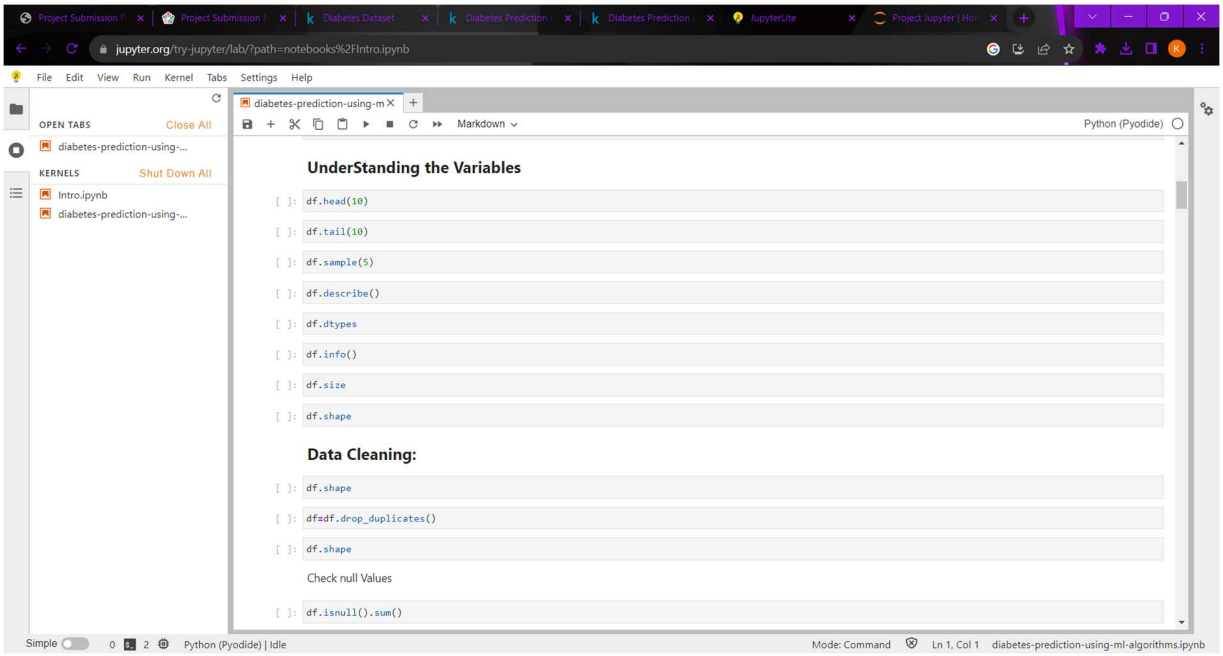
```
Python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# 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)
```

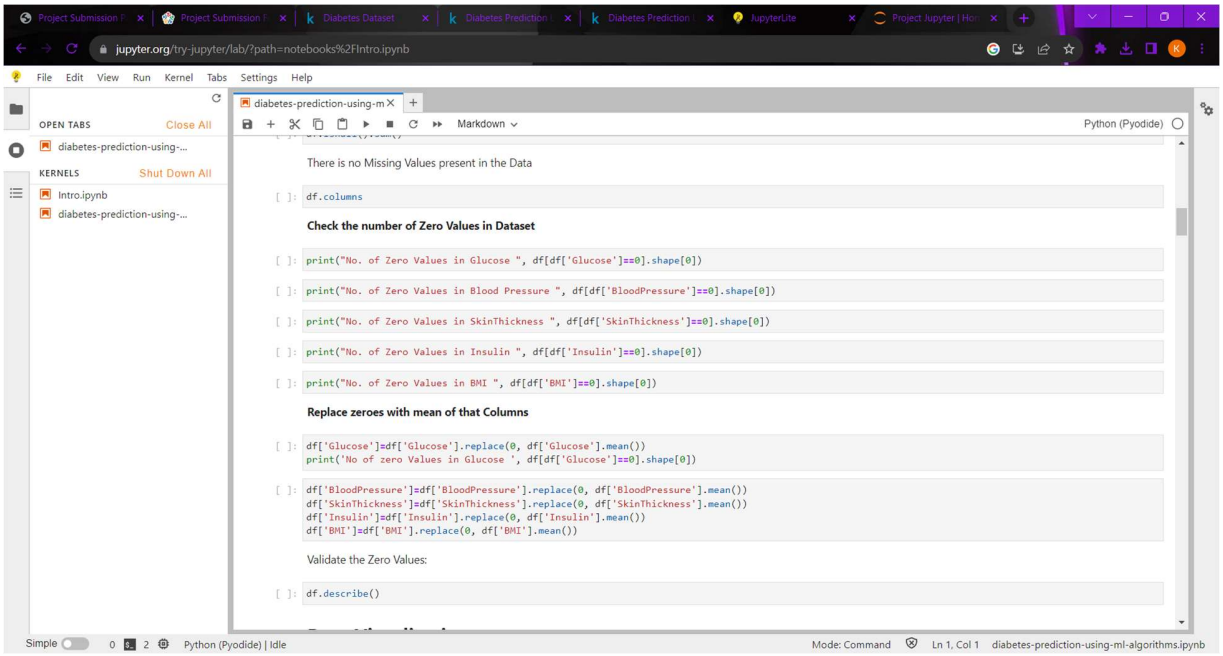
```
# Create the random forest model
model = RandomForestClassifier()

# 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
```

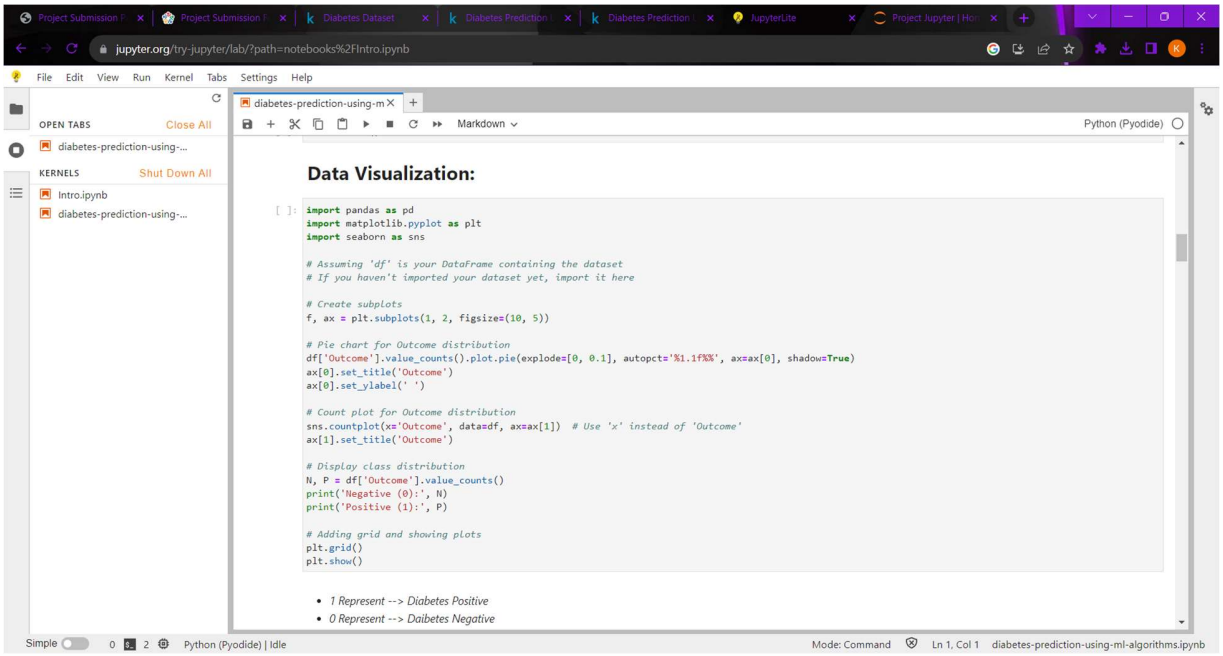
This is just a basic example, and you may need to adjust the parameters of the random forest model or try other algorithms to get the best results.

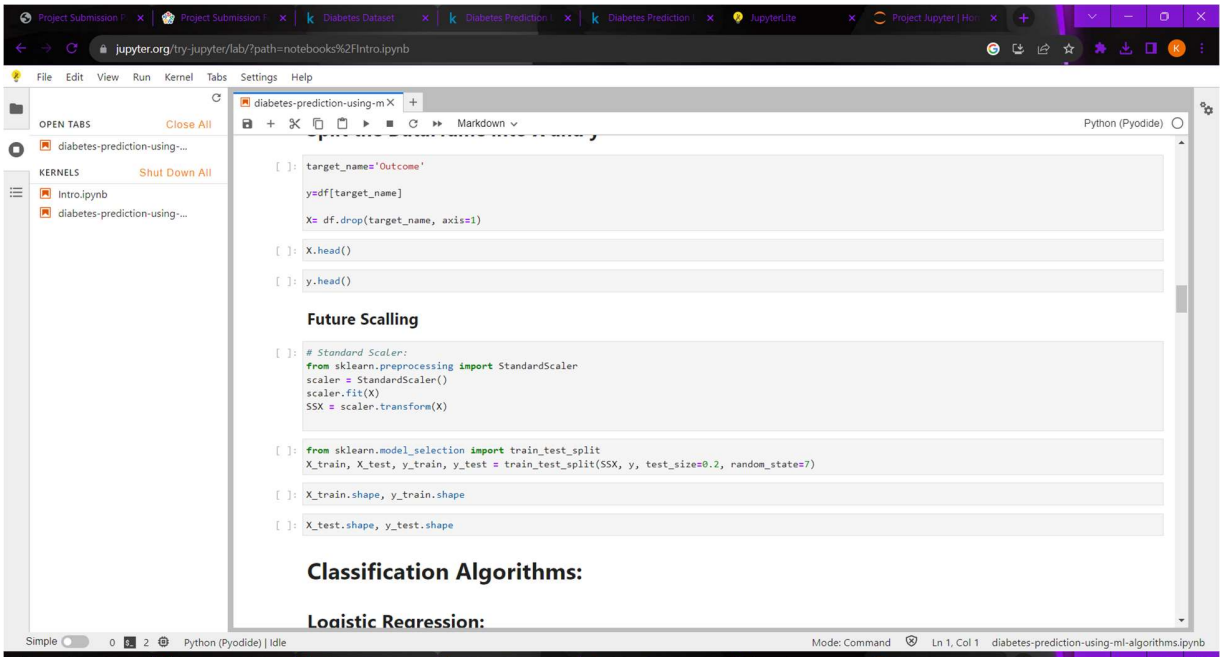
PROGRAMS

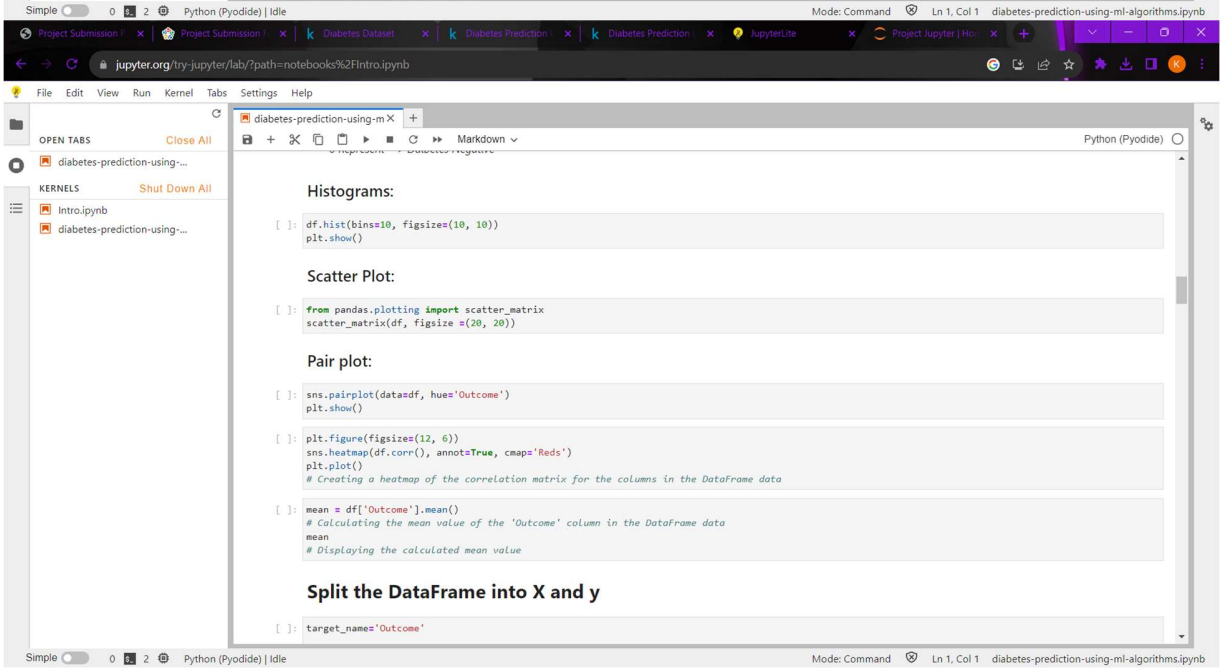


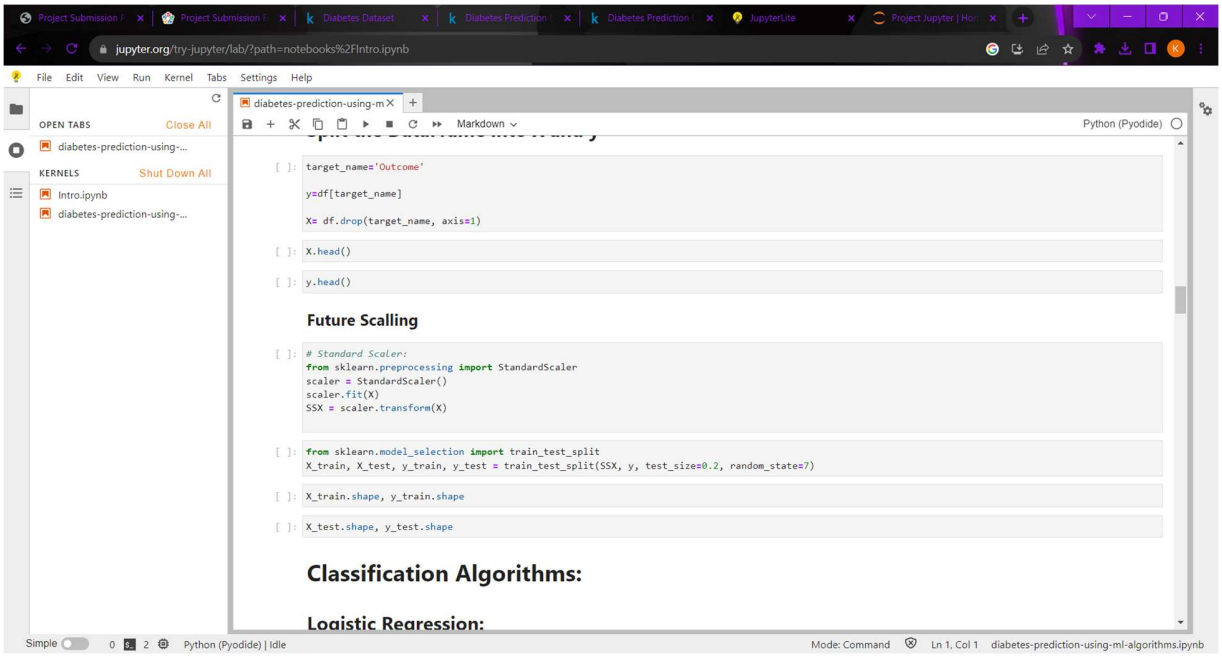

jupyter.org/try-jupyter/lab/?path=notebooks%2FIntro.ipynb
File Edit View Run Kernel Tabs Settings Help
OPEN TABS Close All
diabetes-prediction-using-...
KERNELS Shut Down All
Intro.ipynb
diabetes-prediction-using-...
diabetes-prediction-using-m
Markdown
Python (Pyodide)
UnderStanding the Variables
df.head(10)
df.tail(10)
df.sample(5)
df.describe()
df.dtypes
df.info()
df.size
df.shape
Data Cleaning:
df.shape
df=df.drop_duplicates()
df.shape
Check null Values
df.isnull().sum()
Simple
Python (Pyodide) | Idle
Mode: Command
Ln 1, Col 1
diabetes-prediction-using-ml-algorithms.ipynb

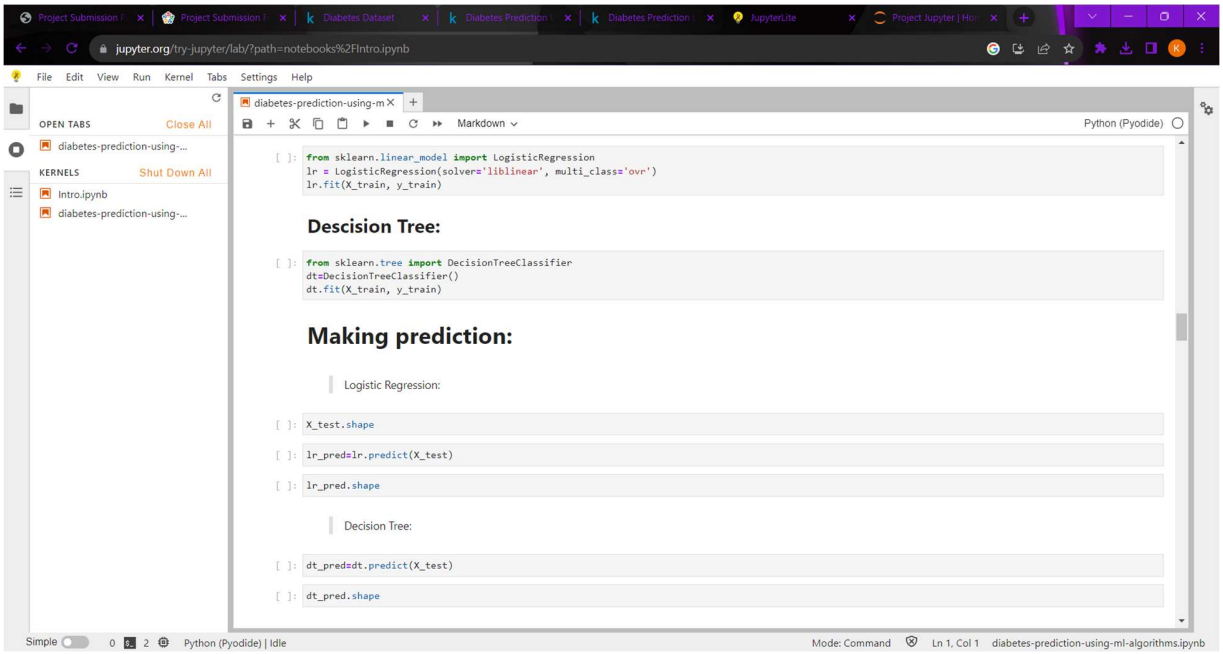

jupyter.org/try-jupyter/lab/?path=notebooks%2FIntro.ipynb
File Edit View Run Kernel Tabs Settings Help
OPEN TABS Close All
diabetes-prediction-using-...
KERNELS Shut Down All
Intro.ipynb
diabetes-prediction-using-...
diabetes-prediction-using-m
Markdown
Python (Pyodide)
There is no Missing Values present in the Data
df.columns
Check the number of Zero Values in Dataset
print("No. of Zero Values in Glucose ", df[df['Glucose']==0].shape[0])
print("No. of Zero Values in Blood Pressure ", df[df['BloodPressure']==0].shape[0])
print("No. of Zero Values in SkinThickness ", df[df['SkinThickness']==0].shape[0])
print("No. of Zero Values in Insulin ", df[df['Insulin']==0].shape[0])
print("No. of Zero Values in BMI ", df[df['BMI']==0].shape[0])
Replace zeroes with mean of that Columns
df['Glucose']=df['Glucose'].replace(0, df['Glucose'].mean())
print('No of zero Values in Glucose ', df[df['Glucose']==0].shape[0])
df['BloodPressure']=df['BloodPressure'].replace(0, df['BloodPressure'].mean())
df['SkinThickness']=df['SkinThickness'].replace(0, df['SkinThickness'].mean())
df['Insulin']=df['Insulin'].replace(0, df['Insulin'].mean())
df['BMI']=df['BMI'].replace(0, df['BMI'].mean())
Validate the Zero Values:
df.describe()
Simple 0 2 Python (Pyodide) | Idle
Mode: Command Ln 1, Col 1 diabetes-prediction-using-ml-algorithms.ipynb

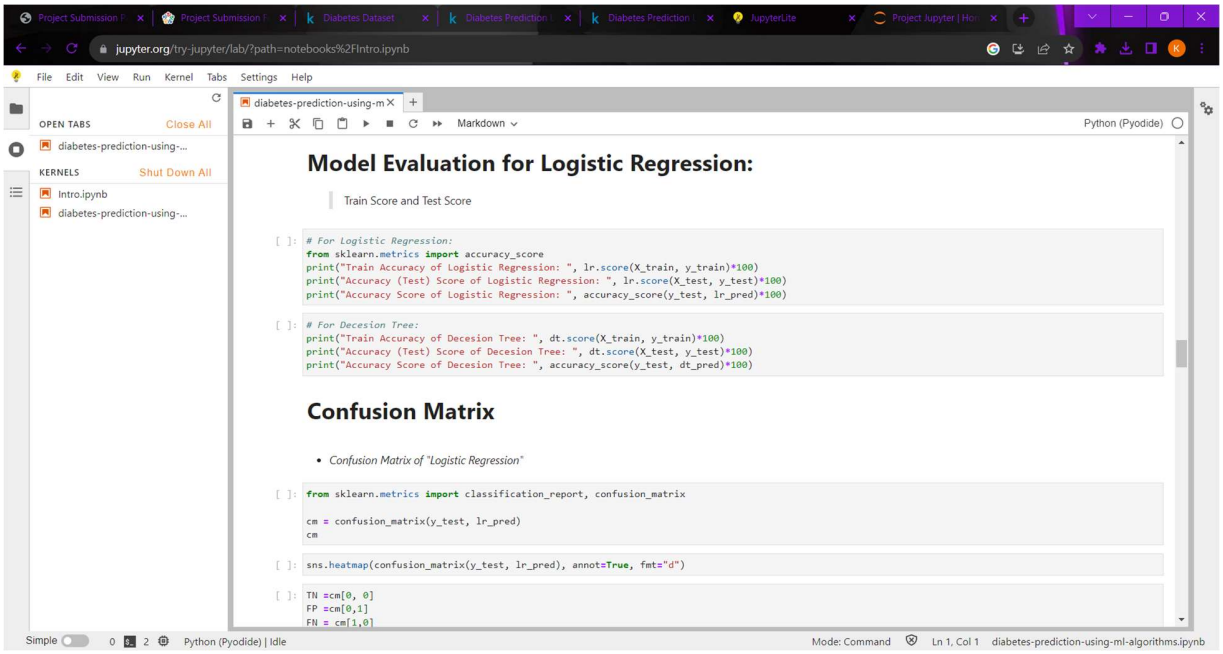

jupyter.org/try-jupyter/lab/?path=notebooks%2FIntro.ipynb
File Edit View Run Kernel Tabs Settings Help
OPEN TABS Close All
diabetes-prediction-using-...
KERNELS Shut Down All
Intro.ipynb
diabetes-prediction-using-...
diabetes-prediction-using-m
Markdown
Python (Pyodide)
Data Visualization:
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Assuming 'df' is your DataFrame containing the dataset
# If you haven't imported your dataset yet, import it here
# Create subplots
f, ax = plt.subplots(1, 2, figsize=(10, 5))
Pie chart for Outcome distribution
df['Outcome'].value_counts().plot.pie(explode=[0, 0.1], autopct='%1.1f%%', ax=ax[0], shadow=True)
ax[0].set_title('Outcome')
ax[0].set_ylabel(' ')
Count plot for Outcome distribution
sns.countplot(x='Outcome', data=df, ax=ax[1]) # Use 'x' instead of 'Outcome'
ax[1].set_title('Outcome')
Display class distribution
N, P = df['Outcome'].value_counts()
print('Negative (0):', N)
print('Positive (1):', P)
# Adding grid and showing plots
plt.grid()
plt.show()
1 Represent --> Diabetes Positive
0 Represent --> Daibetes Negative
Simple 0 2 Python (Pyodide) | Idle
Mode: Command Ln 1, Col 1 diabetes-prediction-using-ml-algorithms.ipynb

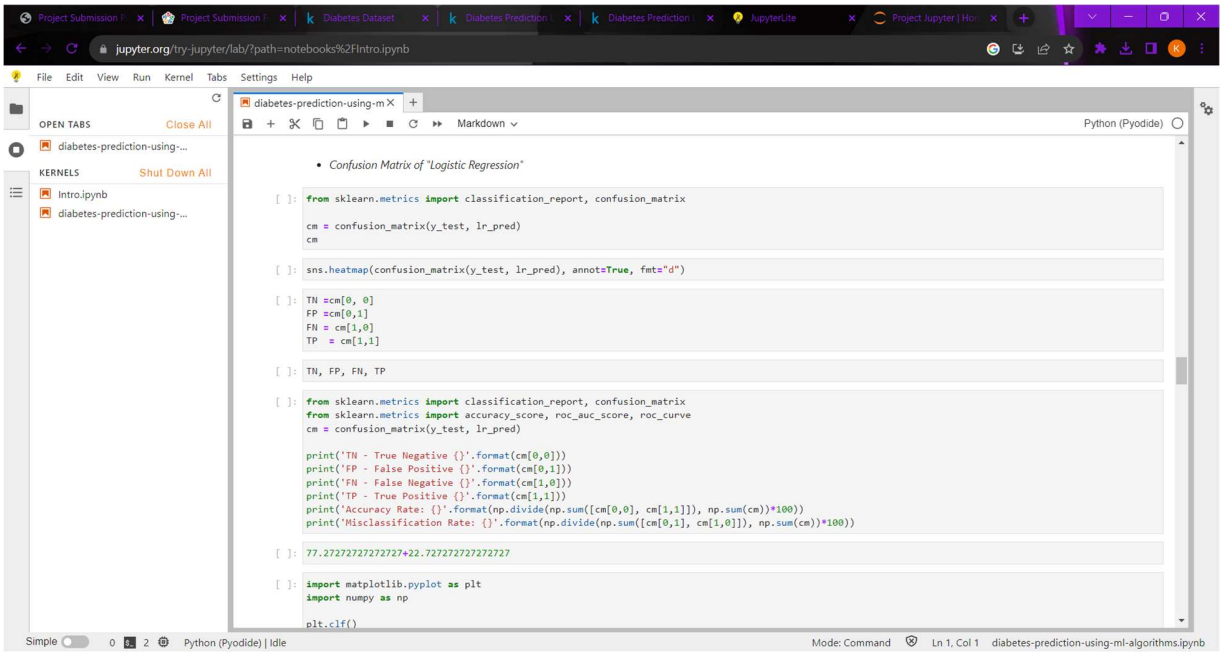

jupyter.org/try-jupyter/lab/?path=notebooks%2FIntro.ipynb
File Edit View Run Kernel Tabs Settings Help
OPEN TABS Close All
diabetes-prediction-using-...
KERNELS Shut Down All
Intro.ipynb
diabetes-prediction-using-...
diabetes-prediction-using-m
Markdown
Python (Pyodide)
target_name='Outcome'
y=df[target_name]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

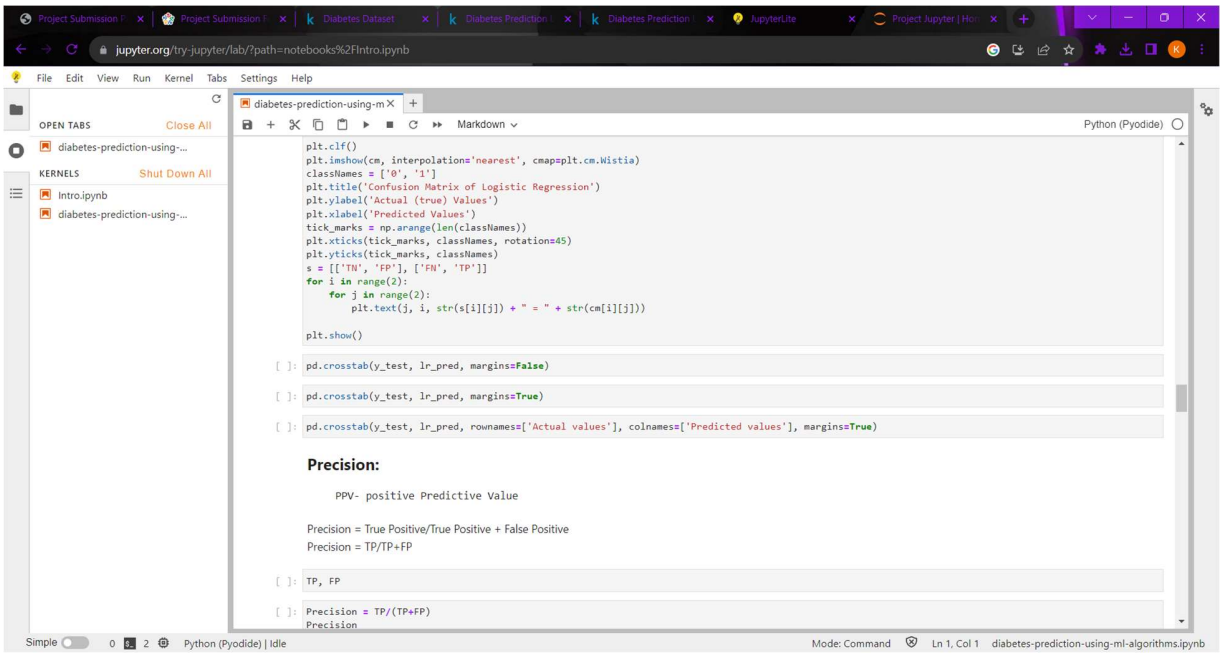

jupyter.org/try-jupyter/lab/?path=notebooks%2FIntro.ipynb
File Edit View Run Kernel Tabs Settings Help
OPEN TABS Close All
diabetes-prediction-using-...
KERNELS Shut Down All
Intro.ipynb
diabetes-prediction-using-...
diabetes-prediction-using-m
Markdown
Python (Pyodide)
Histograms:
df.hist(bins=10, figsize=(10, 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'
Simple
Python (Pyodide) | Idle
Mode: Command
Ln 1, Col 1
diabetes-prediction-using-ml-algorithms.ipynb

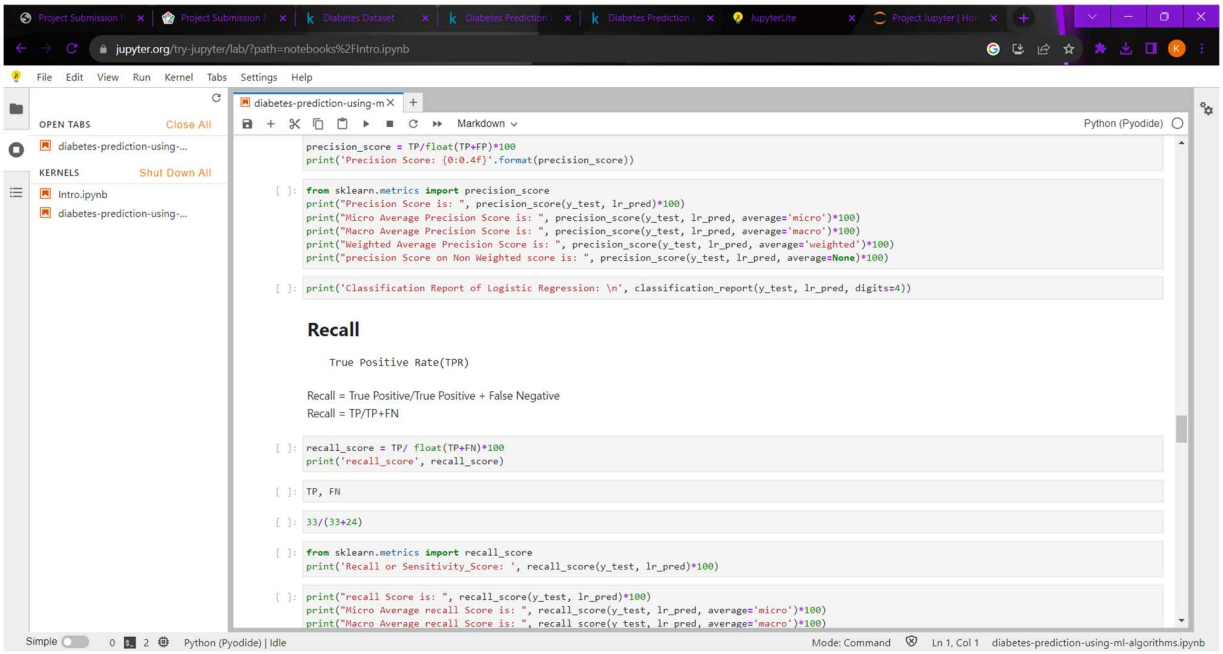

jupyter.org/try-jupyter/lab/?path=notebooks%2FIntro.ipynb
File Edit View Run Kernel Tabs Settings Help
OPEN TABS Close All
diabetes-prediction-using-...
KERNELS Shut Down All
Intro.ipynb
diabetes-prediction-using-...
diabetes-prediction-using-m
Markdown
Python (Pyodide)
target_name='Outcome'
y=df[target_name]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

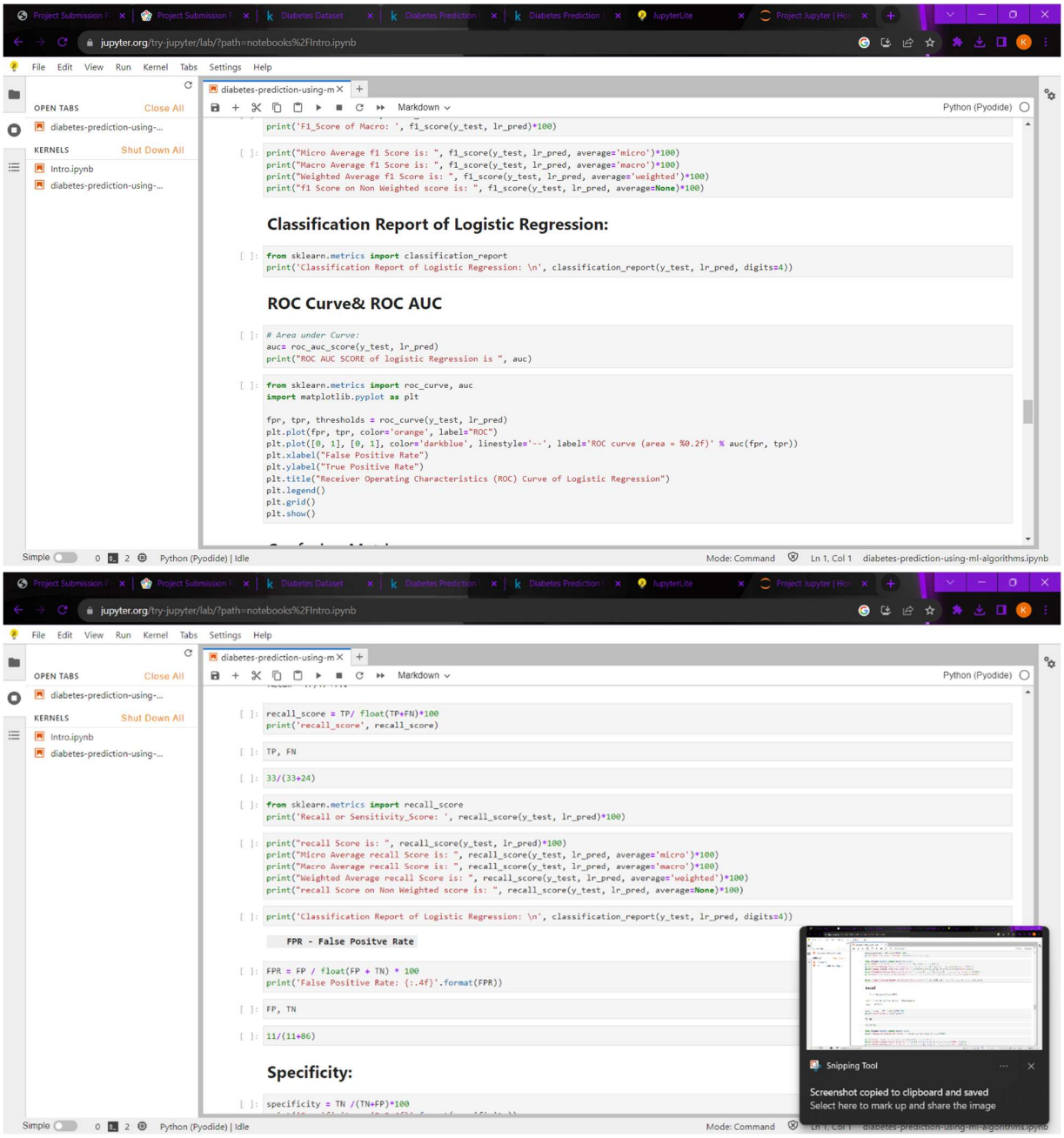

jupyter.org/try-jupyter/lab/?path=notebooks%2FIntro.ipynb
File Edit View Run Kernel Tabs Settings Help
OPEN TABS Close All
diabetes-prediction-using-...
KERNELS Shut Down All
Intro.ipynb
diabetes-prediction-using-...
diabetes-prediction-using-m
Markdown
Python (Pyodide)
from sklearn.linear_model import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


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

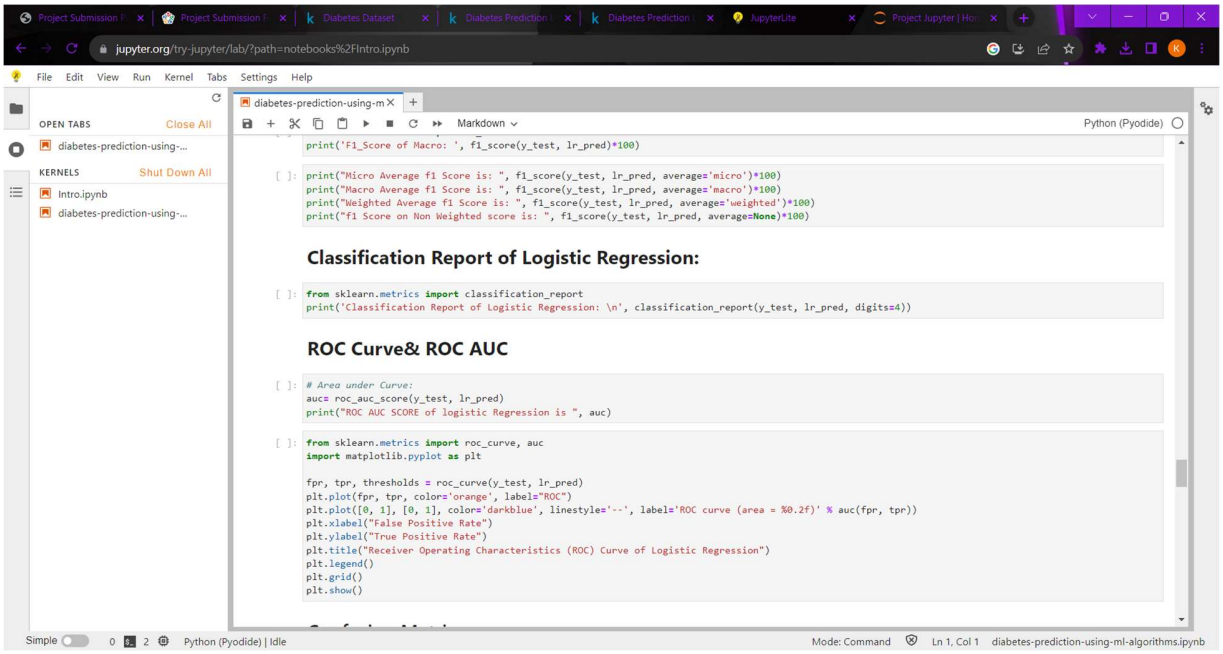

jupyter.org/try-jupyter/lab/?path=notebooks%2FIntro.ipynb
File Edit View Run Kernel Tabs Settings Help
OPEN TABS Close All
diabetes-prediction-using-...
KERNELS Shut Down All
Intro.ipynb
diabetes-prediction-using-...
diabetes-prediction-using-m
Markdown
Python (Pyodide)
Confusion Matrix of "Logistic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

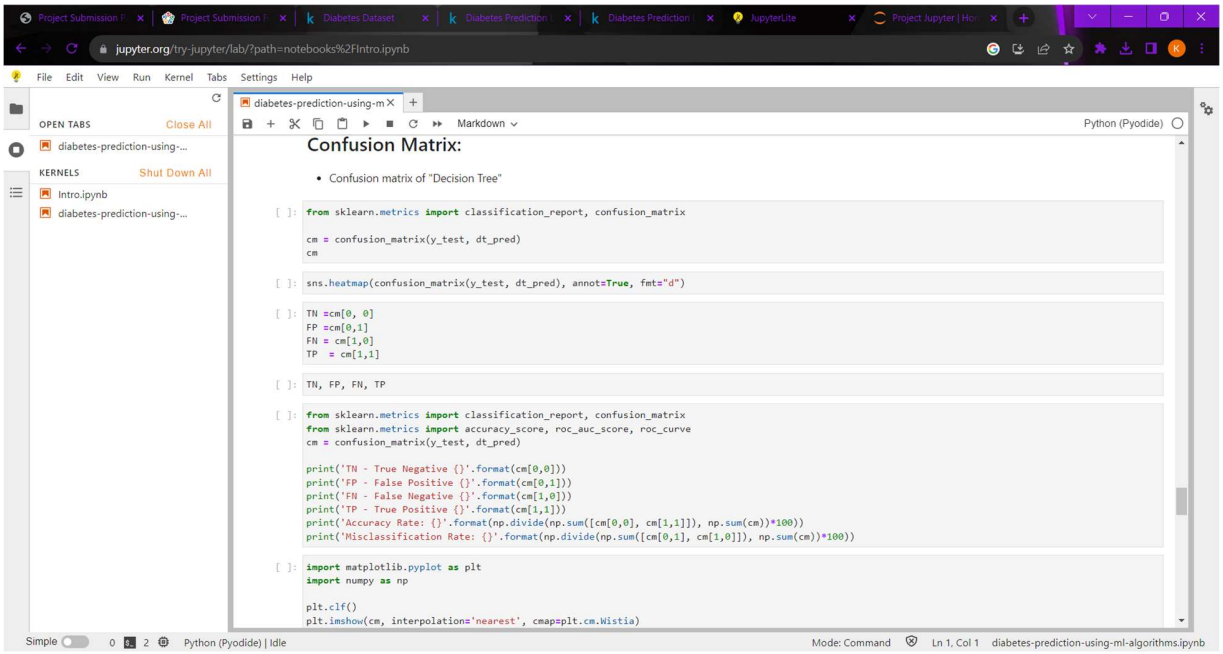

jupyter.org/try-jupyter/lab/?path=notebooks%2FIntro.ipynb
File Edit View Run Kernel Tabs Settings Help
OPEN TABS Close All
diabetes-prediction-using-...
KERNELS Shut Down All
Intro.ipynb
diabetes-prediction-using-...
diabetes-prediction-using-m
Markdown
Python (Pyodide)
plt.clf()
plt.imshow(cm, interpolation='nearest',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

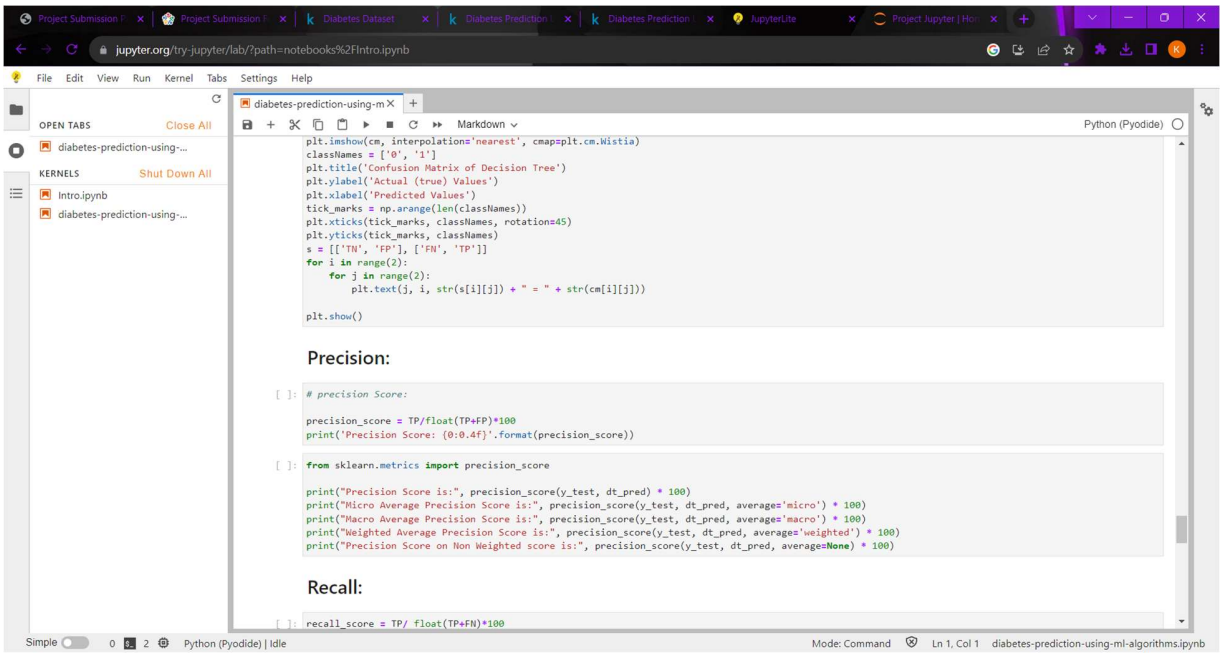

jupyter.org/try-jupyter/lab/?path=notebooks%2FIntro.ipynb
File Edit View Run Kernel Tabs Settings Help
OPEN TABS Close All
diabetes-prediction-using-...
KERNELS Shut Down All
Intro.ipynb
diabetes-prediction-using-...
diabetes-prediction-using-m
Markdown
Python (Pyodide)
precision_score = TP/float(TP+FP)*100
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)
Simple
Python (Pyodide) | Idle
Mode: Command
Ln 1, Col 1
diabetes-prediction-using-ml-algorithms.ipynb

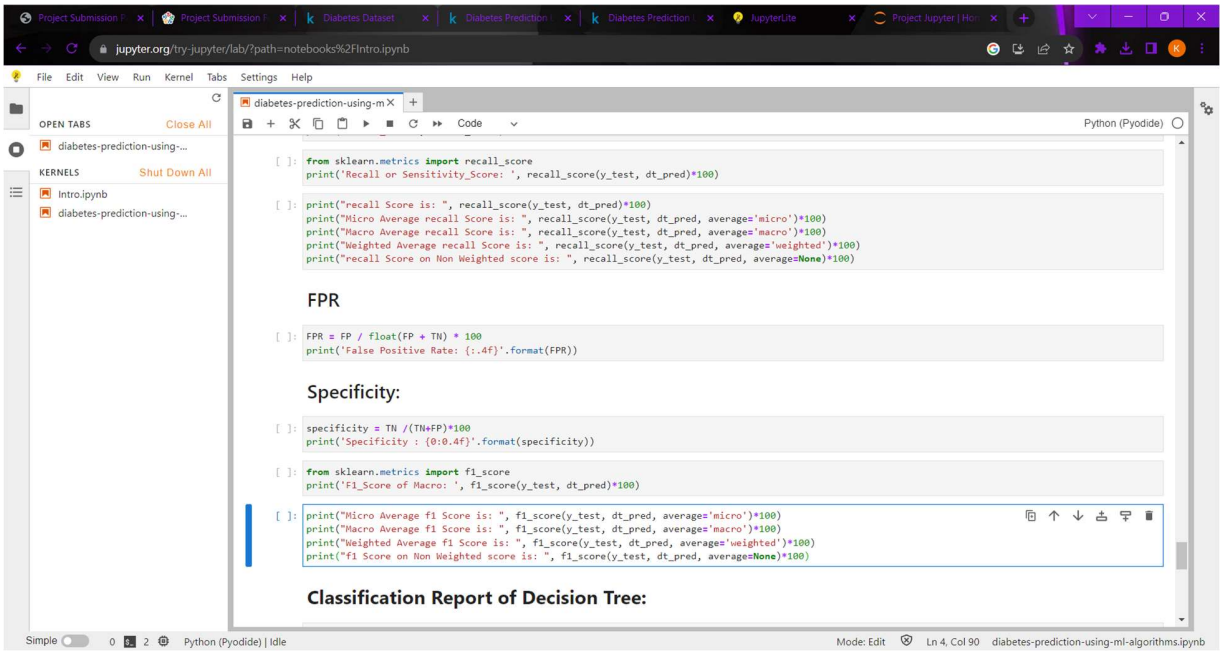

jupyter.org/try-jupyter/lab/?path=notebooks%2FIntro.ipynb
File Edit View Run Kernel Tabs Settings Help
OPEN TABS Close All
diabetes-prediction-using-...
KERNELS Shut Down All
Intro.ipynb
diabetes-prediction-using-...
diabetes-prediction-using-m
Markdown
Python (Pyodide)
print('F1_Score of Macro: ', f1_score(y_test, 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()
Simple
Python (Pyodide) | Idle
Mode: Command
Ln 1, Col 1
diabetes-prediction-using-ml-algorithms.ipynb

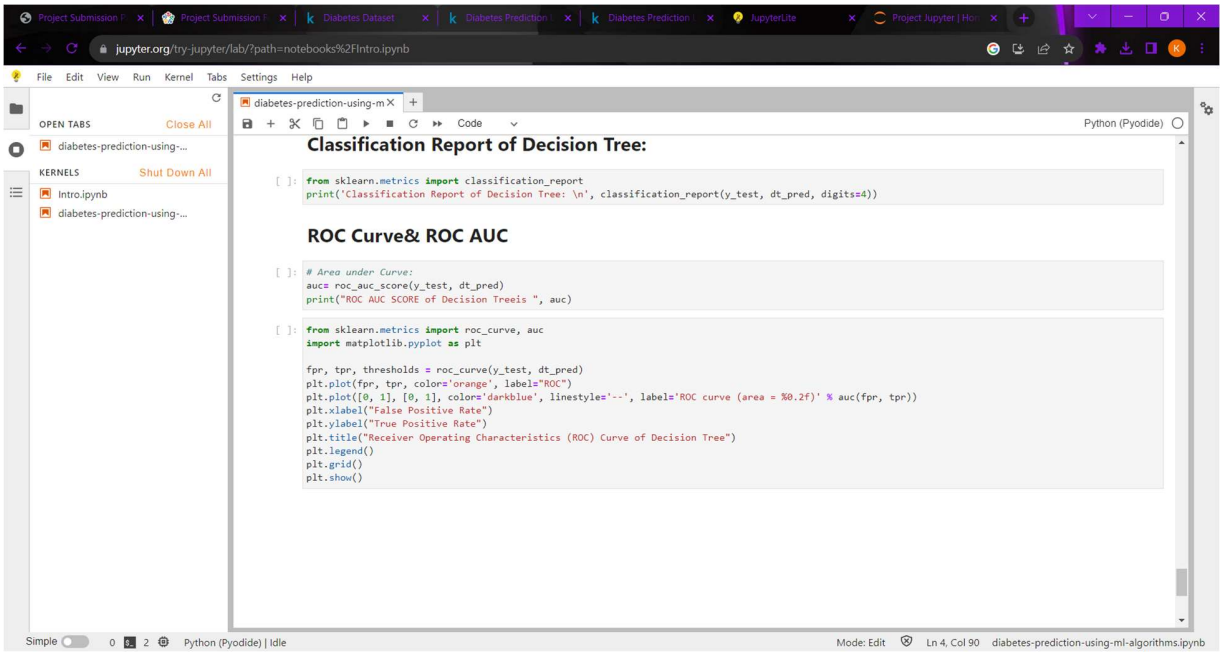
jupyter.org/try-jupyter/lab/?path=notebooks%2FIntro.ipynb
File Edit View Run Kernel Tabs Settings Help
OPEN TABS Close All
diabetes-prediction-using-...
KERNELS Shut Down All
Intro.ipynb
diabetes-prediction-using-...
diabetes-prediction-using-m
Markdown
Python (Pyodide)
recall_score = TP/ float(TP+FN)*100
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
Simple
Python (Pyodide) | Idle
Mode: Command


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

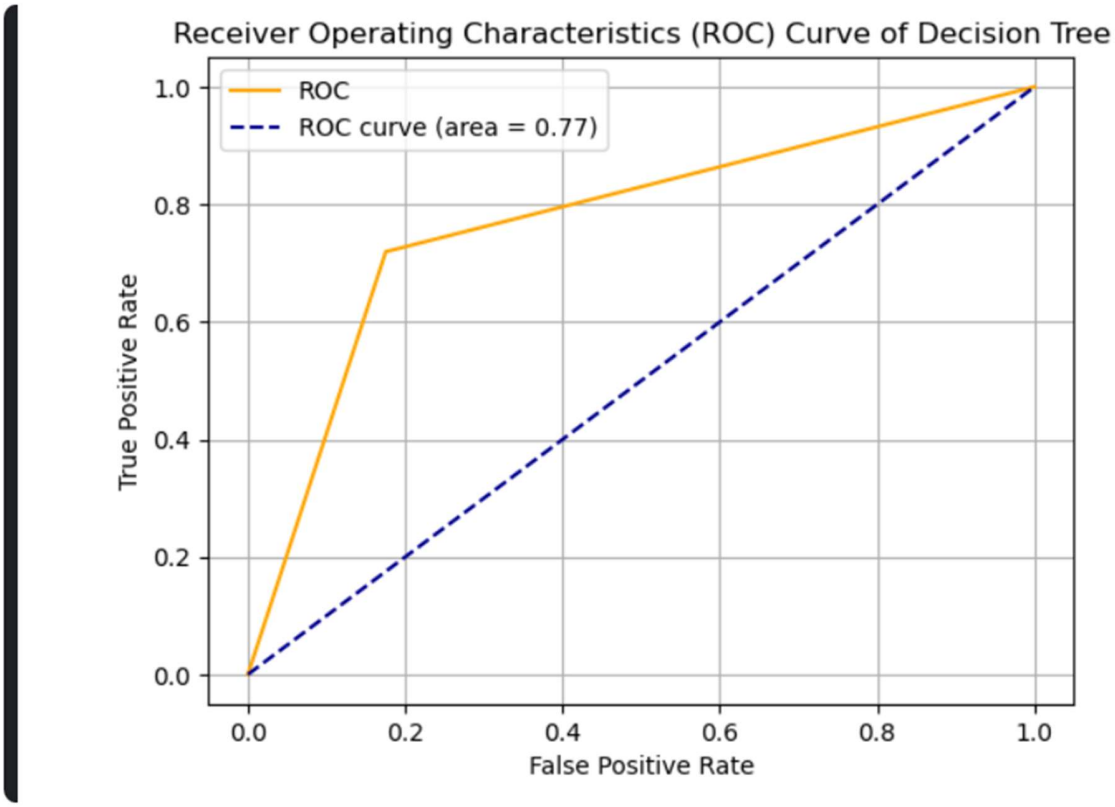

jupyter.org/try-jupyter/lab/?path=notebooks%2FIntro.ipynb
File Edit View Run Kernel Tabs Settings Help
OPEN TABS Close All
diabetes-prediction-using-...
KERNELS Shut Down All
Intro.ipynb
diabetes-prediction-using-...
diabetes-prediction-using-m
Markdown
Python (Pyodide)
Confusion Matrix:
Confusion matrix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


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


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



ACCURACY WITH GRAP 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

ADVANTAGES & DIADVANTAGES:

Advantages of AI-based diabetes prediction:

- **Early detection:**
  AI-based diabetes prediction can help to detect diabetes earlier, when it is more treatable and manageable.

- **Personalized treatment:**
  AI can help to develop personalized treatment plans for patients with diabetes, based on their individual risk factors and needs.

- **Improved outcomes:**
  AI can help to improve the overall health outcomes for patients with diabetes, by reducing the risk of complications and helping patients to live longer, healthier lives.

- **Increased access to healthcare:**
  AI-based diabetes prediction can be used to screen large populations for diabetes risk factors, even in remote or underserved areas. This can help to increase access to healthcare for people who may not otherwise have access to diabetes screening and prevention services.

Benefits of AI-based diabetes prediction:

- **Reduced healthcare costs**:
  By detecting diabetes earlier and preventing complications, AI-based diabetes prediction can help to reduce the overall cost of healthcare for patients with diabetes.

- **Improved quality of life**:

By helping people to manage their diabetes more effectively, AI-based diabetes prediction can help to improve their quality of life.

- **Increased productivity**:

  By reducing the risk of complications from diabetes, AI-based diabetes prediction can help people to stay in the workforce and be more productive.

Disadvantages of AI-based diabetes prediction:

- **Privacy and security concerns:**

  AI-based diabetes prediction systems collect and process sensitive patient data. It is important to ensure that this data is protected from unauthorized access and use.

- **Transparency and accountability:**

  AI-based diabetes prediction systems should be transparent about how they work and how they make decisions. This is important for building trust with patients and caregivers.

- **Equity and inclusion:**

  AI-based diabetes prediction systems should be designed and evaluated in a way that ensures that they are fair and equitable for all patients, regardless of race, ethnicity, socioeconomic status, or other factors.

- **Potential for bias:**

  AI models can be biased, depending on the data that they are trained on. It is important to be aware of this potential bias and to take steps to mitigate it. Overall, the advantages of AI-based diabetes prediction outweigh the disadvantages.

  AI-based diabetes prediction has the potential to revolutionize the way we diagnose and manage diabetes. However, it is important to be aware of the potential disadvantages and to take steps to mitigate them.

CONCLUSION:

AI-based diabetes prediction has the potential to revolutionize the way we diagnose and manage diabetes. AI models can be trained to identify individuals at high risk of developing diabetes, diagnose diabetes early, and assess a patient's risk of complications. This information can be used to develop personalized treatment plans for patients and improve their overall health outcomes.

AI-based diabetes prediction is still in its early stages of development, but it has already made significant progress. AI models have been shown to be accurate in predicting diabetes risk and complications. AI-based diabetes prediction systems are also being deployed in clinical settings to help healthcare professionals diagnose and manage diabetes.

Here are some of the key challenges and opportunities for AI-based diabetes prediction:

Challenges:

- **Data privacy and security:**
  AI-based diabetes prediction systems collect and process sensitive patient data. It is important to ensure that this data is protected from unauthorized access and use.

- **Transparency and accountability**:
  AI-based diabetes prediction systems should be transparent about how they work and how they make decisions. This is important for building trust with patients and caregivers.

- **Equity and inclusion:**
  AI-based diabetes prediction systems should be designed and evaluated in a way that ensures that they are fair and equitable for all patients, regardless of race, ethnicity, socioeconomic status, or other factors.

- **Potential for bias**:
  AI models can be biased, depending on the data that they are trained on. It is important to be aware of this potential bias and to take steps to mitigate it.

Opportunities:

- **Improved accuracy and personalization**:

AI models can be trained on large datasets of patient data to improve their accuracy and personalization. This can lead to earlier detection and more effective treatment of diabetes.

- **Integration with other technologies:**

  AI-based diabetes prediction systems can be integrated with other technologies, such as wearable devices and electronic health records. This can allow for more continuous and personalized monitoring of a patient's risk of diabetes and diabetes complications.

- **Expanded access to healthcare:**

  AI-based diabetes prediction systems can be used to screen large populations for diabetes risk factors, even in remote or underserved areas. This can help to expand access to healthcare for people who may not otherwise have access to diabetes screening and prevention services.

Overall, AI-based diabetes prediction has the potential to significantly improve the lives of people with diabetes. By addressing the challenges and seizing the opportunities, we can create a future where AI is used to help everyone live a longer, healthier life.