Project group number - 04

Diabetes classification

By team: 1.) Lokesh Ram

2.) Pranay Singh

3.) Musthaffa

4.) Mayank

5.) Raja Sagar

6.) Yash Shetty

Table of Contents-

1.) BACKGROUND-

Here first of all we have a data set which is already given to us and it has been attached with the report as well.

This project discusses the limitations of traditional classification methods and the need for more refined approaches. The section emphasizes the advancements in machine learning and data mining techniques for diabetes classification, with a focus on their potential benefits such as identifying subtypes, predicting disease progression, and personalizing treatment. It mentions related research and sets clear objectives for the project, emphasizing its significance in advancing diabetes research and improving patient care.

This is a classification problem and to solve it we have implemented two main methods which are:

- Logistic regression
- Decision tree

We here are using some advanced data science and Machine learning libraries which include:

- Pandas as pd
- Numpy as np

- Matplotlib.pyplot as plt
- Graphviz
- sklearn *

2.) Objective and Scope:

Objective: The objective of this diabetic classification project is to develop an accurate and reliable machine learning model that can classify diabetic patients into different subtypes based on their clinical and genetic characteristics. The model aims to enhance the understanding of diabetes heterogeneity, enable personalized treatment strategies, and improve patient outcomes.

Scope: The scope of this project encompasses the collection and preprocessing of a comprehensive dataset containing clinical and genetic information of diabetic patients. Various machine learning algorithms will be explored and evaluated for their effectiveness in diabetic classification. The model will be trained and validated using appropriate techniques, and its performance will be assessed based on metrics such as accuracy, precision, recall, and F1-score. The project focuses on developing a classification model specifically for type 2 diabetes, utilizing both clinical and genetic features to achieve a robust and generalizable solution.

3.) Tools Used:

1. Python: Python is a widely used programming language for machine learning projects due to its extensive libraries and frameworks.

2. Jupyter Notebook: It appears that the project you linked is implemented in a Jupyter Notebook (.ipynb file), which is an interactive coding environment that allows for data exploration, code execution, and result visualization.

3. Scikit-learn: Scikit-learn is a popular machine learning library in Python that provides a wide range of algorithms and tools for classification, regression, and other tasks.

4. Pandas: Pandas is a data manipulation library that offers data structures and functions for efficiently handling and analyzing structured data.

5. NumPy: NumPy is a fundamental library for numerical computing in Python. It provides support for large, multi-dimensional arrays and a collection of mathematical functions to operate on these arrays.

6. Matplotlib and Seaborn: These libraries are commonly used for data visualization, allowing you to create plots, charts, and graphs to explore.

7. Model Evaluation Metrics: Depending on the project, we used various evaluation metrics like accuracy, precision to assess the performance of the classification model.

4.) Description:

So firstly we load the csv file in a pandas datafile, Which gets referred to as ‘df’ The csv file has 770 rows and 09 columns.

```
csv_file='diabetes.csv'
df = pd.read_csv(csv_file)
```

```
df.iloc[:,:-1].describe().style.background_gradient(axis=0,cmap='Accent')
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000

```
df.head(10).style.bar(axis=0)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.600000	0.627000	50	1
1	1	85	66	29	0	26.600000	0.351000	31	0
2	8	183	64	0	0	23.300000	0.672000	32	1
3	1	89	66	23	94	28.100000	0.167000	21	0
4	0	137	40	35	168	43.100000	2.288000	33	1
5	5	116	74	0	0	25.600000	0.201000	30	0
6	3	78	50	32	88	31.000000	0.248000	26	1
7	10	115	0	0	0	35.300000	0.134000	29	0
8	2	197	70	45	543	30.500000	0.158000	53	1
9	8	125	96	0	0	0.000000	0.232000	54	1

5.) Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is a crucial step in any data analysis project as it enables data scientists to understand the data, identify patterns,relationships, and anomalies, and prepare the data for modelling.

```
print("dimension of diabetes data: {}".format(df.shape))
```

```
dimension of diabetes data: (768, 9)
```

```
df.info()
```

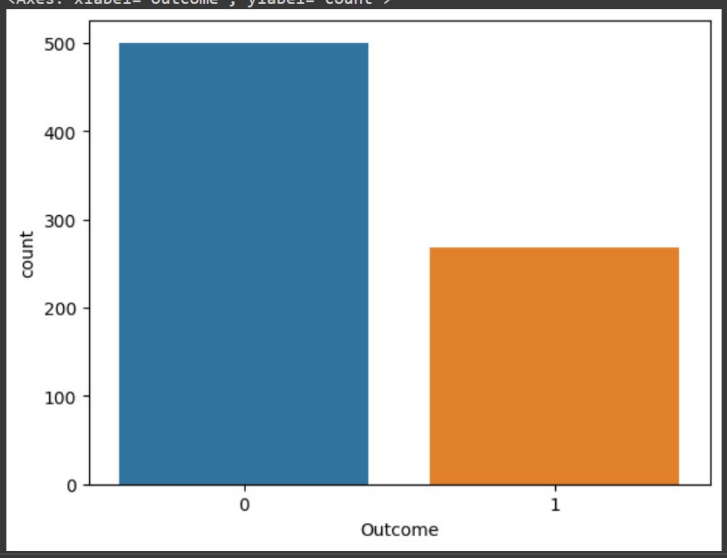
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   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
```

- Dataset is small and well labeled. There are no null values present.
- very suitable to supervised machine learning formulation.
- This is a binary classification problem, where we have 2 classes in the target **(y)** (i.e. `df['Outcome']`) and the medical conditions can be used as the feature (**X**).

Variable Analysis and Plotting according to their data.

```
[ ]  sns.countplot(x=df['Outcome'])
```

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



Preparing data for machine learning

```
X = df.iloc[:,:-1].values   ##features selection
y = df.iloc[:,-1].values    ## target selection
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y, stratify=y, random_state=27)
```

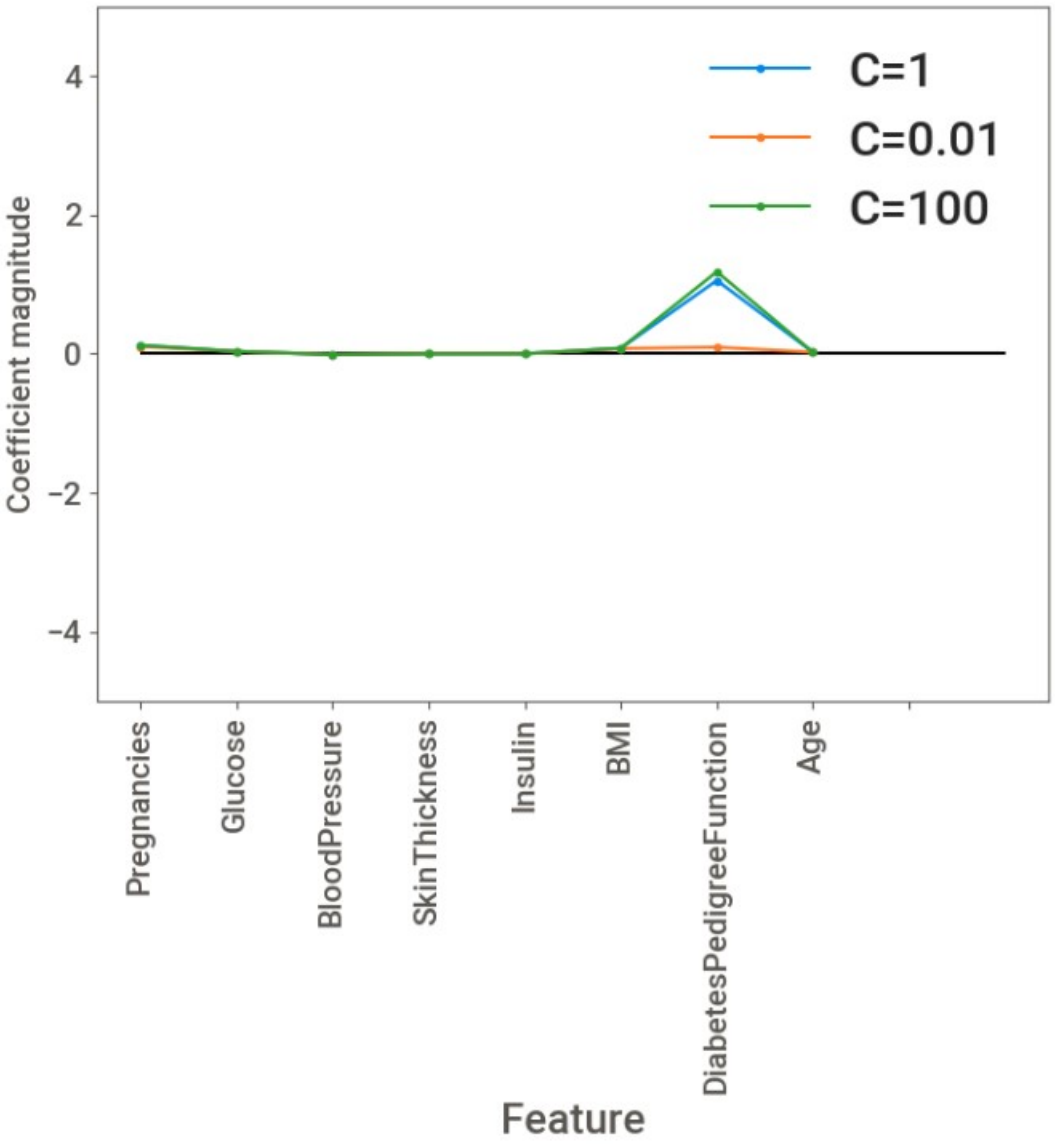
- It is importannt to use stratify inside **train_test_split**. It keeps the same distribution of target same within train and testing datasets. if variable `y` is a binary categorical variable with values 0 and 1 and there are 25% of zeros and 75% of ones, `stratify=y` will make sure that random split has 40% of 0's and 60% of 1's

Machine learning Models

Logistic Regression

```
plt.figure(figsize=(8,6))
Clist=[1,0.01,100]

for C in Clist :

    logreg = LogisticRegression(C=C,solver='newton-cg').fit(X_train, y_train) #keeping C=1 a
    y_train_pred = logreg.predict(X_train)
    y_pred = logreg.predict(X_test)

    print('C : {} Training set accuracy: {:.3f}'.format(C,accuracy_score(y_train, y_train_pred)))
    print('C : {} Test set accuracy: {:.3f}'.format(C,accuracy_score(y_test, y_pred)))

    print('C : {} Training set F1-score: {:.3f}'.format(C,f1_score(y_train, y_train_pred)))
    print('C : {} Test set F1-score: {:.3f}'.format(C, f1_score(y_test, y_pred)))
    print('\n')

    diabetes_features = [x for i,x in enumerate(df.columns) if i!=8]
    plt.plot(logreg.coef_.T, marker='o', label=f"C={C}")

plt.xticks(range(df.shape[1]), diabetes_features, rotation=90)
plt.hlines(0, 0, df.shape[1])
plt.ylim(-5, 5)
plt.xticks(size=15)
plt.yticks(size=15)
plt.xlabel("Feature",size=20)
plt.ylabel("Coefficient magnitude",size=15)
plt.legend(frameon=False)
```

```
C : 1 Training set accuracy: 0.778
C : 1 Test set accuracy: 0.776
C : 1 Training set F1-score: 0.646
C : 1 Test set F1-score: 0.650


C : 0.01 Training set accuracy: 0.776
C : 0.01 Test set accuracy: 0.766
C : 0.01 Training set F1-score: 0.645
C : 0.01 Test set F1-score: 0.634


C : 100 Training set accuracy: 0.774
C : 100 Test set accuracy: 0.771
C : 100 Training set F1-score: 0.641
C : 100 Test set F1-score: 0.645


<matplotlib.legend.Legend at 0x12e2ac850>
```

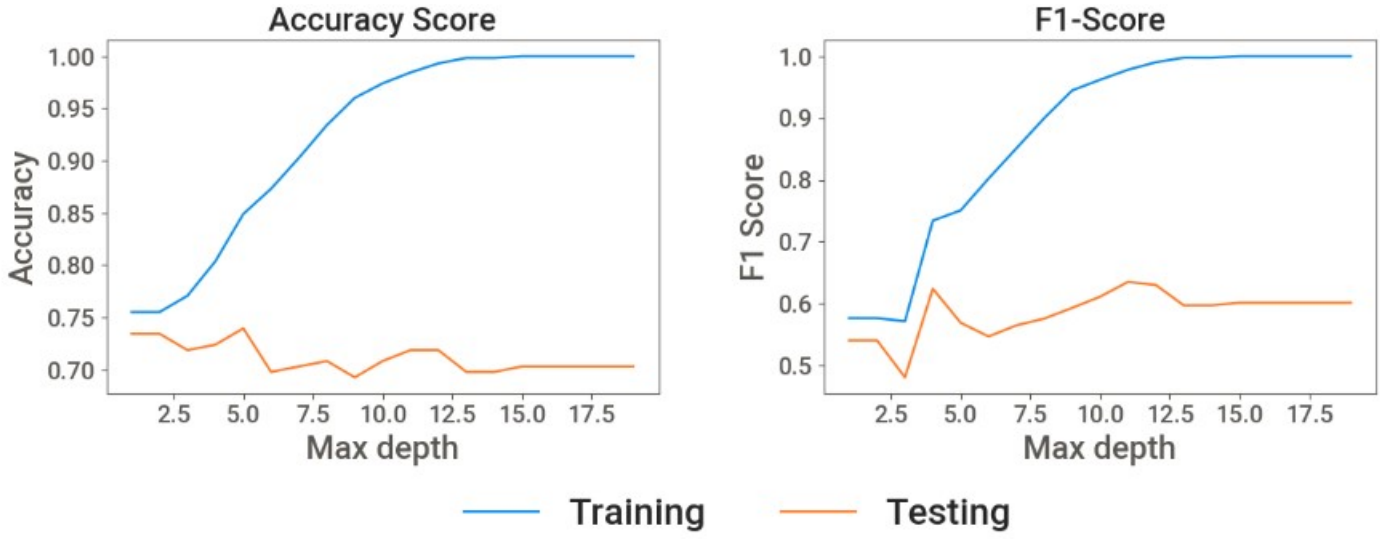


The default value of C=1 provides with 78% accuracy on training and 77% accuracy on test set.

Decision Tree

```
max_depth=range(1,20)
training_accuracy = []
test_accuracy = []
training_f1 = []
test_f1 = []

for depth in max_depth :
    tree = DecisionTreeClassifier(random_state=0, max_depth=depth, min_samples_leaf=1).fit(X_train, y_train)
    y_train_pred = tree.predict(X_train)
    y_pred = tree.predict(X_test)

    training_accuracy.append(accuracy_score(y_train,y_train_pred))
    test_accuracy.append(accuracy_score(y_test, y_pred))

    training_f1.append(f1_score(y_train,y_train_pred))
    test_f1.append(f1_score(y_test, y_pred))
```

```
fig = plt.figure(figsize=(14,10))
fig.subplots_adjust(hspace=0.5, wspace=0.3)

fig.add_subplot(2,2,1)
plt.plot(max_depth, training_accuracy, label='training accuracy')
plt.plot(max_depth, test_accuracy, label='test accuracy')
plt.ylabel('Accuracy',size=20)
plt.xlabel('Max depth',size=20)
plt.xticks(size=15)
plt.yticks(size=15)
plt.title('Accuracy Score',size=20, weight='bold')
plt.legend([],frameon=False)

fig.add_subplot(2,2,2)
plt.plot(max_depth, training_f1)
plt.plot(max_depth, test_f1)
plt.ylabel('F1 Score',size=20)
plt.xlabel('Max depth',size=20)
plt.xticks(size=15)
plt.yticks(size=15)
plt.title('F1-Score',size=20,weight='bold')
plt.legend(['Training','Testing'],frameon=False, bbox_to_anchor=(0.4,-0.2), ncol=2);
```



The training accuracy on the after training set is 100%, while the test set accuracy is much worse. This means tree is overfitting and not generalizing well to new data. We set `max_depth=4`, limiting the depth of the tree decreases overfitting. This leads to a lower accuracy on the training set, but an improvement on the test set.

```
tree = DecisionTreeClassifier(max_depth=4, min_samples_leaf=1, random_state=0).fit(X_train, y_train)
y_pred=tree.predict(X_test)

print("Accuracy on test: {:.3f}".format(accuracy_score(y_pred, y_test)))
print("F1-score on test set: {:.3f}".format(f1_score(y_pred, y_test)))
```

```
Accuracy on test: 0.724
F1-score on test set: 0.624
```

Confusion matrix

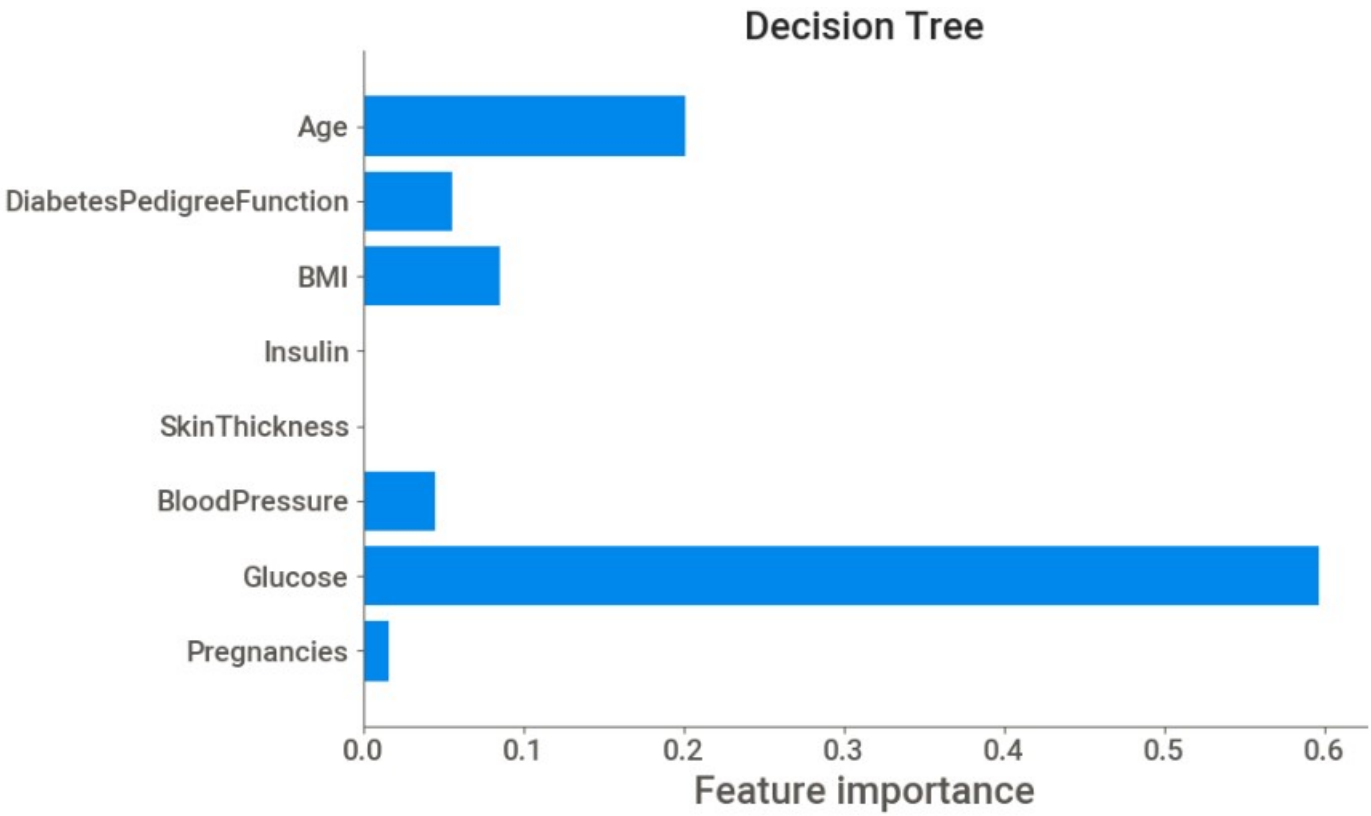
```
conf_mat = confusion_matrix(y_test,y_pred)
normalized_confusion_matrix(y_test,conf_mat,'Decision Tree')
```

Feature importance in Decision trees

Feature importance rates how important each feature is for the decision a tree makes. It is a number between 0 and 1

```
0 means “not used at all”
1 means “perfectly predicts the target”
```

```
def plot_feature_importances(model, figure):
    n_features = 8
    plt.figure(figsize=(10,6))
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), diabetes_features,size=15)
    plt.xticks(size=15)
    plt.xlabel('Feature importance',size=20,)
    #plt.ylabel('Feature',size=20)
    plt.ylim(-1, n_features)
    sns.despine(top=True)
    plt.title(f'{figure}',size=20)
    plt.tight_layout()
    plt.savefig(f'feature-image{figure}.png',dpi=300)

plot_feature_importances(tree,'Decision Tree')
```



Feature `Glucose` is the most important feature.

6)Result:

```
print(f"Accuracy = {accuracy_score(Y_test,pred1)*100}")
print(f"Confusion Matrix =\n {confusion_matrix(Y_test,pred1)}")
print(f"Classification Report =\n {classification_report(Y_test,pred1)}")
```

```
Accuracy = 79.22077922077922
Confusion Matrix =
 [[88 15]
 [17 34]]
Classification Report =
              precision    recall  f1-score   support

           0       0.84      0.85      0.85       103
           1       0.69      0.67      0.68        51

    accuracy                           0.79       154
   macro avg       0.77      0.76      0.76       154
weighted avg       0.79      0.79      0.79       154
```

```
[ ] pd.DataFrame(np.c_[Y_test,pred1],columns=['Actual','Predicted'])
```

	Actual	Predicted
0	1	0
1	1	1
2	0	0
3	1	0
4	0	0
...	...	...
149	1	1
150	1	0
151	1	1
152	0	0
153	1	1

154 rows × 2 columns

7) Conclusion:

- Upon conducting a rigorous 10-fold cross-validation on the dataset, the performance comparison revealed that both logistic regression and neural networks exhibited strong predictive capabilities for the diabetic classification task. However, logistic regression outperformed neural networks, yielding a higher F1-score, which is a more comprehensive metric for evaluating model performance.

- Surprisingly, when analyzing the confusion matrices, it became evident that the decision tree algorithm showcased the highest success in accurately detecting instances of diabetes. The decision tree's ability to effectively classify diabetic cases highlights its potential as a valuable tool in this domain.

- Furthermore, feature selection analysis provided crucial insights, revealing that the "Glucose" feature emerged as the most critical factor in successfully predicting the presence of diabetes. This finding underscores the importance of "Glucose" as a key diagnostic indicator and suggests its significance in informing accurate and reliable diabetic classification.

- Overall, these results emphasize the potential efficacy of logistic regression and decision tree algorithms in the classification of diabetes, with "Glucose" identified as a vital feature for precise prediction. These findings contribute to the advancement of diabetic classification methodologies and may have implications for improving diagnostic accuracy and patient outcomes in clinical settings.