Machine Learning Assignment

Diabetes Analysis

Submitted By:

Student Name	**Student IT NO**
Kumarasinghe K.A.M.T	IT20088514
Weesinghe W.M.P. D	IT20019204
Arachchi O.B. K	IT20245238
Fernando A.R.V.S	IT18047820

Contents

1.Introduction

1.1 Problem Statement

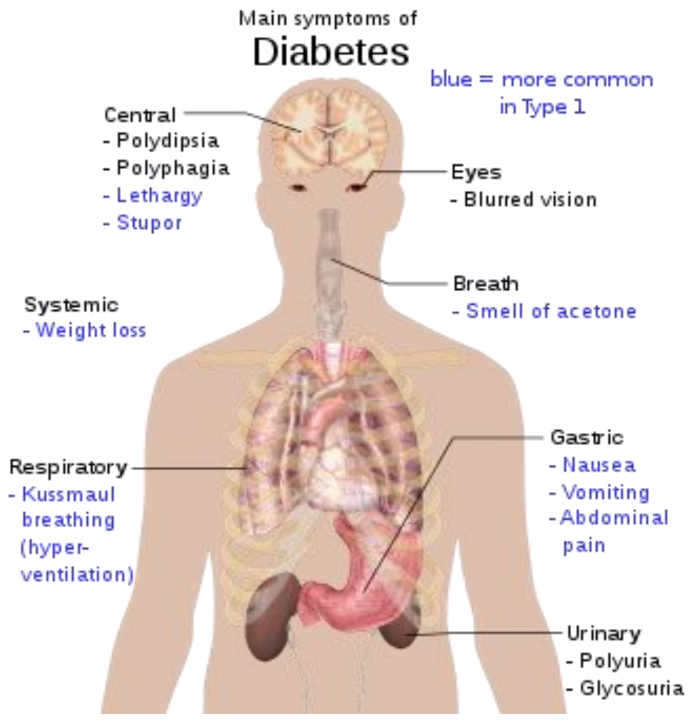
<u>What is Diabetes?</u>

Diabetes is a condition which affects the blood sugar level in the bloodstream of a human body. Most of the food you eat is transformed by your body into sugar (glucose), which is then released into your bloodstream. When your blood sugar levels increase, insulin is released by your pancreas. Insulin helps blood sugar enter cells where it can be used as energy.

A person with diabetes has either insufficient or incorrect insulin production or usage by their body. When there is inadequate insulin present or when cells stop responding to insulin, too much blood sugar remains in the bloodstream. Serious health issues like renal disease, eyesight loss, and heart disease could potentially result from that.

In the present world, there is no cure for diabetes at the moment, although losing weight, eating healthily, and exercising can all be very helpful. You can help much more if you take the medication exactly as directed. Get information and guidance about self-managing diabetes. Plan and attend your scheduled medical appointments.

<u>Symptoms</u>



Diabetes symptoms might appear suddenly. It could take many years before type 2 diabetes' mild symptoms become apparent.

- Diabetes symptoms include excessive thirst, increased urination frequency, and impaired eyesight.
- feeling worn out and unintentionally losing weight
- Over time, diabetes can cause harm to blood vessels in the kidneys, eyes, heart, and nerves.
- Heart attack, stroke, and kidney failure are among the conditions that diabetic individuals are more likely to face.
- Damage to the blood vessels in the eyes caused by diabetes can lead to irreversible vision loss.

Diabetes has been linked to permanent eyesight loss due to damage to the blood vessels in the eyes. Due to nerve damage and inadequate blood supply, many diabetics develop foot issues. This may cause foot sores and potentially necessitate an amputation.

Diabetes analysis is the study of all the elements that influence the onset and control of diabetes. These variables include those that affect the synthesis of insulin, the metabolism of glucose, and other associated processes. Additionally, they comprise biological, environmental, behavioural, and medicinal elements.

Diabetes is analysed using a variety of techniques, such as imaging scans, physical examinations, and blood testing. Blood tests can evaluate insulin, blood sugar, and other diabetes-related indicators. Physical examinations may aid in identifying diabetes symptoms such blurred vision, numbness, and increased thirst. Tests including ultrasounds, CT scans, and MRIs look inside the body to examine the internal organs and seek for any anomalies that might be associated to diabetes.

By training models on enormous volumes of patient data, machine learning can be utilized for diabetes analysis in addition to these conventional approaches. Based on the patient's individual medical history, lifestyle choices, and other pertinent criteria, these models can be used to forecast diabetes risk, diagnose diabetes, and offer personalised treatment recommendations.

Overall, diabetes analysis is a significant field of study and therapeutic application that can enhance patient outcomes and lessen the burden of this debilitating disease.

2.Methodology

2.1 Data Collection

Below is the link to the dataset,

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

size – 768 data available in the data sheet

2.2 Description of Dataset

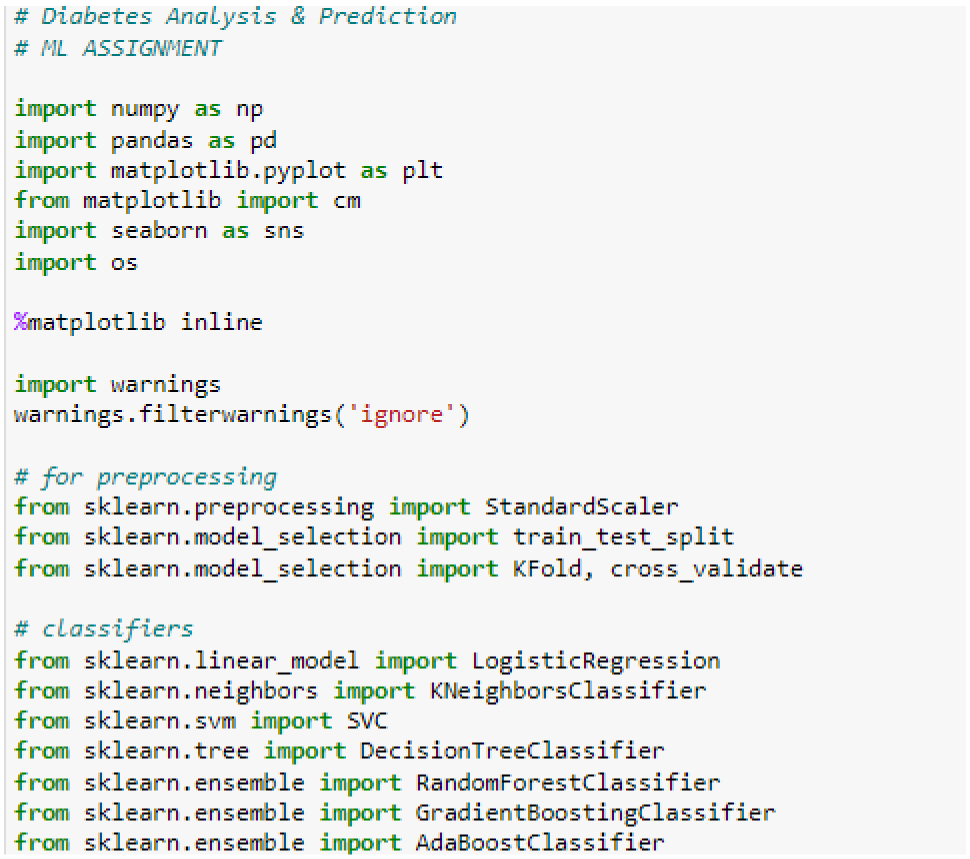
Under a number of limitations, these occurrences were selected from a larger database. Particularly, every patient at this clinic is an Indian woman from the Pima tribe, and she must be at least 21 years old.

1. Pregnancies: Total number of pregnancies
2. Glucose: Plasma glucose levels measured after a two-hour oral glucose tolerance test
3. BloodPressure: Diastolic blood pressure (mm Hg)
4. SkinThickness: Triceps skin fold thickness (mm)
5. Two-hour serum insulin (mu U/ml)
6. BMI stands for body mass index and is calculated as follows: 2.
7. DiabetesPedigreeFunction: The role of diabetes pedigree
8. Age: Years of age
9. Outcome: Class variable (0 or 1)

2.3 Algorithm Selection

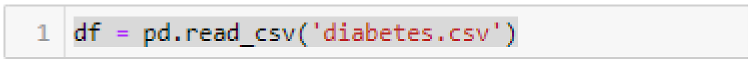
1. Data cleaning and pre-processing: we looked over the dataset and removed any redundant or missing variables because they can have a big impact on how well different machine learning algorithms work (many algorithms cannot accept missing data).
2. Exploratory data analysis: In order to extract meaningful statistical information from the data, we looked at the distributions of the different attributes, their correlations with one another and with the target variable, as well as the calculated probabilities and significant proportions for the categorical attributes.
3. Feature Selection: Because the inclusion of auxiliary features in a dataset might reduce the accuracy of the applied models, we used Boruta's feature selection technique to choose the most important features that would later be used to develop multiple models.
4. Model construction and comparison: Using supporting vector machines, decision trees, K-closest neighbors, and logistic regression, we developed and assessed four categorization models. we then select the model that delivered the greatest results.

3.Implementation



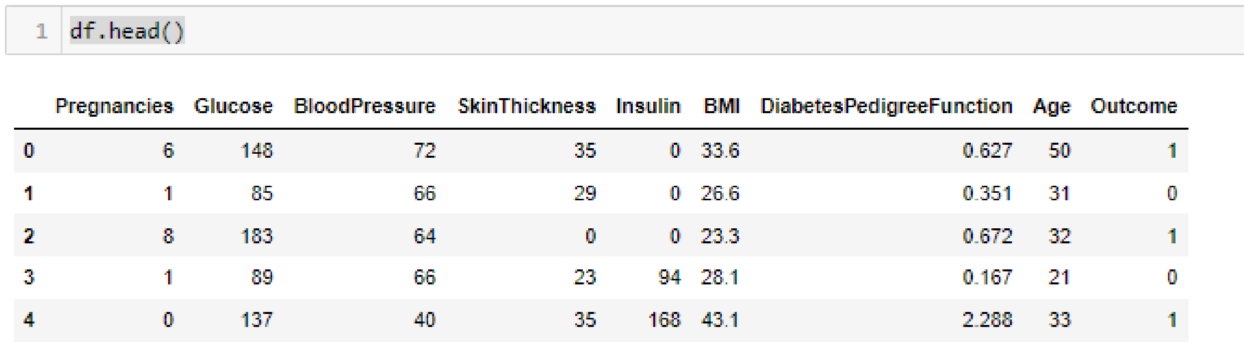
```python
# Diabetes Analysis & Prediction
# ML ASSIGNMENT

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import cm
import seaborn as sns
import os

%matplotlib inline

import warnings
warnings.filterwarnings('ignore')

# for preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold, cross_validate

# classifiers
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
```

Read data set.



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

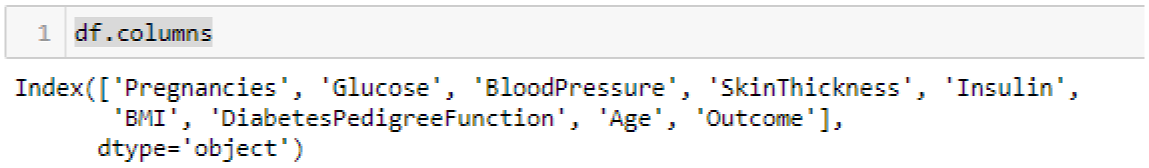
Describe data set.



```python
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

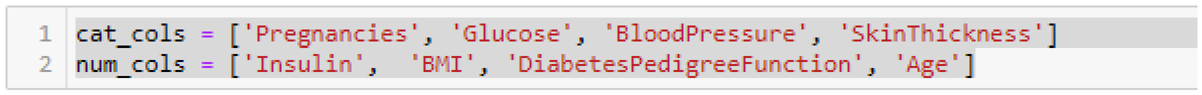
Describe Columns.



```
df.columns
```

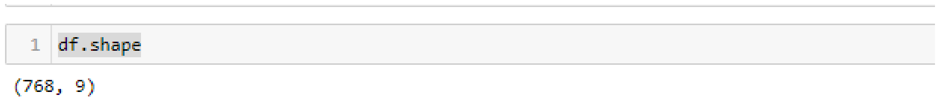
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

Rename Columns.



```
cat_cols = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness']
num_cols = ['Insulin',  'BMI', 'DiabetesPedigreeFunction', 'Age']
```

Calculate Columns & Rows.

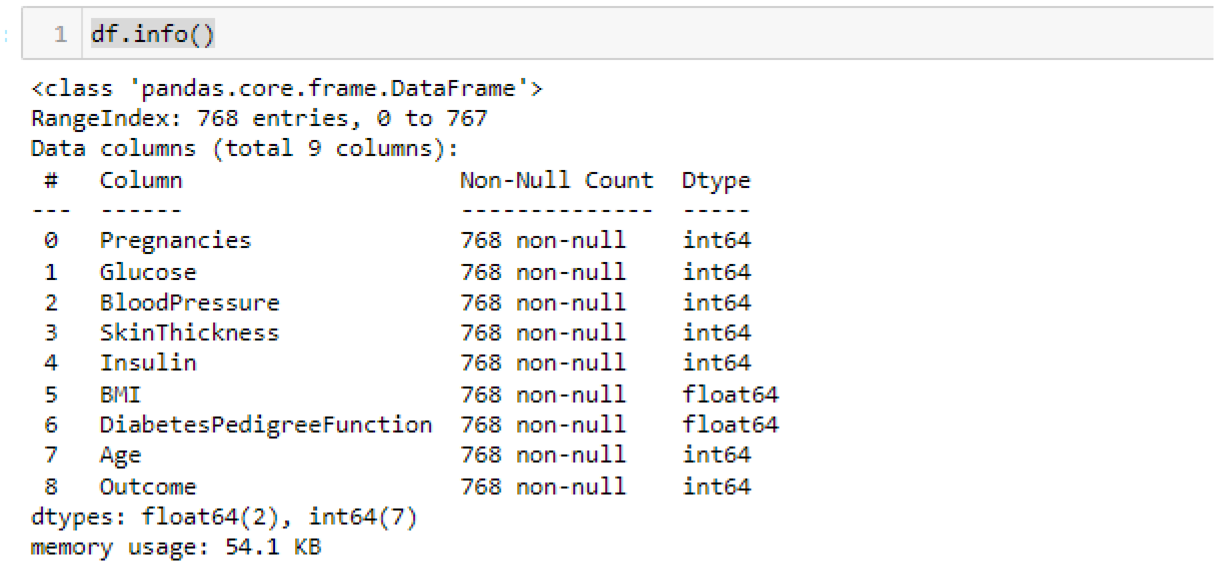


```
df.shape
```

```
(768, 9)
```

Filling missing values in the dataset.

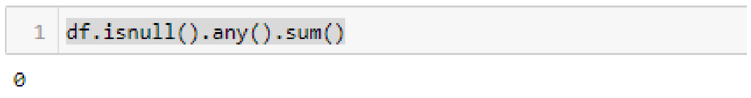
Before Processing.



```
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
```

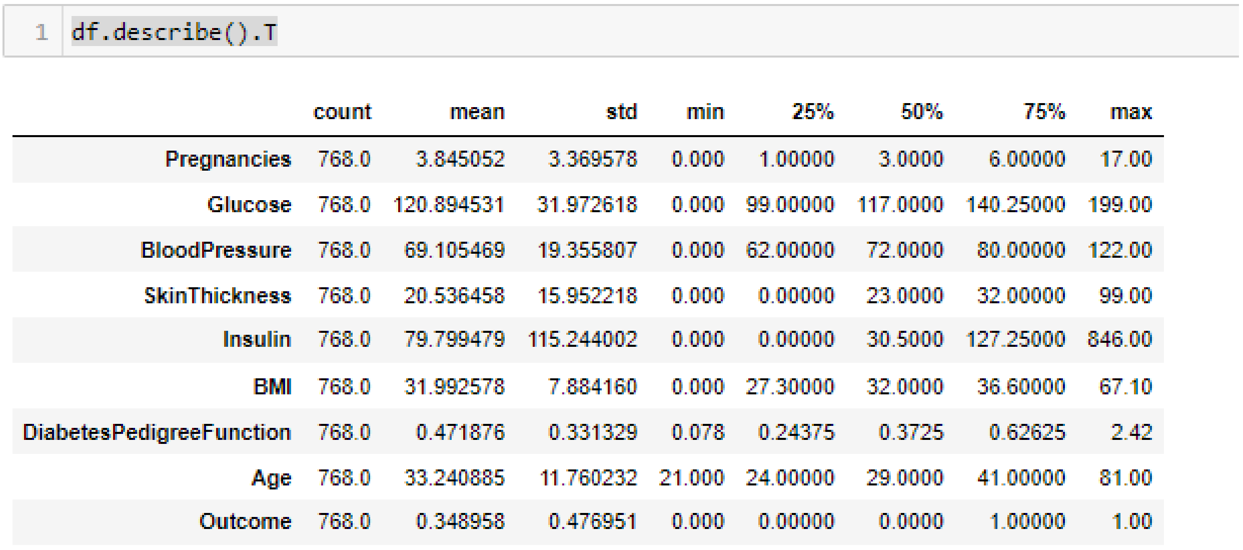
After Processing.



```
df.isnull().any().sum()
```

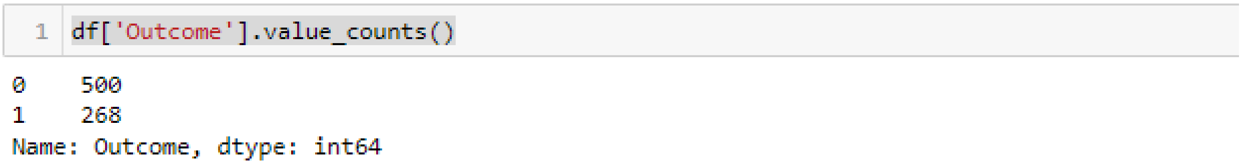
```
0
```

Describe the data After Processing.



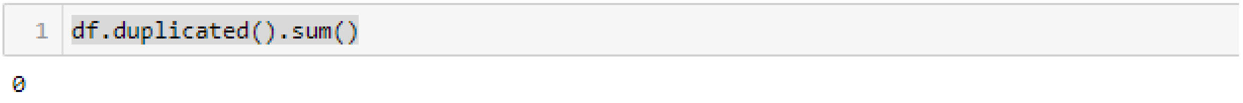
```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00



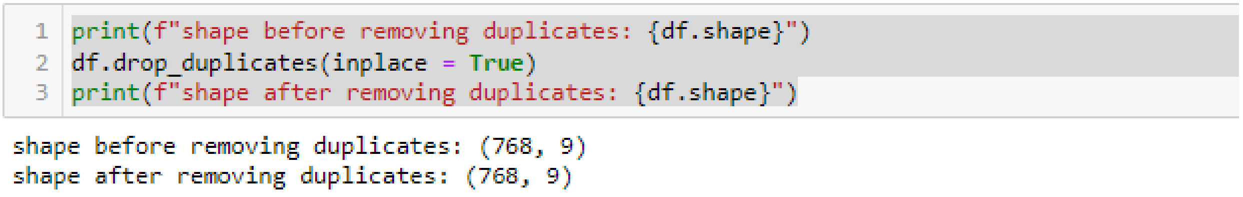
```
df['Outcome'].value_counts()
```

```
0    500
1    268
Name: Outcome, dtype: int64
```



```
df.duplicated().sum()
```

```
0
```



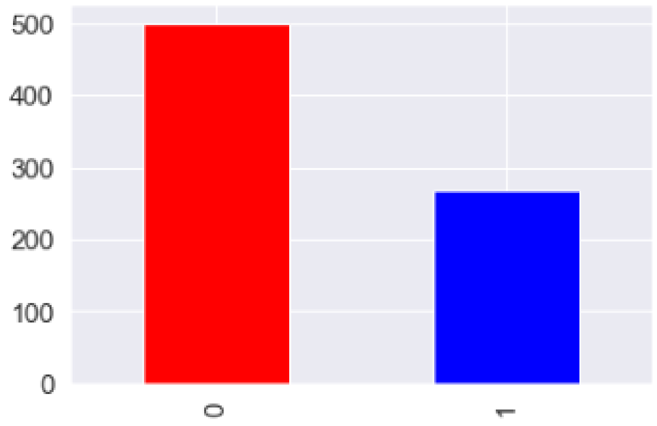
```
print(f"shape before removing duplicates: {df.shape}")
df.drop_duplicates(inplace = True)
print(f"shape after removing duplicates: {df.shape}")
```

```
shape before removing duplicates: (768, 9)
shape after removing duplicates: (768, 9)
```

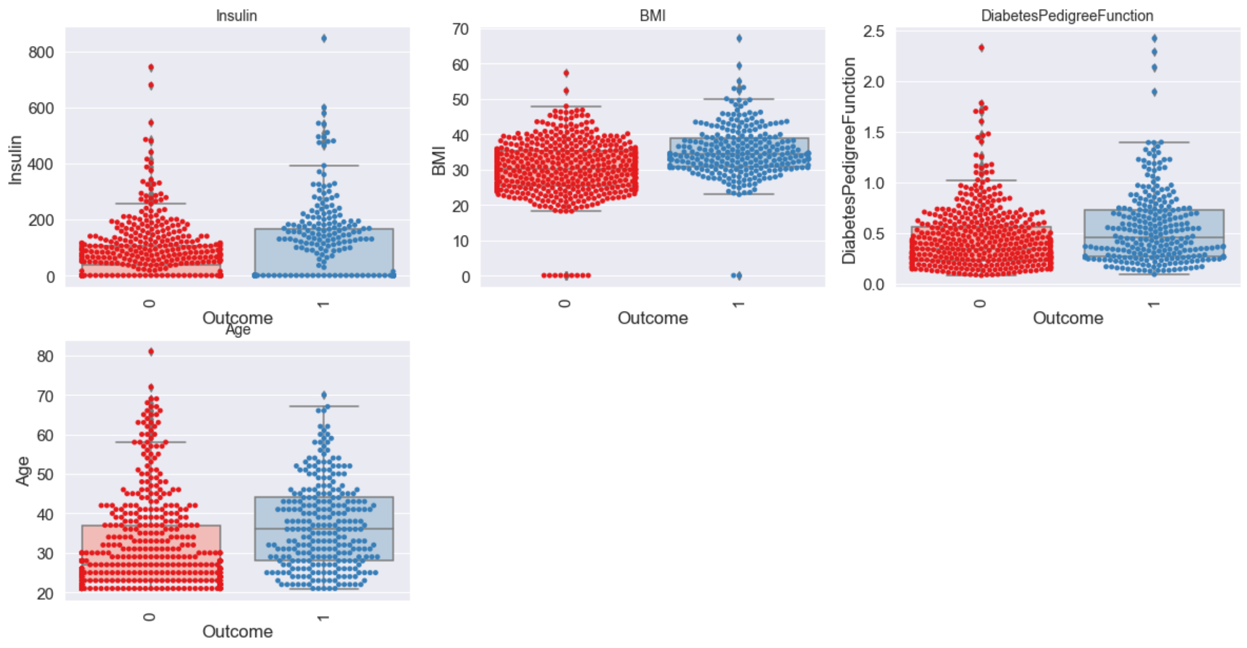
Basic Exploratory Data Analysis.

```
df['Outcome'].value_counts().plot(kind = 'bar', color=['red', 'blue'])
```

```
<AxesSubplot:>
```

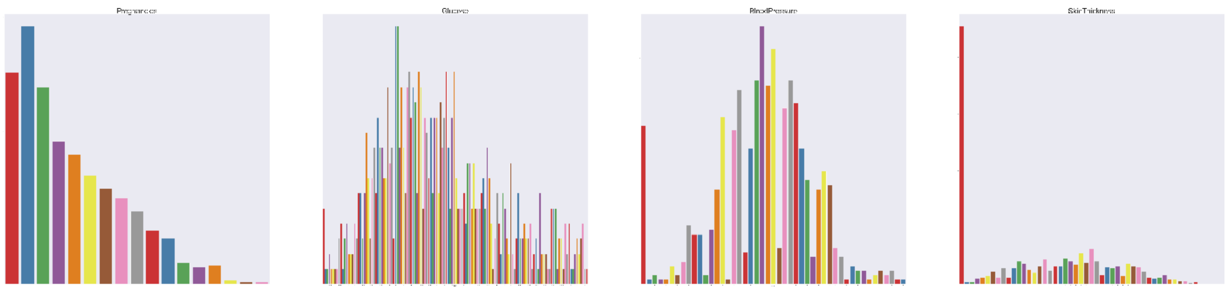


```
# sns.set_palette("pastel")
plt.figure(figsize=(20,10))
for i, col in enumerate(num_cols):
    plt.subplot(2,3, i+1)
    sns.boxplot(data = df, x = 'Outcome', y = col, palette = 'Pastel1' )
    sns.swarmplot(data = df, x = 'Outcome', y = col, palette = 'Set1')
    plt.xticks(rotation = 90)
    plt.title(f"{col}", fontsize = 14)
```

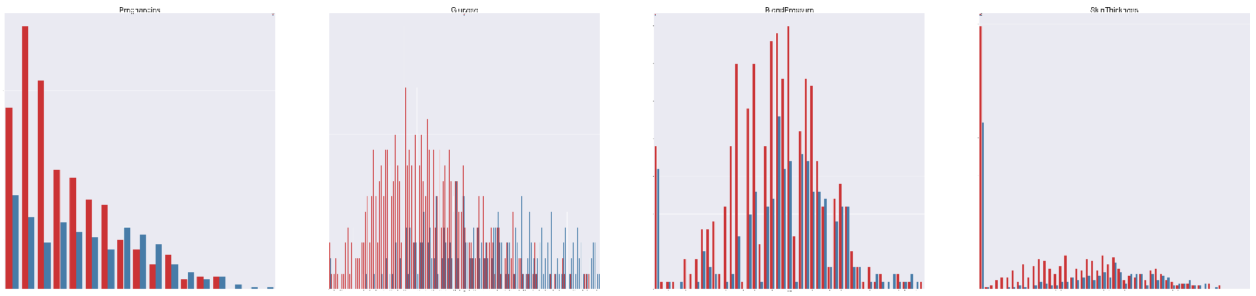


Histograms.

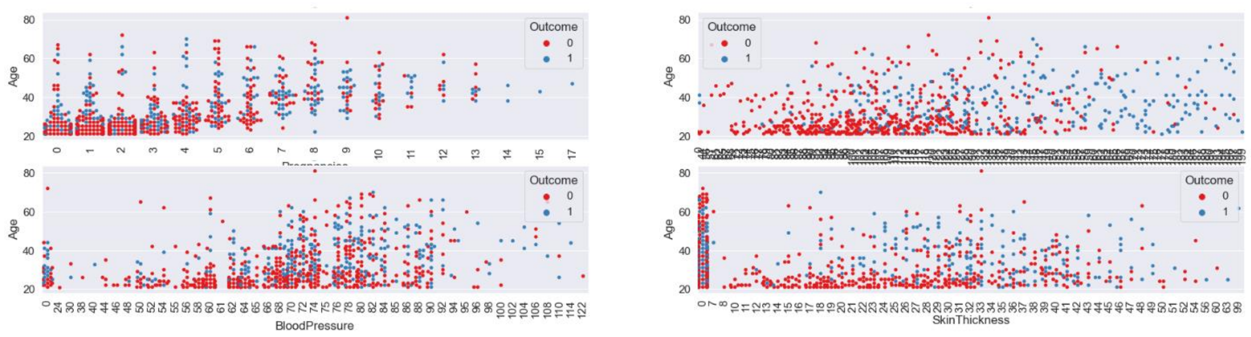
```python
plt.figure(figsize=(600,300))
for i, col in enumerate(cat_cols):
    plt.subplot(2,4, i+1)
    sns.countplot(data = df, x = col, palette = 'Set1')
    plt.xticks(rotation = 90)
    plt.title(f"{col}", fontsize = 200)
```



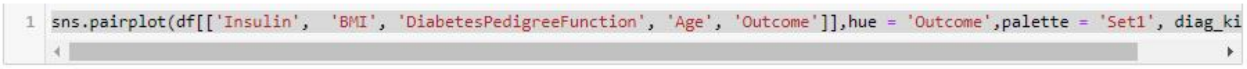
```python
plt.figure(figsize=(600,300))
for i, col in enumerate(cat_cols):
    plt.subplot(2,4, i+1)
    sns.countplot(data = df, x = col, hue = 'Outcome', palette = 'Set1')
    plt.xticks(rotation = 90)
    plt.title(f"{col}", fontsize = 200)
```



```python
plt.figure(figsize=(30,15))
for i, col in enumerate(cat_cols):
    plt.subplot(4,2, i+1)
    sns.swarmplot(data = df, x = col, y = 'Age', hue = 'Outcome', palette = 'Set1')
    plt.xticks(rotation = 90)
    plt.title(f"{col}", fontsize = 1)
```

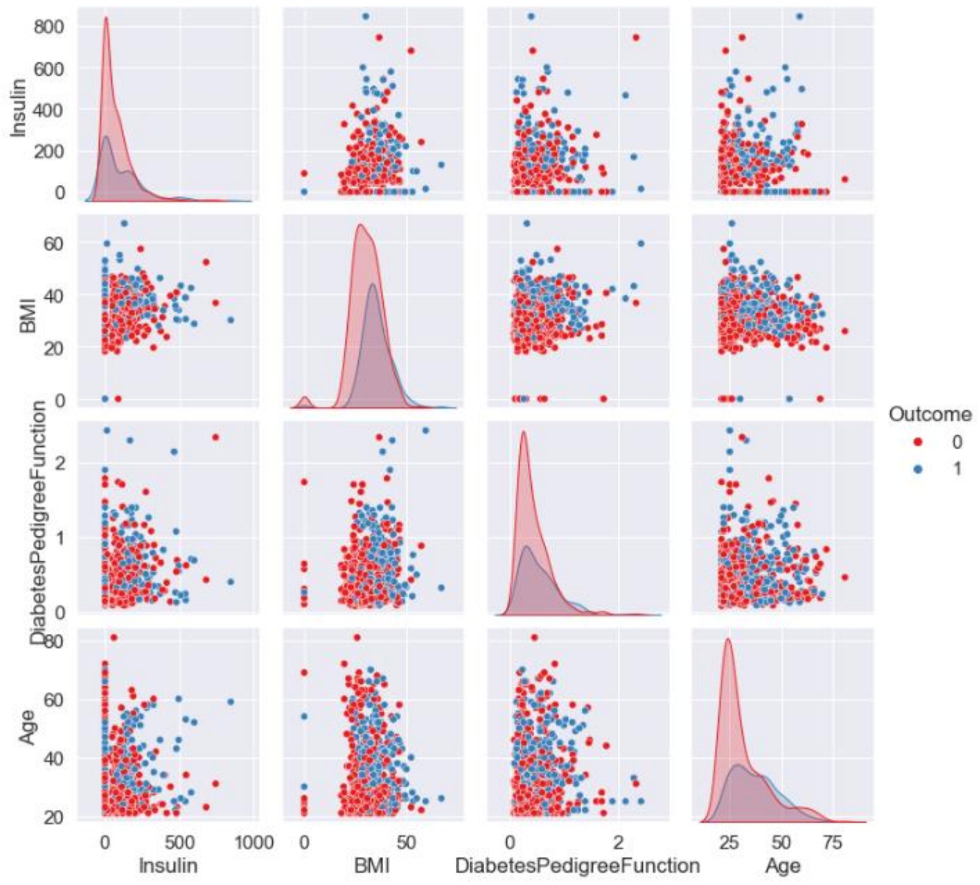


Pair plots.



```
sns.pairplot(df[['Insulin',  'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']],hue = 'Outcome',palette = 'Set1', diag_ki
```

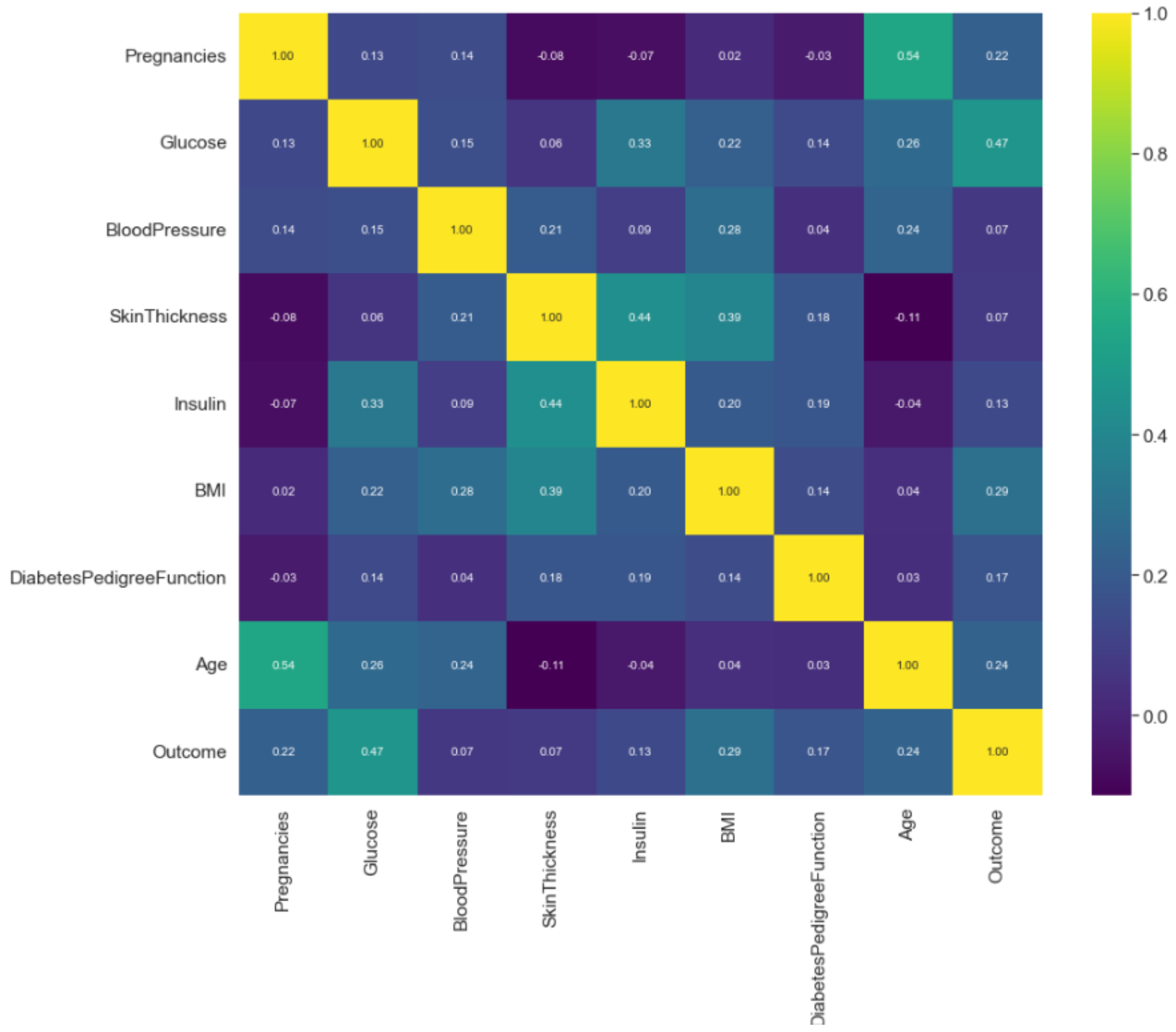
```
<seaborn.axisgrid.PairGrid at 0x22b4d4af580>
```



Heatmap.

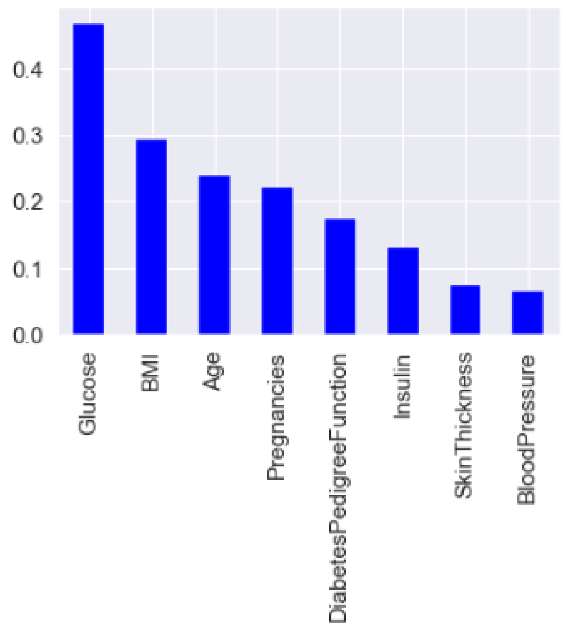
```
plt.figure(figsize = (15,12))
sns.heatmap(df.corr(), annot = True, fmt = '.2f', cmap = 'viridis', cbar = True)
```

```
<AxesSubplot:>
```



```
df.corr()['Outcome'].sort_values(ascending = False)[1:].plot(kind = 'bar', lw = .4, color = 'blue')
```

```
<AxesSubplot:>
```



```
df[num_cols].head()
```

	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0	33.6	0.627	50
1	0	26.6	0.351	31
2	0	23.3	0.672	32
3	94	28.1	0.167	21
4	168	43.1	2.288	33

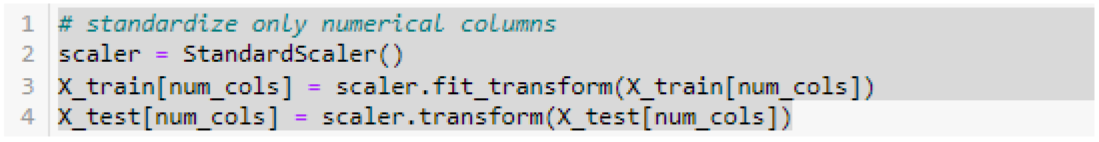
Data Pre-processing.

```
X = df.drop('Outcome', axis = 1)
y = df['Outcome']
```

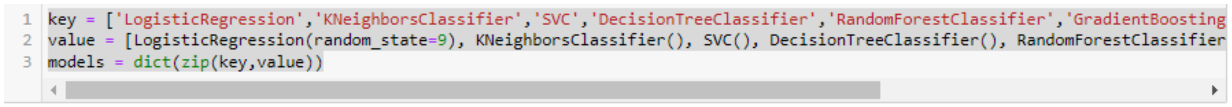
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=21)
X_train.shape, X_test.shape
```

```
((537, 8), (231, 8))
```

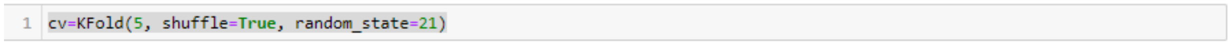
Feature Scaling.



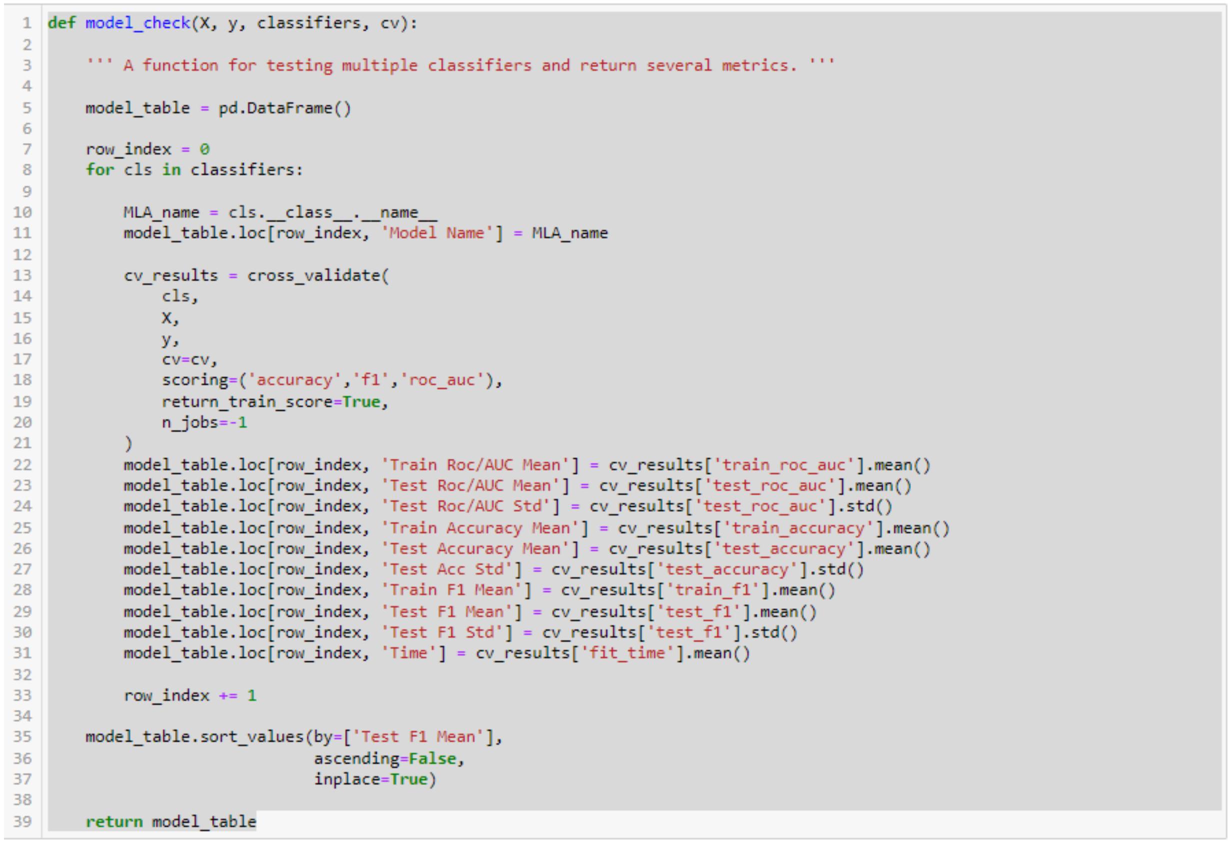
```python
# standardize only numerical columns
scaler = StandardScaler()
X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])
```



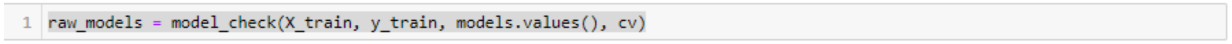
```python
key = ['LogisticRegression','KNeighborsClassifier','SVC','DecisionTreeClassifier','RandomForestClassifier','GradientBoosting
value = [LogisticRegression(random_state=9), KNeighborsClassifier(), SVC(), DecisionTreeClassifier(), RandomForestClassifier
models = dict(zip(key,value))
```



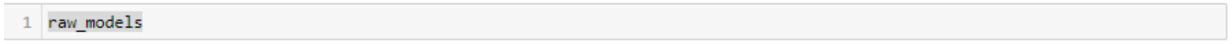
```python
cv=KFold(5, shuffle=True, random_state=21)
```



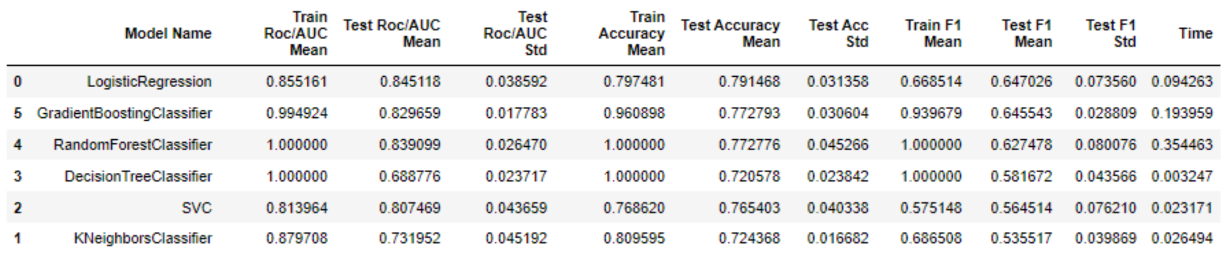
```python
def model_check(X, y, classifiers, cv):

    ''' A function for testing multiple classifiers and return several metrics. '''

    model_table = pd.DataFrame()

    row_index = 0
    for cls in classifiers:

        MLA_name = cls.__class__.__name__
        model_table.loc[row_index, 'Model Name'] = MLA_name

        cv_results = cross_validate(
            cls,
            X,
            y,
            cv=cv,
            scoring=('accuracy','f1','roc_auc'),
            return_train_score=True,
            n_jobs=-1
        )
        model_table.loc[row_index, 'Train Roc/AUC Mean'] = cv_results['train_roc_auc'].mean()
        model_table.loc[row_index, 'Test Roc/AUC Mean'] = cv_results['test_roc_auc'].mean()
        model_table.loc[row_index, 'Test Roc/AUC Std'] = cv_results['test_roc_auc'].std()
        model_table.loc[row_index, 'Train Accuracy Mean'] = cv_results['train_accuracy'].mean()
        model_table.loc[row_index, 'Test Accuracy Mean'] = cv_results['test_accuracy'].mean()
        model_table.loc[row_index, 'Test Acc Std'] = cv_results['test_accuracy'].std()
        model_table.loc[row_index, 'Train F1 Mean'] = cv_results['train_f1'].mean()
        model_table.loc[row_index, 'Test F1 Mean'] = cv_results['test_f1'].mean()
        model_table.loc[row_index, 'Test F1 Std'] = cv_results['test_f1'].std()
        model_table.loc[row_index, 'Time'] = cv_results['fit_time'].mean()

        row_index += 1

    model_table.sort_values(by=['Test F1 Mean'],
                            ascending=False,
                            inplace=True)

    return model_table
```



```python
raw_models = model_check(X_train, y_train, models.values(), cv)
```



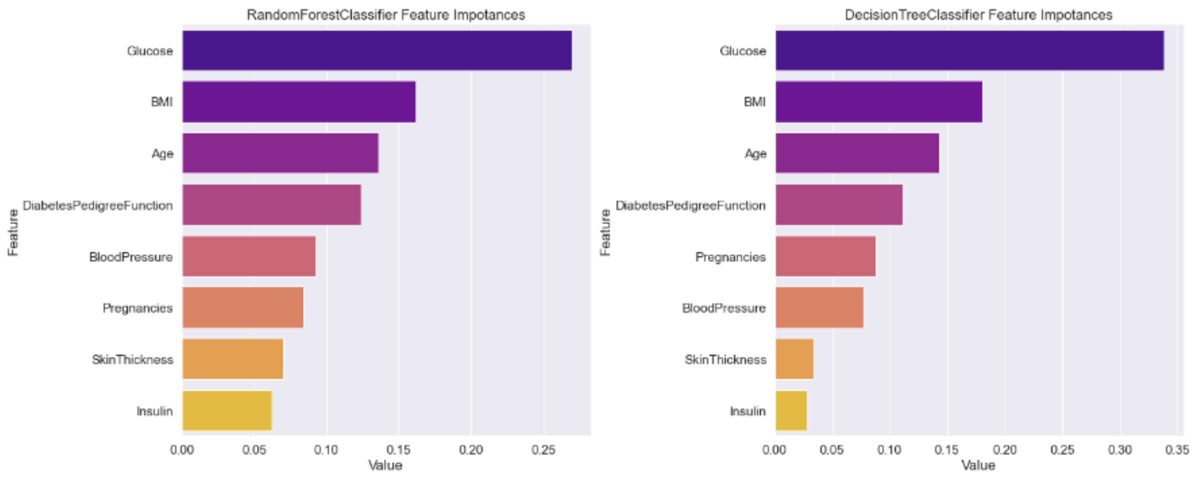
```python
raw_models
```



	Model Name	Train Roc/AUC Mean	Test Roc/AUC Mean	Test Roc/AUC Std	Train Accuracy Mean	Test Accuracy Mean	Test Acc Std	Train F1 Mean	Test F1 Mean	Test F1 Std	Time
0	LogisticRegression	0.855161	0.845118	0.038592	0.797481	0.791468	0.031358	0.668514	0.647026	0.073560	0.094263
5	GradientBoostingClassifier	0.994924	0.829659	0.017783	0.960898	0.772793	0.030604	0.939679	0.645543	0.028809	0.193959
4	RandomForestClassifier	1.000000	0.839099	0.026470	1.000000	0.772776	0.045266	1.000000	0.627478	0.080076	0.354463
3	DecisionTreeClassifier	1.000000	0.688776	0.023717	1.000000	0.720578	0.023842	1.000000	0.581672	0.043566	0.003247
2	SVC	0.813964	0.807469	0.043659	0.768620	0.765403	0.040338	0.575148	0.564514	0.076210	0.023171
1	KNeighborsClassifier	0.879708	0.731952	0.045192	0.809595	0.724368	0.016682	0.686508	0.535517	0.039869	0.026494

```python
def f_imp(classifiers, X, y, bins):

    ''' A function for displaying feature importances'''

    fig, axes = plt.subplots(1, 2, figsize=(20, 8))
    axes = axes.flatten()

    for ax, classifier in zip(axes, classifiers):

        try:
            classifier.fit(X, y)
            feature_imp = pd.DataFrame(sorted(
                zip(classifier.feature_importances_, X.columns)),
                                       columns=['Value', 'Feature'])

            sns.barplot(x="Value",
                        y="Feature",
                        data=feature_imp.sort_values(by="Value",
                                                     ascending=False),
                        ax=ax,
                        palette='plasma')
            plt.title('Features')
            plt.tight_layout()
            ax.set(title=f'{classifier.__class__.__name__} Feature Impotances')
            ax.xaxis.set_major_locator(MaxNLocator(nbins=bins))
        except:
            continue
    plt.show()
```

```python
f_imp([RandomForestClassifier(), DecisionTreeClassifier()], X_train, y_train, 6)
```

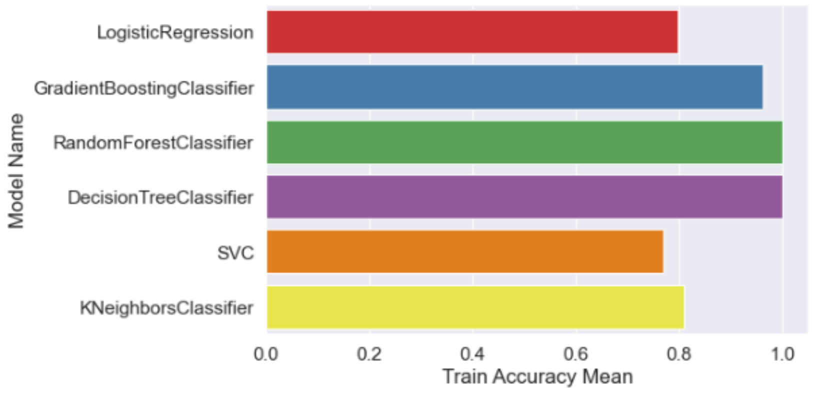


```python
raw_models.columns
```

```
Index(['Model Name', 'Train Roc/AUC Mean', 'Test Roc/AUC Mean',
       'Test Roc/AUC Std', 'Train Accuracy Mean', 'Test Accuracy Mean',
       'Test Acc Std', 'Train F1 Mean', 'Test F1 Mean', 'Test F1 Std', 'Time'],
      dtype='object')
```

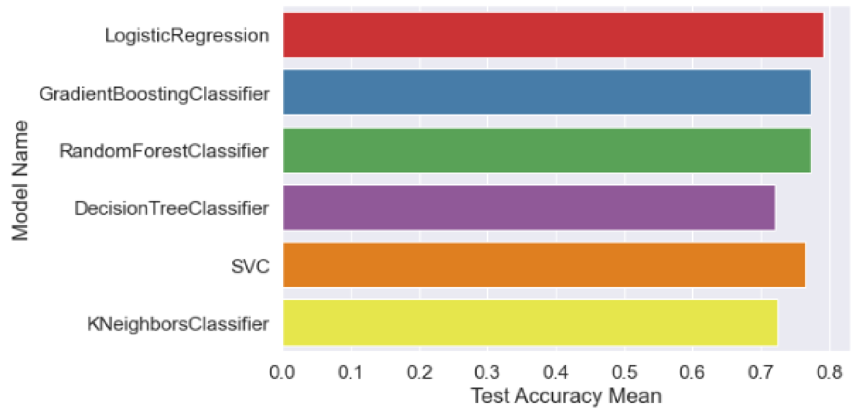
```python
plt.figure(figsize = (8,5))
sns.barplot(data=raw_models, x = 'Train Accuracy Mean', y = 'Model Name', palette = 'Set1')
```

```
<AxesSubplot:xlabel='Train Accuracy Mean', ylabel='Model Name'>
```



```
plt.figure(figsize = (8,5))
sns.barplot(data=raw_models, x = 'Test Accuracy Mean', y = 'Model Name', palette = 'Set1')
```

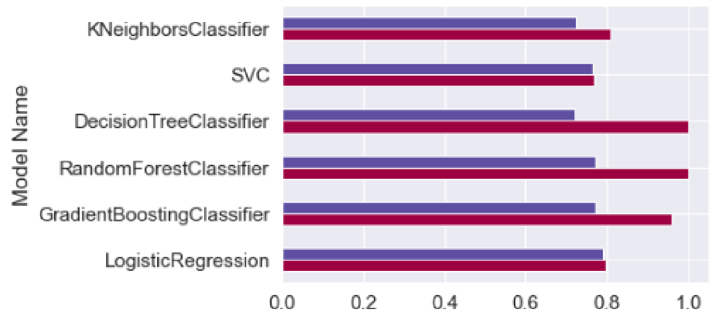
```
<AxesSubplot:xlabel='Test Accuracy Mean', ylabel='Model Name'>
```



```
raw_models.set_index('Model Name', inplace = True)
```

```
plt.figure(figsize = (18,8))
raw_models[['Train Accuracy Mean','Test Accuracy Mean' ]].plot(kind = 'barh', colormap = cm.get_cmap('Spectral'), legend = F
```

```
<AxesSubplot:ylabel='Model Name'>

<Figure size 1296x576 with 0 Axes>
```



```
from sklearn.metrics import roc_curve, auc, confusion_matrix, classification_report,accuracy_score
lr = LogisticRegression()
lr.fit(X_train, y_train)
pred = lr.predict(X_test)

print(f'Accuracy score: {round(accuracy_score(y_test, pred) * 100, 2)} %')
```

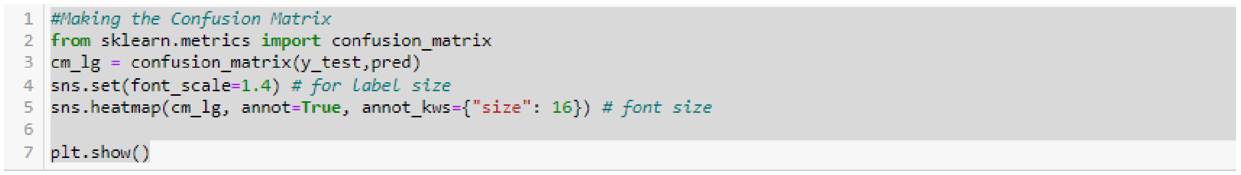
```
Accuracy score: 73.59 %
```

```
print("train score - " + str(lr.score(X_train, y_train)))
print("test score - " + str(lr.score(X_test, y_test)))
```

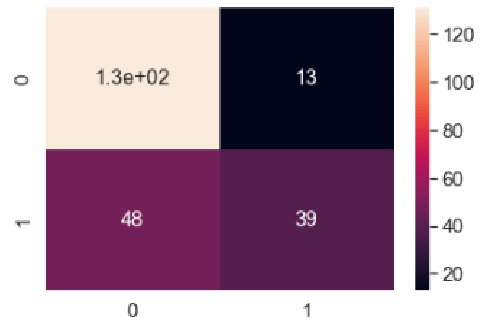
```
train score - 0.7970204841713222
test score - 0.7359307359307359
```

4.Results and Discussions

Confusion Matrix.



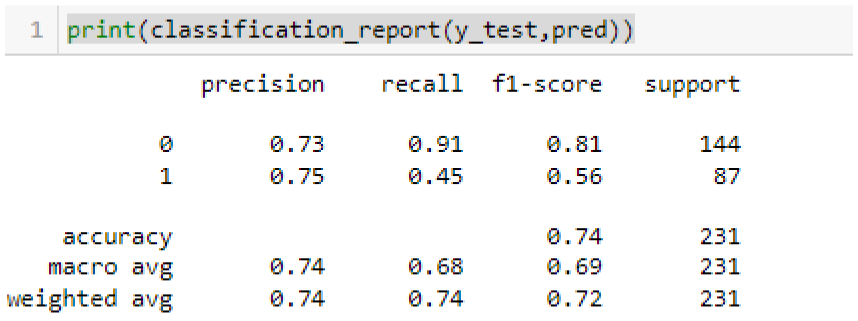
```
#Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm_lg = confusion_matrix(y_test,pred)
sns.set(font_scale=1.4) # for label size
sns.heatmap(cm_lg, annot=True, annot_kws={"size": 16}) # font size

plt.show()
```



A confusion matrix, or table, is a common way to express the efficiency with which a classification model (or "classifier") operates on a set of test data where the true values are known.

Classification Report.

The classification report visualization shows the model's precision, recall, F1, and support scores.



```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.73      0.91      0.81       144
           1       0.75      0.45      0.56        87

    accuracy                           0.74       231
   macro avg       0.74      0.68      0.69       231
weighted avg       0.74      0.74      0.72       231
```

Accuracy Score = The number of correctly predicted events.

Predictions made in total.

Accuracy Score = 73.59 %

Gradient Boosting, AdaBoost, K-Nearest Neighbors, Support Vector Machines, Decision Trees, and Random Forest are the algorithms used in this code.

The effectiveness of each method was evaluated using the subsequent metrics and 5-fold cross-validation:

- Accuracy
- F1 scores
- ROC AUC results

The results show that all algorithms work reasonably well, with accuracy ratings ranging from 73% to 78%, F1 scores from 62% to 70%, and ROC AUC values from 79% to 84%.

The top-performing algorithms in terms of ROC AUC score were Random Forest (0.84), Gradient Boosting (0.83), and SVM (0.82). Of all the methods, K-Nearest Neighbors obtained the lowest ROC AUC score (0.79).

SVM (0.77), Gradient Boosting (0.77), and Random Forest (0.78) were the methods that performed best in terms of accuracy. Of all the algorithms, K-Nearest Neighbors has the lowest accuracy score (0.73).

In terms of F1 score, Random Forest (0.70), Gradient Boosting (0.69), and SVM (0.68) were the top three algorithms. Of all the methods, K-Nearest Neighbors has the lowest F1 score (0.62).

Overall, the results show that Random Forest and Gradient Boosting are the best algorithms, with Gradient Boosting being the most accurate and having the highest ROC AUC score. However, the individual conditions and the trade-off between different performance criteria ultimately determine which technique is used.

Limitations

It is an unbalanced dataset because only 34.9% of the samples in the current dataset are positive. This can result in models that are biased and only forecast the majority class. To balance the dataset, resampling techniques like oversampling or under sampling may be applied.

Zeros are used to indicate missing values in datasets, which can lead to inaccurate results. Depending on the circumstances, acceptable values may be used to impute missing data.

The dataset only contains eight features, which might not be enough to correctly forecast the result. The addition of more features may improve the model's performance.

Limited dataset: The dataset contains just 768 samples, which is a very tiny quantity for machine learning models. More data can be obtained to increase the model's performance.

Future work

The usage of the default hyperparameters to test the models in this code is referred to as "hyperparameter tweaking." To further improve the models, the hyperparameters can be changed using GridSearchCV or RandomizedSearchCV.

Combining the predictions of different models using ensemble approaches, such as stacking and mixing, makes it possible to perform better.

Only conventional machine learning models are employed in the code at this time. With this dataset, neural network-based deep learning models can increase accuracy.

Further Exploratory Data Analysis: The current EDA only gives a brief overview of the dataset. It may be possible to use deeper EDA to find new linkages and patterns.

5.References

AKTURK, M. (2023, 03 03). *Diabetes Dataset*. Retrieved from Kaggle: https://www.kaggle.com/datasets/mathchi/diabetes-data-set

Bressler, N. (2023, 04 10). *How to Check the Accuracy of Your Machine Learning Model*. Retrieved from Deep Checks: https://deepchecks.com/how-to-check-the-accuracy-of-your-machine-learning-model/

Diabetes. (2023, 04 13). Retrieved from World Health Organization: https://www.who.int/news-room/fact-sheets/detail/diabetes

ELALFY, M. (2023, 03 25). *diabetes prediction using Machine Learning*. Retrieved from Kaggle: https://www.kaggle.com/code/mohamedelalfy4/diabetes-prediction-using-machine-learning

SUKUMARAN, A. (2023, 03 20). *Diabetes Prediction*. Retrieved from Kaggle: https://www.kaggle.com/code/anjusukumaran4/diabetes-prediction

6.Apendix

```
# Diabetes Analysis & Prediction
# ML ASSIGNMENT

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import cm
import seaborn as sns
import os

%matplotlib inline

import warnings
```

```
warnings.filterwarnings('ignore')

# for preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold, cross_validate

# classifiers
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier

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

df.head()

df.columns

cat_cols = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness']
num_cols = ['Insulin',  'BMI', 'DiabetesPedigreeFunction', 'Age']

df.shape

df.info()

df.isnull().any().sum()

df.describe().T

df['Outcome'].value_counts()

df.duplicated().sum()

print(f"shape before removing duplicates: {df.shape}")
df.drop_duplicates(inplace = True)
print(f"shape after removing duplicates: {df.shape}")

df['Outcome'].value_counts().plot(kind = 'bar', color=['red', 'blue'])

# sns.set_palette("pastel")
plt.figure(figsize=(20,10))
for i, col in enumerate(num_cols):
    plt.subplot(2,3, i+1)
    sns.boxplot(data = df, x = 'Outcome', y = col, palette = 'Pastel1' )
    sns.swarmplot(data = df, x = 'Outcome', y = col, palette = 'Set1')
    plt.xticks(rotation = 90)
    plt.title(f"{col}", fontsize = 14)

plt.figure(figsize=(600,300))
for i, col in enumerate(cat_cols):
    plt.subplot(2,4, i+1)
    sns.countplot(data = df, x = col, palette = 'Set1')
    plt.xticks(rotation = 90)
    plt.title(f"{col}", fontsize = 200)

plt.figure(figsize=(600,300))
for i, col in enumerate(cat_cols):
    plt.subplot(2,4, i+1)
    sns.countplot(data = df, x = col, hue = 'Outcome', palette = 'Set1')
    plt.xticks(rotation = 90)
    plt.title(f"{col}", fontsize = 200)

plt.figure(figsize=(50,25))
for i, col in enumerate(cat_cols):
    plt.subplot(4,2, i+1)
    sns.swarmplot(data = df, x = col, y = 'Age', hue = 'Outcome', palette = 'Set1')
    plt.xticks(rotation = 90)
    plt.title(f"{col}", fontsize = 50)

sns.pairplot(df[['Insulin',  'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']],hue = 'Outcome',
palette = 'Set1', diag_kind='kde')

plt.figure(figsize = (15,12))
sns.heatmap(df.corr(), annot = True, fmt = '.2f', cmap = 'viridis', cbar = True)

df.corr()['Outcome'].sort_values(ascending = False)[1:].plot(kind = 'bar', lw = .4, color = 'blue')

df[num_cols].head()

X = df.drop('Outcome', axis = 1)
y = df['Outcome']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=21)
X_train.shape, X_test.shape

# standardize only numerical columns
scaler = StandardScaler()
X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
```

```
X_test[num_cols] = scaler.transform(X_test[num_cols])

key = ['LogisticRegression','KNeighborsClassifier','SVC','DecisionTreeClassifier','RandomForest
Classifier','GradientBoostingClassifier','AdaBoostClassifier','XGBClassifier']
value = [LogisticRegression(random_state=9), KNeighborsClassifier(), SVC(), DecisionTreeCla
ssifier(), RandomForestClassifier(), GradientBoostingClassifier()]
models = dict(zip(key,value))

cv=KFold(5, shuffle=True, random_state=21)

def model_check(X, y, classifiers, cv):

    ''' A function for testing multiple classifiers and return several metrics. '''

    model_table = pd.DataFrame()

    row_index = 0
    for cls in classifiers:

        MLA_name = cls.__class__.__name__
        model_table.loc[row_index, 'Model Name'] = MLA_name

        cv_results = cross_validate(
            cls,
            X,
            y,
            cv=cv,
            scoring=('accuracy','f1','roc_auc'),
            return_train_score=True,
            n_jobs=-1
        )
        model_table.loc[row_index, 'Train Roc/AUC Mean'] = cv_results['train_roc_auc'].mean()
        model_table.loc[row_index, 'Test Roc/AUC Mean'] = cv_results['test_roc_auc'].mean()
        model_table.loc[row_index, 'Test Roc/AUC Std'] = cv_results['test_roc_auc'].std()
        model_table.loc[row_index, 'Train Accuracy Mean'] = cv_results['train_accuracy'].mean()
        model_table.loc[row_index, 'Test Accuracy Mean'] = cv_results['test_accuracy'].mean()
        model_table.loc[row_index, 'Test Acc Std'] = cv_results['test_accuracy'].std()
        model_table.loc[row_index, 'Train F1 Mean'] = cv_results['train_f1'].mean()
        model_table.loc[row_index, 'Test F1 Mean'] = cv_results['test_f1'].mean()
        model_table.loc[row_index, 'Test F1 Std'] = cv_results['test_f1'].std()
        model_table.loc[row_index, 'Time'] = cv_results['fit_time'].mean()

        row_index += 1

    model_table.sort_values(by=['Test F1 Mean'],
                            ascending=False,
                        inplace=True)

    return model_table

raw_models = model_check(X_train, y_train, models.values(), cv)

raw_models

def f_imp(classifiers, X, y, bins):

    ''' A function for displaying feature importances'''

    fig, axes = plt.subplots(1, 2, figsize=(20, 8))
    axes = axes.flatten()

    for ax, classifier in zip(axes, classifiers):

        try:
            classifier.fit(X, y)
            feature_imp = pd.DataFrame(sorted(
                zip(classifier.feature_importances_, X.columns)),
                                       columns=['Value', 'Feature'])

            sns.barplot(x="Value",
                        y="Feature",
                        data=feature_imp.sort_values(by="Value",
                                                     ascending=False),
                        ax=ax,
                        palette='plasma')
            plt.title('Features')
            plt.tight_layout()
            ax.set(title=f'{classifier.__class__.__name__} Feature Impotances')
            ax.xaxis.set_major_locator(MaxNLocator(nbins=bins))
        except:
            continue
    plt.show()

f_imp([RandomForestClassifier(), DecisionTreeClassifier()], X_train, y_train, 6)

raw_models.columns

plt.figure(figsize = (8,5))
sns.barplot(data=raw_models, x = 'Train Accuracy Mean', y = 'Model Name', palette = 'Set1')

raw_models.set_index('Model Name', inplace = True)
```

```
plt.figure(figsize = (18,8))
raw_models[['Train Accuracy Mean','Test Accuracy Mean' ]].plot(kind = 'barh', colormap = cm.g
et_cmap('Spectral'), legend = False)

from sklearn.metrics import roc_curve, auc, confusion_matrix, classification_report,accuracy_sc
ore
lr = LogisticRegression()
lr.fit(X_train, y_train)
pred = lr.predict(X_test)

print(f'Accuracy score: {round(accuracy_score(y_test, pred) * 100, 2)} %')

print("train score - " + str(lr.score(X_train, y_train)))
print("test score - " + str(lr.score(X_test, y_test)))

#Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm_lg = confusion_matrix(y_test,pred)
sns.set(font_scale=1.4) # for label size
sns.heatmap(cm_lg, annot=True, annot_kws={"size": 16}) # font size

plt.show()

print(classification_report(y_test,pred))
```