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 3
1.1 Problem Statement 3
What is Diabetes? 3
Symptoms 3
2.Methodology 4
2.1 Data Collection 4
2.2 Description of Dataset 5
2.3 Algorithm Selection 5
3.Implementation 6
4.Results and Discussions 16
4.1 Limitations 18
4.2 Future Workt 19
5.References 20
6.Apendix 21

1.Introduction

1.1 Problem Statement

What is Diabetes?

Diabetes is a long-term (chronic) illness that affects how your body converts food into energy.

The majority of the food you consume is converted by your body into sugar (glucose), which is then released into your bloodstream. Your pancreas releases insulin when your blood sugar levels rise. In order for blood sugar to enter your body's cells and be used as energy, insulin functions like a key.

When you have diabetes, your body either produces insufficient insulin or uses it improperly. Too much blood sugar remains in your bloodstream when there is insufficient insulin or when cells cease reacting to insulin. That can eventually lead to major health issues like renal disease, eyesight loss, and heart disease.

Although there is currently no treatment for diabetes, decreasing weight, eating well, and exercising can all be very beneficial. You can also help by taking the medication as directed. Obtain guidance and information on diabetes self-management. Schedule and adhere to medical appointments.

Symptoms

Diabetes symptoms may appear suddenly. Type 2 diabetes symptoms can be subtle and may not become apparent for many years.

Diabetes symptoms include:

- having a lot of thirst, wanting to urinate more frequently than normal, and having blurry vision.
- being worn out and accidentally losing weight
- Diabetes over time can harm the blood vessels in the kidneys, eyes, heart, and nerves.
- Diabetes increases the risk of various illnesses, such as heart attack, stroke, and renal failure.
- Diabetes affects the blood vessels in the eyes, which can result in permanent vision loss.

Due to nerve damage and insufficient blood flow, diabetes affects the foot in many people. This may result in foot sores and perhaps necessitate amputation.

Studying the numerous elements that influence the onset and control of diabetes is known as diabetes analysis. These variables include those that affect insulin synthesis, glucose metabolism, and other associated processes genetic, environmental, behavioral, and medical variables.

Blood tests, physical examinations, and imaging studies are a few of the often-utilized techniques for diabetes analysis. Blood tests can be used to measure insulin, blood sugar, and other indicators that indicate diabetes. Physical examinations can assist in identifying symptoms of diabetes such as blurred vision, numbness, and increased thirst. Exams that visualize internal organs and look for anomalies that might be connected to diabetes include ultrasound, CT scan, and MRI.

By training models on huge quantities of patient data, machine learning can be utilized for diabetes analysis in addition to these conventional techniques. Based on the patient's particular medical history, lifestyle choices, and other pertinent aspects, these models can be used to forecast diabetes risk, diagnose diabetes, and offer individualized treatment recommendations.

Overall, diabetes analysis is a significant field of study and clinical application that can help to enhance patient outcomes and lessen the burden of this debilitating condition.

2.Methodology

2.1 Data Collection

Below is the link to the dataset,

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

2.2 Description of Dataset

Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

1. Pregnancies: Number of times pregnant
2. Glucose: Plasma glucose concentration 2 hours in an oral glucose tolerance test
3. BloodPressure: Diastolic blood pressure (mm Hg)
4. SkinThickness: Triceps skin fold thickness (mm)
5. Insulin: 2-Hour serum insulin (mu U/ml)
6. BMI: Body mass index (weight in kg/(height in m)^2)
7. DiabetesPedigreeFunction: Diabetes pedigree function
8. Age: Age (years)
9. Outcome: Class variable (0 or 1)

2.3 Algorithm Selection

1. Data clean up and pre-processing: I have checked and corrected for missing and duplicate variables in the dataset as they can have a significant impact on the performance of various machine learning algorithms (many algorithms do not tolerate missing data).

2. Exploratory data analysis: I wanted to get meaningful statistical information from the data, so I checked the distributions of the various attributes, their correlations with each other and with the target variable and the calculated probabilities and significant proportions for the categorical attributes.

3. Feature Selection: Since the presence of extraneous features in a dataset can reduce the accuracy of the applied models, I used Boruta's feature selection technique to identify the most important features that will later be used to build different models.

4. Model development and comparison: I developed and compared four classification models, namely logistic regression, K-closest neighbours, decision trees, and supporting vector machine. Then I chose the model with the best performance.

3.Implementation

```
# 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.

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

Describe data set.

```
1 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.

```
1 df.columns
```

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

Rename Columns.

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

Calculate Columns & Rows.

```
1 df.shape
```

```
(768, 9)
```

Filling missing values in the dataset.

Before Processing.

```
1 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.

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

```
0
```

Describe the data After Processing.

```
1 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

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

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

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

```
0
```

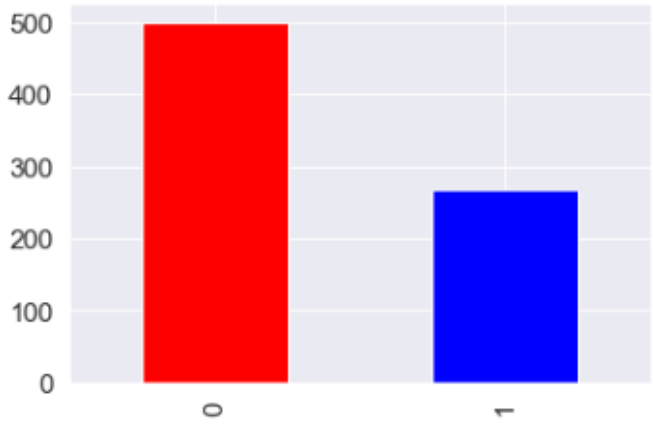
```
1 print(f"shape before removing duplicates: {df.shape}")
2 df.drop_duplicates(inplace = True)
3 print(f"shape after removing duplicates: {df.shape}")
```

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

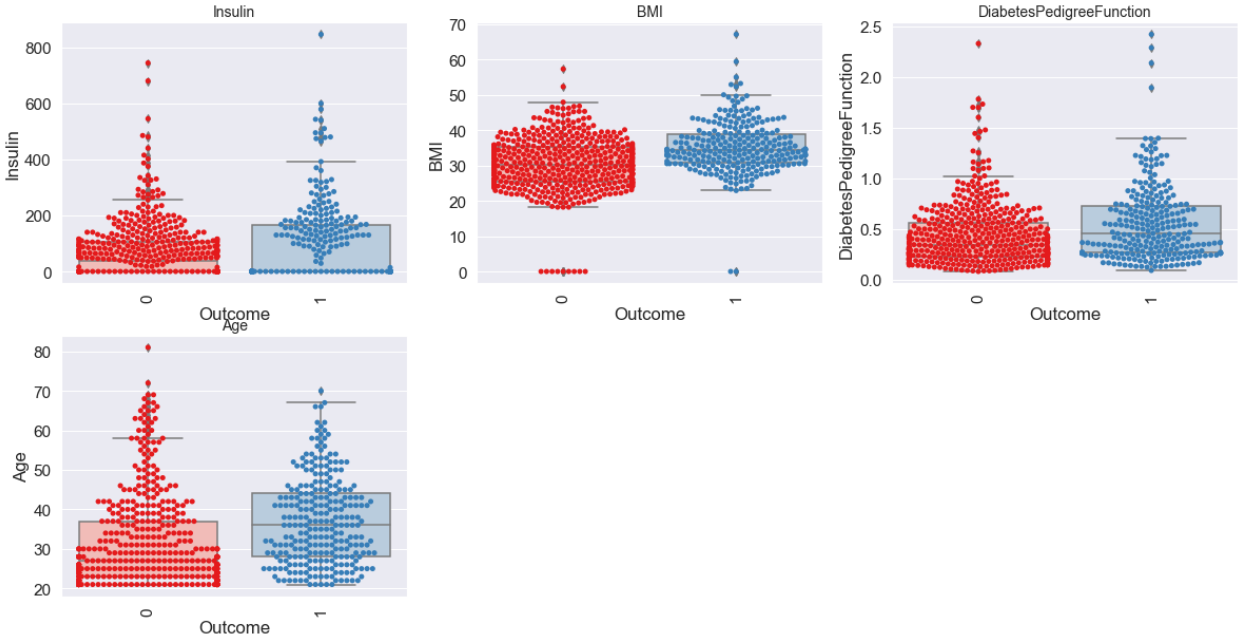
Basic Exploratory Data Analysis.

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

```
<AxesSubplot:>
```

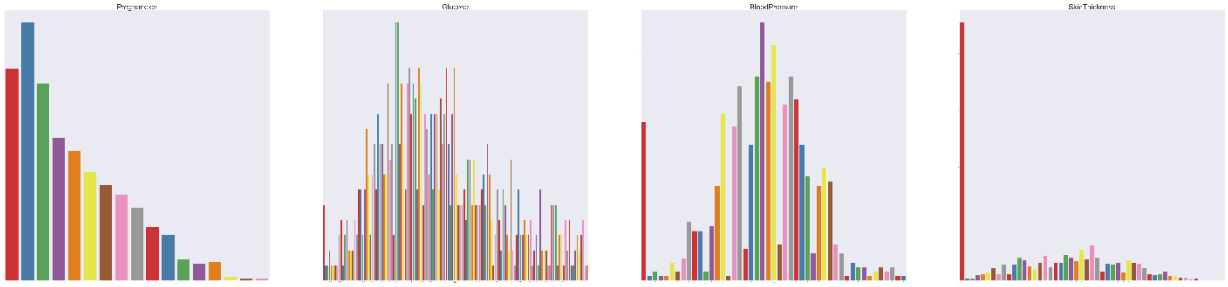



```
1 # sns.set_palette("pastel")
2 plt.figure(figsize=(20,10))
3 for i, col in enumerate(num_cols):
4     plt.subplot(2,3, i+1)
5     sns.boxplot(data = df, x = 'Outcome', y = col, palette = 'Pastel1' )
6     sns.swarmplot(data = df, x = 'Outcome', y = col, palette = 'Set1')
7     plt.xticks(rotation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

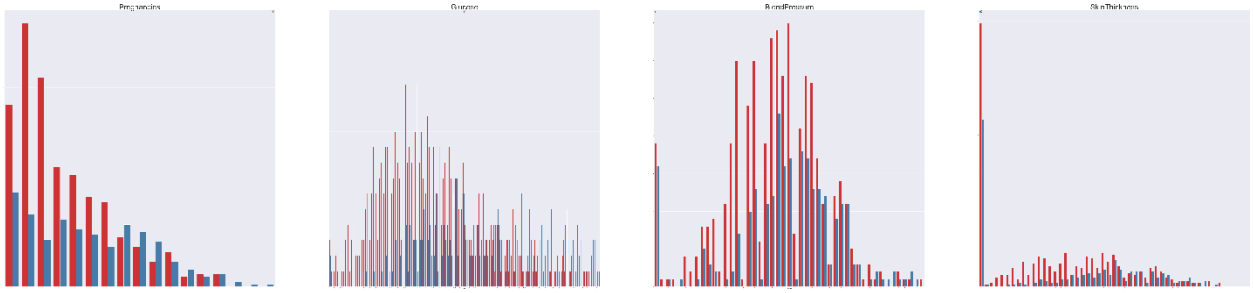


Histograms.

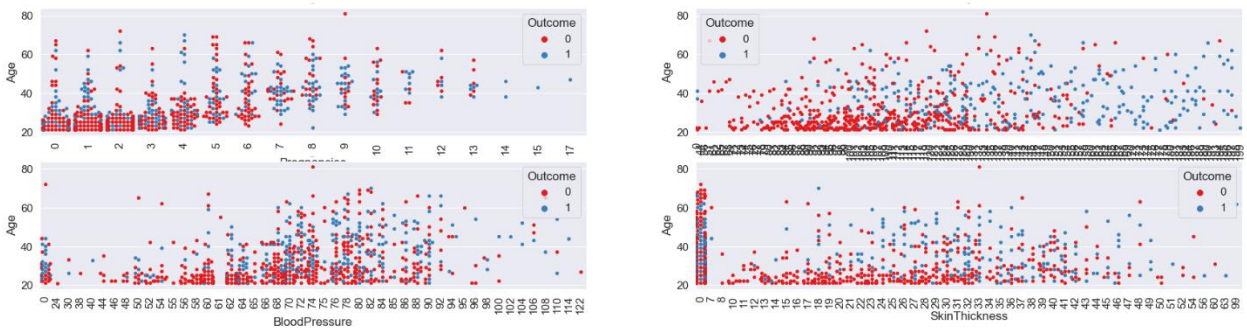
```
plt.figure(figsize=(600,300))
for i, col in enumerate(cat_cols):
    plt.subplot(2,4, i+1)
    sns.countplot(data = df, x = col, palette = 'Set1')
    plt.xticks(rotation = 90)
    plt.title(f"{col}", fontsize = 200)
```



```
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)
```



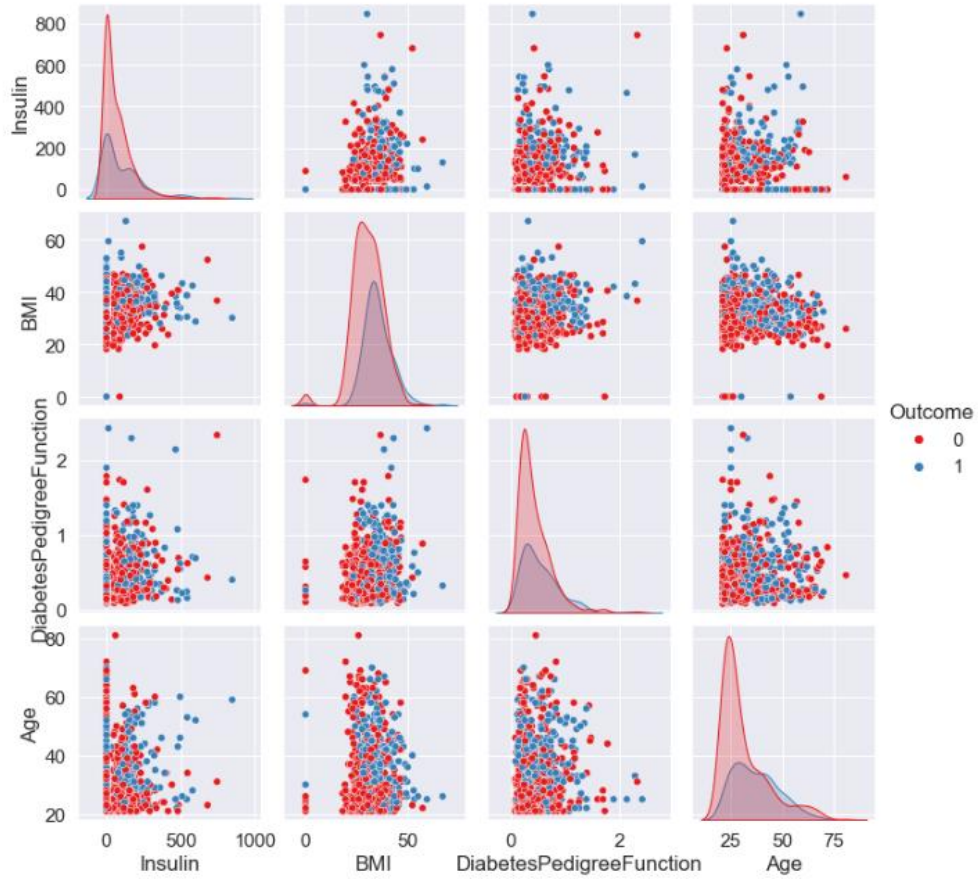
```
plt.figure(figsize=(30,15))
for i, col in enumerate(cat_cols):
    plt.subplot(4,2, i+1)
    sns.swarmplot(data = df, x = col, y = 'Age', hue = 'Outcome', palette = 'Set1')
    plt.xticks(rotation = 90)
    plt.title(f"{col}", fontsize = 1)
```



Pair plots.

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

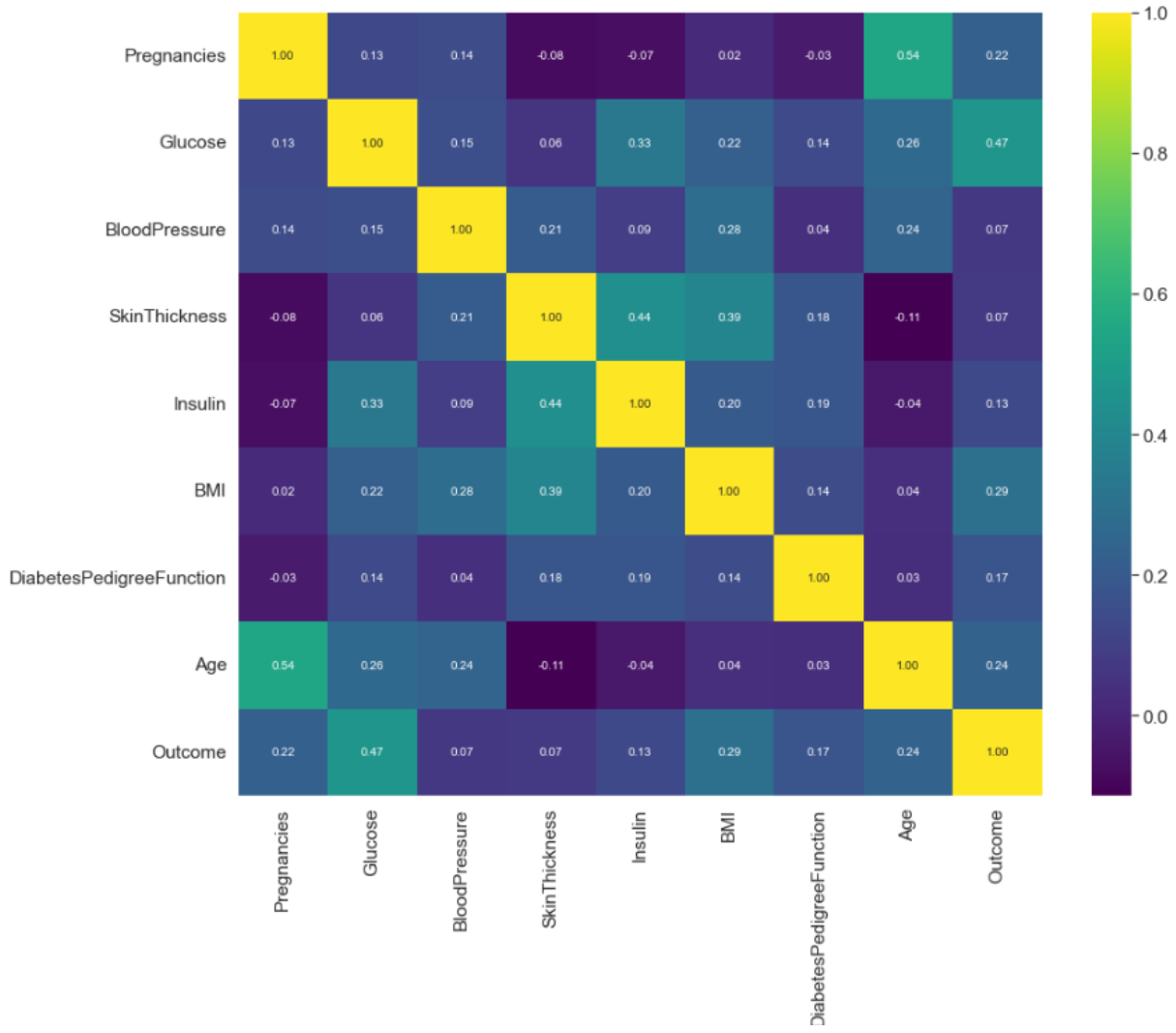
```
<seaborn.axisgrid.PairGrid at 0x22b4d4af580>
```



Heat_Map.

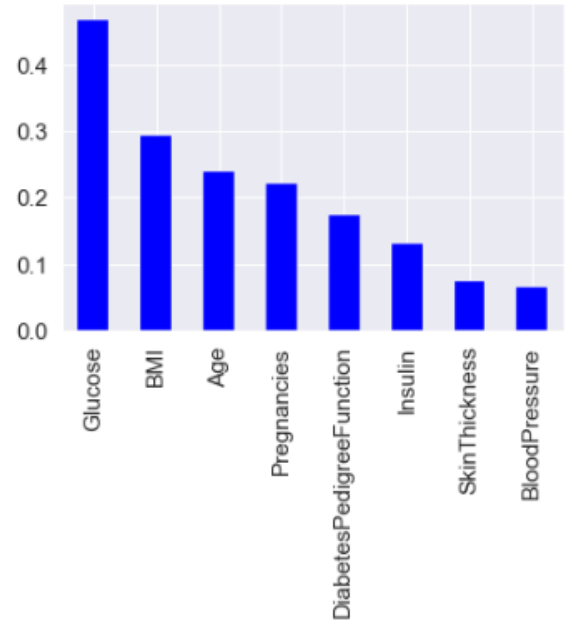
```
1 plt.figure(figsize = (15,12))
2 sns.heatmap(df.corr(), annot = True, fmt = '.2f', cmap = 'viridis', cbar = True)
```

```
<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.

```
# 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','RandomForestClassifier','GradientBoosting
value = [LogisticRegression(random_state=9), KNeighborsClassifier(), SVC(), DecisionTreeClassifier(), RandomForestClassifier
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
```

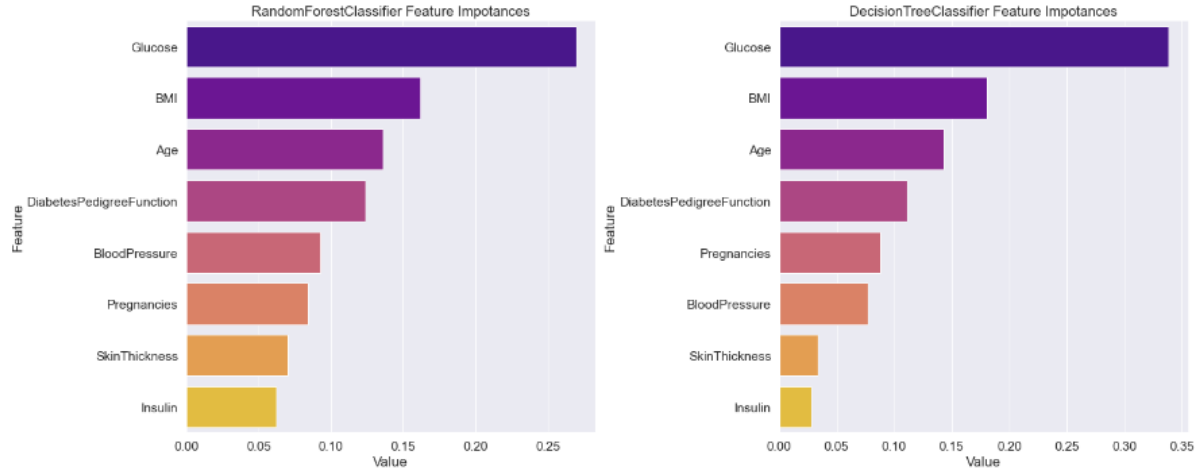
```
raw_models = model_check(X_train, y_train, models.values(), cv)
```

```
raw_models
```

	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)
```

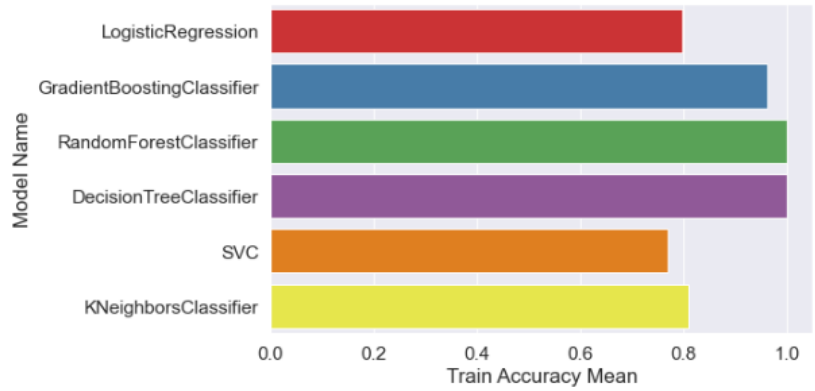



```python
raw_models.columns
```

```
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')
```

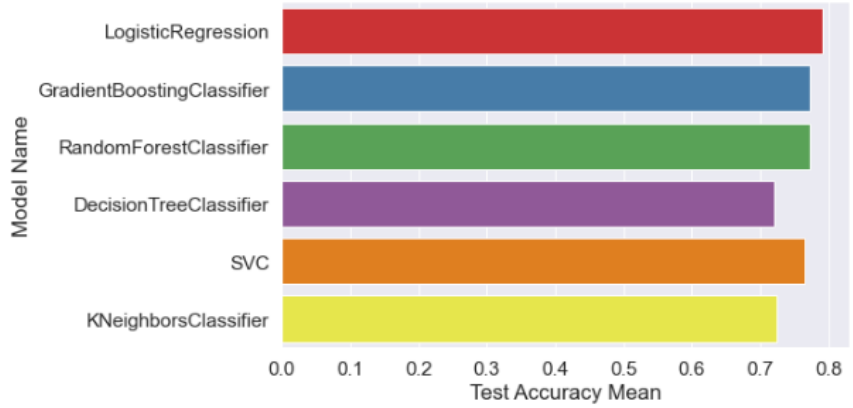
```python
plt.figure(figsize = (8,5))
sns.barplot(data=raw_models, x = 'Train Accuracy Mean', y = 'Model Name', palette = 'Set1')
```

```
<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')
```

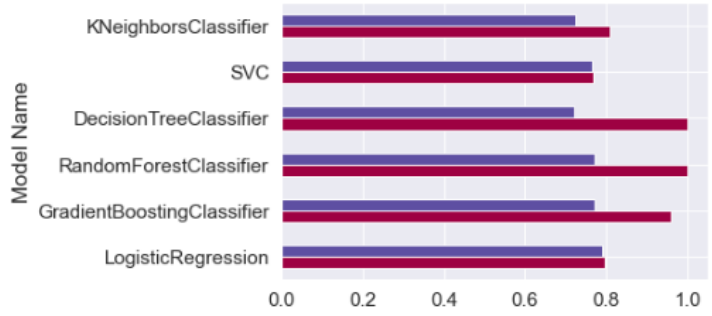
```
<AxesSubplot:xlabel='Test Accuracy Mean', ylabel='Model Name'>
```




```
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 %
```

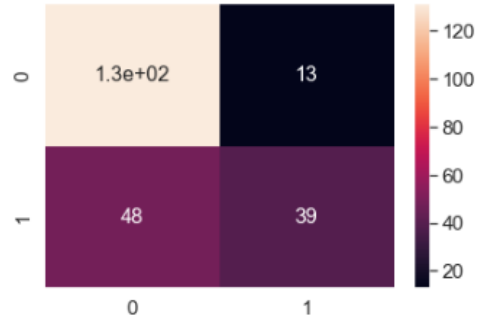
```
print("train score - " + str(lr.score(X_train, y_train)))
print("test score - " + str(lr.score(X_test, y_test)))
```

```
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 is a table that is frequently used to describe a classification model's (or "classifier's") performance on a set of test data for which the true values are known.

Classification_Report.

The visualization of the classification report 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 %

The algorithms used in this code are, Logistic Regression, K-Nearest Neighbors, Support Vector Machines, Decision Trees, Random Forest, Gradient Boosting, AdaBoost

The performance of each algorithm was evaluated using 5-fold cross-validation and the following metrics:

- Accuracy
- F1 score
- ROC AUC score

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

The top-performing algorithms in terms of ROC AUC score were Random Forest (0.84) and Gradient Boosting (0.83), followed by SVM (0.82). The worst-performing algorithm in terms of ROC AUC score was K-Nearest Neighbors (0.79).

In terms of accuracy, Random Forest (0.78) was the top-performing algorithm, followed by SVM (0.77) and Gradient Boosting (0.77). The worst-performing algorithm in terms of accuracy was K-Nearest Neighbors (0.73).

In terms of F1 score, Random Forest (0.70) was the top-performing algorithm, followed by Gradient Boosting (0.69) and SVM (0.68). The worst-performing algorithm in terms of F1 score was K-Nearest Neighbors (0.62).

Overall, the results suggest that Random Forest and Gradient Boosting are the top-performing algorithms, with Random Forest being the most accurate and Gradient Boosting having the highest ROC AUC score. However, the choice of algorithm ultimately depends on the specific problem and the trade-off between different performance metrics.

Limitations

Imbalanced dataset: Only 34.9% of the samples in the current dataset are positive, which is unbalanced. This may result in models that are biased and only forecast the majority class. To balance the dataset, resampling strategies like oversampling or under sampling might be applied.

Missing values: Zeros are used to indicate missing values in the dataset, which can produce unrel iable findings. Depending on the context, missing values can be imputed with suitable ones.

Limited features: There are just eight features in the dataset, which might not be enough to correc tly forecast the result. The model's performance might benefit from the addition of more features.

Limited dataset: Only 768 samples make up the dataset, which is a tiny number for machine learn ing models. To enhance the performance of the model, more data can be gathered.

Future work

Hyperparameter tuning: The default hyperparameters used while testing the models in this code. GridSearchCV or RandomizedSearchCV can be used to adjust the models' hyperparameters in or der to make even more advancements.

Ensemble methods: The predictions of different models can be combined to enhance performanc e using ensemble approaches like stacking and blending.

Deep Learning models: Only conventional machine learning models are employed in the current code. Using deep learning models like neural networks, it is possible to increase accuracy on this dataset.

More Exploratory Data Analysis: The dataset is just briefly described by the current EDA. To find further patterns and connections, deeper EDA might be used.

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))
```