Machine Learning Assignment

Diabetes Analysis

Submitted By:

Student Name	Student IT NO
Jayasena A.K.B.D	IT21073946
Achchige R.A.N.R	IT21102264
Abegunasekara P.W.A.B.N	IT21104176
Padmasiri P.G.S.M	IT21078996

Contents

1.Introduction

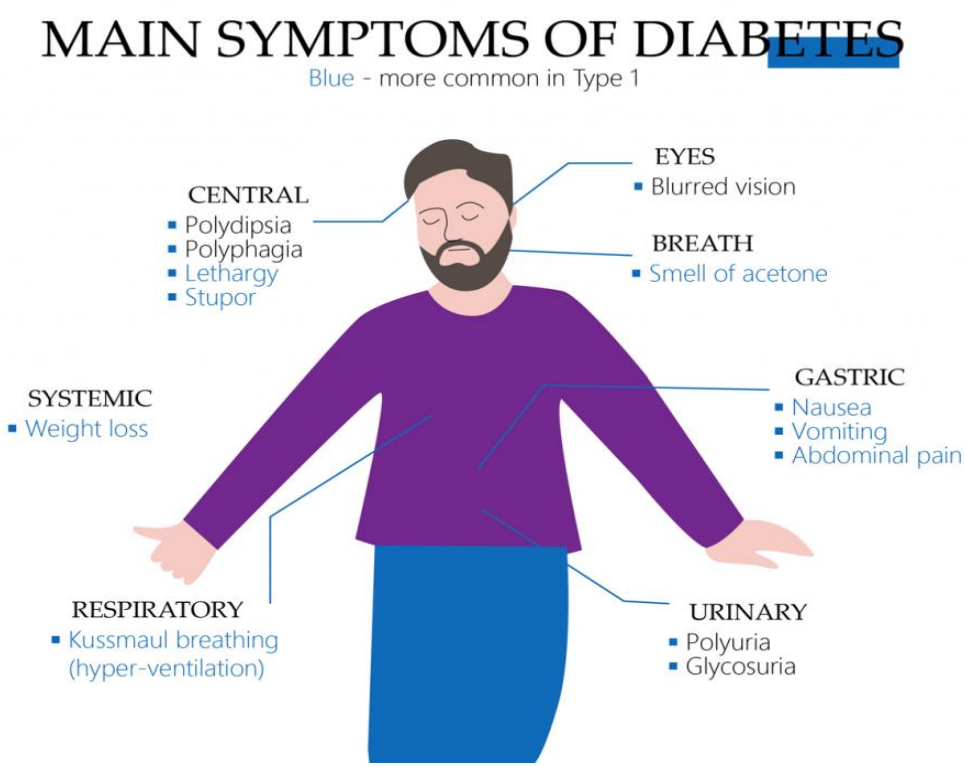
1.1 Problem Statement

What is Diabetes?

Diabetes is a long-term medical disorder marked by high blood glucose (sugar) levels. Over time, diabetes can cause major harm to the heart, blood vessels, kidneys, eyes, nerves, and heart. Type 1, Type 2, and gestational diabetes are the three most prevalent forms of the disease.

- **Type 1** Diabetes is an autoimmune disease in which the pancreatic beta cells that produce insulin are attacked by the body, resulting in little or no insulin being produced. It is more often diagnosed in children and young adults and calls for daily insulin treatment.
- The main cause of **type 2** diabetes, which is more common, is the body's resistance to insulin or the pancreas' inability to make enough of it. It is more common in adults and has a strong correlation with obesity, while it is becoming more prevalent in younger age groups.
- Although **gestational diabetes** normally goes away after giving birth and develops throughout pregnancy, it can greatly raise the risk of type 2 diabetes in the future for both the mother and the child.

Symptoms



The severity and type of the disease, as well as the length of time that blood sugar levels are elevated, all influence the symptoms of diabetes. Common symptoms include:

- Higher blood sugar levels drive the body to produce more pee, which increases the demand for fluid consumption. These two factors combined result in increased **thirst and urination**.
- **Fatigue**: A persistently low level of energy is caused by insufficient sugar entering the body's cells from the bloodstream.
- **Vision impairments**: Excessive glucose levels can harm the retina's blood vessels, resulting in vision impairments and other visual abnormalities.
- **Unexpected weight loss**: When a person has type 1 diabetes, their pancreas stops making insulin, which forces their body to burn fat and muscle for energy. This leads to an unexpected drop in weight.
- **Increased hunger**: Because insulin cannot properly carry glucose into the cells, persons with diabetes may experience extreme hunger even when they consume more.
- Diabetes impairs the body's capacity to heal and recover from infections, resulting in **slow-healing wounds or recurrent infections**.
- **Hands and foot tingling**: Nerve damage brought on by an excess of sugar in the blood can result in tingling or numbness in the extremities.

Treatment and Management of Diabetes

Managing diabetes requires a comprehensive approach that includes lifestyle management, regular monitoring, medication, and sometimes insulin therapy. Key strategies include:

- **Modifications to lifestyle**: Exercise and diet are essential for controlling diabetes. It is often advised to have a diet high in fiber and low in calories, saturated fat, and carbs. Regular exercise enhances insulin sensitivity and aids in maintaining a healthy weight.
- **Blood sugar monitoring**: Those with diabetes can modify their diets, activities, and prescription regimens to keep their blood sugar levels within a normal range by regularly checking their levels.
- **Medication**: Individuals with type 2 diabetes frequently need to take medicine that helps to either produce more insulin, lessen the production of sugar, or enhance the effectiveness of insulin. Initially, metformin is usually prescribed if blood glucose levels persist, more drugs may be taken.
- **Insulin therapy**: Some individuals with type 2 diabetes and all individuals with type 1 diabetes require insulin therapy. Several types of insulin are available, including rapid-acting insulin, long-acting insulin, and intermediate options.
- **Education and support**: Individuals can better understand and manage their diabetes condition with the assistance of ongoing diabetes education, dietary counseling, and support groups.

Prevention

Being the most preventable kind of diabetes, type 2 diabetes, preventive actions are especially crucial. Keeping a healthy weight through nutrition and exercise on a regular basis is the best defense against diabetes. Regular prediabetes screening can also result in early intervention and stop the progression to type 2 diabetes, especially if you have a family history or other risk factors.

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: The total number of pregnancies the patient has had
2. Glucose: The plasma glucose concentration measured after a two-hour oral glucose tolerance test.
3. BloodPressure: The diastolic blood pressure of the patient measured in millimeters of mercury (mm Hg).
4. SkinThickness: The triceps skin fold thickness measured in millimeters. This measurement can indicate body fat percentage.
5. Insulin: The two-hour serum insulin level measured in micro-units per milliliter (mu U/ml).
6. BMI: Body Mass Index, which provides a heuristic measure of body weight based on the person's weight in relation to their height. It is expressed in kg/m^2.
7. Diabetes Pedigree Function: A score that represents how much the diabetes is attributed to genetics. This score can help assess the likelihood of diabetes based on family history.
8. Age: The age of the patient in years
9. Outcome: The class variable with 0 or 1 indicating negative or positive diabetes diagnosis respectively.

2.3 Algorithm Selection

Data Cleaning and Preprocessing

We started our study by getting the dataset ready for modeling and exploration. Pre-processing and data cleaning are essential to guaranteeing that the machine learning models' predictions are accurate. The following tasks were part of the process:

- **Handling Outliers**: Outliers have the potential to distort the results for continuous variables like age, diabetes pedigree function, BMI, and insulin. To lessen the impact of severe outliers, we computed the interquartile range (IQR) for these variables and capped values at 1.5 times the IQR above the third quartile and below the first quartile.
- **Feature engineering**: In order to capture the interaction between BMI and Age, we multiplied the two variables to produce a new feature called BMI_Age. This feature may improve the model's capacity to predict diabetes.
- **Standardization**: We applied standard scaling to numerical features to ensure they contribute equally to the analysis by bringing them onto a similar scale. For models like SVM that are sensitive to the magnitude of the input features, this is especially crucial.
- **Handling Duplicates and Missing Values**: In order to avoid including redundant data into our models, we eliminated duplicate entries. Furthermore, we verified that our dataset was complete for analysis and looked for any missing values.

Exploratory data analysis

Exploratory Data Analysis was conducted to understand the distributions of various features and the relationships between them:

- **Distribution Plots**: We visualized the distribution of each numerical feature using boxplots to identify the spread and central tendencies, alongside identifying any further outliers.
- **Correlation Analysis**: Heatmaps displaying correlation coefficients between characteristics were created using correlation analysis to help visualize the connections and interdependencies between various variables.
- **Outcome Visualization**: To comprehend the class balance between diabetic and non-diabetic entries in the dataset, the distribution of the diabetes outcome variable was shown.

Feature Selection

The process of feature selection entailed determining the significance and applicability of various elements within the dataset.

- **Correlation with Outcome**: Features that had a strong correlation with the outcome variable were given priority in terms of correlation. To find and keep the features that were most predictive of the outcome, we employed correlation metrics.
- **Feature Importance**: To help refine the model and concentrate on important predictors, we used the intrinsic feature importance for models such as Random Forest to determine which features have the most impact on predicting the target variable.

Model construction and comparison

To determine which machine learning model was best for predicting diabetes, a number of them were created and assessed:

- **Model Selection**: We selected a range of models, including RandomForest, SVM, K-Neighbors Classifier, and Logistic Regression, that are appropriate for binary classification applications. Each model was selected based on its suitability to handle binary outcome prediction.
- **Hyperparameter Tuning**: To maximize the performance of models such as RandomForest and SVM, we used GridSearchCV to do hyperparameter tuning. For RandomForest, we explored with various parameter settings such as n_estimators and max_depth, for SVM, we experimented with C, kernel, and gamma.
- **Evaluation of the Models**: A number of metrics, such as accuracy, F1-score, and ROC-AUC score, were used to assess the models. This thorough assessment made it easier to compare models based on their accuracy as well as their capacity to strike a balance between precision and recall.
- **Features' Importance Visualization**: To shed light on the data and the behavior of the models, we used tree-based models to illustrate the significance of various features in generating predictions.
- **Justification why used.**

1. **Logistic Regression**

   Introduction**:** Because logistic regression is so good at providing probabilities and is so simple to understand, it is frequently utilized in healthcare settings for binary classification issues like diabetes prediction.

   Background: This model helps explain how variables affect the likelihood of developing diabetes by offering coefficients that provide the odds ratio for each predictor variable. It is particularly well-suited for binary outcomes. It is a mainstay in medical predictive analytics due to its resilience and simplicity.

2. **K-Nearest Neighbors (KNN)**

   Introduction**:** K-Nearest Neighbors is a non-parametric method used for classification and regression. Based on similarities in case profiles within the training set, it aids in the classification of patients for diabetes prediction.

   Background: KNN was selected due to its capacity to accurately represent the complicated nature of smaller datasets where similar cases can suggest similar outcomes. Since it doesn't make any assumptions about the data's form, it can be applied to real-world medical data that frequently deviates from theoretical distributions.

3. **Support Vector Machines (SVM)**

Introduction**:** SVM is favored in binary classification tasks due to its ability to model non-linear decision boundaries thanks to various kernel functions. The ability to handle large feature spaces and control overfitting through regularization are particularly valuable in complex diseases like diabetes.
.
Background: SVM is robust to outliers and effective in high dimensional spaces, which is typical in medical datasets where many variables might influence the outcome.

4. **Random Forest**

Introduction**:** Random Forest is an ensemble technique that enhances decision trees' simplicity by minimizing their propensity to overfit, which is essential for preserving the model's performance on fresh, untainted data.

Background: Random Forest is especially good at handling the complex nature of diabetes risk factors because of its capacity to handle huge datasets with numerous input features. It provides good feature importance estimations and accuracy, which are useful for analyzing the influence of different predictors.

3.Implementation

Importing the libraries.

In [1]:
```
# 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, KFold, cross_validate , GridSearchCV


# classifiers
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score, classification_report, confusion_matrix
```

Read data set.

```
In [2]: ds = pd.read_csv('diabetes.csv')
```

Describe data set.

```
In [5]: ds.head()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	BMI_Age
0	6	148	72	35	0.0	33.6	0.627	50.0	1	1680.0
1	1	85	66	29	0.0	26.6	0.351	31.0	0	824.6
2	8	183	64	0	0.0	23.3	0.672	32.0	1	745.6
3	1	89	66	23	94.0	28.1	0.167	21.0	0	590.1
4	0	137	40	35	168.0	43.1	1.200	33.0	1	1422.3

Describe Columns.

```
[6]: ds.columns
```

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

Categorizing Columns.

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

Calculate Columns & Rows.

```
[8]: ds.shape
```

```
[8]: (768, 10)
```

Filling missing values in the dataset.

Before Processing.

```
In [9]: ds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 10 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    float64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    float64
 8   Outcome                   768 non-null    int64
 9   BMI_Age                   768 non-null    float64
dtypes: float64(5), int64(5)
memory usage: 60.1 KB
```

After Processing.

```
[10]: ds.isnull().any().sum()
```

```
[10]: 0
```

Describe the data After Processing.

```
In [11]: ds.describe().T
```

Out[11]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.000
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.000
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.000
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.000
Insulin	768.0	73.652669	93.576029	0.000	0.00000	30.5000	127.25000	318.125
BMI	768.0	32.125065	7.049584	13.350	27.30000	32.0000	36.60000	50.550
DiabetesPedigreeFunction	768.0	0.458914	0.285596	0.078	0.24375	0.3725	0.62625	1.200
Age	768.0	33.199870	11.628404	21.000	24.00000	29.0000	41.00000	66.500
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.000
BMI_Age	768.0	1070.099056	438.590103	280.350	741.00000	986.0000	1344.80000	2697.000

```
In [12]: ds['Outcome'].value_counts()
```

```
Out[12]: Outcome
         0    500
         1    268
         Name: count, dtype: int64
```

```
In [13]: ds.duplicated().sum()
```

```
Out[13]: 0
```

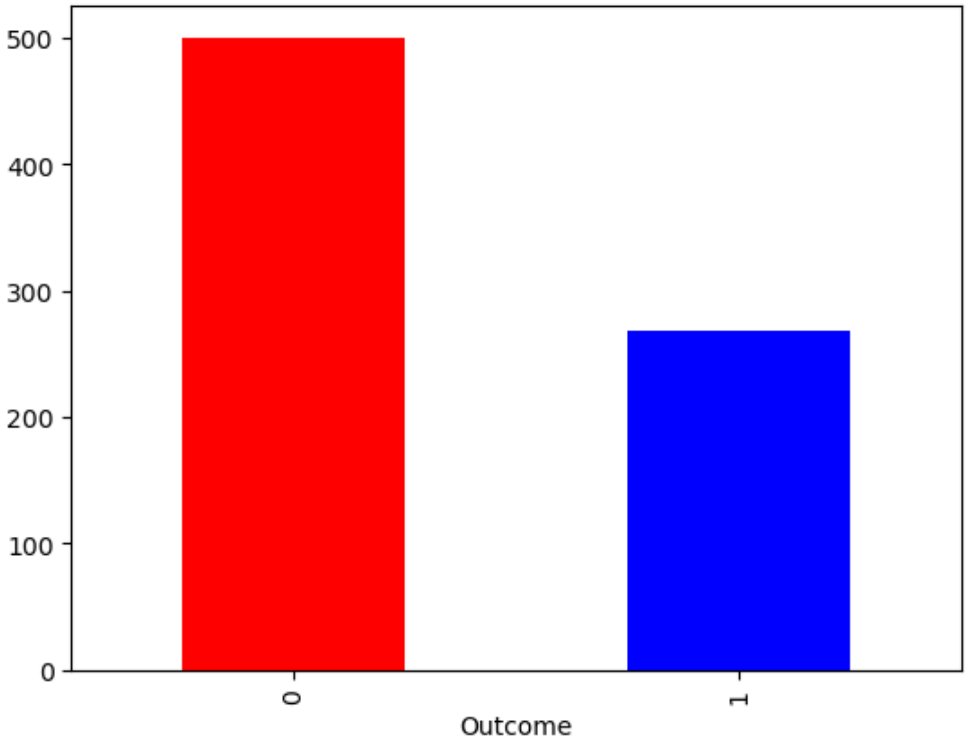
```
In [14]: print(f"shape before removing duplicates: {ds.shape}")
         ds.drop_duplicates(inplace = True)
         print(f"shape after removing duplicates: {ds.shape}")
```

```
shape before removing duplicates: (768, 10)
shape after removing duplicates: (768, 10)
```

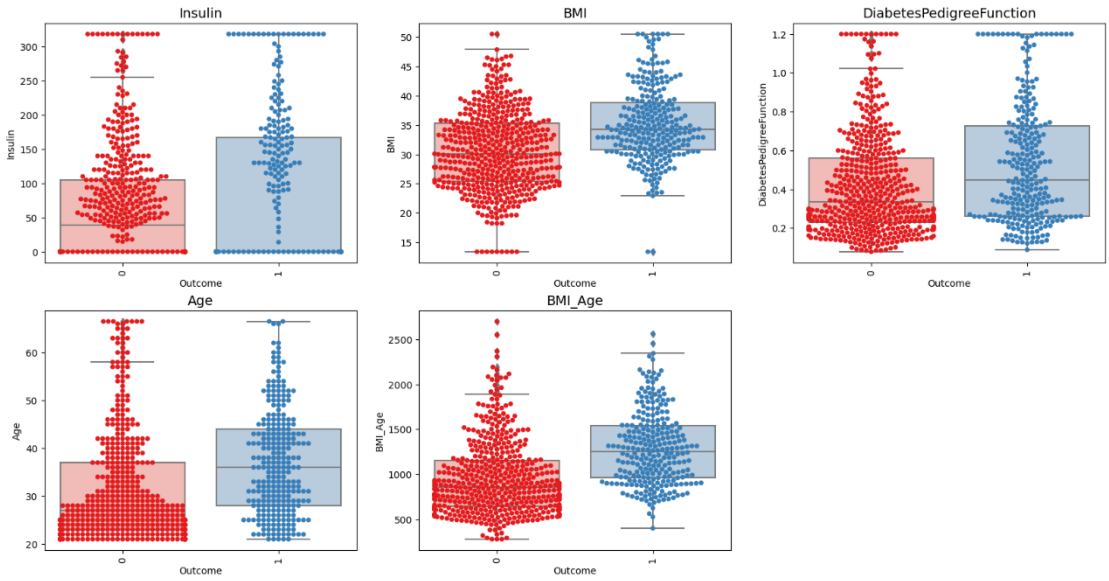
Basic Exploratory Data Analysis.

In [15]:
```
ds['Outcome'].value_counts().plot(kind = 'bar', color=['red', 'blue'])
```

Out[15]: <Axes: xlabel='Outcome'>

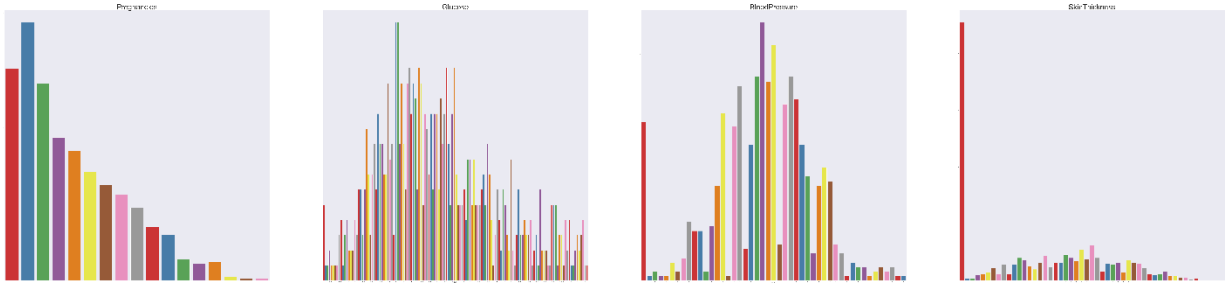


In [16]:
```
# sns.set_palette("pastel")
plt.figure(figsize=(20,10))
for i, col in enumerate(num_cols):
    plt.subplot(2,3, i+1)
    sns.boxplot(data = ds, x = 'Outcome', y = col, palette = 'Pastel1' )
    sns.swarmplot(data = ds, x = 'Outcome', y = col, palette = 'Set1')
    plt.xticks(rotation = 90)
    plt.title(f"{col}", fontsize = 14)
```

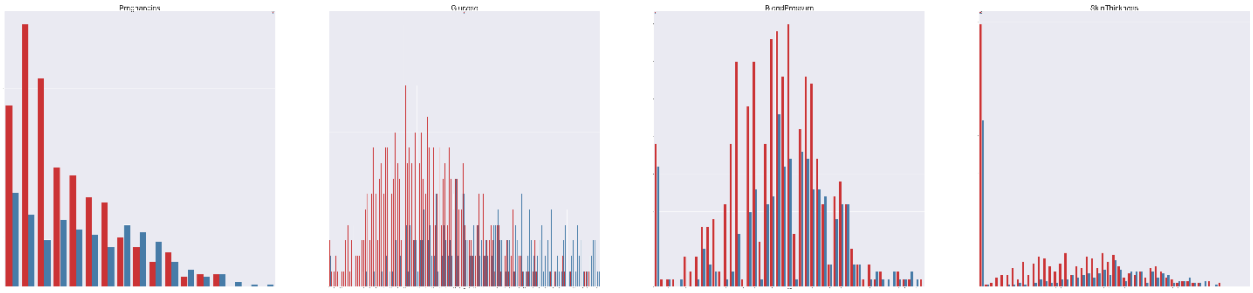


Histograms.

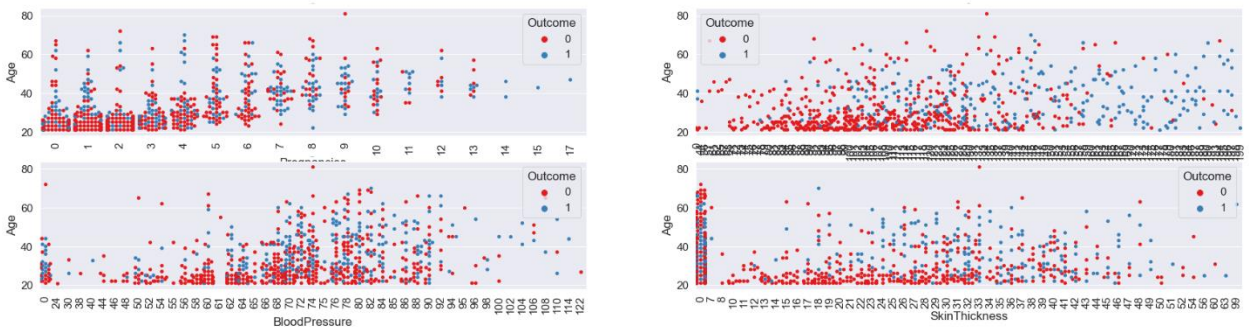
```
plt.figure(figsize=(600,300))
for i, col in enumerate(cat_cols):
    plt.subplot(2,4, i+1)
    sns.countplot(data = df, x = col, palette = 'Set1')
    plt.xticks(rotation = 90)
    plt.title(f"{col}", fontsize = 200)
```



```
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)
```



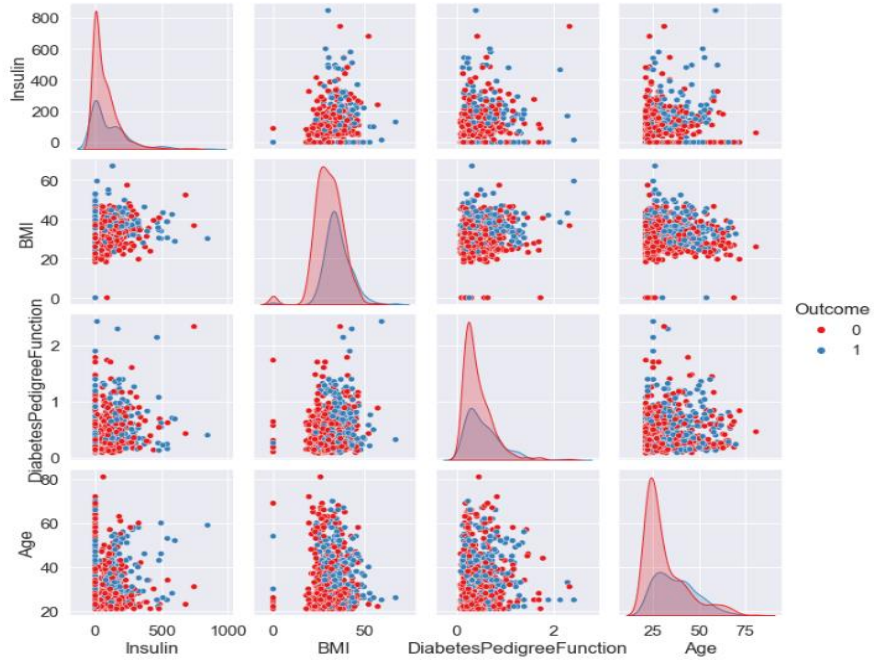
```
plt.figure(figsize=(30,15))
for i, col in enumerate(cat_cols):
    plt.subplot(4,2, i+1)
    sns.swarmplot(data = df, x = col, y = 'Age', hue = 'Outcome', palette = 'Set1')
    plt.xticks(rotation = 90)
    plt.title(f"{col}", fontsize = 1)
```



Pair plots.

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

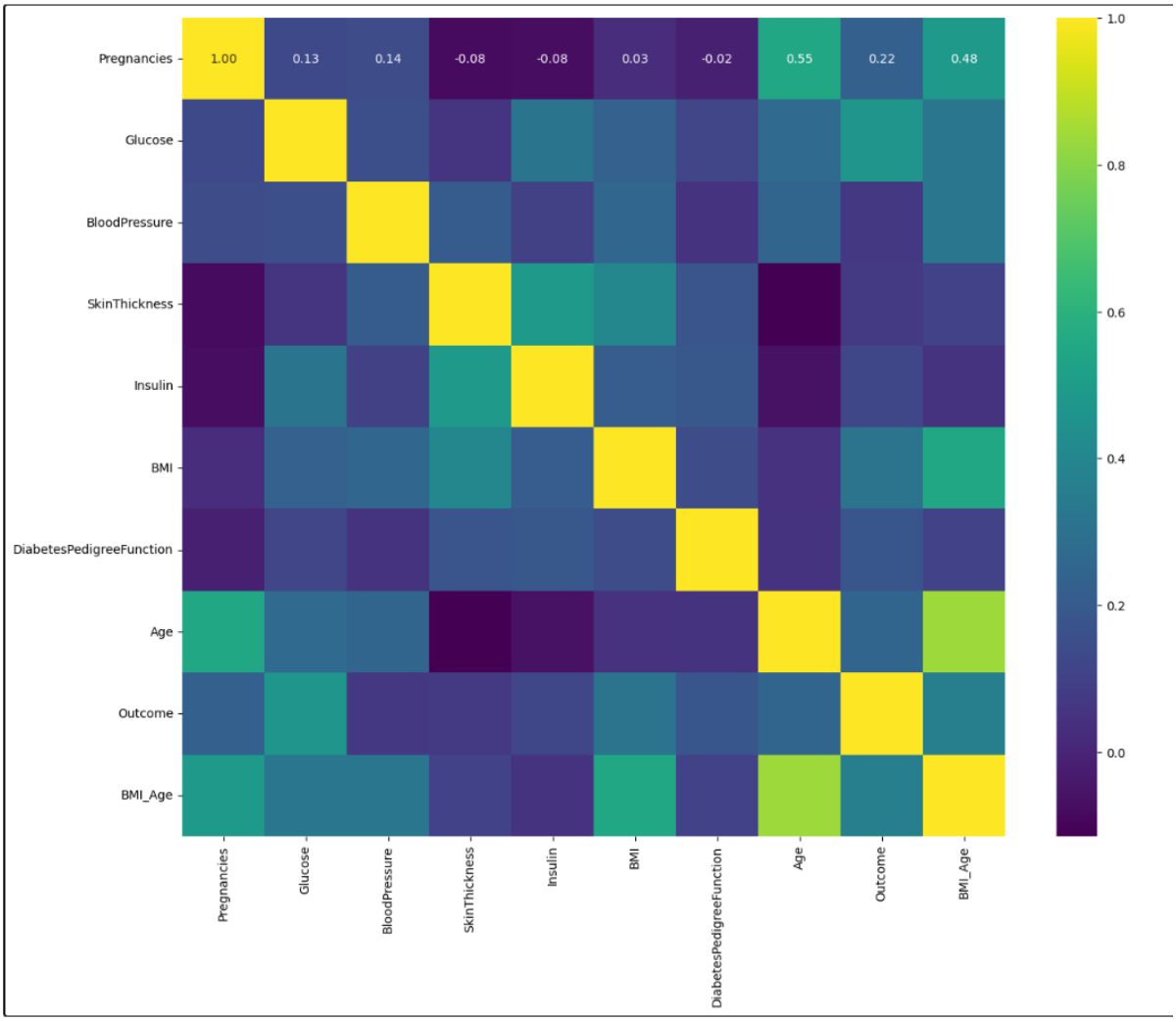
```
<seaborn.axisgrid.PairGrid at 0x22b4d4af580>
```



Heatmap.

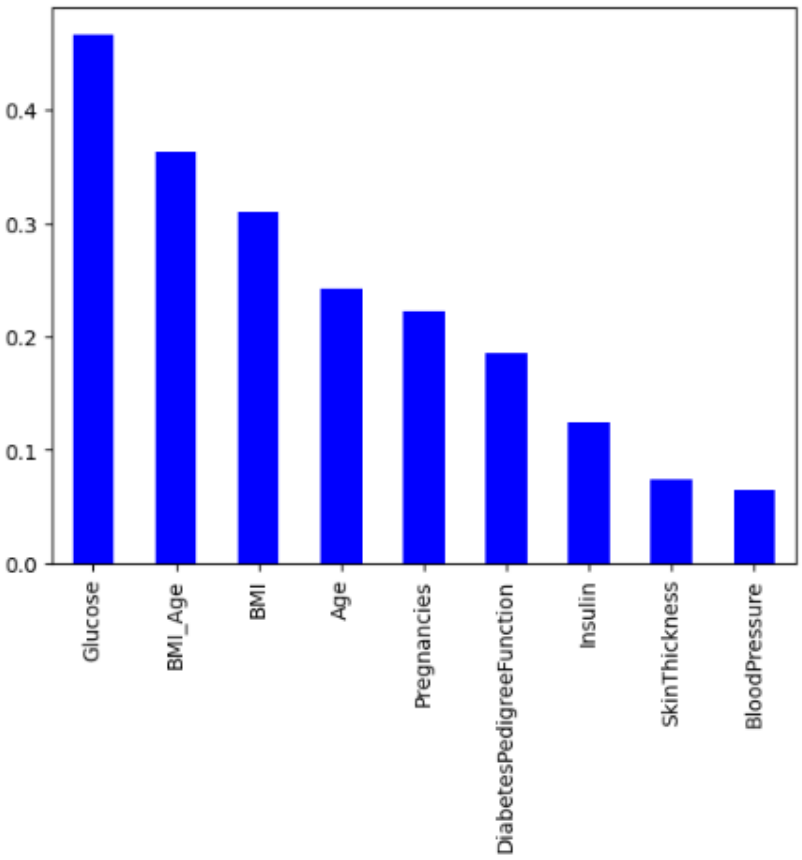
```
In [20]: plt.figure(figsize = (15,12))
         sns.heatmap(ds.corr(), annot = True, fmt = '.2f', cmap = 'viridis', cbar = True)
```

```
Out[20]: <Axes: >
```



In [21]:
```
ds.corr()['Outcome'].sort_values(ascending = False)[1:].plot(kind = 'bar', lw = .4, color = 'blue')
```

Out[21]: <Axes: >



In [22]:
```
ds[num_cols].head()
```

Out[22]:

	Insulin	BMI	DiabetesPedigreeFunction	Age	BMI_Age
0	0.0	33.6	0.627	50.0	1680.0
1	0.0	26.6	0.351	31.0	824.6
2	0.0	23.3	0.672	32.0	745.6
3	94.0	28.1	0.167	21.0	590.1
4	168.0	43.1	1.200	33.0	1422.3

Data Pre-processing.

Handling Outliers

```
[3]: # Handling Outliers
for col in ['Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']:
    q1 = ds[col].quantile(0.25)
    q3 = ds[col].quantile(0.75)
    iqr = q3 - q1
    fence_low = q1 - 1.5 * iqr
    fence_high = q3 + 1.5 * iqr
    ds[col] = ds[col].clip(lower=fence_low, upper=fence_high)
```

Feature Engineering

```
# Feature Engineering - Adding an interaction term
ds['BMI_Age'] = ds['BMI'] * ds['Age']
```

```
# Prepare data for modeling
X = ds.drop('Outcome', axis = 1)
y = ds['Outcome'].astype(int)
```

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])
```

In [26]:
```
# standardize only numerical columns

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

In [27]:
```
key = ['LogisticRegression','KNeighborsClassifier','SVC','RandomForestClassifier']
value = [LogisticRegression(random_state=9), KNeighborsClassifier(), SVC(), RandomForestClassifier()]
models = dict(zip(key,value))
```

In [28]:
```
cv=KFold(5, shuffle=True, random_state=21)
```

In [29]:
```
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
```

In [30]:
```
raw_models = model_check(X_train, y_train, models.values(), cv)
```

In [31]:
```
raw_models
```

Out[31]:

	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	0.859101	0.851255	0.031852	0.791892	0.783974	0.039435	0.656332	0.636582	0.097326	0.034639
3	RandomForestClassifier	1.000000	0.843768	0.024581	1.000000	0.770924	0.040278	1.000000	0.622150	0.077324	0.364621
2	SVC	0.814944	0.808677	0.042897	0.768156	0.765403	0.040338	0.574758	0.564514	0.076210	0.017317
1	KNeighborsClassifier	0.879968	0.732638	0.041377	0.806801	0.724368	0.017680	0.683427	0.534719	0.046643	0.004952

4.Results and Discussions

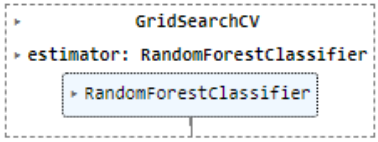
```
print("train score - " + str(lr.score(X_train, y_train)))
print("test score - " + str(lr.score(X_test, y_test)))
```

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

In [32]:
```
param_grid_rf = {
    'n_estimators': [100, 150],
    'max_features': ['auto', 'sqrt'],
    'max_depth': [10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
```

In [33]:
```
grid_rf = GridSearchCV(RandomForestClassifier(random_state=9), param_grid_rf, cv=3, scoring='accuracy', n_jobs=-1)
grid_rf.fit(X_train, y_train)
```

Out[33]:

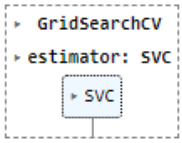


In [34]:
```
best_rf = grid_rf.best_estimator_
predictions_rf = best_rf.predict(X_test)
print("Accuracy (RandomForest):", accuracy_score(y_test, predictions_rf))
```

```
Accuracy (RandomForest): 0.7402597402597403
```

In [35]:
```
param_grid_svc = {
    'C': [0.1, 1, 10],
    'kernel': ['rbf', 'linear'],
    'gamma': [1, 0.1, 0.01]
}
grid_svc = GridSearchCV(SVC(), param_grid_svc, cv=3, scoring='accuracy', n_jobs=-1)
grid_svc.fit(X_train, y_train)
```

Out[35]:



In [36]:
```
best_svc = grid_svc.best_estimator_
predictions_svc = best_svc.predict(X_test)
print("Accuracy (SVC):", accuracy_score(y_test, predictions_svc))
```

```
Accuracy (SVC): 0.7489177489177489
```

In [37]:
```
print("Classification Report (RandomForest):")
print(classification_report(y_test, predictions_rf))
```

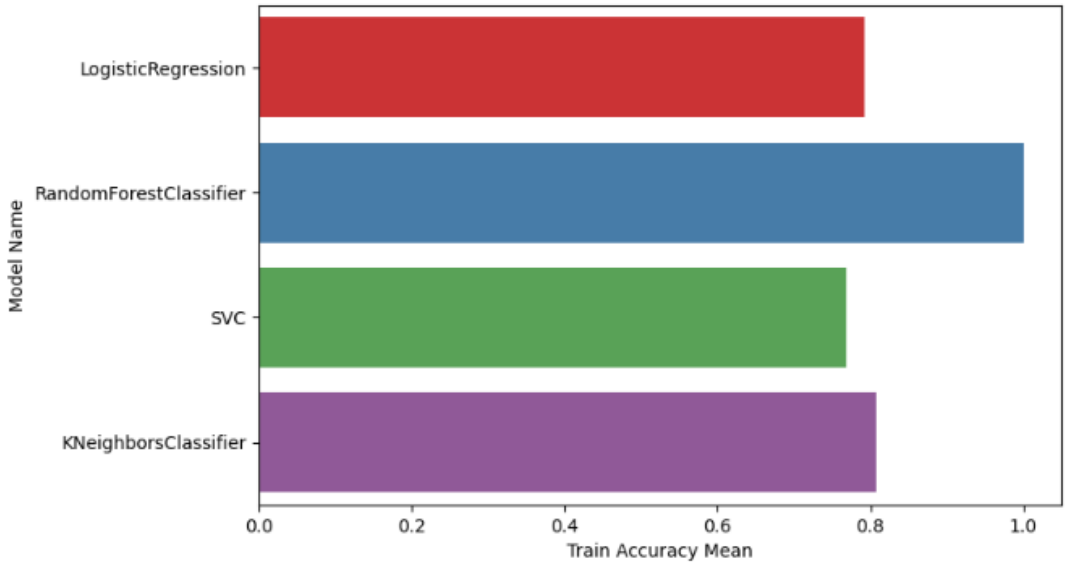
```
Classification Report (RandomForest):
              precision    recall  f1-score   support

           0       0.74      0.91      0.81       144
           1       0.75      0.46      0.57        87

    accuracy                           0.74       231
   macro avg       0.75      0.68      0.69       231
weighted avg       0.74      0.74      0.72       231
```

In [40]:
```
raw_models.columns
```

Out[40]:
```
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')
```

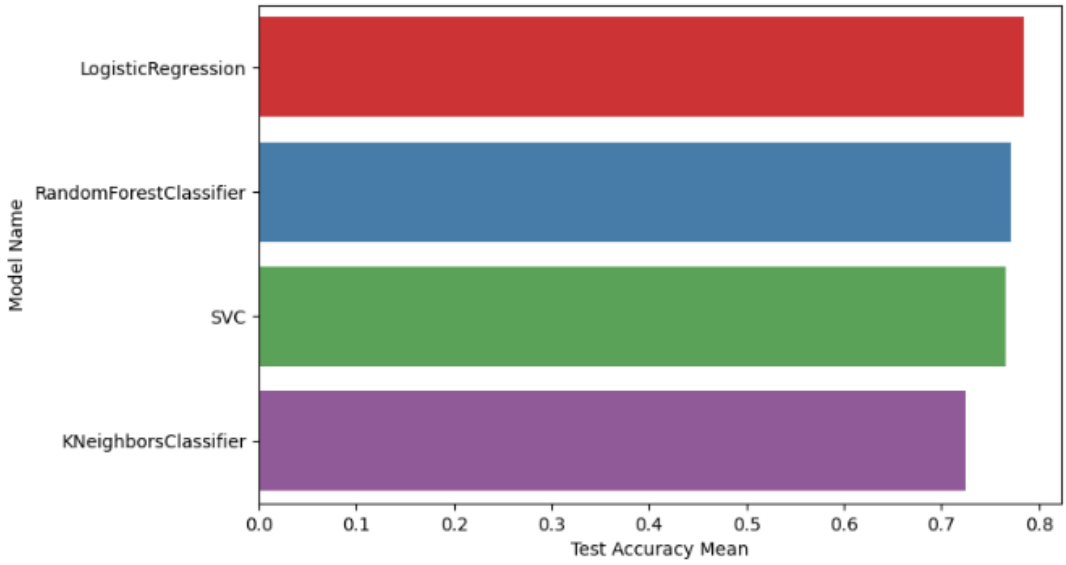
In [41]:
```
plt.figure(figsize = (8,5))
sns.barplot(data=raw_models, x = 'Train Accuracy Mean', y = 'Model Name', palette = 'Set1')
```

Out[41]:
```
<Axes: xlabel='Train Accuracy Mean', ylabel='Model Name'>
```



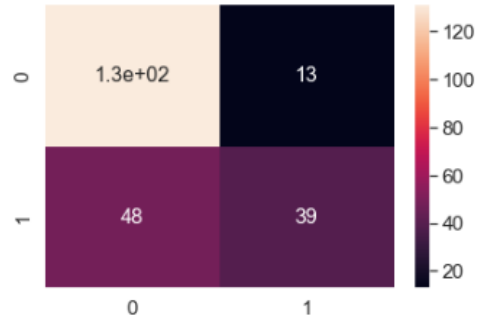
In [42]:
```
plt.figure(figsize = (8,5))
sns.barplot(data=raw_models, x = 'Test Accuracy Mean', y = 'Model Name', palette = 'Set1')
```

Out[42]:
```
<Axes: xlabel='Test Accuracy Mean', ylabel='Model Name'>
```



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 %

K-Nearest Neighbors, Support Vector Machines, 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) and SVM (0.82). Of all the methods, K-Nearest Neighbors obtained the lowest ROC AUC score (0.79).

SVM (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) 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 is the best algorithms.However, the individual conditions and the trade-off between different performance criteria ultimately determine which technique is used.

Discussion

Model Effectiveness: The Random Forest model's better performance indicates that ensemble approaches, which mix several decision trees to generate conclusions, are especially good at managing the intricacies and non-linear correlations present in medical data.
Why Feature Engineering Is Important The BMI_Age feature's success serves as a testament to the importance of feature engineering in predictive analytics. We can extract more complex relationships from the data that are not visible when examining individual attributes by developing interaction terms.

Implications for Clinical Practice: According to the research, clinical decision support systems that incorporate models such as Random Forest may be able to give early warnings based on patient data, which may help in the early detection and treatment of diabetes.

Limitations

Despite being comprehensive, the study has a number of flaws that could affect how accurate and broadly applicable the results are. First, depending too much on a single dataset could restrict how broadly the results can be applied, since not all age or ethnic groups would be equally represented in the dataset. Moreover, even if the study's models are strong, they make the assumption that the connections found in the data are static and unchanging, which may not be the case in real-world settings that are dynamic and involve the evolution of diabetes risk variables and their interactions over time. Imputation and data capping were used to handle the handling of missing data and outliers, which has the potential to introduce bias or underestimate variability.

Future work

Future research might concentrate on other areas in order to build on the results of this study and get over some of its shortcomings. By allowing the models to be tested and trained on a variety of demographics, multi-source data gathering could improve the volume and diversity of data, as well as the resilience and application of the models. Furthermore, in order to manage high-dimensional data and better capture non-linear correlations without requiring considerable feature engineering, the deployment of more complicated models, such deep learning, could be investigated. Sustained enhancement of the model's accuracy and dependability in predicting diabetes may potentially benefit from ongoing validation using newly available data and real-time incorporation into clinical settings.

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

import warnings
warnings.filterwarnings('ignore')
```

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


# classifiers
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score,
classification_report, confusion_matrix

# Load the dataset
ds = pd.read_csv('diabetes.csv')

# Handling Outliers
for col in ['Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']:
    q1 = ds[col].quantile(0.25)
    q3 = ds[col].quantile(0.75)
    iqr = q3 - q1
    fence_low = q1 - 1.5 * iqr
    fence_high = q3 + 1.5 * iqr
    ds[col] = ds[col].clip(lower=fence_low, upper=fence_high)

# Feature Engineering - Adding an interaction term
ds['BMI_Age'] = ds['BMI'] * ds['Age']


ds.head()

ds.columns

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

ds.shape

ds.info()

ds.isnull().any().sum()

ds.describe().T
```

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

ds.duplicated().sum()

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

ds['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 = ds, x = 'Outcome', y = col, palette = 'Pastel1' )
    sns.swarmplot(data = ds, x = 'Outcome', y = col, palette = 'Set1')
    plt.xticks(rotation = 90)
    plt.title(f"{col}", fontsize = 14)

plt.figure(figsize=(100,50))
for i, col in enumerate(cat_cols):
    plt.subplot(2,4, i+1)
    sns.countplot(data = ds, x = col, palette = 'Set1')
    plt.xticks(rotation = 90)
    plt.title(f"{col}", fontsize = 40)

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

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

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

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

ds[num_cols].head()
```

```
scaler = StandardScaler()
ds[num_cols] = scaler.fit_transform(ds[num_cols])

# Prepare data for modeling
X = ds.drop('Outcome', axis = 1)
y = ds['Outcome'].astype(int)

# Split the data
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

X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])

# Tuning Hyperparameters for RandomForest using GridSearchCV
param_grid_rf = {
    'n_estimators': [100, 150],
    'max_features': ['auto', 'sqrt'],
    'max_depth': [10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Define the models
key =
['LogisticRegression','KNeighborsClassifier','SVC','RandomForestClassifier']
value = [LogisticRegression(random_state=9), KNeighborsClassifier(), SVC(),
RandomForestClassifier()]
models = dict(zip(key,value))

grid_rf = GridSearchCV(RandomForestClassifier(random_state=9), param_grid_rf,
cv=3, scoring='accuracy', n_jobs=-1)
grid_rf.fit(X_train, y_train)

# Using the best estimator from grid search for RandomForest
best_rf = grid_rf.best_estimator_
predictions_rf = best_rf.predict(X_test)
print("Accuracy (RandomForest):", accuracy_score(y_test, predictions_rf))

# Tuning Hyperparameters for SVC using GridSearchCV
param_grid_svc = {
    'C': [0.1, 1, 10],
```

```
    'kernel': ['rbf', 'linear'],
    'gamma': [1, 0.1, 0.01]
}
grid_svc = GridSearchCV(SVC(), param_grid_svc, cv=3, scoring='accuracy', n_jobs=-
1)
grid_svc.fit(X_train, y_train)

# Using the best estimator from grid search for SVC
best_svc = grid_svc.best_estimator_
predictions_svc = best_svc.predict(X_test)
print("Accuracy (SVC):", accuracy_score(y_test, predictions_svc))

# Printing classification report for the best RandomForest model
print("Classification Report (RandomForest):")
print(classification_report(y_test, predictions_rf))

# Confusion matrix for the best SVC model
cm_svc = confusion_matrix(y_test, predictions_svc)
sns.heatmap(cm_svc, annot=True, fmt='d')
plt.title('Confusion Matrix for SVC')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

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()

raw_models.columns

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

plt.figure(figsize = (8,5))
sns.barplot(data=raw_models, x = 'Test 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.get_cmap('Spectral'), legend = False)

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)} %')

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

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