Diabetes Prediction Using Machine Learning

Diabetes is a medical disorder that impacts how well our body uses food as fuel. Most food we eat daily is converted to sugar, commonly known as glucose, and then discharged into the bloodstream. Our pancreas releases insulin when the blood sugar levels rise.

Diabetes can cause blood sugar levels to rise if it is not continuously and carefully managed, which raises the chance of severe side effects like heart attack and stroke. We, therefore, choose to forecast using Python machine learning.

Steps

1. Installing the Libraries
2. Importing the Dataset
3. Filling the Missing Values
4. Exploratory Data Analysis
5. Feature Engineering
6. Implementing Machine Learning Models
7. Predicting Unseen Data
8. Concluding the Report

Installing the Libraries

We first have to import the most popular Python libraries, which we will use for implementing machine learning algorithms in the first step of building the project, including Pandas, Seaborn, Matplotlib, and others.

We will use Python because it is the most adaptable and powerful programming language for data analysis purposes. In the world of software development, we also use Python.

Code

```
# Import libraries
import numpy as np # for linear algebra
import pandas as pd # for data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns # for data visualization
import matplotlib.pyplot as plt # to plot data visualization charts
from collections import Counter
import os

# Modeling Libraries
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score
from sklearn.preprocessing import QuantileTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV, cross_val_score, StratifiedKFold, learning_curve, train_test_split
```

```
from sklearn.svm import SVC
```

The Sklearn toolkit is incredibly practical and helpful and has practical applications. It offers a vast selection of ML models and algorithms.

Importing the Dataset

We are using the Diabetes Dataset from Kaggle for this study. The National Institute of Diabetes and Digestive and Kidney Diseases is the original source of this database.

Code

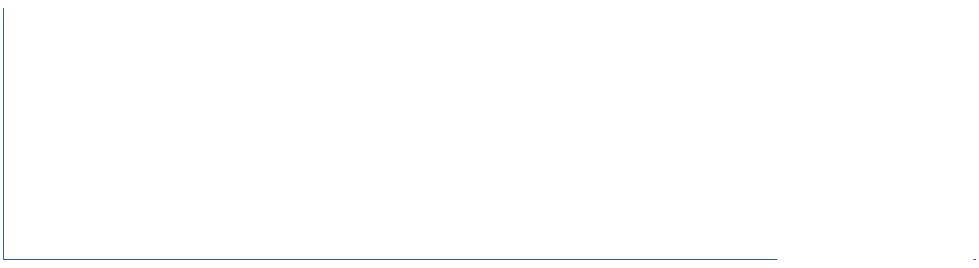
```
# Importing the dataset from Kaggle
data = pd.read_csv("../input/pima-indians-diabetes-database/diabetes.csv")

# First step is getting familiar with the structure of the dataset
data.info()
```

Output

```
<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
```

As we can see, all of the columns are integers, except for the BMI and DiabetesPedigreeFunction. The target variable is the labels with values of 1 and 0. A person's diabetes status is indicated by a one or a zero.

Code



```
# Showing the top 5 rows of the dataset
data.head()
```

Output

	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

Filling the Missing Values

The next step is cleaning the dataset, which is a crucial step in data analysis. When modelling and making predictions, missing data can result in incorrect results.

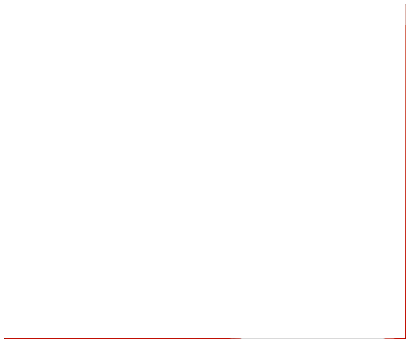
Code

```
# Exploring the missing values in the diabetes dataset
data.isnull().sum()
```

Output

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

We found no missing values in the dataset, yet independent features like skin thickness, insulin, blood pressure, ;and glucose each have some 0 values, which is practically impossible. A particular column's mean or median scores must be used to replace unwanted 0 values.



Code

```
# Replacing 0 values with the mean of that column

# Replacing 0 values of Glucose
data['Glucose'] = data['Glucose'].replace(0, data['Glucose'].median())

# Filling 0 values of Blood Pressure
data['BloodPressure'] = data['BloodPressure'].replace(0, data['BloodPressure'].median())
```

```
# Replacing 0 values in BMI
data['BMI'] = data['BMI'].replace(0, data['BMI'].mean())

# Replacing the missing values of Insulin and SkinThickness
data['SkinThickness'] = data['SkinThickness'].replace(0, data['SkinThickness'].mean())
data['Insulin'] = data['Insulin'].replace(0, data['Insulin'].mean())
data.head()
```

Output

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35.000000	79.799479	33.6	0.627	50	1
1	1	85	66	29.000000	79.799479	26.6	0.351	31	0
2	8	183	64	20.536458	79.799479	23.3	0.672	32	1
3	1	89	66	23.000000	94.000000	28.1	0.167	21	0
4	0	137	40	35.000000	168.000000	43.1	2.288	33	1

Let's now examine the data statistics.

Code

```
# Reviewing the dataset statistics
data.describe()
```

Output

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Ou
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	76
mean	3.845052	121.656250	72.386719	26.606479	118.660163	32.450805	0.471876	33.240885	0.3
std	3.369578	30.438286	12.096642	9.631241	93.080358	6.875374	0.331329	11.760232	0.4
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.0
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000	0.243750	24.000000	0.0
50%	3.000000	117.000000	72.000000	23.000000	79.799479	32.000000	0.372500	29.000000	0.0
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.0
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.0

Now our dataset is free of missing and unwanted values.

Exploratory Data Analysis

We will demonstrate analytics using the Seaborn GUI in this tutorial.

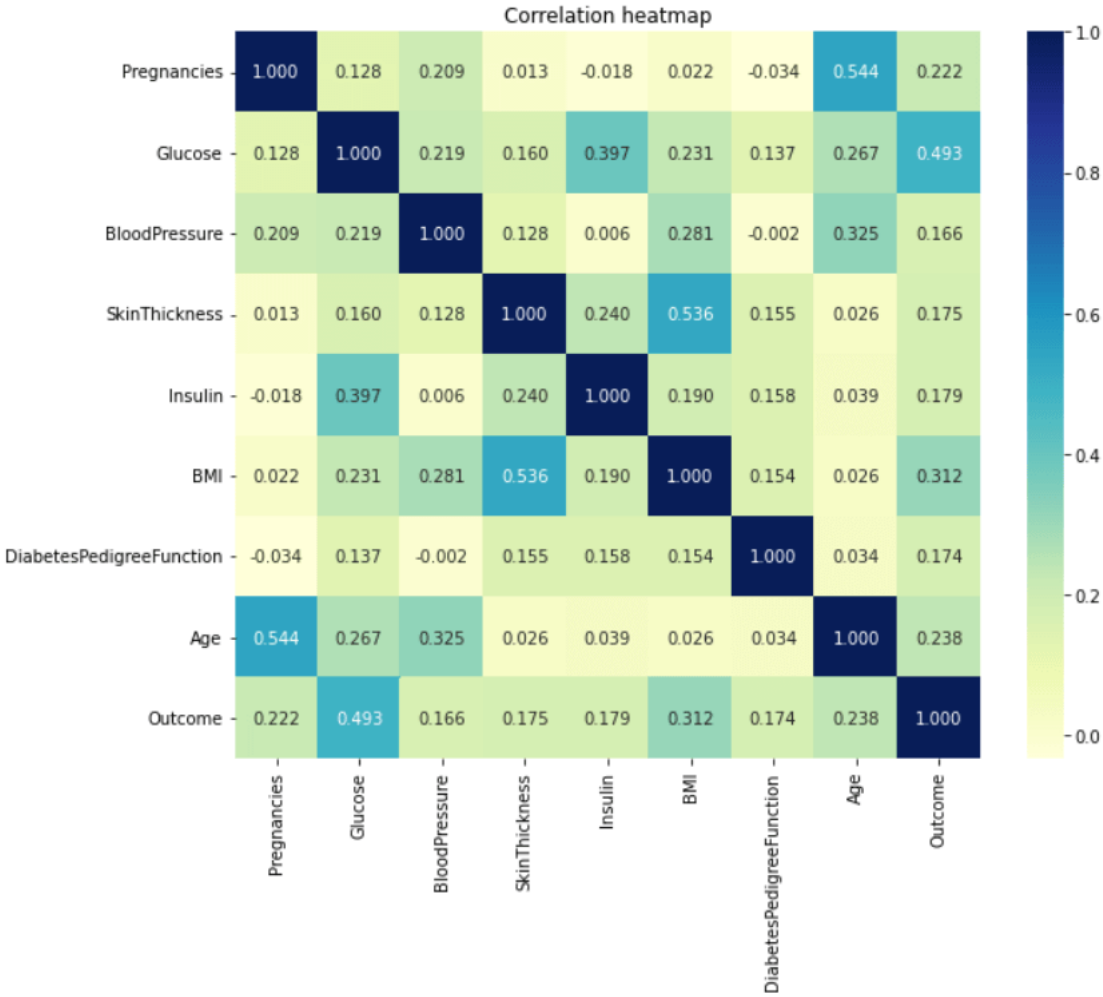
Correlation

Correlation is the relationship between two or more variables. Finding the important features and cleaning the dataset before we begin modelling also helps make the model efficient.

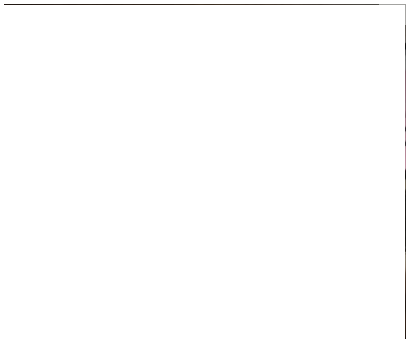
Code

```
# Correlation plot of the independent variables

plt.figure(figsize = (10, 8))
sns.heatmap(data.corr(), annot = True, fmt = ".3f", cmap = "YlGnBu")
plt.title("Correlation heatmap")
```

Output



Observations show that characteristics like pregnancy, glucose, BMI, and age are more closely associated with outcomes. I demonstrated a detailed illustration of these aspects in the following phases.



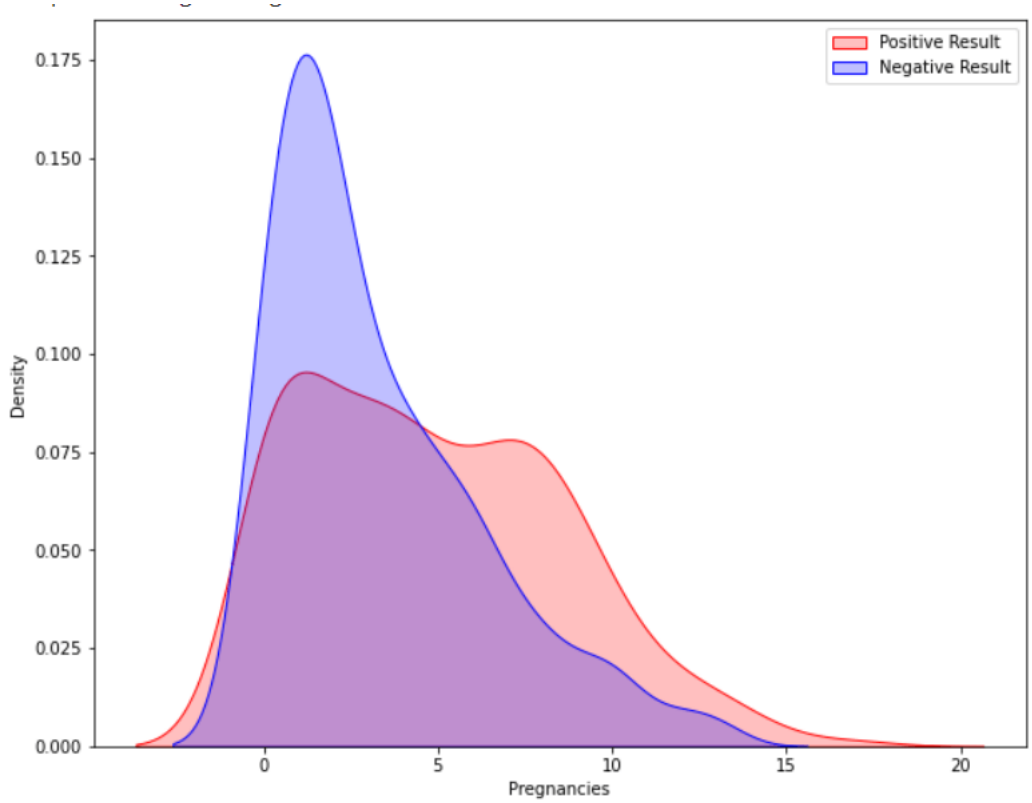
Pregnancy

Code

```
# Exploring Pregnancy and target variables together

plt.figure(figsize = (10, 8))
```

```
# Plotting density function graph of the pregnancies and the target variable
kde = sns.kdeplot(data["Pregnancies"][data["Outcome"] == 1], color = "Red", shade = True)
kde = sns.kdeplot(data["Pregnancies"][data["Outcome"] == 0], ax = kde, color = "Blue", shade= True)
kde.set_xlabel("Pregnancies")
kde.set_ylabel("Density")
kde.legend(["Positive Result", "Negative Result"])
```

Output

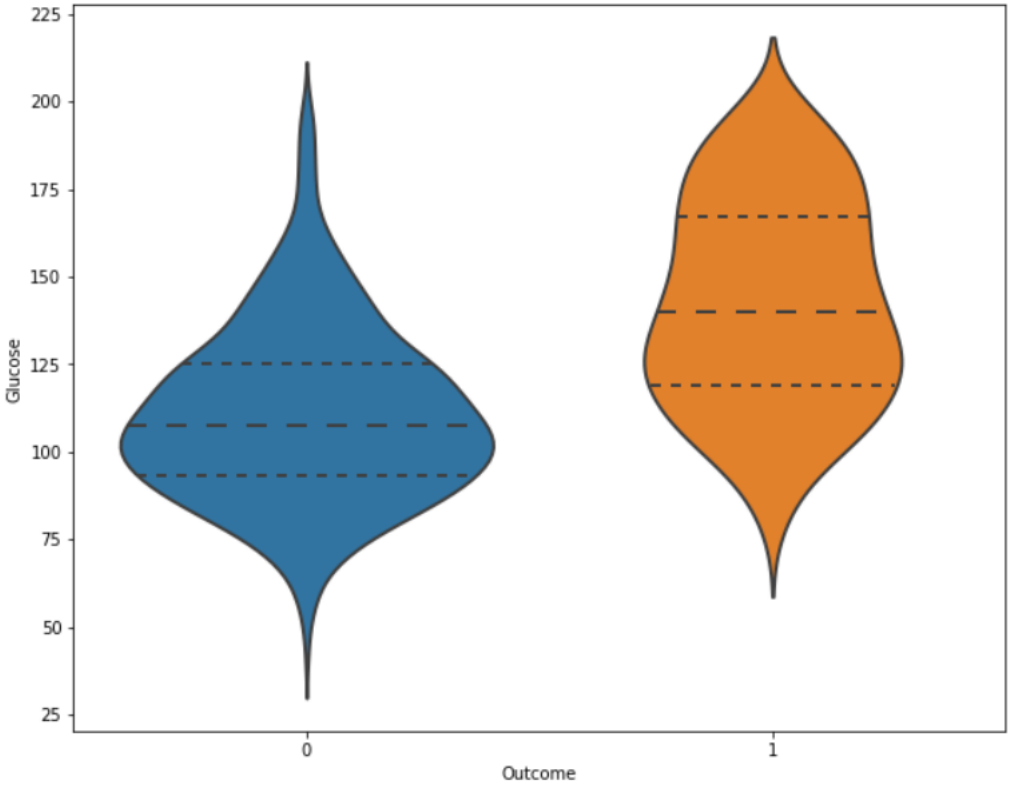


According to the data, women having diabetes have given birth to healthy infants. However, the risk for future complications can be decreased by managing diabetes. The risk of pregnancy issues, such as hypertension, depression, preterm birth, birth abnormalities, and pregnancy loss, is increased if women have uncontrolled diabetes.

Glucose


```
# Exploring the Glucose and the Target variables together
plt.figure(figsize = (10, 8))
sns.violinplot(data = data, x = "Outcome", y = "Glucose",
               split = True, inner = "quart", linewidth = 2)
```

Output

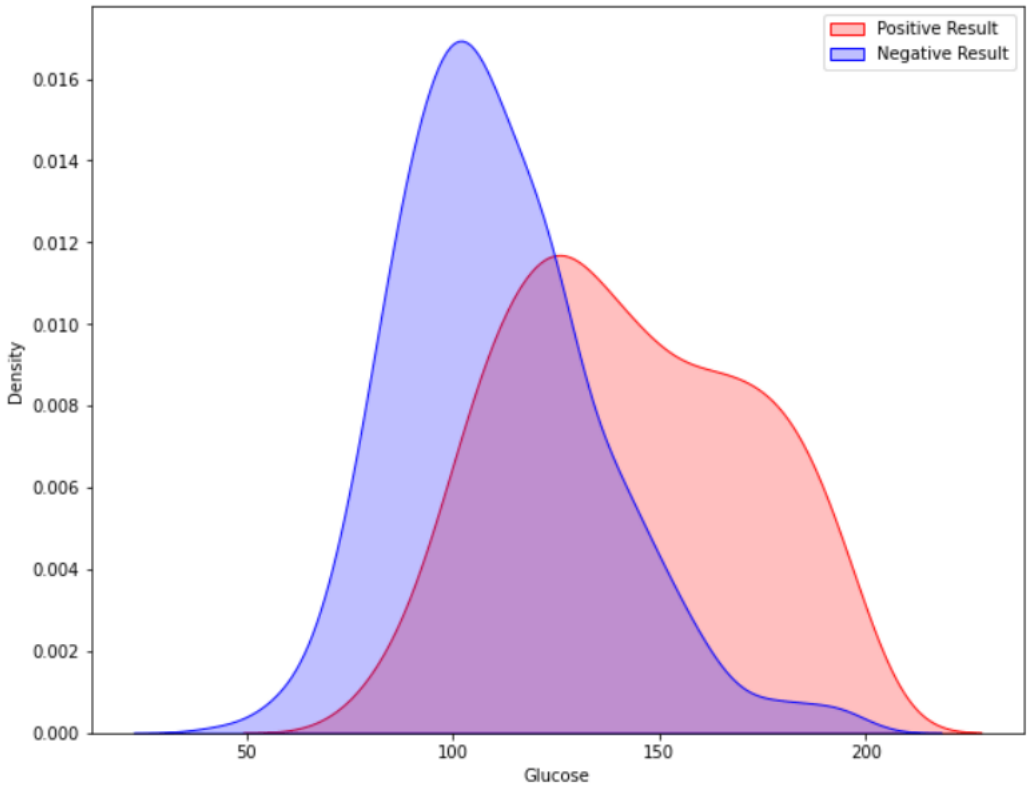


The likelihood of developing diabetes gradually climbs with glucose levels.

Code

```
# Exploring the density function plot of the Glucose levels

plt.figure(figsize = (10, 8))
kde = sns.kdeplot(data["Glucose"][data["Outcome"] == 1], color = "Red", shade = True)
kde = sns.kdeplot(data["Glucose"][data["Outcome"] == 0], ax = kde, color = "Blue", shade= True)
kde.set_xlabel("Glucose")
kde.set_ylabel("Density")
kde.legend(["Positive Result","Negative Result"])
```

Output



Implementing Machine Learning Models

We will test many machine learning models and compare their accuracy in this part. After that, we will tune the hyperparameters on models with good precision.

We will use sklearn.preprocessing to convert the data into quantiles before dividing the dataset.

Code

```
# Transforming the data into quartiles
quartile  = QuantileTransformer()
X = quartile.fit_transform(data)
dataset = quartile.transform(X)
dataset = pd.DataFrame(X)
dataset.columns =['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
# Showing the top 5 rows of the transformed dataset
dataset.head()
```

Output

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.747718	0.810300	0.494133	0.801825	0.380052	0.591265	0.750978	0.889831	1.0
1	0.232725	0.091265	0.290091	0.644720	0.380052	0.213168	0.475880	0.558670	0.0
2	0.863755	0.956975	0.233377	0.308996	0.380052	0.077575	0.782269	0.585398	1.0
3	0.232725	0.124511	0.290091	0.505867	0.662973	0.284224	0.106258	0.000000	0.0
4	0.000000	0.721643	0.005215	0.801825	0.834420	0.926988	0.997392	0.606258	1.0

Data Splitting

We will now divide the data into a training and testing dataset. We will use the training and testing datasets to train and evaluate different models. We will also perform cross-validation for multiple models before predicting the testing data.