Ai based diabetes prediction using machine learning

Data cleaning and testing:

What is Diabetes?

Diabetes is a disease that occurs when the blood glucose level becomes high, which ultimately leads to other health problems such as heart diseases, kidney disease, etc. Diabetes is caused mainly due to the consumption of highly processed food, bad consumption habits, etc. According to WHO, the number of people with diabetes has been increased over the years.

Prerequisites

- Python 3.+
- Anaconda (Scikit Learn, Numpy, Pandas, Matplotlib, Seaborn)
- Jupyter Notebook.
- Basic understanding of supervised machine learning methods: specifically classification.

Data Preparation

As a Data Scientist, the most tedious task which we encounter is the acquiring and the preparation of a data set. Even though there is an abundance of data in this era, it is still hard to find a suitable data set that suits the problem you are trying to tackle. If there aren't any suitable data sets to be found, you might have to create your own.

Data Exploration

When encountered with a data set, first we should analyze and "**get to know**" the data set. This step is necessary to familiarize with the data, to gain some understanding of the potential features and to see if data cleaning is needed.

First, we will import the necessary libraries and import our data set to the Jupyter notebook. We can observe the mentioned columns in the data set.

Import the package:

First I have import the my dataset of the packages.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import scale, StandardScaler

from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score

import warnings

warnings.simplefilter(action = "ignore")

- Import all packages using diabetes dataset.

First I have import numpy:

- Import Numpy as np is an incredibly powerful and versatile numerical computing library for Python and it is capable of numerous mathematical operations. When you import Numpy as 'np', you unlock the full potential of Numpy.
- The numpy used in the numerical value can be calculated and the functions can be used in the package was imported.
- The numpy is install in the command prompt using this command is **pip install numpy.**then you will import the package is **import numpy as np.**
- Numpy can be used in the numerical python the extension is np.
- The numpy using array can be performed in the numpy as np.the dataset will be numerical value can be inserted for the dataset the all dataset print the array the numpy was imported and all the functions can be performed.

Import the pandas:

The **import pandas** portion of the code tells Python to bring the pandas data analysis library into your current environment.

The **as pd** portion of the code then tells Python to give pandas the alias of **pd**. This allows you to use pandas functions by simply typing pd.function_name rather than pandas.function_name.

Once you've imported pandas, you can then use the functions built in it to create and analyze data.

The pandas was used in the dataframe and series all pandas functions can be used in the pandas as pd.

Import the matplotlib as plt:

importing the matplotlib module, defines x and y values for a plots, plots the data using the plot() function and it helps to display the plot by using the show() function . The plot() creates a line plot by connecting the points defined by x and y values.

Import the seaborn sns:

The **import seaborn** portion of the code tells Python to bring the Seaborn library into your current environment.

The **as sns** portion of the code then tells Python to give Seaborn the alias of **sns**. This allows you to use Seaborn functions by simply typing sns.function_name rather than seaborn.function_name.

Once you've imported Seaborn, you can then use the functions built in it to quickly visualize data.

Code:

```
diabetes = pd.read_csv('D:diabetes/diabetes.csv')
diabetes.columns
```

output:

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

Important: It should be noted that the above data set contains only limited features, where as in reality numerous features come into play.

We can examine the data set using the pandas' **head()** method.

- **diabetes.head(700)**

	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	
...	...	...	...	...	...	...	...	...	...	...
755	1	128	88	39	110	36.5	1.057	37	1	
756	7	137	90	41	0	32.0	0.391	39	0	

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
757		0	123	72	0	0	36.3	0.258	52	1
758		1	106	76	0	0	37.5	0.197	26	0
759		6	190	92	0	0	35.5	0.278	66	1

760 rows × 9 columns

We can find the dimensions of the data set using the panda Dataframes' 'shape' attribute.

- **diabetes.shape**

output

(768, 9)

We can observe that the data set contain 768 rows and 9 columns. '*Outcome*' is the column which we are going to predict, which says if the patient is diabetic or not. 1 means the person is diabetic and 0 means a person is not. We can identify that out of the 768 persons, 500 are labeled as 0 (non-diabetic) and 268 as 1 (diabetic)

diabetes ['Outcome'].value_counts()

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

diabetes.groupby('Outcome').mean()

output:

Outcome	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200	0.429734	31.190000
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	0.550500	37.067164

Missing or Null Data points

We can find any missing or null data points of the data set (if there is any) using the following pandas function.

diabetes.isnull().sum()
diabetes.isna().sum()

We can observe that there are no data points missing in the data set. If there were any, we should deal with them accordingly.

df.isnull().sum()

output

Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	0

dtype: int64

Unexpected Outliers

When analyzing the histogram we can identify that there are some outliers in some columns. We will further analyze those outliers and determine what we can do about them.

Blood pressure:

By observing the data we can see that there are 0 values for blood pressure. And it is evident that the readings of the data set seem wrong because a living person cannot have a diastolic blood pressure of zero. By observing the data we can see 35 counts where the value is 0.

```
print("Total : ", diabetes[diabetes.BloodPressure == 0].shape[0])Total:35  print(diabetes
[diabetes.BloodPressure == 0].groupby('Outcome')['Age'].count())

Outcome
0    19
1    16
Name: Age, dtype: int64
```

Plasma glucose levels:

Even after fasting glucose levels would not be as low as zero. Therefore zero is an invalid reading. By observing the data we can see 5 counts where the value is 0.

```
print("Total : ", diabetes[diabetes.Glucose == 0].shape[0])Total:5
print(diabetes[diabetes.Glucose == 0].groupby('Outcome')['Age'].count())Total:5

Outcome
0    3
1    2
Name: Age, dtype: int64
```

Skin Fold Thickness:

For normal people, skin fold thickness can't be less than 10 mm better yet zero. Total count where value is 0: 227.

```
print("Total : ", diabetes[diabetes.SkinThickness == 0].shape[0])Total :227
print(diabetes[diabetes.SkinThickness == 0].groupby('Outcome')['Age'].count())

Outcome
0    139
1     88
Name: Age, dtype: int64
```

BMI: Should not be 0 or close to zero unless the person is really underweight which could be life-threatening.

```
print("Total : ", diabetes[diabetes.BMI == 0].shape[0])Total :11  print(diabetes[diabetes.BMI == 0].groupby('Outcome')['Age'].count())

Outcome
0    9
1    2
Name: Age, dtype: int64
```

Insulin: In a rare situation a person can have zero insulin but by observing the data, we can find that there is a total of 374 counts.

```
print("Total : ", diabetes[diabetes.Insulin == 0].shape[0])Total :374
print(diabetes[diabetes.Insulin == 0].groupby('Outcome')['Age'].count())

Outcome
0    236
1    138
Name: Age, dtype: int64
```

Here are several ways to handle invalid data values :

1. Ignore/remove these cases: This is not actually possible in most cases because that would mean losing valuable information. And in this case “skin thickness” and “insulin” columns mean to have a lot of invalid points. But it might work for “BMI”, “glucose ”and “blood pressure” data points.

2. Put average/mean values: This might work for some data sets, but in our case putting a mean value to the blood pressure column would send a wrong signal to the model.

3. Avoid using features: It is possible to not use the features with a lot of invalid values for the model. This may work for “skin thickness” but it's hard to predict that.

By the end of the data cleaning process, we have come to the conclusion that this given data set is incomplete. Since this is a demonstration for machine learning we will proceed with the given data with some minor adjustments.

We will remove the rows which the “BloodPressure”, “BMI” and “Glucose” are zero.

```
diabetes_mod = diabetes[(diabetes.BloodPressure != 0) & (diabetes.BMI != 0) &
(diabetes.Glucose != 0)]
print(diabetes_mod.shape)

output:
(724, 9)
```

```
diabetes_scores=lof.negative_outlier_factor_
np.sort(diabetes_scores)[0:30]
```

output

```
array([-3.30445978, -2.48884101, -2.28758733, -2.10500141, -2.05369597,
    -2.02885837, -2.01096252, -2.00720763, -1.98655427, -1.95338702,
    -1.91601291, -1.88815728, -1.8134966 , -1.80857804, -1.74187579,
    -1.73154315, -1.71639102, -1.71372358, -1.67587303, -1.64102097,
    -1.63498158, -1.62215678, -1.61146741, -1.59344933, -1.54582494,
    -1.54285259, -1.51413703, -1.49974262, -1.49619189, -1.48877158])
```

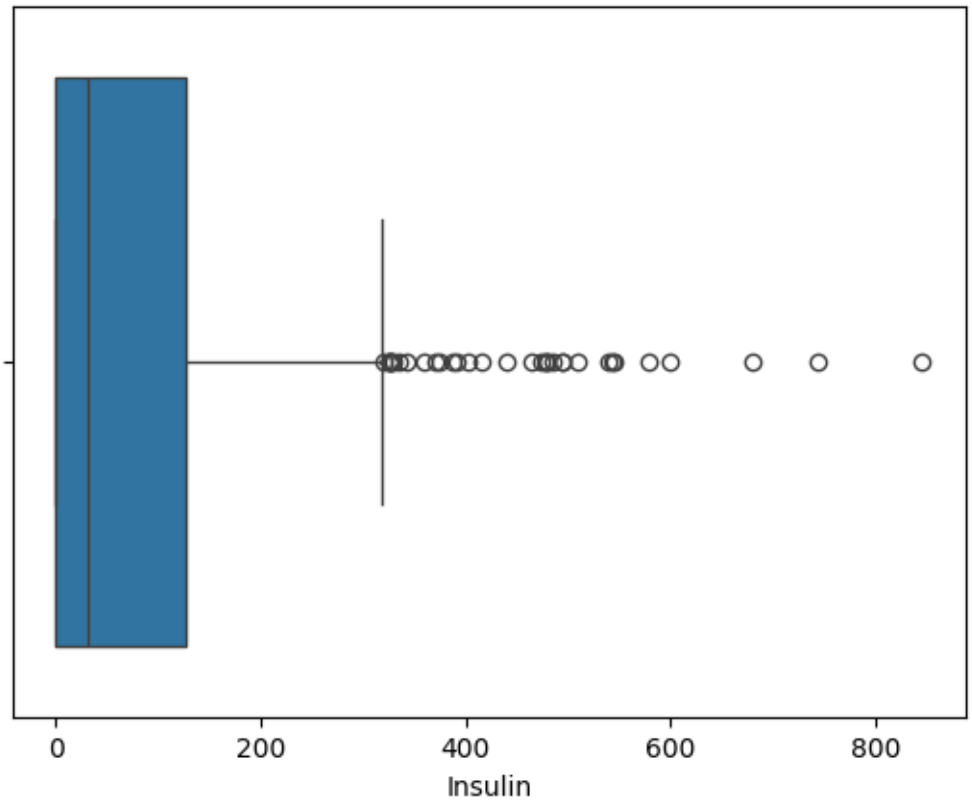
```
for feature in df:

    Q1 = df[feature].quantile(0.25)
    Q3 = df[feature].quantile(0.75)
    IQR = Q3-Q1
    lower = Q1- 1.5*IQR
    upper = Q3 + 1.5*IQR

    if df[(df[feature] > upper)].any(axis=None):
        print(feature,"yes")
    else:
print(feature, "no")
```

output
Pregnancies yes
Glucose no
BloodPressure yes
SkinThickness yes
Insulin yes
BMI yes
DiabetesPedigreeFunction yes
Age yes
Outcome no

```
import seaborn as sns
sns.boxplot(x = df["Insulin"]);
```


0
200
400
600
800
Insulin