AI-Based Diabetes Prediction System

Au952721104002-R.Abel Prince.

INTRODUCTION

Diabetes Dataset:

1. **Patient Information:**
    - Demographic data: Age, gender, ethnicity, location, etc.
    - Medical history: Previous diagnoses, family history, lifestyle factors, etc.
2. **Clinical Measurements:**
    - Blood sugar levels (glucose): Fasting blood glucose, postprandial glucose, HbA1c levels, etc.
    - Blood pressure: Systolic and diastolic blood pressure readings.
    - Body measurements: Weight, height, body mass index (BMI), waist circumference, etc.
3. **Medical Tests and Results:**

- Lipid profile: Cholesterol levels, triglycerides, HDL, LDL, etc.
- Kidney function tests: Creatinine levels, estimated glomerular filtration rate (eGFR), etc.
- Liver function tests: ALT, AST, bilirubin levels, etc.

4. **Medication and Treatment Data:**
    - Type of diabetes (e.g., Type 1, Type 2) and duration of diagnosis.
    - Insulin or other medications prescribed, dosage, and frequency.
    - History of insulin injections, oral medications, or other treatments.
5. **Diet and Physical Activity:**
    - Dietary habits: Daily caloric intake, types of food consumed, dietary restrictions, etc.
    - Physical activity: Exercise routines, activity levels, sedentary behavior, etc.
6. **Complications and Comorbidities:**
    - Presence of diabetic complications (e.g., neuropathy, retinopathy, nephropathy).
    - Other medical conditions or diseases coexisting with diabetes.

Importance of Diabetes Datasets:

1. **Research and Insights:**
    - Datasets provide valuable insights into diabetes prevalence, risk factors, progression, and potential interventions.
    - Researchers analyze the data to identify patterns and correlations, aiding in the development of new treatments and prevention strategies.
2. **Personalized Medicine:**
    - Data helps tailor treatment plans for individual patients based on their unique characteristics, improving treatment effectiveness and patient outcomes.
3. **Public Health Planning:**
    - Governments and healthcare organizations utilize the data to plan public health initiatives, allocate resources, and design awareness campaigns to tackle diabetes at a population level.
4. **Machine Learning and Predictive Models:**

- Data scientists use diabetes datasets to develop machine learning models that predict the risk of diabetes, monitor disease progression, and optimize treatment plans.

Table of Contents

1. First look to the Dataset

Business Problem: It is desired to develop a machine learning model that can predict whether people have diabetes when their characteristics are specified. You are expected to perform the necessary data analysis and feature engineering steps before developing the model.

Story of Dataset: The dataset is part of the large dataset held at the National Institutes of Diabetes-Digestive-Kidney Diseases in the USA. Data used for diabetes research on Pima Indian women aged 21 and over living in Phoenix, the 5th largest city of the State of Arizona in the USA.

Variables: The target variable is specified as **"outcome"**; **1** indicates **positive** diabetes test result, **0** indicates **negative**.

- **Pregnancies:** The number of pregnancies
- **Glucose:** 2-hour plasma glucose concentration in the oral glucose tolerance test
- **Blood Pressure:** Blood Pressure (Small blood pressure) (mmHg)
- **SkinThickness:** Skin Thickness
- **Insulin:** 2-hour serum insulin (mu U/ml)
- **DiabetesPedigreeFunction:** A function that calculates the probability of having diabetes according to the descendants
- **BMI:** Body mass index

- **Age:** Age (year)
- **Outcome:** Have the disease (1) or not (0)

2. EDA (Exploratory Data Analysis)

This is the part of the explore the dataset. The outliers and missing value analyses will be done in this part but the only point is to observe these in this part. The operation and editing for these will be done in Part 3: Pre-processing.

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
# !pip install missingno
import missingno as msno
from datetime import date
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import LocalOutlierFactor
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, StandardScaler,
RobustScaler


import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/diabetes/diabetes.csv
```

In [2]:

```
# Adjustment of visibility of Datafreames
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.float_format', lambda x: '%.3f' % x)
pd.set_option('display.width', 500)
```

In [3]:

```
df = pd.read_csv("../input/diabetes/diabetes.csv")
df.head()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.600	0.627	50	1
1	1	85	66	29	0	26.600	0.351	31	0
2	8	183	64	0	0	23.300	0.672	32	1
3	1	89	66	23	94	28.100	0.167	21	0
4	0	137	40	35	168	43.100	2.288		

2.1 Analyse outliers and missing values

Now, check the missing values and outliers. The only process in EDA is to analyze these values. Editing of missing values and outliers will be done in the next part.

Before analyzing, firstly, an explanation of these terms will be done, the codes will be placed right after each explanation.

Outliers

Values that deviate considerably from the general trend in the data are called outliers. Especially in linear problems, the effects of outliers are more severe. They have less impact on tree methods, but still, need to be considered.

How are outliers determined?: The critical point is to determine the acceptable threshold value, which are up limit and low limit. After determining the threshold value, outliers are caught based on these values. Methods by which we can catch the threshold value:

- Industry knowledge
- Standard deviation approach
- Z-score approximation
- **Boxplot(interquartile range-IQR) method (as univariate)**

- **LOF Method => Multivariate**

We will emphasize and examine Boxplot(IQR Method) and LOF Method to analyse. After getting outliers, they may be deleted, or reassigned with thresholds(pressing outliers values to thresholds). These solution approaches will be in the next part.

In [12]:

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

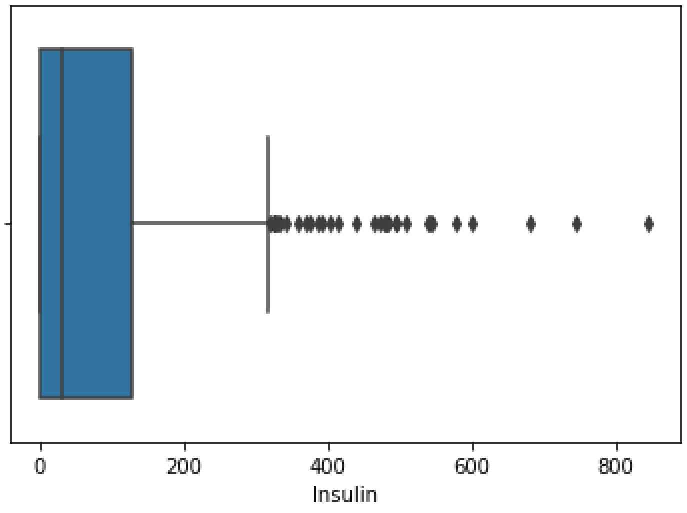
Out[12]:

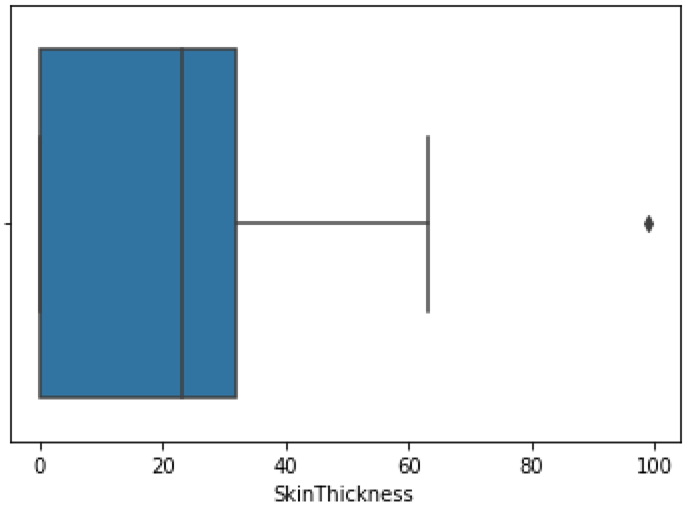
	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.000	3.845	3.370	0.000	1.000	3.000	6.000	17.000
Glucose	768.000	120.895	31.973	0.000	99.000	117.000	140.250	199.000
BloodPressure	768.000	69.105	19.356	0.000	62.000	72.000	80.000	122.000
SkinThickness	768.000	20.536	15.952	0.000	0.000	23.000	32.000	99.000
Insulin	768.000	79.799	115.244	0.000	0.000	30.500	127.250	846.000
BMI	768.000	31.993	7.884	0.000	27.300	32.000	36.600	67.100
DiabetesPedigreeFunction	768.000	0.472	0.331	0.078	0.244	0.372	0.626	2.420
Age	768.000	33.241	11.760	21.000	24.000	29.000	41.000	81.000
Outcome	768.000	0.349	0.477	0.000	0.000	0.000	1.000	1.000

```
# Boxplot
sns.boxplot(x=df["Insulin"])
plt.show()

sns.boxplot(x=df["SkinThickness"])
```

```
plt.show()
```





3. Pre-processing (Solve Outliers and Missing Values)

In part 2, the analysis has been completed. Examination of thresholds for outliers and how to reach missing values have been done. The next step is to interfere with these and prepare the dataset. This part will be separated into 2, like part 2. Firstly, the outliers problem will be examined, then, missing values will be processed.

Our threshold values can be affected after the operations on the missing value. Also, an outlier method, LOF, cannot be used when the dataset includes NaN values. Therefore, the missing value part will be taken firstly on preprocessing.

Missing Values

It mentioned on analyze part that there is 3 solution to missing values.

- **Deleting:** It means dropping the rows that include missing values. Especially in a small dataset, it creates a loss of information. For example, the dataset has 768 rows, but Insuline has 374 missing values. If missing values are dropped, half of the dataset has been lost, and the information will be lost, as well. If the dataset would be large, and there are a couple of missing values that can be sacrificed, deleting can be an option.

In [31]:

```
df_new = df.copy() # copy dataset to see effect without damage the main datas
et
df_new.shape
```

Out[31]:

```
(768, 9)
```

In [32]:

```
df_new.dropna(inplace=True)
df_new.shape
```

Out[32]:

```
(392, 9)
```

- **Value Assignment Methods**: We can fill the NaN values with the column's mean, median, mode, etc. If the distribution is homogeneous, filling with median or mean is logical. Also, in some scenarios, filling with a specific number like 0 would make sense.

In [33]:

```
df_fill = df.apply(lambda x: x.fillna(x.median()) if x.dtype != "O" else x, a
xis=0)
df_fill.head(10) # after filling
```

Out[33]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.000	72.000	35.000	125.000	33.600	0.627	50.000	1
1	1	85.000	66.000	29.000	125.000	26.600	0.351	31.000	0
2	8	183.000	64.000	29.000	125.000	23.300	0.672	32.000	1

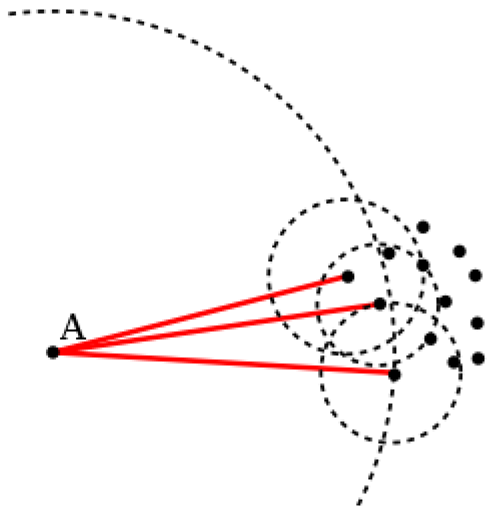
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
3	1	89.000	66.000	23.000	94.000	28.100	0.167	21.000	0
4	0	137.000	40.000	35.000	168.000	43.100	2.288	33.000	1
5	5	116.000	74.000	29.000	125.000	25.600	0.201	30.000	0
6	3	78.000	50.000	32.000	88.000	31.000	0.248	26.000	1
7	10	115.000	72.000	29.000	125.000	35.300	0.134	29.000	0
8	2	197.000	70.000	45.000	543.000	30.500	0.158	53.000	1
9	8	125.000	96.000	29.000	125.000	32.300	0.232		

Outliers

We mentioned on analyze part that there are 2 ways to solve this problem: IQR Method for the find outliers on univariate, and LOF Method for getting multivariate outliers.

Firstly, LOF Method will be explained because it controls the variables' meaningfulness to each other and gives a few outliers. Then, IQR Method will be implemented.

- **Local Outlier Factor (LOF):** Another approach is the Local Outlier Factor. It helps us to define outliers accordingly by ordering the observations based on the density at their location. The local density of a point means the neighborhoods around that point. If a point is significantly less dense than its neighbors, then that point is in a moresparse region, so there may be an outlier. The LOF method allows us to calculate a distance score based on neighborhoods.



4. Feature Extraction

When a data set is prepared, not only existing variables are tried to be edited, but also new, meaningful variables have to be created. New columns sometimes can be created with mathematical operations, sometimes named a numerical value to categorical, or categorical values' ranges, etc. This process is known as **Feature Engineering**, and this is one of the critical parts of data preparation.

In [56]:

```
# pregrancy cannot be float, it occurs due to calculation of IQR
df["Pregnancies"] = df["Pregnancies"].apply(lambda x: int(x))
```

In [57]:

```
# create categorical columns from numerical columns

# if bins are 0, 3, 6 => 0 values become NaN due to bins
df["NumOfPreg"] = pd.cut(df["Pregnancies"], bins=[-1, 3, 6, df["Pregnancies"]
.max()], labels=["Normal", "Above Normal","Extreme"])
df["AgeGroup"] = pd.cut(df["Age"], bins=[18, 25, 40, df["Age"].max()], labels
=["Young", "Mature", "Old"])
df["GlucoseGroup"] = pd.qcut(df["Glucose"], 3, labels=["Low", "Medium", "High
"])
df["Patient"] = np.where(df["Outcome"] == 1, "Yes", "No")
```

In [58]:

```
# example of mathematical expression

"""Assume there is a variable named "BMIns", and it can be found with the mul
tiplication of BMI and Insuline.
Create and add it to data frame"""
```

```
df["BMIns"] = df["BMI"]*df["Insulin"] # numerical
df["BMInsGroup"] = pd.qcut(df["BMIns"], 3, labels=["Low", "Medium", "High"])
# categorical
```
In [59]:

```
df.head()
```
Out[59]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	AgeGroup	P
0	6	148.000	72.000	35.000	125.000	33.600	0.627	50.000	1	Old	
1	1	85.000	66.000	29.000	125.000	26.600	0.351	31.000	0	Mature	
2	8	183.000	64.000	29.000	125.000	23.300	0.672	32.000	1	Mature	
3	1	89.000	66.000	23.000	114.500	28.100	0.167	21.000	0	Young	
4	0	137.000	40.000	35.000	134.500	43.100	1.200				

5. Encoding & Scaling

Encoding

Changing the representation of variables. There are different types of encoding:
If there are 2 classes, it is called **"Binary Encoding"**, if there are more than 2 classes, it is called **"Label Encoding"**.

It converts in alphabetical order, giving 0 to the first one it sees.

In [60]:

```
df_enc = df.copy()

le = LabelEncoder()
le.fit_transform(df_enc["Patient"])[0:5]
```
Out[60]:

```
array([1, 0, 1, 0, 1])
```
In [61]:

```
# let's say we forgot which 0 and which 1, inverse_transform is used to detec
t this
le.inverse_transform([0, 1])
```

Out[61]:

```
array(['No', 'Yes'], dtype=object)
```

In [62]:

```
# detect variables that have 2 unique numbers for Binary Encoding
binary_cols = [col for col in df.columns if df[col].dtype not in [int, float]
               and df[col].nunique() == 2]
binary_cols
```

Out[62]:

```
['Patient']
```

6. Model

It is not within the scope of this topic, we are only doing it to see the model establishment while preparing the data.

Patient carries the same information with Outcome. It was created as an example of feature engineering. Also, BMIns is not a real feature, it was another example of feature engineering using mathematical expression. Therefore, these 2 variables will be dropped.

KNN algorithm will be used.

In [72]:

```
y = df["Outcome"]
X = df.drop(["Outcome", "Patient", "BMIns", "BMInsGroup_Medium", "BMInsGroup_
High"], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, ran
dom_state=17)

from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(random_state=46).fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
accuracy_score(y_pred, y_test)
```

```
0.8104575163398693
```

Challenge loading and pre-processing in dataset

Loading the Dataset:

1. **Determine the Dataset Format:** Identify the format of the dataset you're working with, such as CSV, Excel, JSON, SQL, etc.
2. **Choose a Programming Language and Library:** Choose a programming language like Python and a suitable library for handling the dataset. Common libraries include Pandas (for structured data), NumPy (for numerical operations), and scikit-learn (for machine learning).
3. **Import the Necessary Libraries:** Import the required libraries in your chosen programming language.

   Example (Python with Pandas for a CSV dataset):
   ```
   import pandas as pd
   ```
4. **Load the Dataset:** Use the appropriate function or method to load the dataset into your code.

   Example (Loading a CSV file into a Pandas DataFrame):
   ```
   df = pd.read_csv('path/to/your/dataset.csv')
   ```

Preprocessing the Dataset:

1. **Handling Missing Data:** Check for and handle missing or null values in the dataset, either by removing or imputing them.

   Example (Dropping rows with missing values in Pandas):
   ```
   df.dropna(inplace=True)
   ```
2. **Data Cleaning and Transformation:** Clean the dataset by removing irrelevant columns, formatting data types, or transforming features as needed.

   Example (Converting a column to a numeric type in Panda)
   ```
   df['numeric_column'] = pd.to_numeric(df['numeric_column'])
   ```
3. **Feature Scaling or Normalization:** Normalize or scale the features to ensure they have a similar range, which can improve model performance.

   Example (Using Min-Max scaling in scikit-learn):
   ```
   from sklearn.preprocessing import MinMaxScaler
   scaler = MinMaxScaler ()
   df[['feature1', 'feature2']] = scaler.fit_transform(df[['feature1', 'feature2']])
   ```
4. **Encoding Categorical Variables:** Encode categorical variables into numerical representations (e.g., one-hot encoding) to make them suitable for machine learning algorithms.

Example (One-hot encoding in Pandas):
```
df = pd.get_dummies(df, columns=['categorical_column'])
```

5. **Splitting into Training and Testing Sets:** Split the dataset into training and testing sets to evaluate the model's performance.

   Example (Using scikit-learn for train-test split):

```
 from sklearn.model_selection import train_test_split

X = df.drop('target_column', axis=1)

y = df['target_column']

X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.2,
random_state=42)
```

How to Overcome the Challenge loading and Pre-processing in diabetes

1. **Understand the Data:** Start by understanding the structure and nature of the diabetes dataset you're working with. Know the features (independent variables) and the target variable (dependent variable) you want to predict.
2. **Data Cleaning:**
   - Deal with missing values: Identify and handle any missing or null values in the dataset. You can impute missing values using techniques like mean, median, or interpolation.
   - Remove duplicates: Check for and remove any duplicate records in the dataset.
3. **Data Exploration and Analysis:**
   - Visualize the data to understand patterns and relationships using plots, histograms, box plots, etc.
   - Explore statistical summaries and descriptive statistics to understand the characteristics of the dataset.
4. **Feature Engineering:**
   - Create new features if needed based on domain knowledge to improve the predictive power of the model.
   - Transform features, if necessary, through scaling, normalization, or encoding (e.g., one-hot encoding for categorical variables).
5. **Data Splitting:**

- Divide the dataset into training, validation, and test sets. The training set is used to train the model, the validation set helps in tuning hyperparameters, and the test set evaluates the final model.

6. **Handling Imbalanced Data (if applicable):**
    - In some cases, diabetes datasets might be imbalanced, meaning one class (e.g., diabetic) is more prevalent than the other. Use techniques like oversampling, undersampling, or generating synthetic data to balance the dataset.
7. **Data Loading and Preprocessing for Models:**
    - Load the preprocessed data into the appropriate data structures for your chosen machine learning or statistical model.
    - Ensure the data is in a format that the model can accept. For example, reshape or reformat data as needed.
8. **Normalization and Standardization:**
    - Normalize or standardize numerical features to ensure that they are on a similar scale, which can help improve model performance.
9. **Save Preprocessing Steps:**
    - Save the preprocessing steps and transformations, so they can be applied consistently to new data when making predictions with the trained model