AI BASED DIABETES PREDICTION SYSTEM USING MACHINE LEARNING

TEAM MEMBER

922321106012:R.HEMALATHA

Phase 3 submission document

Project Title: Diabetes Prediction System

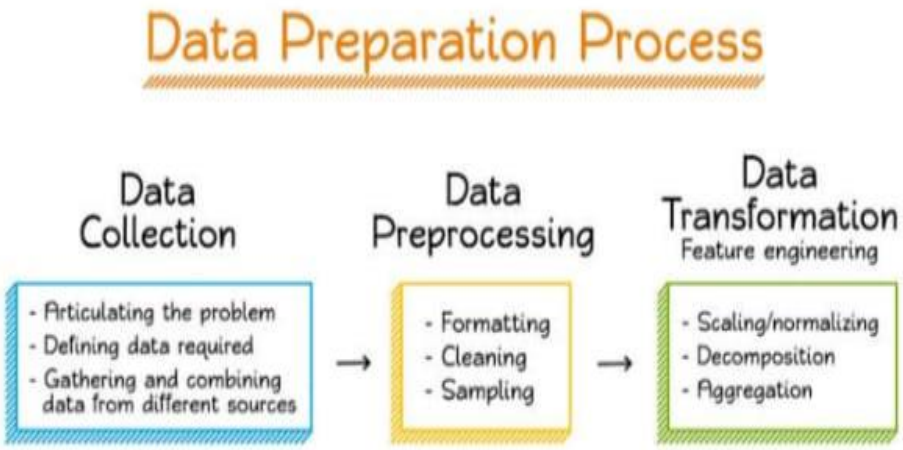
Phase 3: Development Part 1

Topic: Start building the diabetes prediction model by loading and pre-processing the dataset.

Diabetes Prediction System

Introduction:

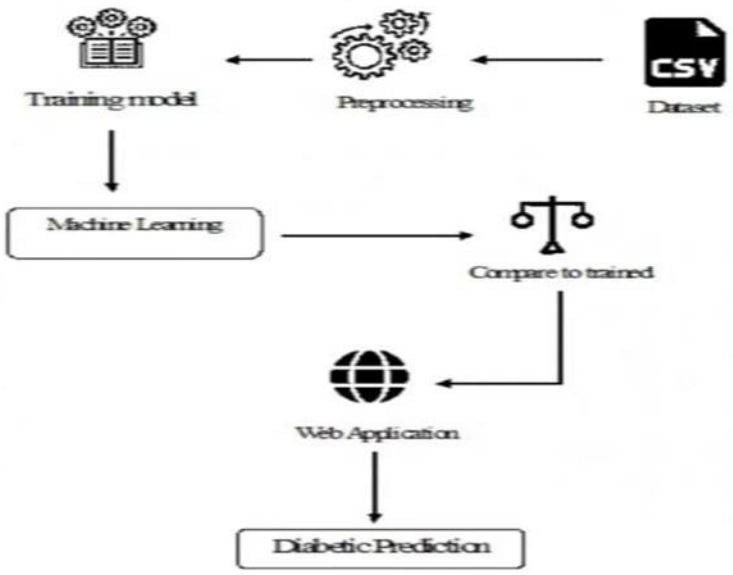
An AI-based diabetes prediction system is a computer program that uses artificial intelligence (AI) to predict whether a person is likely to develop diabetes. AI is a field of computer science that deals with the creation of intelligent agents, which are systems that can reason, learn, and act autonomously. AI-based diabetes prediction systems work by analyzing a person's medical data, such as their age, weight, blood pressure, and blood sugar levels. The system then uses this data to train a machine learning model, which is a type of AI algorithm that can learn from data and make predictions. Once the model is trained, it can be used to predict whether a person is likely to develop diabetes in the future.



AI-based diabetes prediction systems have a number of advantages over traditional methods of diabetes prediction.

First, they are more accurate. AI systems can analyze large amounts of data and identify patterns that would be difficult or impossible for humans to spot. Second, AI systems are more efficient. They can quickly analyze a person's data and make a prediction, without the need for human intervention. Third, AI systems are more objective. They are not biased by personal factors, such as a doctor's experience or intuition.

AI-based diabetes prediction systems are still under development, but they have the potential to revolutionize the way that diabetes is diagnosed and managed. By predicting who is at risk of developing diabetes, these systems can help to prevent the disease from developing in the first place. They can also help people with diabetes to better manage their condition and avoid complications.



Given Dataset:

	A	B	C	D	E	F	G	H	I
1	Pregnanci	Glucose	BloodPres	SkinThickr	Insulin	BMI	DiabetesF	Age	Outcome
2	6	148	72	35	0	33.6	0.627	50	1
3	1	85	66	29	0	26.6	0.351	31	0
4	8	183	64	0	0	23.3	0.672	32	1
5	1	89	66	23	94	28.1	0.167	21	0
6	0	137	40	35	168	43.1	2.288	33	1
7	5	116	74	0	0	25.6	0.201	30	0
8	3	78	50	32	88	31	0.248	26	1
9	10	115	0	0	0	35.3	0.134	29	0
10	2	197	70	45	543	30.5	0.158	53	1
11	8	125	96	0	0	0	0.232	54	1
12	4	110	92	0	0	37.6	0.191	30	0
13	10	168	74	0	0	38	0.537	34	1
14	10	139	80	0	0	27.1	1.441	57	0
15	1	189	60	23	846	30.1	0.398	59	1
16	5	166	72	19	175	25.8	0.587	51	1
17	7	100	0	0	0	30	0.484	32	1
18	0	118	84	47	230	45.8	0.551	31	1
19	7	107	74	0	0	29.6	0.254	31	1
20	1	103	30	38	83	43.3	0.183	33	0
21	1	115	70	30	96	34.6	0.529	32	1
22	3	126	88	41	235	39.3	0.704	27	0
23	8	99	84	0	0	35.4	0.388	50	0
24	7	196	90	0	0	39.8	0.451	41	1

769 Rows *9 Columns

DATASET LINK:
https://1drv.ms/x/s!AhQBfiMJZbYSi2Z8ifTdYFtnhilQ

Necessary step to follow:

To build an AI-based diabetes prediction system using machine learning with Python, you can follow these steps.

1)Install the necessary Python libraries:

You will need to install the following Python libraries.

a) NumPy

b) Pandas

c) Scikit-learn

Program:

```
Import numpy as np
Import pandas as pd
From sklearn.model_selection import train_test_split
From sklearn.ensemble import RandomForestClassifier
```

2)Load the diabetes dataset:

You can download the diabetes dataset from Kaggle.

Program:

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

3)Prepare the data for machine learning:

This may involve cleaning the data, removing outliers, and imputing missing values.

Program:

```
# Handle missing values
```

```
# If there are any missing values in the dataset, you can impute them using a variety of methods. For example, you could use the mean or median of the column to impute missing values.
# ...
# Encode categorical variables
# If there are any categorical variables in the dataset, you need to encode them before feeding them to the machine learning algorithm. You can use a variety of methods to encode categorical variables, such as one-hot encoding or label encoding.
# For this example, we will use label encoding.
Label_encoder = LabelEncoder()
For col in diabetes_df.columns:
If diabetes_df[col].dtype == 'object':
Diabetes_df[col] = label_encoder.fit_transform(diabetes_df[col])

# Scale numerical variables
# It is important to scale numerical variables before feeding them to the machine learning algorithm. This will help to ensure that all of the variables are on the same scale and that no one variable dominates the model.
# We will use the StandardScaler class from scikit-learn to scale the numerical variables in the dataset.
Scaler = StandardScaler()
Diabetes_df = scaler.fit_transform(diabetes_df)
# Save the preprocessed data
Diabetes_df.to_csv('preprocessed_diabetes.csv', index=False)
```

4)Split the data into training and testing sets:

You should split the data into training and testing sets in a ratio of 70:30. The training set will be used to train the machine learning model, and the testing set will be used to evaluate the model's performance

Program:

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42).
```

5)Choose a machine learning algorithm:

There are many different machine learning algorithms that can be used for diabetes prediction. Some popular choices include support vector machines (SVMs), random forests, and logistic regression.

Program:

```
# Create a random forest classifier
Clf = RandomForestClassifier()
```

6)Train the machine learning model:

This involves feeding the training data to the algorithm and allowing it to learn the patterns associated with diabetes.

Program:

```
# Train the model
Clf.fit(X_train, y_train)
```

7)Evaluate the machine learning model:

Once the model is trained, you need to evaluate its performance on the testing set. This will give you an idea of how well the model will generalize to new data.

Program:

```
# Evaluate the model
Y_pred = clf.predict(X_test)
Accuracy = np.mean(y_pred == y_test)
Print('Accuracy:', accuracy)
```

8)Deploy the machine learning model:

Once you are satisfied with the model's performance, you can deploy it to production. This may involve integrating the model into a software application or web service.

9)Exploratory Data Analysis (EDA):

Perform EDA to understand your data better. This includes checking for missing values, exploring the data's statistics and visualizing it to identify patterns.

Program:

```
# Plot the distribution of each feature
For col in diabetes_df.columns:
Plt.hist(diabetes_df[col])
Plt.xlabel(col)
Plt.ylabel('Count')
Plt.title(f'Distribution of {col}')
Plt.show()
# Calculate the correlation between the features
Diabetes_df.corr()
# Plot a heatmap of the correlation matrix
Plt.matshow(diabetes_df.corr())
Plt.colorbar()
Plt.title('Correlation matrix')
Plt.show()
```

This is just a basic example, and you may want to add other EDA techniques, such as:

- Identifying outliers
- Identifying missing values
- Checking for normality
- Identifying feature relationships
- Identifying important features

You can use the insights gained from EDA to improve your machine learning model by:

- Handling outliers and missing values
- Transforming non-normal features
- Selecting the most important features

By performing EDA on your data, you can gain a better understanding of your data and build a better machine learning model.

Importance of loading and pre-processing dataset:

- Pre-processing helps to improve the accuracy, reduce the time and resources required to train the model, prevent overfitting and improve the interpretability of the model.
- Loading the data is the data that we need to load for starting any of the ML project. With respect to data, the most common format of data for ML projects is CSV (Comma separated Values).

Challenges involved in loading and preprocessing of diabetes prediction system:

Loading and preprocessing data for a diabetes prediction system can be challenging due to various factors, including the nature of healthcare data and the complexity of the prediction task. Here are some specific challenges you might encounter:

1)Data Quality and Completeness:

Healthcare data can be incomplete or contain missing values, which can affect the accuracy of predictions. Ensuring data quality and completeness is crucial.

2)Imbalanced Data:

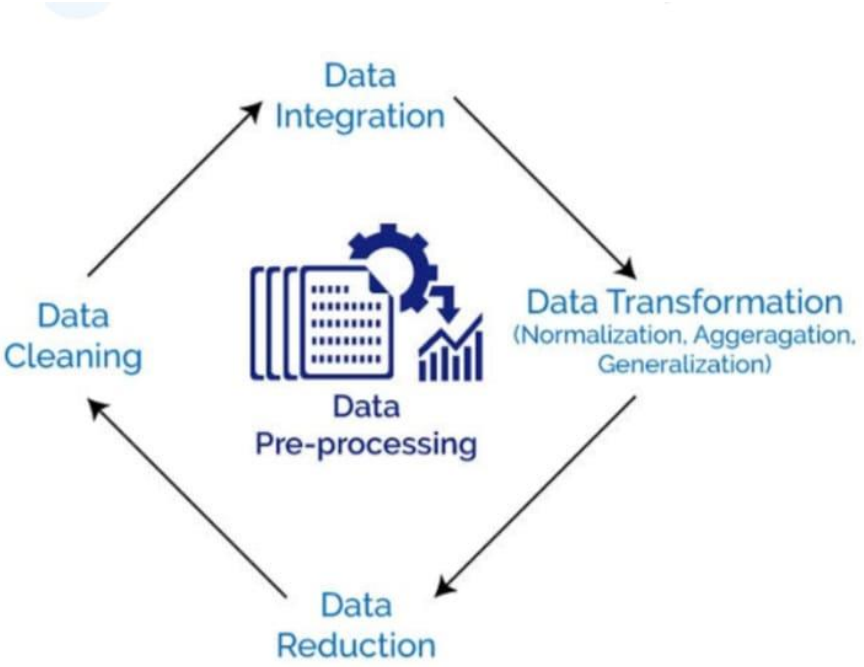
In many healthcare datasets, there is an imbalance between diabetes-positive and diabetes-negative cases. The model may be biased toward the majority class without proper handling

3)Data Privacy and Security:

Healthcare data is highly sensitive, and there are strict regulations like HIPAA in the United States and GDPR in Europe. Handling and securing patient data while preserving privacy is challenging.

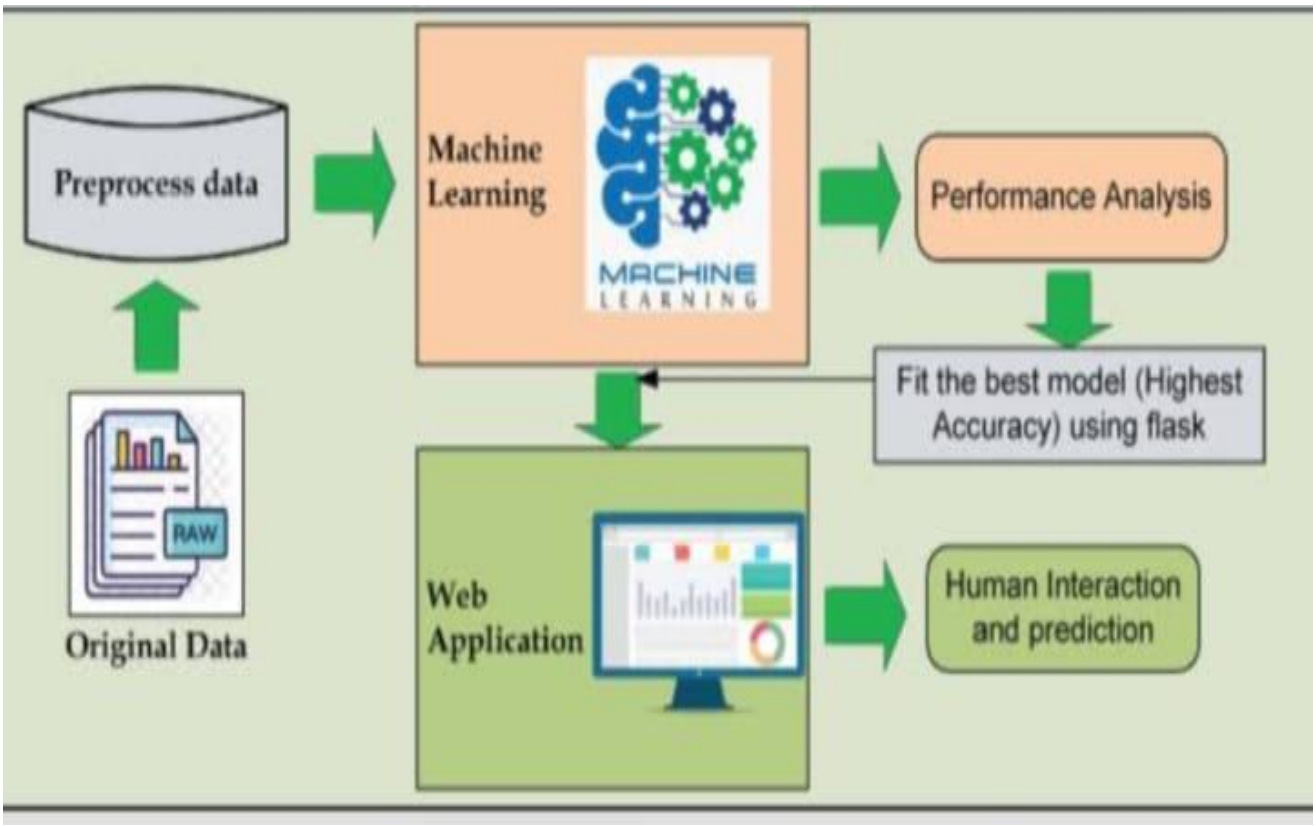
4)Feature Engineering:

Selecting the most relevant features or variables for prediction is often a complex task. Medical data may have numerous features, and determining which ones are informative can be challenging.



5)Categorical Data:

Healthcare data often includes categorical variables such as gender, medication types, and medical procedures. Converting these into a numerical format that machine learning models can understand is a challenge.



How to overcome the challenges of loading and pre-processing a diabetes prediction dataset:

1)Data Quality and Missing Values:

- ✓ Challenge: Real-world healthcare data often contains missing or inaccurate values.
- ✓ Solution:

- Carefully inspect and clean the data, filling in missing values using appropriate techniques such as mean imputation or regression imputation.
- Outlier detection and handling can help improve data quality.
- Consider domain knowledge to identify and correct anomalies.

2)Feature Selection:

✓ Challenge: Choosing the right set of features is crucial for model performance.

✓ Solution:

- Use feature engineering techniques, domain expertise, and feature importance rankings to select the most relevant features.
- Apply dimensionality reduction methods like Principal Component Analysis (PCA) to reduce the number of features while preserving important information.

3)Categorical Data Encoding:

✓ Challenge: Healthcare data often includes categorical variables that need to be transformed into numerical format.

✓ Solution:

- Apply one-hot encoding or label encoding to convert categorical data into a format suitable for machine learning models.
- Ensure proper handling of rare categories to prevent data leakage.

PYTHON PROGRAM:

1) Exploratory Data Analysis

In [1]:

```
#Installation of required libraries
import numpy as np
import pandas as pd
import statsmodels.api as sm
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_
score
from sklearn.metrics import confusion_matrix, accuracy_score, mean_squared_
error, r2_score, roc_auc_score, roc_curve, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from lightgbm import LGBMClassifier
from sklearn.model_selection import KFold
import warnings
warnings.simplefilter(action = "ignore")
```

In [2]:

```
#Reading the dataset
df = pd.read_csv("../input/pima-indians-diabetes-database/diabetes.csv")
```

In [3]:

```
# The first 5 observation units of the data set were accessed.
```

df.head()

Out[3]:

	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

In [4]:

 # The size of the data set was examined. It consists of 768 observation units and 9 variables.

df.shape

Out[4]:

(768, 9)

In [5]:

#Feature information

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
```

In [6]:

```
# Descriptive statistics of the data set accessed.
df.describe([0.10,0.25,0.50,0.75,0.90,0.95,0.99]).T
```

Out[6]:

	count	mean	std	min	10%	25%	50%	75%	90%	95%	99%	max
Pregnancies	768.0	3.845052	3.369578	0.000	0.000	1.000000	3.0000	6.000000	9.0000	10.00000	13.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	85.0000	99.00000	117.0000	140.25000	167.0000	181.00000	196.00000	199.00

	count	mean	std	min	10%	25%	50%	75%	90%	95%	99%	max
BloodPressure	768.0	69.105469	19.355807	0.000	54.0000	62.000000	72.000	80.00000	88.000	90.00000	106.000000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.000	0.000000	23.000	32.00000	40.000	44.00000	51.33000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.000	0.000000	30.5000	127.25000	210.0000	293.00000	519.90000	846.00
BMI	768.0	31.992578	7.884160	0.000	23.600	27.300000	32.000	36.60000	41.5000	44.39500	50.75900	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.165	0.24375	0.3725	0.62625	0.8786	1.13285	1.69833	2.42
Age	768.0	33.240885	11.760232	21.000	22.0000	24.000000	29.000	41.00000	51.000	58.00000	67.00000	81.00

	count	mean	std	min	10%	25%	50%	75%	90%	95%	99%	max
Outcome	768.0	0.348958	0.476951	0.000	0.000	0.000000	0.000000	1.000000	1.00000	1.000000	1.000000	1.00

In [7]:

```
# The distribution of the Outcome variable was examined.
df["Outcome"].value_counts()*100/len(df)
```

Out[7]:

```
0    65.104167
1    34.895833
Name: Outcome, dtype: float64
```

In [8]:

```
# The classes of the outcome variable were examined.
df.Outcome.value_counts()
```

Out[8]:

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

In [9]:

```
# The histagram of the Age variable was reached.
df["Age"].hist(edgecolor = "black");
```

In[10]:

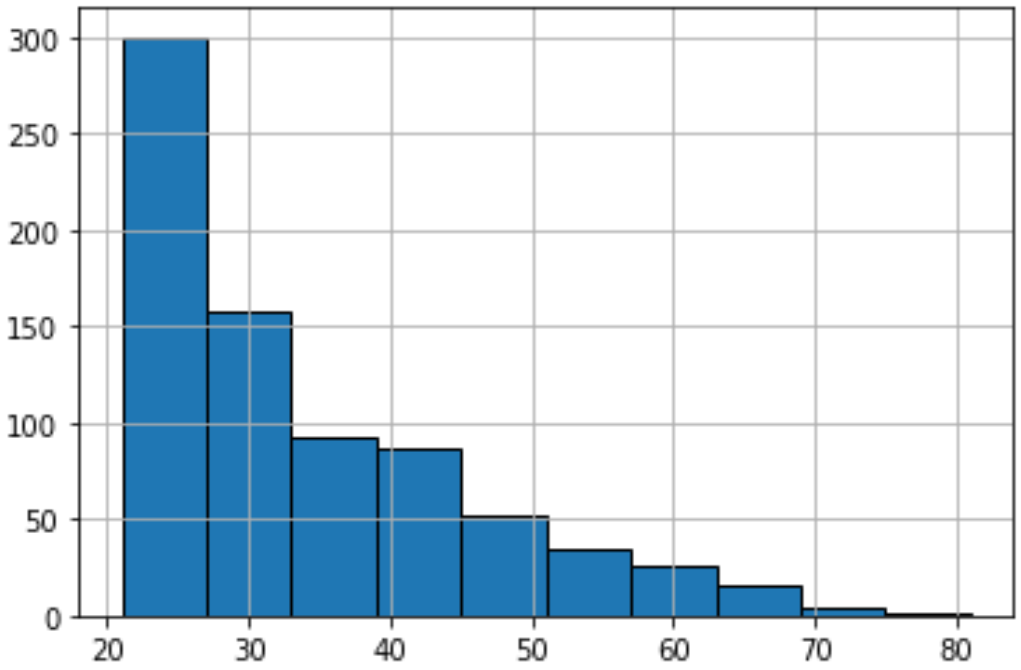
```
print("Max Age: " + str(df["Age"].max()) + " Min Age: " + str(df["Age"].min())
)
```

Max Age: 81 Min Age: 21

In [11]:

linkcode

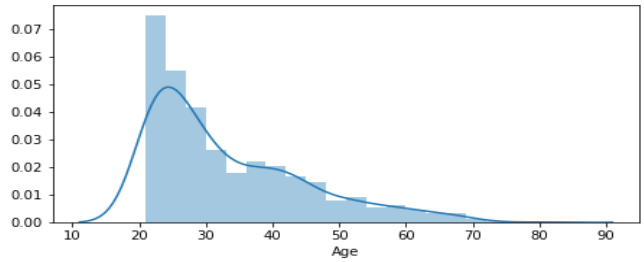
```
# Histogram and density graphs of all variables were accessed.
fig, ax = plt.subplots(4,2, figsize=(16,16))
sns.distplot(df.Age, bins = 20, ax=ax[0,0])
```

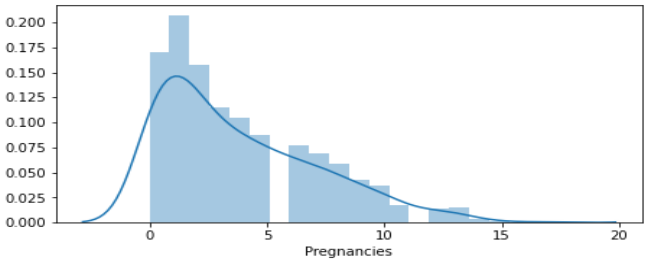


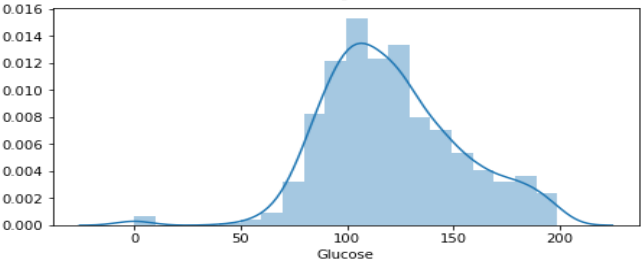

```
sns.distplot(df.Pregnancies, bins = 20, ax=ax[0,1])
sns.distplot(df.Glucose, bins = 20, ax=ax[1,0])
sns.distplot(df.BloodPressure, bins = 20, ax=ax[1,1])
sns.distplot(df.SkinThickness, bins = 20, ax=ax[2,0])
sns.distplot(df.Insulin, bins = 20, ax=ax[2,1])
sns.distplot(df.DiabetesPedigreeFunction, bins = 20, ax=ax[3,0])
sns.distplot(df.BMI, bins = 20, ax=ax[3,1])
```

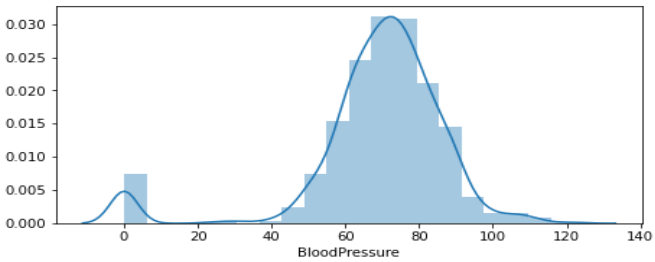
Out[11]:

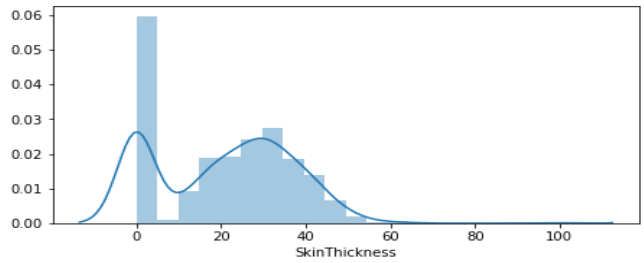
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f77b83d5950>
```

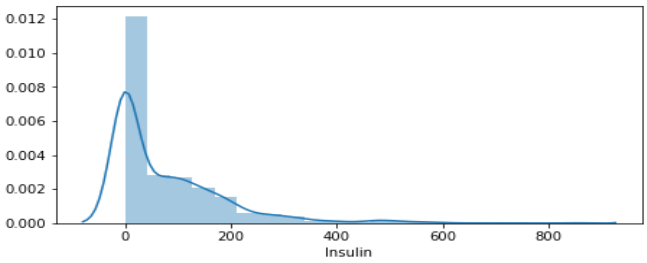


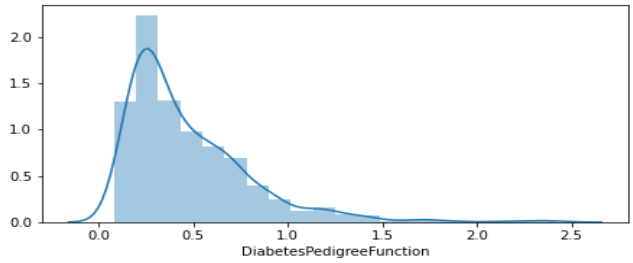


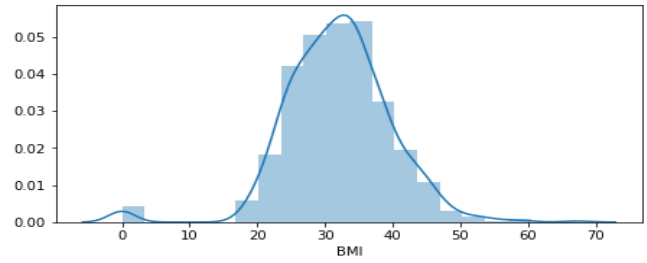












In[12]:

```
df.groupby("Outcome").agg({"Pregnancies":"mean"})
```

Out[12]:

Pregnancies	
Outcome	
0	3.298000
1	4.865672

In[13]:

```
df.groupby("Outcome").agg({"Age":"mean"})
```

Out[13]:

	Age
Outcome	
0	31.190000
1	37.067164

In[14]:

```
df.groupby("Outcome").agg({"Age":"max"})
```

Out[14]:

	Age
Outcome	
0	81
1	70

In[15]:

```
df.groupby("Outcome").agg({"Insulin":  "mean"})
```

Out[15]:

	Insulin
Outcome	
0	68.792000
1	100.335821

In[16]:

```
df.groupby("Outcome").agg({"Insulin":  "max"})
```

Out[16]:

	Insulin
Outcome	
0	744
1	846

In[17]:

```
df.groupby("Outcome").agg({"Glucose":  "mean"})
```

Out[17]:

	Glucose
Outcome	
0	109.980000
1	141.257463

In[18]:

```
df.groupby("Outcome").agg({"Glucose":  "max"})
```

Out[18]:

	Glucose
Outcome	
0	197

	Glucose
Outcome	
1	199

In [19]:

```
df.groupby("Outcome").agg({"BMI":  "mean"})
```

Out[19]:

	BMI
Outcome	
0	30.304200
1	35.142537

In[20]:

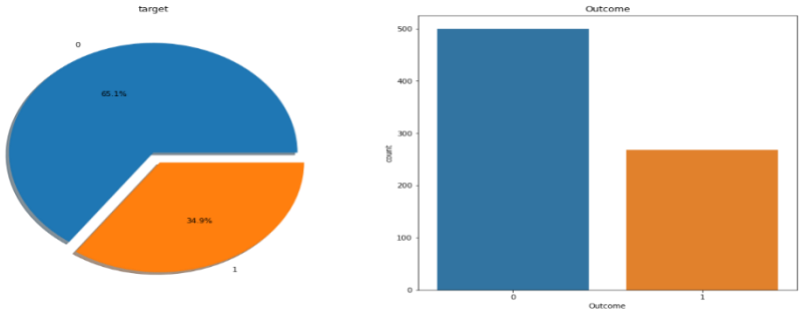
```
# The distribution of the outcome variable in the data was examined and visualized.
f,ax=plt.subplots(1,2,figsize=(18,8))
df['Outcome'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%%',ax=ax[0],
shadow=True)
```

```
ax[0].set_title('target')
ax[0].set_ylabel('')
sns.countplot('Outcome',data=df,ax=ax[1])
ax[1].set_title('Outcome')
plt.show()
```

In[21]:

```
# Access to the correlation of the data set was provided. What kind of relationship is examined between the variables.
# If the correlation value is> 0, there is a positive correlation. While the value of one variable increases, the value of the other variable also increases.
# Correlation = 0 means no correlation.
# If the correlation is <0, there is a negative correlation. While one variable increases, the other variable decreases.
# When the correlations are examined, there are 2 variables that act as a positive correlation to the Salary dependent variable.
# These variables are Glucose. As these increase, Outcome variable increases.
df.corr()
```

Out[21]:



Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

In[22]:

```
# Correlation matrix graph of the data set
f, ax = plt.subplots(figsize= [20,15])
sns.heatmap(df.corr(), annot=True, fmt=".2f", ax=ax, cmap = "magma" )
ax.set_title("Correlation Matrix", fontsize=20)
plt.show()
```

2) Data Preprocessing

2.1) Missing Observation Analysis

We saw on df.head() that some features contain 0, it doesn't make sense here and this indicates missing value Below we replace 0 value by NaN:

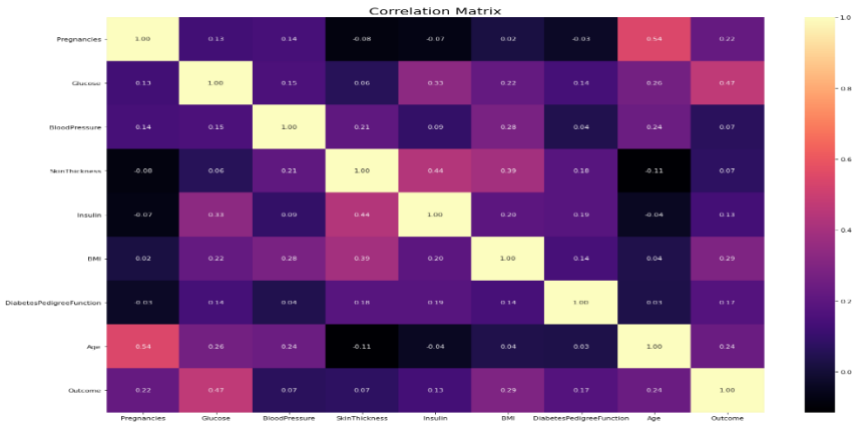
In [23]:

```
df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df[['Glucose','Blood Pressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)
```

In [24]:

```
df.head()
```

Out[24]:



Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
0	6	148.0	72.0	35.0	NaN	33.6	0.627	50	1
1	1	85.0	66.0	29.0	NaN	26.6	0.351	31	0
2	8	183.0	64.0	NaN	NaN	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1

In[25]:

```
# Now, we can look at where are missing values
df.isnull().sum()
```

Out[25]:

```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [26]:

 # Have been visualized using the missingno library for the visualization of missing observations.

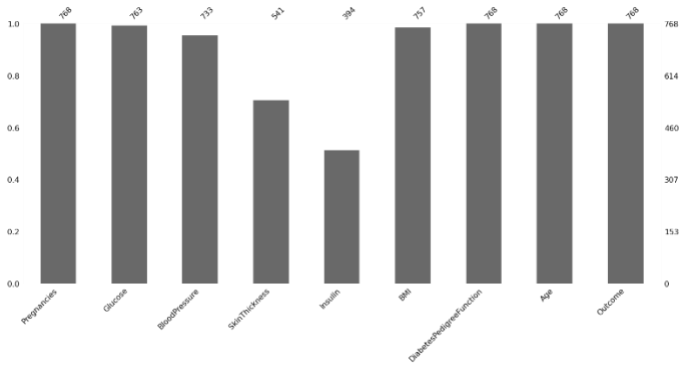
 # Plotting

```
import missingno as msno
msno.bar(df);
```

In[27]:

 # The missing values will be filled with the median values of each variable.

```
def median_target(var):
```




```
    temp = df[df[var].notnull()]
    temp = temp[[var, 'Outcome']].groupby(['Outcome'])[[var]].median().reset_index()
    return temp
```

In [28]:

 # The values to be given for incomplete observations are given the median value of people who are not sick and the median values of people who are sick.

```
columns = df.columns
columns = columns.drop("Outcome")
for i in columns:
    median_target(i)
        df.loc[(df['Outcome'] == 0 ) & (df[i].isnull()), i] = median_target(i)[i][0]
```

```
df.loc[(df['Outcome'] == 1 ) & (df[i].isnull()), i] = median_target(i)[i][1]
```

In [29]:

```
df.head()
```

Out[29]:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
0	6	148.0	72.0	35.0	169.5	33.6	0.627	50	1
1	1	85.0	66.0	29.0	102.5	26.6	0.351	31	0
2	8	183.0	64.0	32.0	169.5	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1

In[30]:

```
# Missing values were filled.
df.isnull().sum()
```

Out[30]:

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
```

```
DiabetesPedigreeFunction        0
Age                             0
Outcome                         0
dtype: int64
```

2.2) Outlier Observation Analysis

In [31]:

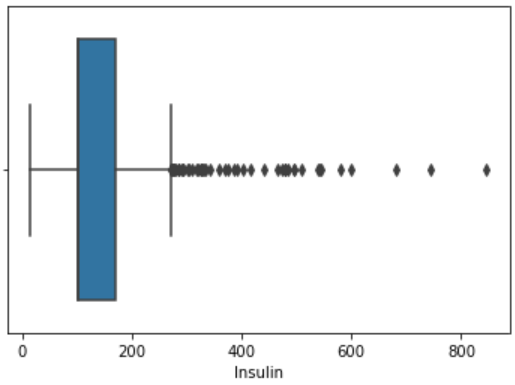
```
# In the data set, there were asked whether there were any outlier obser vations compared to the 25% and 75% quarters.
# It was found to be an outlier observation.
for feature in df:

    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")
```

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

In [32]:

```
# The process of visualizing the Insulin variable with boxplot method was done. We find the outlier observations on the chart.
```

```
import seaborn as sns
```




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

3) Feature Engineering

Creating new variables is important for models. But you need to create a logical new variable. For this data set, some new variables were created according to BMI, Insulin and glucose variables.

In [40]:

```
# According to BMI, some ranges were determined and categorical variables were assigned.
NewBMI = pd.Series(["Underweight", "Normal", "Overweight", "Obesity 1", "Obesity 2", "Obesity 3"], dtype = "category")
df["NewBMI"] = NewBMI
df.loc[df["BMI"] < 18.5, "NewBMI"] = NewBMI[0]
df.loc[(df["BMI"] > 18.5) & (df["BMI"] <= 24.9), "NewBMI"] = NewBMI[1]
df.loc[(df["BMI"] > 24.9) & (df["BMI"] <= 29.9), "NewBMI"] = NewBMI[2]
df.loc[(df["BMI"] > 29.9) & (df["BMI"] <= 34.9), "NewBMI"] = NewBMI[3]
df.loc[(df["BMI"] > 34.9) & (df["BMI"] <= 39.9), "NewBMI"] = NewBMI[4]
df.loc[df["BMI"] > 39.9 ,"NewBMI"] = NewBMI[5]
```

In [41]:

```
linkcode
df.head()
```

Out[41]:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	NewBMI	
0	6	148.0	72.0	35.0	169.5	33.6	0.627	50	1	Obesity 1
1	1	85.0	66.0	29.0	102.5	26.6	0.351	31	0	Overweight
2	8	183.0	64.0	32.0	169.5	23.3	0.672	32	1	Normal
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0	Overweight
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1	Obesity 3

In[42]:

```
# A categorical variable creation process is performed according to the insulin value.
def set_insulin(row):
    if row["Insulin"] >= 16 and row["Insulin"] <= 166:
        return "Normal"
    else:
```

```
        return "Abnormal"
```

In [43]:

```
# The operation performed was added to the dataframe.
df = df.assign(NewInsulinScore=df.apply(set_insulin, axis=1))

df.head()
```

Out[43]:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	NewBMI	NewInsulinScore	
0	6	148.0	72.0	35.0	169.5	33.6	0.627	50	1	Obesity 1	Abnormal
1	1	85.0	66.0	29.0	102.5	26.6	0.351	31	0	Overweight	Normal
2	8	183.0	64.0	32.0	169.5	23.3	0.672	32	1	Normal	Abnormal
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0	Overweight	Normal
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1	Obesity 3	Abnormal

In[44]:

```
# Some intervals were determined according to the glucose variable and t
hese were assigned categorical variables.
NewGlucose = pd.Series(["Low", "Normal", "Overweight", "Secret", "High"], dt
ype = "category")
df["NewGlucose"] = NewGlucose
df.loc[df["Glucose"] <= 70, "NewGlucose"] = NewGlucose[0]
df.loc[(df["Glucose"] > 70) & (df["Glucose"] <= 99), "NewGlucose"] = NewGlu
cose[1]
df.loc[(df["Glucose"] > 99) & (df["Glucose"] <= 126), "NewGlucose"] = NewGl
ucose[2]
df.loc[df["Glucose"] > 126 ,"NewGlucose"] = NewGlucose[3]
```

In [45]:

```
df.head()
```

Out[45]:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	NewBMI	NewInsulinScore	NewGlucose	
0	6	148.0	72.0	35.0	169.5	33.6	0.627	50	1	Obesity 1	Abnormal	Secret
1	1	85.0	66.0	29.0	102.5	26.6	0.351	31	0	Overweight	Normal	Normal
2	8	183.0	64.0	32.0	169.5	23.3	0.672	32	1	Normal	Abnormal	Secret
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0	Overweight	Normal	Normal

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	NewBMI	NewInsulinScore	NewGlucose	
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1	Obesity 3	Abnormal	Secret

Conclusion:

A diabetes prediction system using machine learning (ML) that leverages AI for loading and pre-processing the data set has the potential to be a valuable tool in the field of healthc are. This conclusion is drawn based on several key factors:

- ✓ Data Quality and Accessibility
- ✓ Feature Engineering
- ✓ Model Training
- ✓ Predictive Accuracy
- ✓ Automation and Scalability.
- ✓ Real-time Monitoring
- ✓ Limitations and Ethical Considerations

In conclusion, an AI-based diabetes prediction system that emphasizes loading and pre-processing the dataset holds great promise in the healthcare industry. By automating and enhancing these crucial steps, such systems can improve the accuracy of diabetes predictions and ultimately contribute to better patient care and disease management. However, careful attention must be paid to data quality, ethical considerations, and the ongoing monitoring of these systems to ensure their effectiveness and fairness in practice.