AI Based Diabetes Prediction System

Problem Definition:

The problem is to build an AI-powered diabetes prediction system that uses machine learning algorithms to analyze medical data and predict the likelihood of an individual developing diabetes. The system aims to provide early risk assessment and personalized preventive measures, allowing individuals to take proactive actions to manage their health.

Design Thinking:

Data Collection: We need a dataset containing medical features such as glucose levels, blood pressure, BMI, etc., along with information about whether the individual has diabetes or not.

Data Preprocessing: The medical data needs to be cleaned, normalized, and prepared for training machine learning models.

Feature Selection: We will select relevant features that can impact diabetes risk prediction.

Model Selection: We can experiment with various machine learning algorithms like Logistic Regression, Random Forest, and Gradient Boosting.

Evaluation: We will evaluate the model's performance using metrics like accuracy, precision, recall, F1-score, and ROC-AUC.

Iterative Improvement: We will fine-tune the model parameters and explore techniques like feature engineering to enhance prediction accuracy.

Ensemble Methods:

Ensemble methods combine multiple individual models to generate a final prediction. The idea behind ensemble methods is that the combination of multiple models can produce more accurate and robust predictions compared to a single model.

a. Bagging: Bagging involves training multiple models on different subsets of the training data using bootstrapping. Bagging can help reduce overfitting and improve prediction performance.

b. Boosting: Boosting is an iterative ensemble method in which models are trained sequentially, and each subsequent model focuses on correcting the mistakes made by the previous models.

c. Random Forest: Random Forest is an ensemble method that combines multiple decision trees. Each tree is trained on a random subset of features and data samples.

Feature Engineering and Selection:

Feature engineering involves creating new features or transforming existing features to provide more informative representations of the data. Domain knowledge and insights can be leveraged to engineer relevant features that capture important characteristics of diabetes. Additionally, feature selection techniques such as L1 regularization (Lasso) or tree-based feature importance can help identify the most relevant features, reducing noise and improving prediction performance

Model Interpretability:

Interpretable models can provide insights into the reasoning behind predictions, enhancing transparency and trustworthiness. Techniques such as feature importance analysis, attention mechanisms, or rule-based models can be employed to interpret and explain the model's predictions, enabling better understanding and acceptance by healthcare professionals and patients.

Loading the Dataset:

Downloaded the diabetes.csv dataset from the Kaggle. We loaded a diabetes.csv dataset using the pandas library.

```
[1] import pandas as pd
```

```
[2] dataFrame = pd.read_csv('diabetes.csv')
    dataFrame.head()
```

	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

Data Preprocessing:

Checking whether there are any null values in the dataset

```
dataFrame.isnull()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
763	False	False	False	False	False	False	False	False	False
764	False	False	False	False	False	False	False	False	False
765	False	False	False	False	False	False	False	False	False
766	False	False	False	False	False	False	False	False	False
767	False	False	False	False	False	False	False	False	False

768 rows × 9 columns

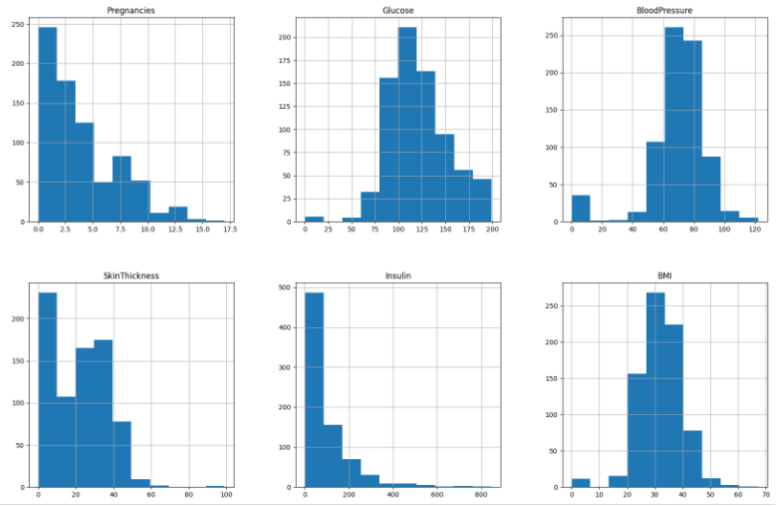
```
[3] dataFrame.isnull().sum()
```

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

Data Visualization:

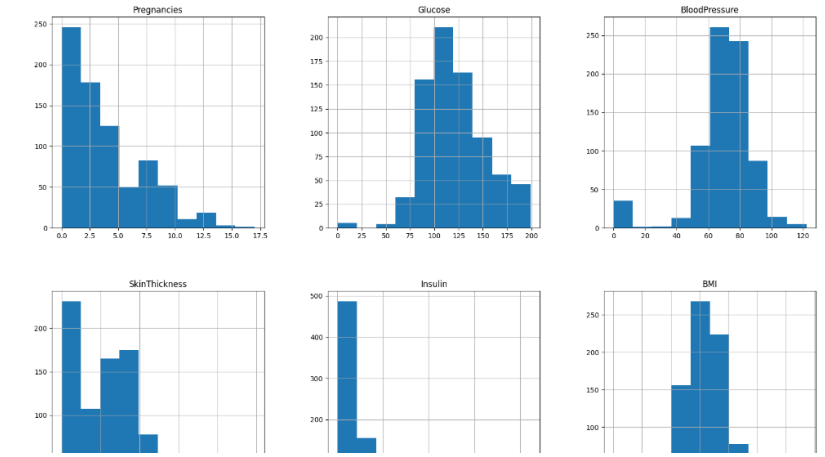
Plotting the distributions after removing the NAN values.

```
[23] p = dataFrame.hist(figsize = (20,20))
```



```
dataFrame['Glucose'].fillna(dataFrame['Glucose'].mean(), inplace = True)
dataFrame['BloodPressure'].fillna(dataFrame['BloodPressure'].mean(), inplace = True)
dataFrame['SkinThickness'].fillna(dataFrame['SkinThickness'].median(), inplace = True)
dataFrame['Insulin'].fillna(dataFrame['Insulin'].median(), inplace = True)
dataFrame['BMI'].fillna(dataFrame['BMI'].median(), inplace = True)
```

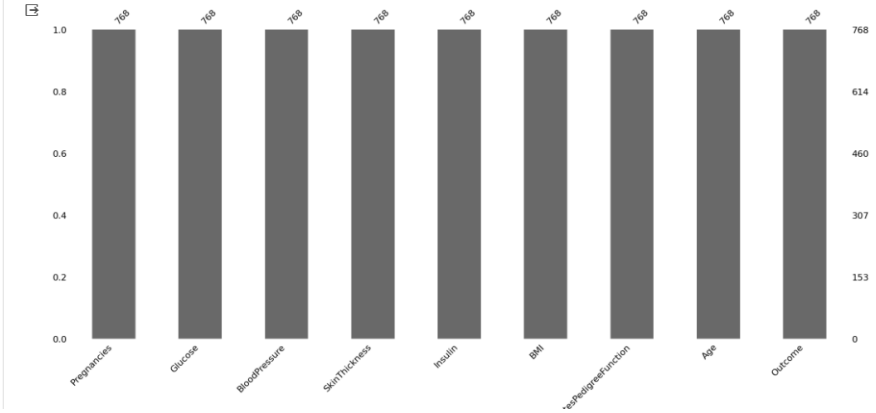
```
p = dataFrame.hist(figsize = (20,20))
```



Plotting Null Count Analysis Plot

```
[21] import missingno as msno
```

```
p = msno.bar(dataFrame)
```



Selecting a machine learning algorithm:

In general, for classification problems, several machine learning algorithms have shown promising results. Few commonly used algorithms were:

Logistic Regression, Random Forest, Support Vector Machines (SVM), Gradient Boosting and Neural Networks

Here, we have chosen the Random Forest algorithm.

Random Forest is an ensemble method that combines multiple decision trees to make accurate predictions. It can handle complex and high-dimensional datasets, making it suitable for diabetes prediction with multiple input features.

[Preprocessing steps, such as handling missing values and feature scaling, should be performed before training the Random Forest model.]

Training the model:

Training the model is a crucial step in machine learning where the model learns patterns and relationships from labeled data to make predictions or classifications.

The training process involves feeding the model with input features and their corresponding target labels, which indicate the desired output or prediction.

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the dataset into a DataFrame
data = pd.read_csv('diabetes.csv')

# Split the dataset into features and labels
X = diabetes_dataframe.drop('Outcome', axis=1)
y = diabetes_dataframe['Outcome']
```

```
[43] # Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Random Forest classifier
model = RandomForestClassifier()

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

```
RandomForestClassifier
RandomForestClassifier()
```

```
[44] # Make predictions on the testing data
y_pred = model.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7532467532467533
```

Evaluating the performance:

Evaluating the performance of a machine learning model is essential to assess its accuracy, reliability, and effectiveness in making predictions or classifications.

Performance evaluation involves measuring the model's performance metrics using test data that was not used during training.

Common performance metrics for classification tasks include accuracy, precision, recall, F1 score

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Make predictions on the testing data
y_pred = model.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Calculate the precision of the model
precision = precision_score(y_test, y_pred)
print("Precision:", precision)

# Calculate the recall of the model
recall = recall_score(y_test, y_pred)
print("Recall:", recall)

# Calculate the F1 score of the model
f1 = f1_score(y_test, y_pred)
print("F1 score:", f1)
```

```
Accuracy: 0.7532467532467533
Precision: 0.6440677966101694
Recall: 0.6909090909090909
F1 score: 0.6666666666666665
```


TEAM MEMBERS

JEFFY J

JENIFER P

PRIYA MAHALAKSHMI P

MALLIKA S

J P College of Engineering