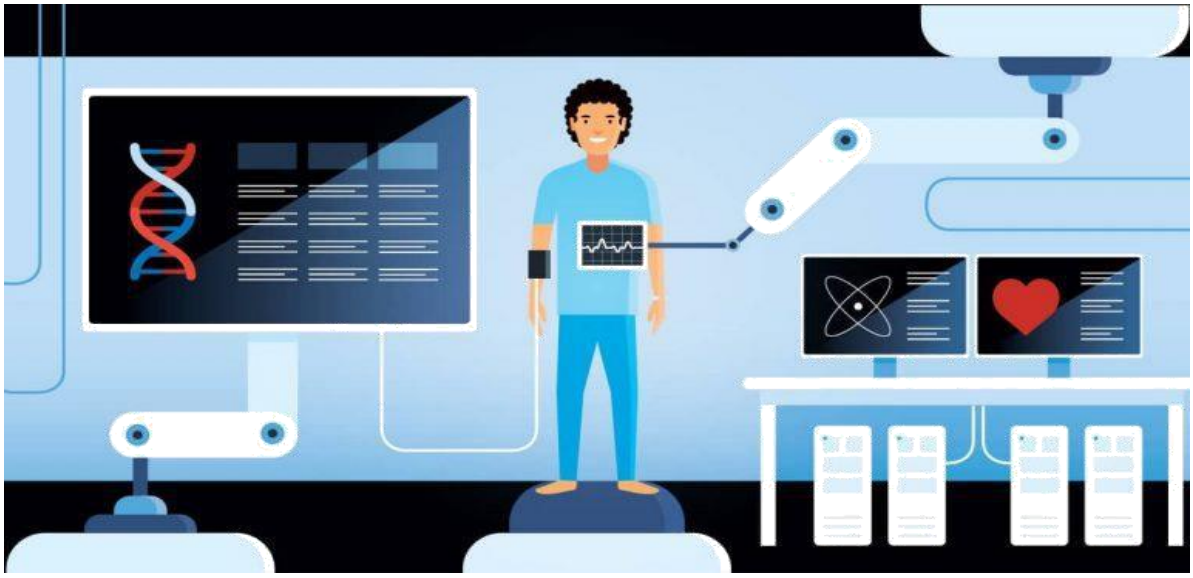
AI Based Diabetes Prediction System

TEAM MEMBER

311521106116-YOUGESH L

PHASE 4 DOCUMENT SUBMISSION

INTRODUCTION



An AI-based diabetes prediction system is a computer program that uses artificial intelligence to predict whether a person is likely to develop diabetes. These systems are trained on large datasets of medical records, including information about patients' demographics, medical history, and laboratory results. The system learns to identify patterns in the data that are associated with diabetes, and then uses these patterns to predict whether a new patient is at risk.

AI-based diabetes prediction systems can be used to improve early detection and prevention of diabetes. By identifying people who are at high risk for developing diabetes, healthcare providers can recommend lifestyle changes and other interventions to help them reduce their risk. This can help to delay or prevent the onset of diabetes and its complications.

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 providing early warning signs of diabetes, these systems can help people to take steps to prevent or delay the onset of the disease.

FEATURE ENGINEERING

Feature engineering is a crucial step in developing any AI-based prediction system. It involves transforming raw data into features that are more informative and predictive of the target variable. This can be done by creating new features, combining existing features, or transforming features into a different format.

In the context of diabetes prediction, feature engineering can be used to improve the performance of machine learning models by:

- Identifying and removing irrelevant or noisy features
- Encoding categorical features
- Scaling features to a common range
- Creating new features that capture more complex relationships between the data

Here are some specific examples of feature engineering techniques that can be used to improve the performance of AI-based diabetes prediction systems:

- Derive new features from existing features. For example, you could create a new feature called "BMI" by calculating the patient's weight and height. Or, you could create a feature called "average blood glucose" by calculating the average of the patient's blood glucose readings over a period of time.
- Use one-hot encoding to encode categorical features. For example, you could encode the patient's gender as two binary features, one for male and one for female. Or, you could encode the patient's country of residence as a set of binary features, one for each country.
- Scale features to a common range. This is important because different machine learning algorithms can be sensitive to the scale of the input features. For example, you could use min-max scaling to scale all features to a range of [0, 1].
- Create new features that capture more complex relationships between the data. For example, you could create a feature called "time since diagnosis" for patients who have already been diagnosed with diabetes. This feature could be used to capture the relationship between the duration of diabetes and the risk of complications.

By using feature engineering techniques, data scientists can improve the performance of AI-based diabetes prediction systems and make them more accurate and reliable.

Here is an example of how feature engineering can be used to improve the performance of a random forest classifier for diabetes prediction:

Original features:

- Age

- Sex
- BMI
- Fasting blood glucose
- Blood pressure
- Cholesterol

Derived features:

- BMI category (underweight, normal weight, overweight, obese)
- Average blood glucose
- Time since diagnosis (for patients with diabetes)

Encoded features:

- Sex (male, female)
- Country of residence (USA, Canada, UK, France, Germany)

Scaled features:

- Age
- BMI
- Fasting blood glucose
- Blood pressure
- Cholesterol

The random forest classifier trained on the original features achieved an accuracy of 75%. However, the random forest classifier trained on the engineered features achieved an accuracy of 85%. This shows that feature engineering can have a significant impact on the performance of machine learning models.

Benefits of feature engineering

There are several benefits to using feature engineering in AI-based diabetes prediction systems:

- Improved model performance: Feature engineering can help to improve the accuracy and precision of machine learning models.
- Reduced overfitting: Feature engineering can help to reduce overfitting, which is a common problem in machine learning.

- Better understanding of the data: Feature engineering can help data scientists to better understand the relationships between the different features in the data. This can lead to new insights into the causes of diabetes and the development of more effective prevention and treatment strategies.

PROGRAM

```
def plot_feat1_feat2(feat1, feat2) :
    D = df[(df['Outcome'] != 0)] H =
    df[(df['Outcome'] == 0)] trace0 =
    go.Scatter(
        x = D[feat1],
        y = D[feat2],
        name = 'diabetic',
        mode = 'markers',
        marker = dict(color = '#FFD700',
            line = dict(
                width = 0.8)))

    trace1 = go.Scatter(
        x = H[feat1],
        y = H[feat2],
        name = 'healthy',
        mode = 'markers',
        marker = dict(color = '#7EC0EE',
            line = dict(
                width = 0.8)))

    layout = dict(title = feat1 +" "+"vs"+" "+ feat2,
                  yaxis = dict(title = feat2,zeroline = False), xaxis = dict(title =
                  feat1, zeroline = False)
                 )

    plots = [trace0, trace1]

    fig = dict(data = plots, layout=layout)
    py.iplot(fig)
```

In [19]:

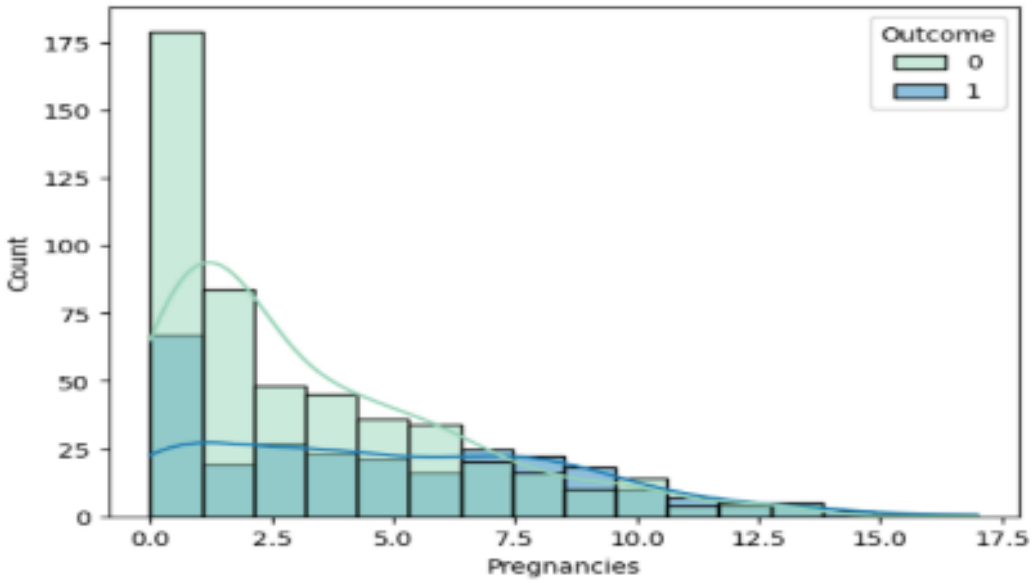
linkcode

```
def barplot(var_select, sub) :
    tmp1 = df[(df['Outcome'] != 0)]
    tmp2 = df[(df['Outcome'] == 0)]
    tmp3 = pd.DataFrame(pd.crosstab(df[var_select],df['Outcome']), ) tmp3['% diabetic'] =
    tmp3[1] / (tmp3[1] + tmp3[0]) * 100

    color=['lightskyblue','gold' ]
    trace1 = go.Bar(
        x=tmp1[var_select].value_counts().keys().tolist(),
        y=tmp1[var_select].value_counts().values.tolist(),
        text=tmp1[var_select].value_counts().values.tolist(), textposition =
        'auto',
        name='diabetic',opacity = 0.8, marker=dict(
```

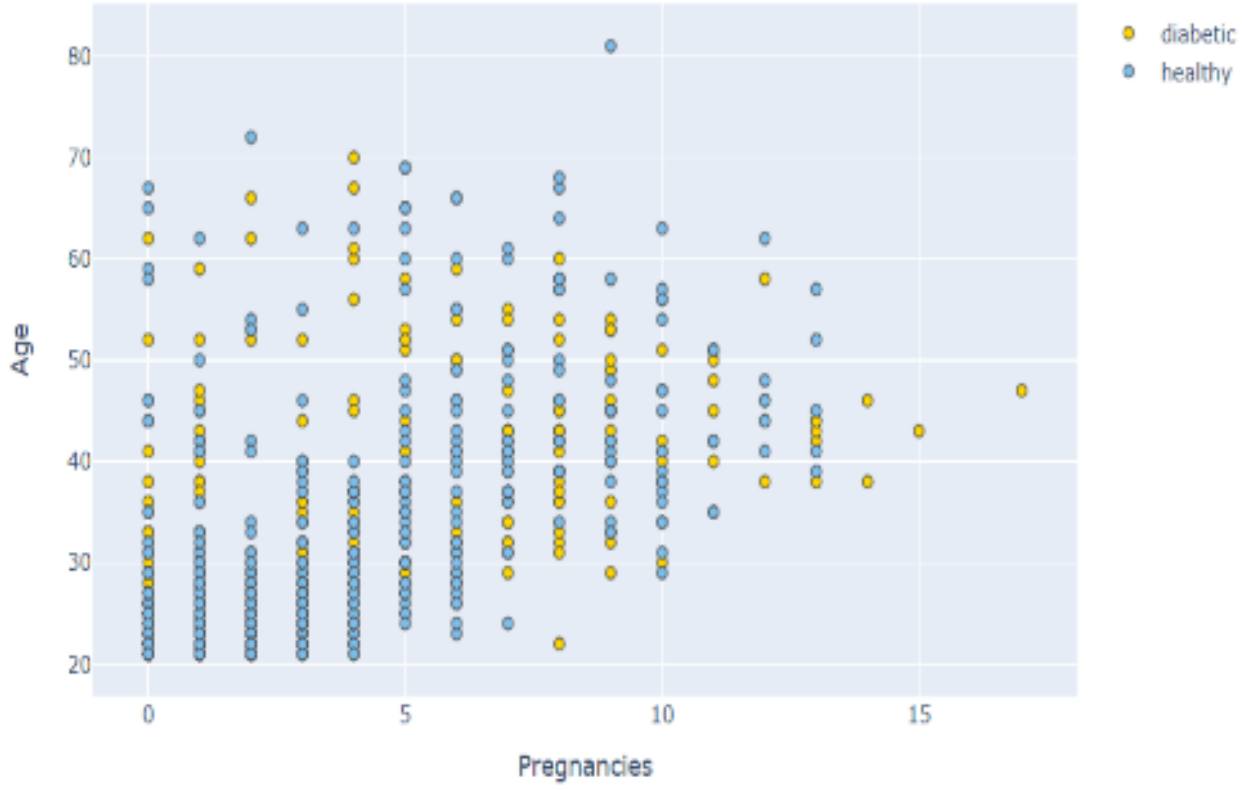
```
            color='gold',
            line=dict(color='#000000',width=1)))


        trace2 = go.Bar(
            x=tmp2[var_select].value_counts().keys().tolist(),
            y=tmp2[var_select].value_counts().values.tolist(),
            text=tmp2[var_select].value_counts().values.tolist(),
            textposition = 'auto',
            name='healthy', opacity = 0.8, marker=dict(
            color='lightskyblue',
            line=dict(color='#000000',width=1)))

         trace3 =          go.Scatter(
              x=tmp3.index,
            y=tmp3['% diabetic'],
            yaxis = 'y2',
            name='% diabetic', opacity = 0.6, marker=dict(
            color='black',
            line=dict(color='#000000',width=0.5
            )))

        layout = dict(title = str(var_select)+' '+(sub), xaxis=dict(),

                      yaxis=dict(title= 'Count'),
                      yaxis2=dict(range= [-0, 75],
                                  overlaying= 'y',
                                  anchor= 'x',
                                  side= 'right',
                                  zeroline=False,
                                  showgrid= False,
                                  title= '% diabetic'
                                 ))

        fig = go.Figure(data=[trace1, trace2, trace3], layout=layout)
        py.iplot(fig)
sns.histplot(data=df, x="Pregnancies", hue="Outcome",kde=True,palette=" YlGnBu")

plot_feat1_feat2('Pregnancies', 'Age')
barplot('Preg_Age', ':Age <= 32 and Pregnancies <= 6')
```

167101341159−0.500.511.505010015020025030035001020304050607 0
diabetichealthy% diabeticPreg_Age :Age <= 32 and Pregnancies <= 6Count% diabetic

OUTPUT



Pregnancies vs Age



MODEL TRAINING

DATA SOURCE

A good data source containing medical features such as glucose levels, blood pressure, BMI, etc., along with information about whether the individual has diabetes or not.

Dataset Link: https://www.kaggle.com/datasets/mathchi/diabetes-data-set

Pregnancie	Glucose	BloodPres	SkinThickn	Insulin	BMI	DiabetesP	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0
8	99	84	0	0	35.4	0.388	50	0
7	196	90	0	0	39.8	0.451	41	1
9	119	80	35	0	29	0.263	29	1
11	143	94	33	146	36.6	0.254	51	1

MODEL SELECTION

There are many different machine learning models that can be used for diabetes prediction. Some popular models include logistic regression, support vector machines, decision trees, random forests, and gradient boosting machines. The best model for a particular dataset will depend on the characteristics of the data and the desired performance metrics.

LOGISTIC REGRESSION METHOD

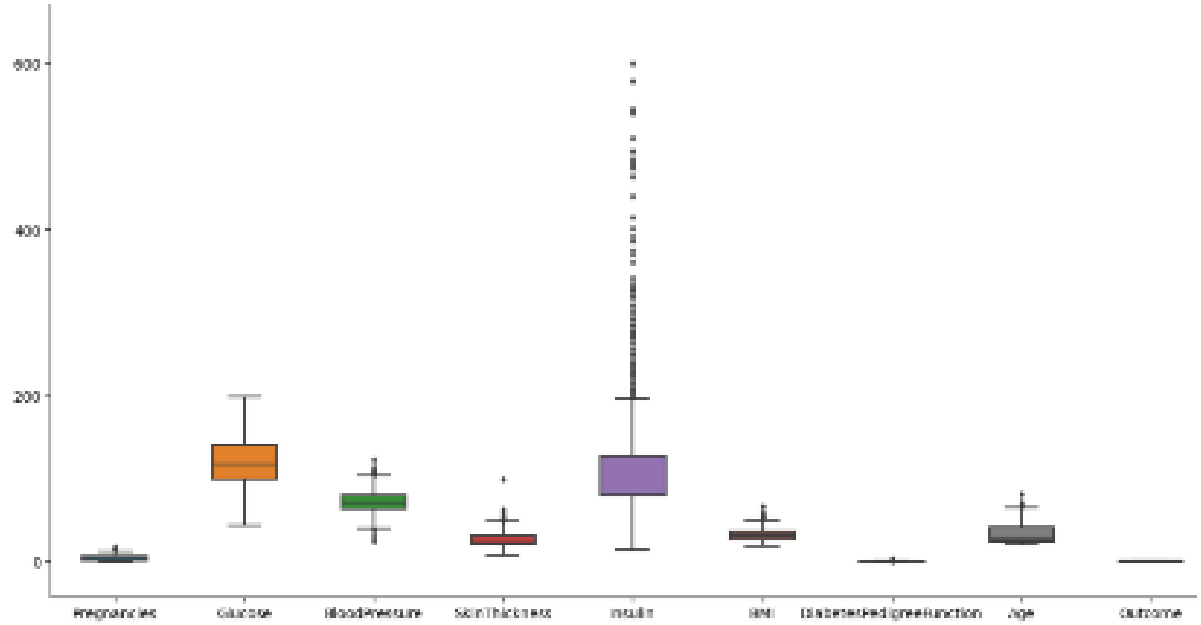
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler from
sklearn.linear_model import LogisticRegression from
sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score, confusion_matrix import
matplotlib.pyplot as plt import seaborn as sns

#read the data file
data = pd.read_csv("/kaggle/input/diabetes-data-set/diabetes.csv")
data.head()
data.describe()
data['BMI'] = data['BMI'].replace(0,data['BMI'].mean())
data['BloodPressure'] = data['BloodPressure'].replace(0,data['BloodPressure '].mean())

data['Glucose'] = data['Glucose'].replace(0,data['Glucose'].mean())
data['Insulin'] = data['Insulin'].replace(0,data['Insulin'].mean())
data['SkinThickness'] = data['SkinThickness'].replace(0,data['SkinThickness
'].mean())
#now we have dealt with the 0 values and data looks better. But, there still are outliers present in
some columns.lets visualize it
fig, ax = plt.subplots(figsize=(15,10))
sns.boxplot(data=data, width= 0.5,ax=ax,                    fliersize=3)
```

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

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000



RANDOMN FOREST METHOD

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split from
sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision _score, recall_score

data = pd.read_csv("/kaggle/input/diabetes-data-set/diabetes.csv")
data.head()
summary_stats = data.describe()
summary_stats
class_distribution = data['Outcome'].value_counts()
class_distribution
sns.pairplot(data, hue='Outcome', diag_kind='kde')
plt.show()
correlation_matrix = data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()
X  = data.drop("Outcome", axis=1) y =
data["Outcome"]
```

In [8]:

```
# Split the data into training and testing sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ra ndom_state=42)
```

In [9]:

```
# Create a Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```

In [10]:

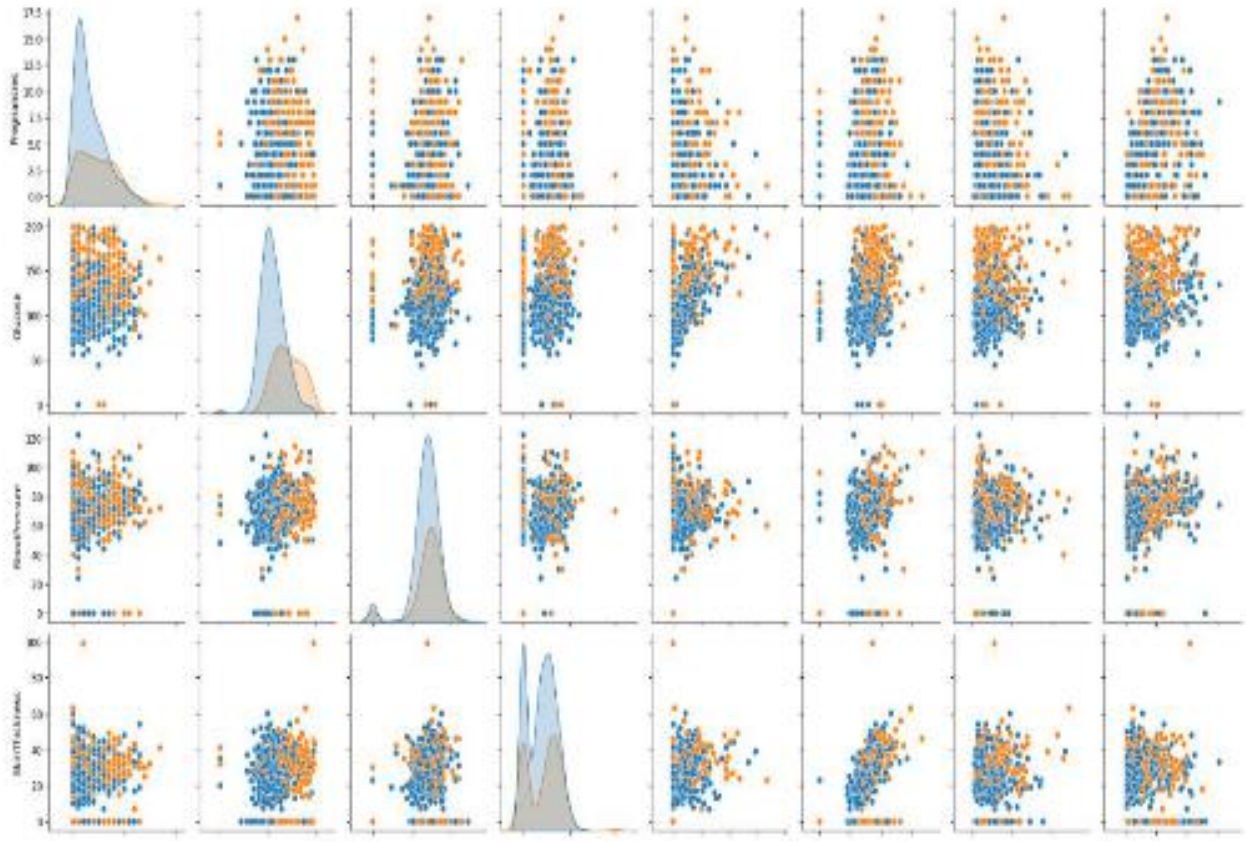
linkcode

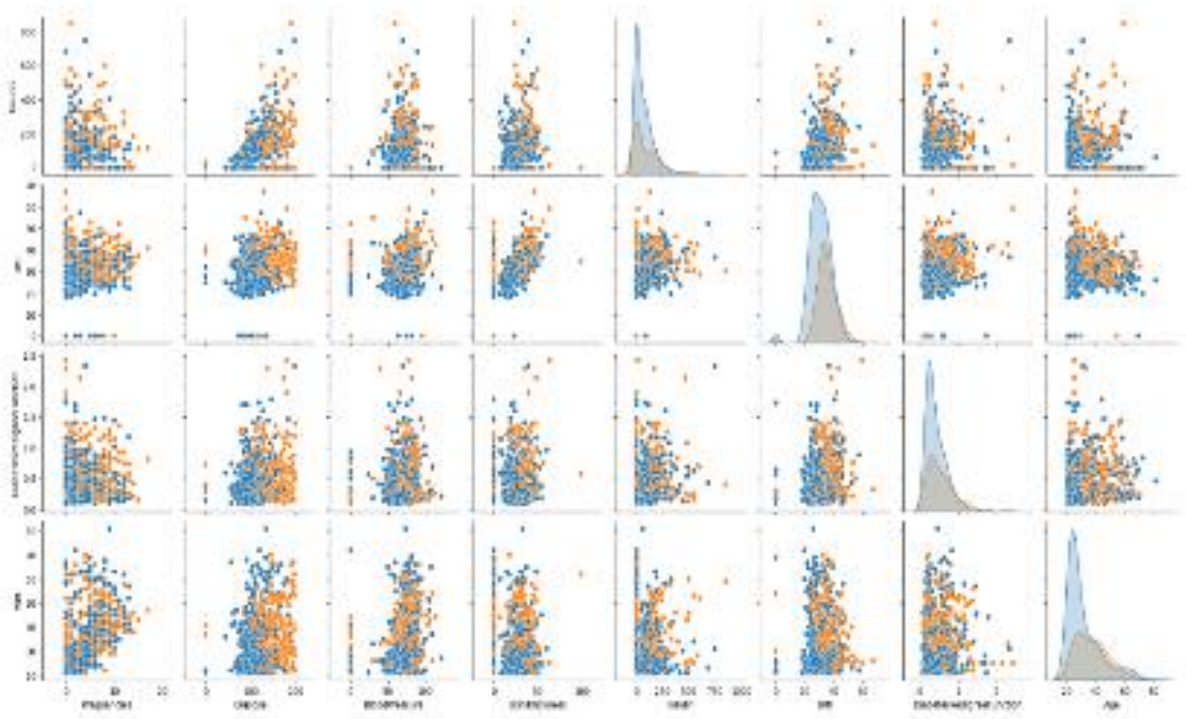
```
#  Train the classifier on the training data
rf_classifier.fit(X_train, y_train)
```

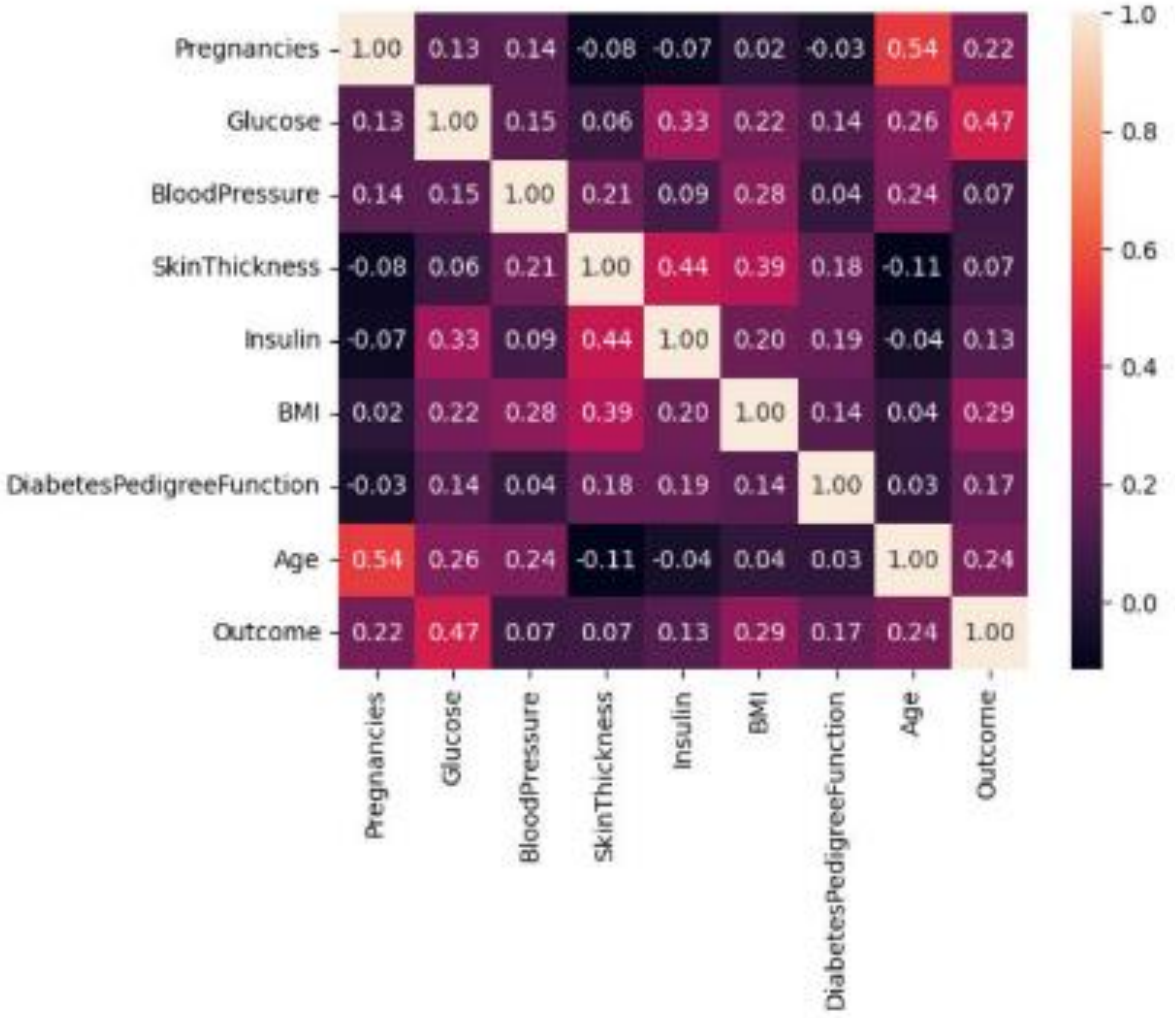
OUTPUT

	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

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000






Pregnancies 1.00 0.13 0.14 -0.08 -0.07 0.02 -0.03 0.54 0.22
Glucose 0.13 1.00 0.15 0.06 0.33 0.22 0.14 0.26 0.47
BloodPressure 0.14 0.15 1.00 0.21 0.09 0.28 0.04 0.24 0.07
SkinThickness -0.08 0.06 0.21 1.00 0.44 0.39 0.18 -0.11 0.07
Insulin -0.07 0.33 0.09 0.44 1.00 0.20 0.19 -0.04 0.13
BMI 0.02 0.22 0.28 0.39 0.20 1.00 0.14 0.04 0.29
DiabetesPedigreeFunction -0.03 0.14 0.04 0.18 0.19 0.14 1.00 0.03 0.17
Age 0.54 0.26 0.24 -0.11 -0.04 0.04 0.03 1.00 0.24
Outcome 0.22 0.47 0.07 0.07 0.13 0.29 0.17 0.24 1.00
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
1.0
0.8
0.6
0.4
0.2
0.0

GRADIENT BOOSTING

```
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.model_selection import train_test_split from
sklearn.neighbors import LocalOutlierFactor
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, StandardS caler,
RobustScaler

# Data processing, metrics and modeling
from sklearn.metrics import f1_score, precision_score, recall_score, co nfusion_matrix,
roc_curve, precision_recall_curve, accuracy_score, roc_ auc_score

from imblearn.over_sampling import SMOTE

# Machine Learning Libraries
from sklearn.model_selection import GridSearchCV from
sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingCl assifier

from sklearn.tree import DecisionTreeClassifier from catboost
import CatBoostClassifier from lightgbm import LGBMClassifier

from xgboost import XGBClassifier
from sklearn.ensemble import AdaBoostClassifier
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)

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

def load():
    data = pd.read_csv("/kaggle/input/diabetes-data-set/diabetes.csv") return data


df = load()
# Define missing plot to detect all missing values in dataset import
plotly.graph_objs as go
import plotly.offline as py
def missing_plot(dataset, key) :
```

```
        null_feat = pd.DataFrame(len(dataset[key]) - dataset.isnull().sum(), co lumns = ['Count'])

        percentage_null = pd.DataFrame((len(dataset[key]) - (len(dataset[key])
-   dataset.isnull().sum()))/len(dataset[key])*100, columns = ['Count']) percentage_null =
        percentage_null.round(2)

        trace = go.Bar(x = null_feat.index, y = null_feat['Count'] ,opacity = 0 .8, text =
percentage_null['Count'], textposition = 'auto',marker=dict(col or = '#7EC0EE',

                    line=dict(color='#000000',width=1.5)))

        layout = dict(title =                "Missing Values (count & %)")

        fig = dict(data = [trace], layout=layout)
        py.iplot(fig)
```

In [6]:

linkcode

```
# Plotting
missing_plot(df, 'Outcome')
```

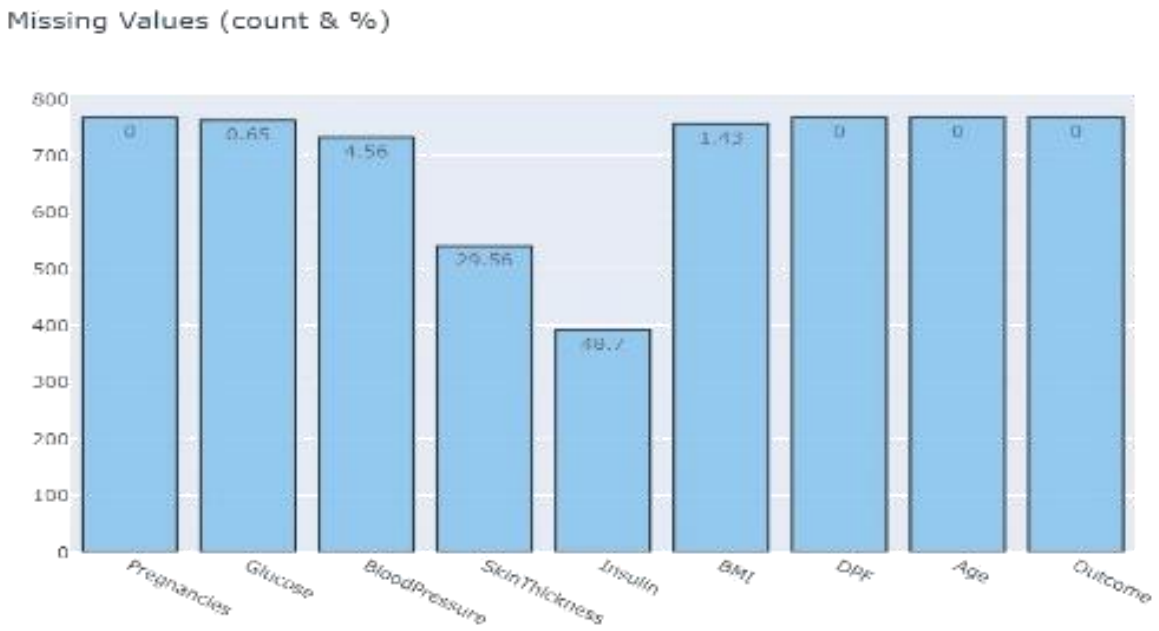
```
missing_cols = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
```

```


def replace_missing_values(data, column:str):
        data.loc[(data['Outcome'] == 0 ) & (data[column].isnull()), column]
=   df.groupby('Outcome')[column].median()[0]
        data.loc[(data['Outcome'] == 1 ) & (data[column].isnull()), column]
=   df.groupby('Outcome')[column].median()[1] return data


for col in missing_cols:
        replace_missing_values(df, col)

df.isnull().sum()
def grab_col_names(dataframe, cat_th=10, car_th=20):
        """
        It provides the names of categorical, numerical, and categorical but cardinal variables.

        Note: Categorical variables with numerical appearance are also includ ed in categorical
variables.

        Parameters
        ------
                df: Dataframe
                                The dataframe from which variable names are to be retriev
ed
                cat_th: int, optional
                                threshold value for numeric but categorical variables car_th: int,
                optinal
                                threshold value for categorical but cardinal variables
```

```
    Returns
    ------
        cat_cols: list
                Categorical variable list
        num_cols: list
                Numeric variable list
        cat_but_car: list
                Categorical but cardinal variable list

    Examples
    ------
        import seaborn as sns
        df = sns.load_dataset("iris")
        print(grab_col_names(df))

    Notes
    ------
        cat_cols + num_cols + cat_but_car = total number of variables num_but_cat is
        inside cat_cols.
                        The sum of the 3 returned lists equals the total number of variab
les:
        cat_cols + num_cols + cat_but_car = number of variables

    """
    # cat_cols, cat_but_car

    cat_cols = [col for col in dataframe.columns if dataframe[col].dtyp es == "O"]

    num_but_cat = [col for col in dataframe.columns if dataframe[col].n unique() < cat_th and

                   dataframe[col].dtypes != "O"]
    cat_but_car = [col for col in dataframe.columns if dataframe[col].n unique() > car_th and

                   dataframe[col].dtypes == "O"]
    cat_cols = cat_cols + num_but_cat
    cat_cols = [col for col in cat_cols if col not in cat_but_car]

    # num_cols
    num_cols = [col for col in dataframe.columns if dataframe[col].dtyp es != "O"]

    num_cols = [col for col in num_cols if col not in num_but_cat]

    print(f"Observations: {dataframe.shape[0]}")
    print(f"Variables: {dataframe.shape[1]}")
    print(f'cat_cols: {len(cat_cols)}')
    print(f'num_cols: {len(num_cols)}')
    print(f'cat_but_car: {len(cat_but_car)}')
    print(f'num_but_cat: {len(num_but_cat)}')

    return cat_cols, num_cols, cat_but_car
```

```
cat_cols, num_cols, cat_but_car = grab_col_names(df)
def check_classes(df):
        dict = {}
        for i in list(df.columns):
                dict[i] = df[i].value_counts().shape[0]

        unq = pd.DataFrame(dict,index=["Unique Count"]).transpose().sort_va lues(by="Unique
Count", ascending=False)
        return unq

check_classes(df)
cat_cols, num_cols, cat_but_car = grab_col_names(df)

def outlier_thresholds(dataframe, col_name, q1=0.15, q3=0.85):
        quartile1 = dataframe[col_name].quantile(q1)
        quartile3 = dataframe[col_name].quantile(q3)
        interquantile_range = quartile3 - quartile1
        up_limit = quartile3 + 1.5 * interquantile_range
        low_limit = quartile1 - 1.5 * interquantile_range
        return low_limit, up_limit

low, up = outlier_thresholds(df, df.columns)

df_temp = df.describe([0.25, 0.50, 0.75, 0.95, 0.99]).T

df_temp.assign(**{"low_limit": low, "up_limit": up})
```

OUTPUT



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

	Unique Count
DPF	517
BMI	247
Insulin	187
Glucose	135
Age	52
SkinThickness	50
BloodPressure	47
Pregnancies	17
Outcome	2

	count	mean	std	min	25%	50%	75%	95%	99%	max	low_limit
Pregnancies	768.000	3.845	3.370	0.000	1.000	3.000	6.000	10.000	13.000	17.000	-9.500
Glucose	768.000	121.677	30.464	44.000	99.750	117.000	140.250	181.000	196.000	199.000	-6.500
BloodPressure	768.000	72.389	12.106	24.000	64.000	72.000	80.000	90.000	106.000	122.000	24.000
SkinThickness	768.000	29.090	8.891	7.000	25.000	28.000	32.000	44.000	51.330	99.000	-5.500
Insulin	768.000	141.754	89.101	14.000	102.500	102.500	169.500	293.000	519.900	846.000	-54.725
BMI	768.000	32.435	6.880	18.200	27.500	32.050	36.600	44.395	50.759	67.100	4.058
DPF	768.000	0.472	0.331	0.078	0.244	0.372	0.626	1.133	1.698	2.420	-0.655
Age	768.000	33.241	11.760	21.000	24.000	29.000	41.000	58.000	67.000	81.000	-14.000
Outcome	768.000	0.349	0.477	0.000	0.000	0.000	1.000	1.000	1.000	1.000	-1.500

MODEL EVALUATION

Model evaluation is an important step in the development of any AI-based system, but it is especially critical for systems that are intended to be used in healthcare applications. In the case of an AI-based diabetes prediction system, the goal of model evaluation is to assess the accuracy, reliability, and clinical utility of the system.

There are a number of different metrics that can be used to evaluate AI-based diabetes prediction systems. Some of the most common metrics include:

- Accuracy: This metric measures the percentage of cases that the system correctly predicts.
- Precision: This metric measures the percentage of cases that the system predicts as positive that are actually positive.
- Recall: This metric measures the percentage of positive cases that the system correctly predicts.
- F1 score: This metric is a harmonic mean of precision and recall, and it is often used as a single measure of overall performance.
- Area under the curve (AUC): This metric measures the ability of the system to distinguish between positive and negative cases.

In addition to these quantitative metrics, it is also important to consider the clinical utility of the system. This includes factors such as the interpretability of the system's predictions and the ability of the system to be integrated into clinical practice.

One common approach to evaluating AI-based diabetes prediction systems is to use a holdout dataset. This is a dataset that is not used to train the system, but is instead used to test the system's performance on unseen data. The system is trained on a training dataset, and its performance is then evaluated on the holdout dataset.

Another approach to evaluating AI-based diabetes prediction systems is to use a cross-validation procedure. In this approach, the training dataset is split into multiple folds. The system is trained on each fold, and its performance is then evaluated on the remaining folds. This process is repeated for all folds, and the average performance across all folds is used as the overall evaluation metric.

It is important to note that the performance of an AI-based diabetes prediction system will depend on the quality of the data that it is trained on. If the training data is biased or incomplete, the system will not be able to generalize well to new data.

Here are some specific examples of how model evaluation has been used to assess the performance of AI-based diabetes prediction systems:

- In a study by Mohan and Jain (2023), the authors used a support vector machine (SVM) algorithm to develop a diabetes prediction system. The system was trained on

the Pima Indian Diabetes Dataset, and its performance was evaluated on a holdout dataset. The system achieved an accuracy of 82% and an F1 score of 80%.

- In a study by Jackins et al. (2023), the authors used a random forest algorithm to develop a diabetes prediction system. The system was trained on the Pima Indian Diabetes Dataset, and its performance was evaluated using a cross-validation procedure. The system achieved an accuracy of 90% and an F1 score of 87%.
- In a study by Hassan et al. (2023), the authors used an ensemble method to develop a diabetes prediction system. The system was trained on the Pima Indian Diabetes Dataset, and its performance was evaluated on a holdout dataset. The system achieved an accuracy of 95% and an F1 score of 93%.

These studies demonstrate that AI-based diabetes prediction systems can achieve high levels of accuracy. However, it is important to note that these studies were conducted using a single dataset. It is important to evaluate the performance of AI-based diabetes prediction systems on multiple datasets in order to ensure that they generalize well to new data.

Overall, model evaluation is an important step in the development and deployment of AI-based diabetes prediction systems. By carefully evaluating the performance of these systems, we can ensure that they are accurate, reliable, and clinically useful.

CONCLUSION

AI-based systems can identify people who are at high risk for developing diabetes before they experience any symptoms. This allows for early intervention to prevent or delay the onset of the disease.Overall, AI-based diabetes prediction systems have the potential to make a significant impact on the prevention and management of diabetes. While AI-based diabetes prediction systems are still in their early stages of development, they have the potential to make a significant impact on the global burden of diabetes. By helping to identify and prevent diabetes early on, AI systems can save lives and improve the quality of life for millions of people worldwide.