AI_BASED DIABETES PREDICTION SYSTEM USING MACHINE LEARNING.

TEAM LEADER

922321106004: AKSHAYASRI

Phase 3: Development part 1

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

AI_BASED DIABETES PREDICTION SYSTEM USING MACHINE LEARNING.

Introduction: In this phase we begin developing the diabetes prediction system by preparing the data and selecting relevant features. Loading and preprocessing data are crucial steps in building an AI-based diabetes prediction system. Proper data handling sets the foundation for developing an accurate and effective predictive model. This phase provides an overview of the significance and process of loading and preprocessing data for such a system.

Given Dataset:

Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	Diabetes Pedigree Function	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

Excel Dataset link:
https://in.docworkspace.com/d/sIETqyObiAYe6s6kG

769 rows *9 columns

Necessary steps to follow:

To create an AI-based diabetes prediction system, you'll need to load and work with a dataset that contains relevant information. Here are the key steps and considerations for loading a dataset:

1.Data Collection: Gather a dataset that includes historical information about individuals, particularly features that are relevant to diabetes prediction. Common features might include age, gender, body mass index (BMI), blood pressure, glucose levels, family history, and more.

2.Data Preprocessing: Clean the dataset to handle missing values, outliers, and inconsistencies. This may involve imputing missing values, scaling features, and encoding categorical variables.

3.Dataset Splitting: Split the dataset into training, validation, and test sets. This helps in training your AI model, tuning hyperparameters, and evaluating its performance.

4. **Feature Selection**: Identify the most relevant features for diabetes prediction. Feature selection techniques can help in reducing dimensionality and improving model efficiency.

5. **Data Normalization**: Normalize or standardize the data to ensure that different features are on a similar scale. This can improve the performance of many machine learning algorithms.

6. **Loading into Your AI System**: Depending on the programming language and libraries you're using; you can load the dataset into your AI system. Popular libraries for this purpose include NumPy, Pandas, and scikit-learn in Python.

- Steps to be continued after loading and preprocessing the dataset.

7. **Building the Model**: Train your AI model, which can be a machine learning model (e.g., logistic regression, decision tree, random forest) or a deep learning model (e.g., neural network). Your model will use the loaded dataset to learn patterns and make predictions.

8.Evaluation: Assess the performance of your model using appropriate metrics like accuracy, precision, recall, F1-score, or area under the ROC curve (AUC). Tweak your model and dataset as needed for better results.

9.Deployment: Once your AI-based diabetes prediction system performs well, you can deploy it in a real-world environment where it can make predictions for new data.

10.Monitoring and Maintenance: Continuously monitor the system's performance and update the dataset and model as new data becomes available.

Loading the Dataset:

Data Collection: Obtain a dataset that contains relevant information for diabetes prediction. You can find such datasets from sources like the National Institute of Diabetes and Digestive and Kidney Diseases (NIDDK) or the UCI Machine Learning Repository.

Choose a Programming Language and Libraries: Select a programming language (e.g., Python) and libraries (e.g., Pandas) to work with your dataset. Python is commonly used for data science and machine learning tasks.

Load the Dataset: Use library functions to load your dataset. For example, in Python, you can use Pandas to read data from CSV, Excel, or other file formats.

Data Preprocessing:

Handling Missing Values: Check for and handle missing values. You can either remove rows with missing values or impute them using techniques like mean, median, or regression imputation.

Dealing with Outliers: Identify and handle outliers in your dataset. You can use statistical methods or visualization techniques to detect outliers and then decide whether to remove or transform them.

Feature Engineering: Create new features or transform existing ones to make them more suitable for your model. For example, you can compute the body mass index (BMI) if it's not already in the dataset.

Data Scaling: Normalize or standardize numerical features to ensure they are on the same scale. This is important, especially if you plan to use algorithms sensitive to feature scaling, like support vector machines or k-nearest neighbors.

Encoding Categorical Variables: If your dataset contains categorical variables, encode them into numerical values. One-hot encoding is a common technique for this purpose.

Split the Data: Divide your dataset into training, validation, and test sets. This is important for model training, hyperparameter tuning, and evaluation.

- With the dataset loaded and preprocessed, you can proceed to build and train your diabetes prediction model using machine learning or deep learning techniques.

PROGRAM:

```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
```

IMPORTING SEABORN LIBRARY FOR STATISTICAL DATA VISUALIZATION

```
import seaborn as sns
```

IMPORTING MATPLOTLIB LIBRARY FOR CREATING PLOTS AND VISUALIZATIONS

```
import matplotlib.pyplot as plt
```

IMPORTING PLOTLY EXPRESS LIBRARY FOR INTERACTIVE VISUALIZATIONS

```
import plotly.express as px
```

Exploratory Data Analysis (EDA): It is a critical step in understanding and gaining insights from your dataset in an AI-based diabetes prediction system.

LOAD AND PREPARE DATA

```
df=pd.read_excel('https://in.docworkspace.com/d/sIETqyObiAYe6s6kG ')
```

UNDERSTANDING THE VARIABLES

INPUT

```
df.head(10)
```

OUTPUT

S:NO	pregnancies	Glucose	Blood pressure	Skin thickness	insulin	BMI	Diabetes pedigree function	Age	Out come
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
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

INPUT

```
df.tail(10)
```

OUTPUT

S:NO	pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	Diabetes pedigree function	Age	Out come
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	2	88	58	26	16	28.4	0.766	22	0
761	9	170	74	31	0	44.0	0.403	43	1
762	9	89	62	0	0	22.5	0.142	33	0
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

INPUT

```
df.sample(5)
```

OUTPUT

S:NO	Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	Diabetes pedigree function	Age	**Out come**
345	8	126	88	36	108	38.5	0.349	49	0
578	10	133	68	0	0	27.0	0.245	36	0
84	5	137	108	0	0	48.8	0.227	37	1
217	6	125	68	30	120	30.0	0.464	32	0
595	0	188	82	14	185	32.0	0.682	22	1

INPUT

```
df.describe()
```

OUTPUT

S:NO	Pregnancies	Glucose	Blood pressure	Skin Thickness	Insulin	BMI	Diabetes Pedigree Function	Age	**Out come**
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

INPUT

```
df.size
```

OUTPUT

```
6912
```

INPUT

```
df.shape
```

OUTPUT

```
(768, 9)
```

Data cleaning: It can be a complex and domain-specific process. The specific steps and techniques you use may vary depending on your dataset and the nature of the data quality issues.

INPUT

```
df=df.drop_duplicates()
```

```
Df.shape
```

OUTPUT

```
(768, 9)
```

CHECK FOR NULL VALUES

INPUT

```
df.isnull().sum()
```

OUTPUT

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

THERE IS NO MISSING VALUES PRESENT IN THE DATA

INPUT

```
df.columns
```

OUTPUT

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

Check the number of Zero Values in Dataset

INPUT

```
print("No. of Zero Values in Glucose ", df[df['Glucose']==0].shape[0])
```

OUTPUT

```
No. of Zero Values in Glucose  5
```

INPUT

```
print("No. of Zero Values in Blood Pressure ",
df[df['BloodPressure']==0].shape[0])
```

OUTPUT

```
No. of Zero Values in Blood Pressure  35
```

INPUT

```
print("No. of Zero Values in SkinThickness ",
df[df['SkinThickness']==0].shape[0])
```

OUTPUT

```
No. of Zero Values in SkinThickness  227
```

INPUT

```
print("No. of Zero Values in Insulin ", df[df['Insulin']==0].shape[0])
```

OUTPUT

```
No. of Zero Values in Insulin  374
```

INPUT

```
print("No. of Zero Values in BMI ", df[df['BMI']==0].shape[0])
```

OUTPUT

```
No. of Zero Values in BMI  11
```

Replace zeroes with mean of that Columns

INPUT

```
df['Glucose']=df['Glucose'].replace(0, df['Glucose'].mean())
print('No of zero Values in Glucose ', df[df['Glucose']==0].shape[0])
```

```
No of zero Values in Glucose  0
```

```
df['BloodPressure']=df['BloodPressure'].replace(0,
df['BloodPressure'].mean())
df['SkinThickness']=df['SkinThickness'].replace(0,
df['SkinThickness'].mean())
```

```
df['Insulin']=df['Insulin'].replace(0, df['Insulin'].mean())
df['BMI']=df['BMI'].replace(0, df['BMI'].mean())
```

Validate the Zero Values:

```
df.describe()
```

OUTPUT

	Pregnancies	glucose	Blood Pressure	Skin Thickness	Insulin	Insulin	Diabetes Pedigree Function	Age	Out come
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.681605	72.254807	26.606479	118.660163	32.450805	0.471876	33.240885	0.348958
std	3.369578	30.436016	12.115932	9.631241	93.080358	6.875374	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	79.799479	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

Data visualization: Data visualization helps you understand the data distribution, relationships between features, and can be particularly useful for feature selection and feature engineering. You can use libraries like Matplotlib, Seaborn, or Plotly in Python to create these visualizations.

Bar Charts: Use bar charts to display the distribution of categorical features such as gender, family history, and medication usage.

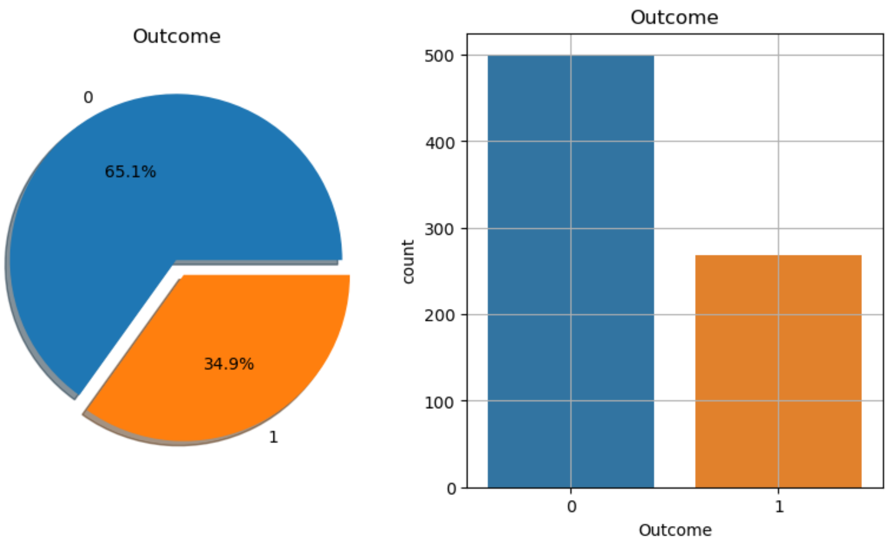
INPUT

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
f, ax = plt.subplots(1, 2, figsize=(10, 5))
df['Outcome'].value_counts().plot.pie(explode=[0, 0.1], autopct='%1.1f%%',
ax=ax[0], shadow=True)
ax[0].set_title('Outcome')
ax[0].set_ylabel(' ')
```

```python
sns.countplot(x='Outcome', data=df, ax=ax[1])  # Use 'x' instead of
'Outcome'
ax[1].set_title('Outcome')
N, P = df['Outcome'].value_counts()
print('Negative (0):', N)
print('Positive (1):', P)
plt.grid()
plt.show()
```

OUTPUT

```
Negative (0): 500
Positive (1): 268
```




```
1 Represent --> Diabetes Positive
0 Represent --> Diabetes Negative
```

Histograms: Visualize the distribution of numerical features like age, glucose levels, BMI, etc. Histograms help you understand the data's central tendencies and spread.

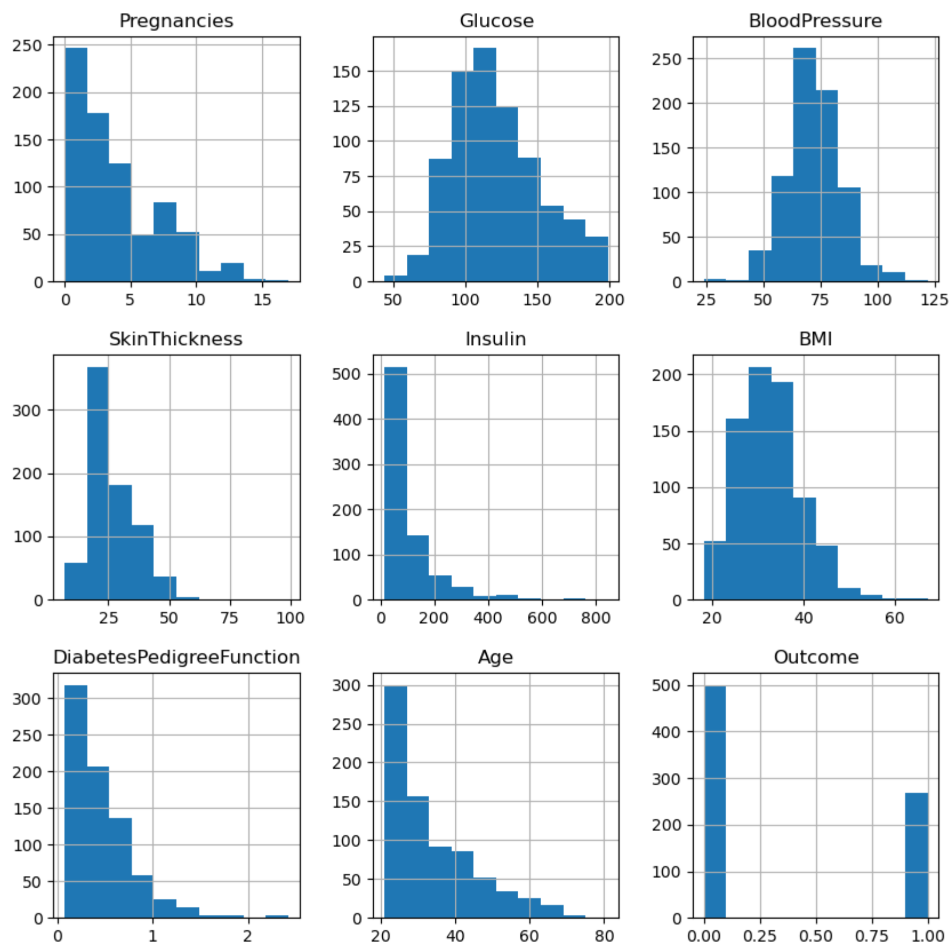
Pie Charts: Use pie charts to visualize the distribution of categorical variables, such as the percentage of people with and without diabetes.

Line Charts: If your data has a temporal aspect, create line charts to observe trends over time. For example, tracking glucose levels over weeks

INPUT

```
df.hist(bins=10, figsize=(10, 10))
plt.show()
```

OUTPUT

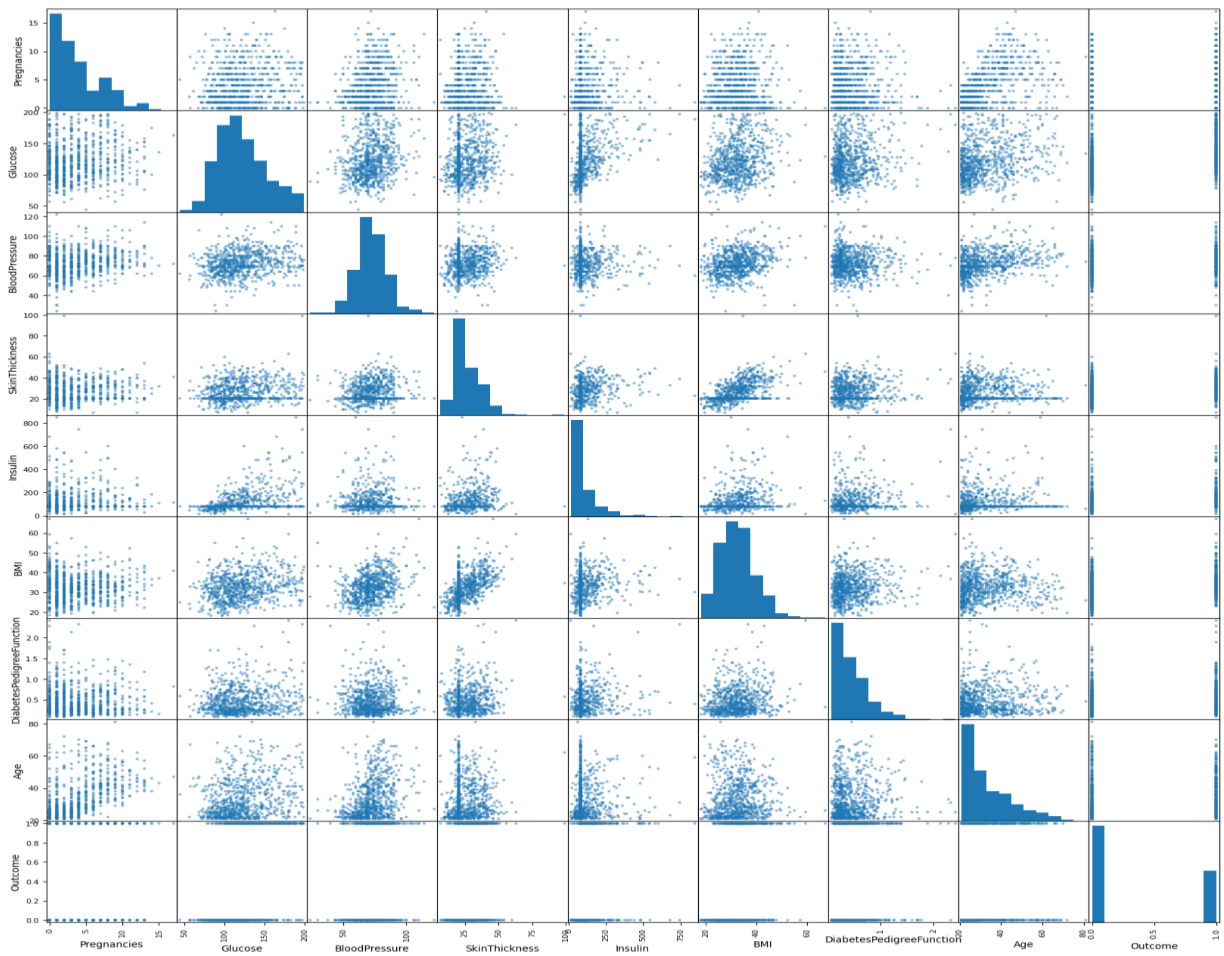


Scatter Plots: Plot relationships between two numerical features to identify correlations and patterns. For instance, you can plot glucose levels against BMI.

INPUT

```
from pandas.plotting import scatter_matrix
scatter_matrix(df, figsize =(20, 20))
```

OUTPUT



Pair Plots: Plot pairs of features against each other, especially when you have multiple numerical features. Pair plots help in visualizing relationships within the dataset.

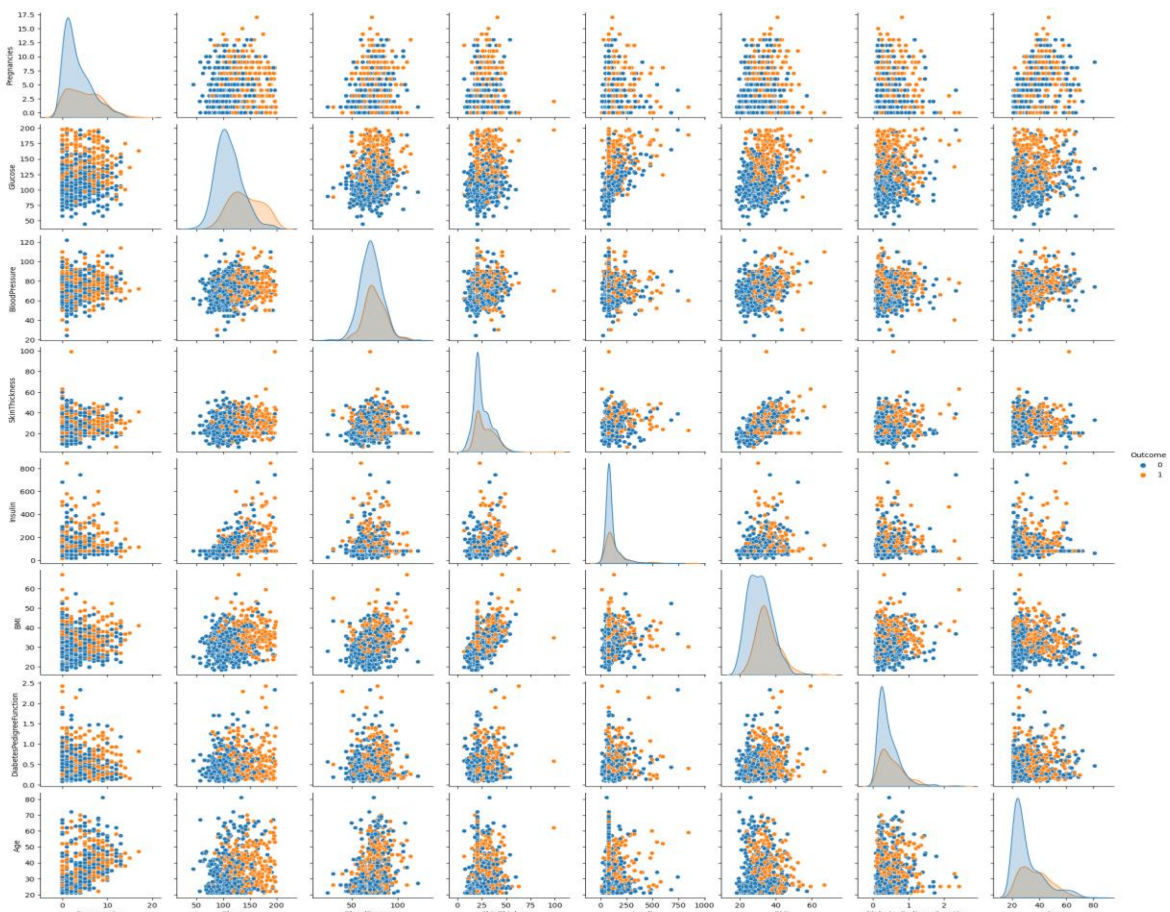
Pair Plots: Plot pairs of features against each other, especially when you have multiple numerical features. Pair plots help in visualizing relationships within the dataset.

Violin Plots: Violin plots combine a box plot and a kernel density plot to visualize the distribution of numerical features.

INPUT

```
sns.pairplot(data=df, hue='Outcome')
plt.show()
```

OUTPUT:

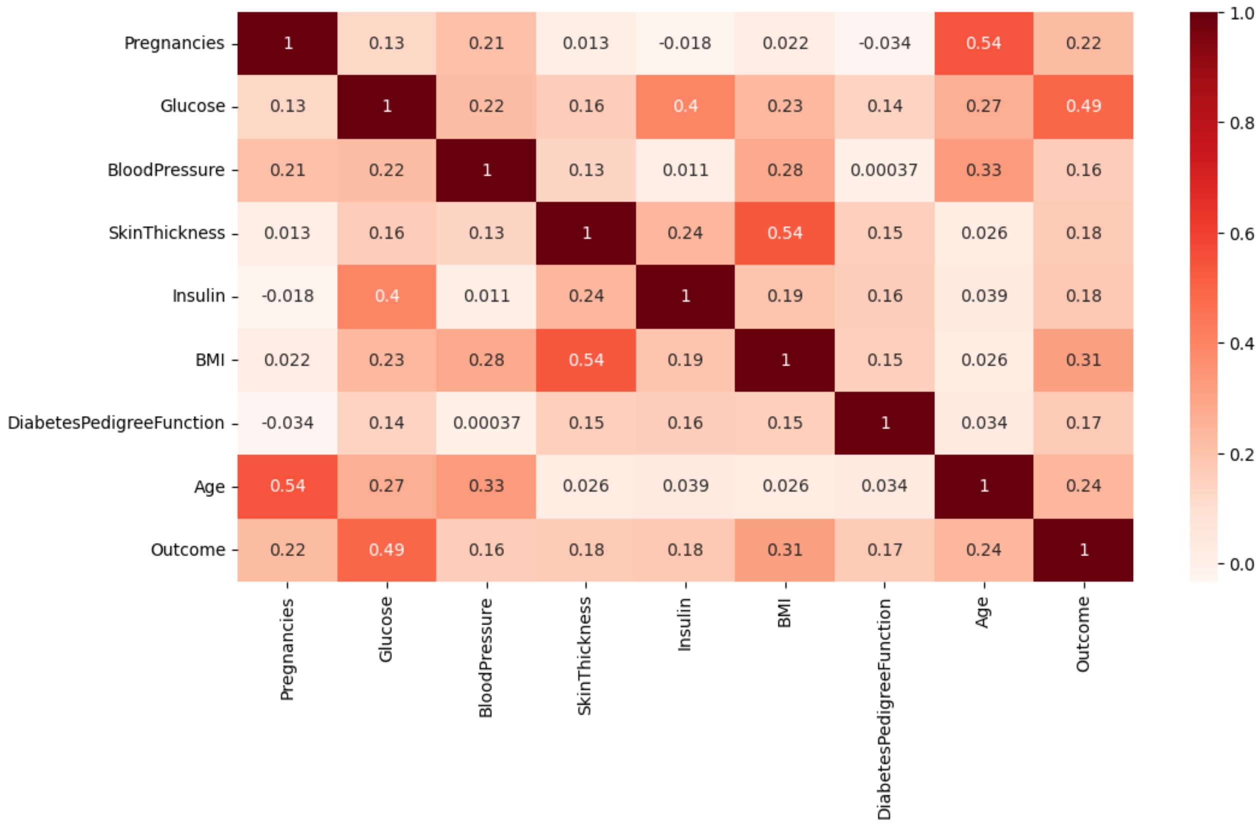


Correlation Heatmaps: Create a heatmap to show the correlation between numerical features. This helps you understand how features relate to each other.

INPUT

```
plt.figure(figsize=(12, 6))
sns.heatmap(df.corr(), annot=True, cmap='Reds')
plt.plot()
```

OUTPUT



INPUT

```
mean = df['Outcome'].mean()
```

OUTPUT

```
0.3489583333333333
```

SPLIT THE DATA FRAME INTO X AND Y

INPUT

```
target_name='Outcome'
y=df[target_name]
X= df.drop(target_name, axis=1)

X.head()
```

OUTPUT

S:NO	Pregnancies	glucose	Blood pressure	Skin thickness	Insulin	BMI	Diabetes Pedigree function	Age
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33

INPUT

```
y.head()
```

OUTPUT

```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

FUTURE SCALING: Scaling an AI-based diabetes prediction system involves a multidisciplinary approach that combines expertise in healthcare, AI, data science, and technology. Continuous learning, adaptation, and collaboration are key elements in the successful scaling of such system

INPUT

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
scaler.fit(X)
SSX = scaler.transform(X)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(SSX, y, test_size=0.2,
random_state=7)
```

```
X_train.shape, y_train.shape
```

OUTPUT

```
((614, 8), (614,))
```

INPUT

```
X_test.shape, y_test.shape
```

OUTPUT

```
((154, 8), (154,))
```

- The following are the steps to be followed after completing the loading and preprocessing of dataset.

Model Building: Choose an appropriate machine learning algorithm for your diabetes prediction task. Common choices include logistic regression, decision trees, random forests, or support vector machines.

Model Training: Train the chosen model on the training dataset using appropriate algorithms and hyperparameters. You may need to perform hyperparameter tuning to optimize model performance.

Model Evaluation: Evaluate the model's performance on the testing dataset using metrics like accuracy, precision, recall, F1-score, and ROC AUC.

Fine-Tuning: Based on the evaluation results, fine-tune the model or try different algorithms to improve predictive performance.

Deployment: Once satisfied with the model's performance, you can deploy it in a real-world setting for diabetes prediction.

Monitoring and Maintenance: Continuously monitor the model's performance and retrain it periodically to ensure it remains accurate and up to date.

Conclusion:

- Understanding the data’s structure, characteristics and any potential issues through exploratory data analysis (EDA) is essential for informed decision-making.
- Data preprocessing emerged as a pivotal aspect of this process. It involves cleaning, transforming, and refining the dataset to ensure that it aligns with the requirements of machine learning algorithms.
- With these foundational steps completed, our dataset is now primed for the subsequent stages of building and training a house price prediction model.