

PROJECT TITLE:

BUILDING A SMARTER AI BASED DIABETES PREDICTION SYSTEM

Phase 5-Project Documentation & Submission

CLEAR OUTLINE :

The AI-Powered Diabetes Prediction System is a pioneering healthcare initiative that leverages the capabilities of artificial intelligence (AI) and machine learning to transform the landscape of diabetes prevention and management. The project encompasses a well-structured approach, beginning with project planning, clearly defining objectives, and ensuring compliance with healthcare regulations and data privacy laws, such as HIPAA.

DATA PROCESSING:

Data collection, the first technical phase, involves gathering a rich and diverse dataset from sources like electronic health records, surveys, wearable devices, and genetic databases. This extensive data is then securely stored, often leveraging cloud-based solutions for scalability and accessibility.

Subsequent data preprocessing ensures data quality and uniformity. Duplicate records are removed, missing values are

addressed, and data is standardized, making it ready for model training. Feature engineering extracts relevant variables such as glucose levels and family history, vital for precise diabetes risk prediction.

MACHINE LEARNING ALGORITHM:

Model development, the core of the project, involves selecting suitable machine learning algorithms, like logistic regression or neural networks, and training them on the prepared data. Continuous learning and updates facilitate the incorporation of new data and medical insights.

FEATURE SELECTION TECHNIQUES:

Privacy and security measures safeguard sensitive patient information, and comprehensive documentation supports transparency and reproducibility. The system's deployment, often integrated with healthcare providers, enables user-friendly interfaces for data input and personalized recommendations.

STEPS:

Step 1: PROJECT PLANNING AND DEFINITION

- **Project Scope:** Clearly define the scope and objectives of the project, including the target population (e.g., adults, children), geographic region, and any specific data sources or technologies to be used.
- **Data Sources:** Identify the sources of medical data, such as electronic health records (EHRs), surveys, wearable devices, and genetic databases.
- **Regulatory Compliance:** Ensure compliance with healthcare regulations and data privacy laws (e.g., HIPAA in the United States).

Step 2: DATA COLLECTION AND INTEGRATION

- **Data Gathering:** Collect a diverse dataset including patient demographics, medical history, lifestyle factors, and genetic information. Collaborate with healthcare institutions and obtain necessary permissions.
- **Data Integration:** Integrate data from various sources into a unified database or data warehouse.

Step 3: DATA PREPROCESSING

- **Data Cleaning:** Remove duplicates, handle missing values, and address outliers in the dataset.
- **Data Transformation:** Normalize and standardize data to ensure uniformity and improve model performance.

Step 4: EXPLORATORY DATA ANALYSIS (EDA)

- **Feature Selection:** Use EDA to identify relevant features or variables that may influence diabetes risk. This can help streamline the modeling process by focusing on the most important factors.

- **Data Visualization:** Create data visualizations such as histograms, scatter plots, and box plots to understand data distributions and uncover patterns or trends.
- **Summary Statistics:** Calculate and examine summary statistics like mean, median, standard deviation, and quantiles to gain insights into data characteristics.
- **Correlation Analysis:** Perform correlation analysis to understand relationships between features, helping identify potential multicollinearity and influential variables.

Step 5: DATA SPLITTING

- **Training and Validation Sets:** Split the preprocessed dataset into training and validation sets. The training set is used to train the predictive model, while the validation set is used to assess model performance during development.
- **Test Set:** Reserve a separate test set, which will be used to evaluate the final model's performance. It should not be used for model development.
- **Cross-Validation (Optional):** Consider using cross-validation techniques like k-fold cross-validation to assess model generalization and robustness. This involves repeatedly splitting the data into training and validation subsets to produce multiple model evaluation.

Step 6: FEATURE ENGINEERING

- **Feature Selection:** Identify relevant features that are likely to influence diabetes risk. This might involve domain expertise and data analysis.
- **Feature Creation:** Create new features if necessary, such as calculated BMI (Body Mass Index) or genetic risk scores.

Step 7: MACHINE LEARNING MODEL DEVELOPMENT

- **Model Selection:** Choose appropriate machine learning algorithms for diabetes prediction. Common choices include logistic regression, decision trees, random forests, and neural networks.
- **Training:** Split the data into training and validation sets. Train the selected models using the training data.

Step 8: MODEL EVALUATION

- **Performance Metrics:** Evaluate model performance using metrics like accuracy, precision, recall, F1-score, and ROC-AUC. Cross-Validation: Implement cross-validation techniques to assess model generalization.
- **Interpretability:** Use model interpretability tools to understand why the model makes specific predictions.

Step 9: PREDICTION AND RISK ASSESSMENT

- **User Interface:** Develop a user-friendly interface where users can input their data.
- **Prediction:** Use the trained model to predict diabetes risk based on user input.
- **Risk Assessment:** Provide users with a risk assessment score and an explanation of how the model arrived at that prediction.

Step 10: PERSONALIZED RECOMMENDATIONS

- **Recommendation Engine:** Create a recommendation engine that offers personalized preventive measures based on the risk assessment.
- **Healthcare Integration:** Integrate the system with healthcare providers to facilitate referrals and follow-up care.

Step 11: CONTINUOUS LEARNING AND UPDATES

- **Data Updates:** Continuously update the model with new data to improve accuracy and adapt to changing healthcare trends.
- **Algorithm Improvements:** Stay informed about the latest research and advancements in diabetes prediction and prevention.

Step 12: PRIVACY AND SECURITY

- **Data Security:** Implement robust security measures to protect patient data, including encryption and access controls.
- **Compliance:** Regularly audit and ensure compliance with data privacy regulations.

Step 13: DEPLOYMENT AND MONITORING

- **Deployment:** Deploy the system in healthcare institutions or make it available to the public, depending on the project's goals.
- **Monitoring:** Continuously monitor system performance, user feedback, and any potential ethical concerns.

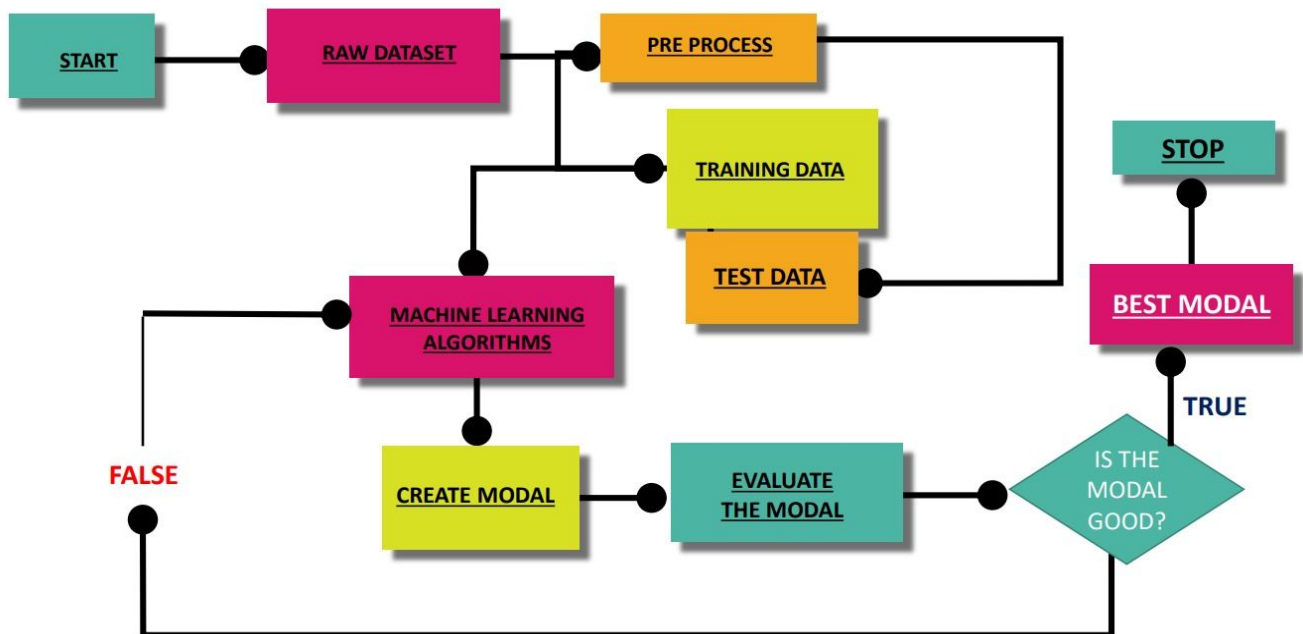
Step 14: DOCUMENTATION AND TRAINING

- **Documentation:** Document the system's architecture, data sources, and algorithms for reference.
- **User Training:** Provide training to healthcare professionals and users on how to use the system effectively.

Step 15: MAINTENANCE AND SUPPORT SYSTEM

- **Maintenance:** Regularly update the system to address bugs, security vulnerabilities, and user feedback.
- **User Support:** Offer user support channels for inquiries, issues, and updates

BLOCK DIAGRAM



DATA SET

Features	Minimum	Maximum	Average
Pregnancies	0	8	1.61
Glucose (mg/dL)	52.2	274	109.39
Blood pressure (mm Hg)	5.9	115	71.09
Skin thickness (mm)	2.9	23.3	10.78
BMI (kg/m ²)	2.61	41.62	22.69
Age (years)	17	77	27.02

Irritability and Mood changes

Changes in mood and irritability can be related to fluctuating blood sugar levels..

Glucose
Check the latest glucose rate of the patient

Weight loss
Losing weight without making any significant changes to diet or exercise

Frequent Infections

Such as skin infections or urinary tract infections

Blood Pressure

Check the BP of the patients



Diabetes-Prediction-using-ml

1 Diabetes Prediction:

The dataset comprises crucial health-related features such as 'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', and 'Age'. The main objective was to predict the 'Outcome' label, which signifies the likelihood of diabetes.

1.1 About the Data:

Data Overview: This is a [diabetes.csv](#) data

1.2 Import Required Libraries:

```
[1]: # Ignore warning messages to prevent them from being displayed during code execution
import warnings
warnings.filterwarnings('ignore')
```

```
[2]: import numpy as np      # Importing the NumPy library for linear algebra operations
import pandas as pd        # Importing the Pandas library for data processing and CSV file handling
```

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

/kaggle/input/diabetes-data-set/diabetes.csv

```
[4]: import seaborn as sns      # Importing the Seaborn library for statistical data visualization
import matplotlib.pyplot as plt # Importing the Matplotlib library for creating plots and visualizations
import plotly.express as px     # Importing the Plotly Express library for interactive visualizations
```

1.3 Exploratory Data Analysis:

1.3.1 Load and Prepare Data:

```
[5]: df=pd.read_csv('/kaggle/input/diabetes-data-set/diabetes.csv')
```

1.3.2 Understanding the Variables

```
[6]: df.head(10)
```

```
[6]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
5	5	116	74	0	0	25.6	
6	3	78	50	32	88	31.0	
7	10	115	0	0	0	35.3	
8	2	197	70	45	543	30.5	
9	8	125	96	0	0	0.0	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1
5	0.201	30	0
6	0.248	26	1
7	0.134	29	0
8	0.158	53	1
9	0.232	54	1

```
[7]: df.tail(10)
```

```
[7]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
758	1	106	76	0	0	37.5	
759	6	190	92	0	0	35.5	
760	2	88	58	26	16	28.4	
761	9	170	74	31	0	44.0	
762	9	89	62	0	0	22.5	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

	DiabetesPedigreeFunction	Age	Outcome
758	0.197	26	0
759	0.278	66	1
760	0.766	22	0
761	0.403	43	1
762	0.142	33	0
763	0.171	63	0
764	0.340	27	0
765	0.245	30	0
766	0.349	47	1
767	0.315	23	0

```
[8]: df.sample(5)
```

```
[8]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI \
760	2	88	58	26	16	28.4
687	1	107	50	19	0	28.3
355	9	165	88	0	0	30.4
187	1	128	98	41	58	32.0
235	4	171	72	0	0	43.6

	DiabetesPedigreeFunction	Age	Outcome
760	0.766	22	0
687	0.181	29	0
355	0.302	49	1
187	1.321	33	1
235	0.479	26	1

```
[9]: df.describe()
```

```
[9]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin \
count	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479
std	3.369578	31.972618	19.355807	15.952218	115.244002
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000
75%	6.000000	140.250000	80.000000	32.000000	127.250000
max	17.000000	199.000000	122.000000	99.000000	846.000000

	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000
mean	31.992578	0.471876	33.240885	0.348958
std	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.078000	21.000000	0.000000
25%	27.300000	0.243750	24.000000	0.000000
50%	32.000000	0.372500	29.000000	0.000000

75%	36.600000	0.626250	41.000000	1.000000
max	67.100000	2.420000	81.000000	1.000000

```
[10]: df.dtypes
```

```
[10]: Pregnancies      int64
      Glucose          int64
      BloodPressure    int64
      SkinThickness    int64
      Insulin          int64
      BMI              float64
      DiabetesPedigreeFunction float64
      Age              int64
      Outcome          int64
      dtype: object
```

```
[11]: 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
```

```
[12]: df.size
```

```
[12]: 6912
```

```
[13]: df.shape
```

```
[13]: (768, 9)
```

1.3.3 Data Cleaning:

```
[14]: df.shape
```

```
[14]: (768, 9)
```

```
[15]: df=df.drop_duplicates()
```

```
[16]: df.shape
```

```
[16]: (768, 9)
```

Check null Values

```
[17]: df.isnull().sum()
```

```
[17]: 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

```
[18]: df.columns
```

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

Check the number of Zero Values in Dataset

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

No. of Zero Values in Glucose 5

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

No. of Zero Values in Blood Pressure 35

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

No. of Zero Values in SkinThickness 227

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

No. of Zero Values in Insulin 374

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

No. of Zero Values in BMI 11

Replace zeroes with mean of that Columns

```
[24]: 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

```
[25]: 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:

```
[26]: df.describe()
```

```
[26]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin \
count	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.681605	72.254807	26.606479	118.660163
std	3.369578	30.436016	12.115932	9.631241	93.080358
min	0.000000	44.000000	24.000000	7.000000	14.000000
25%	1.000000	99.750000	64.000000	20.536458	79.799479
50%	3.000000	117.000000	72.000000	23.000000	79.799479
75%	6.000000	140.250000	80.000000	32.000000	127.250000
max	17.000000	199.000000	122.000000	99.000000	846.000000

	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000
mean	32.450805	0.471876	33.240885	0.348958
std	6.875374	0.331329	11.760232	0.476951
min	18.200000	0.078000	21.000000	0.000000
25%	27.500000	0.243750	24.000000	0.000000
50%	32.000000	0.372500	29.000000	0.000000
75%	36.600000	0.626250	41.000000	1.000000
max	67.100000	2.420000	81.000000	1.000000

1.4 Data Visualization:

```
[27]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your DataFrame containing the dataset
# If you haven't imported your dataset yet, import it here

# Create subplots
f, ax = plt.subplots(1, 2, figsize=(10, 5))

# Pie chart for Outcome distribution
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(' ')

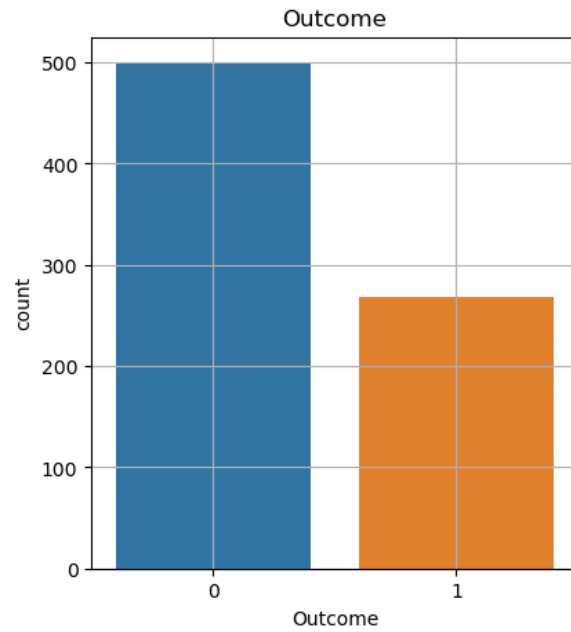
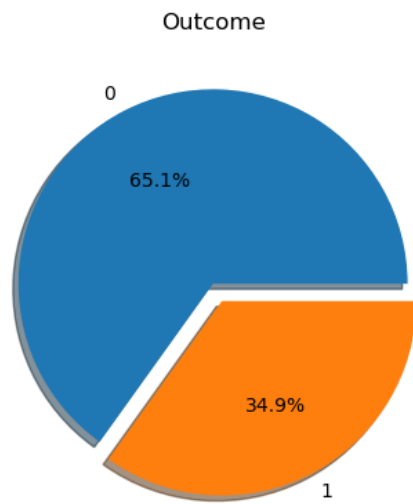
# Count plot for Outcome distribution
sns.countplot(x='Outcome', data=df, ax=ax[1]) # Use 'x' instead of 'Outcome'
ax[1].set_title('Outcome')

# Display class distribution
N, P = df['Outcome'].value_counts()
print('Negative (0):', N)
print('Positive (1):', P)

# Adding grid and showing plots
plt.grid()
plt.show()
```

Negative (0): 500

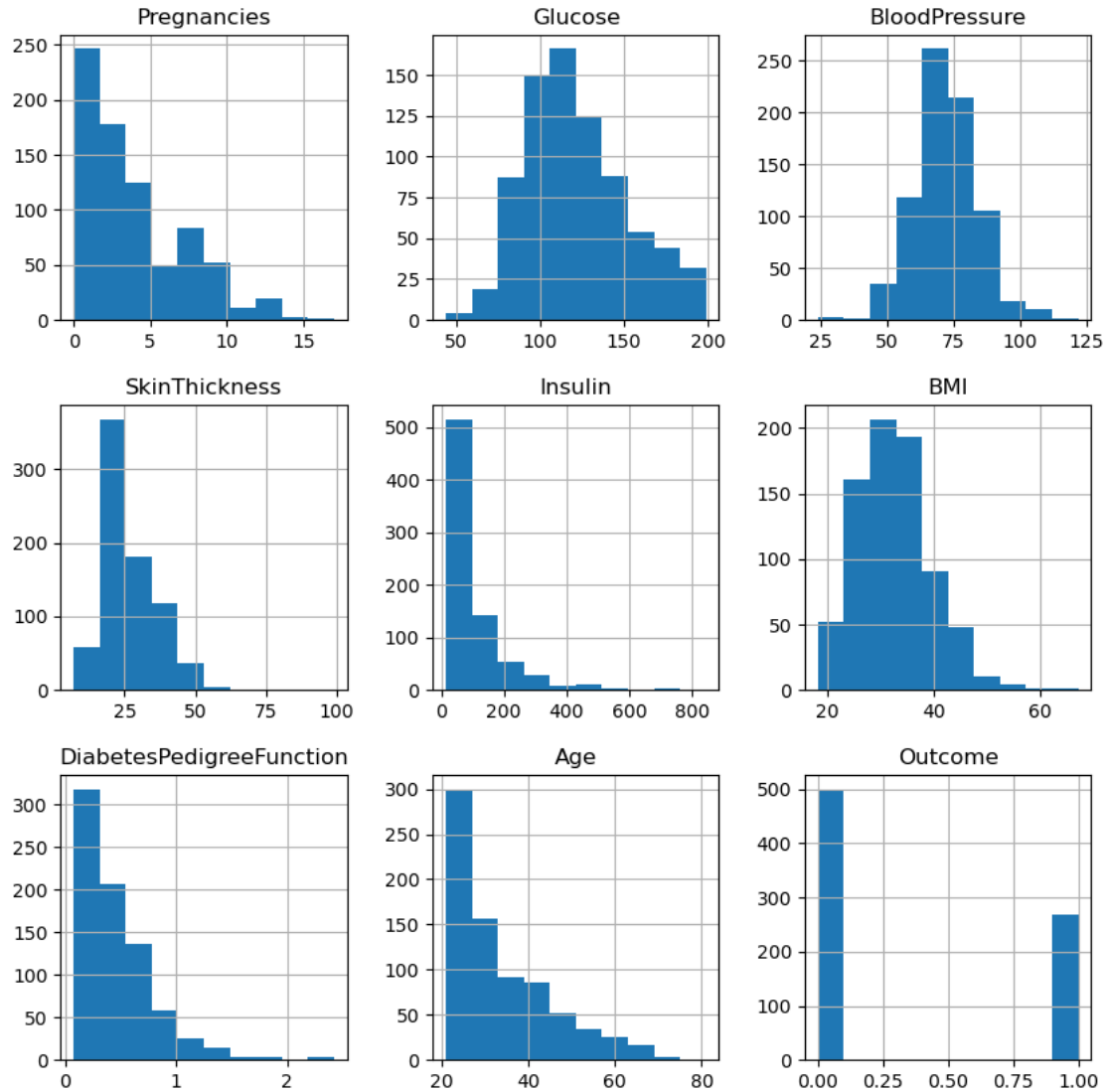
Positive (1): 268



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

1.4.1 Histograms:

```
[28]: df.hist(bins=10, figsize=(10, 10))  
plt.show()
```

1.4.2 Scatter Plot:

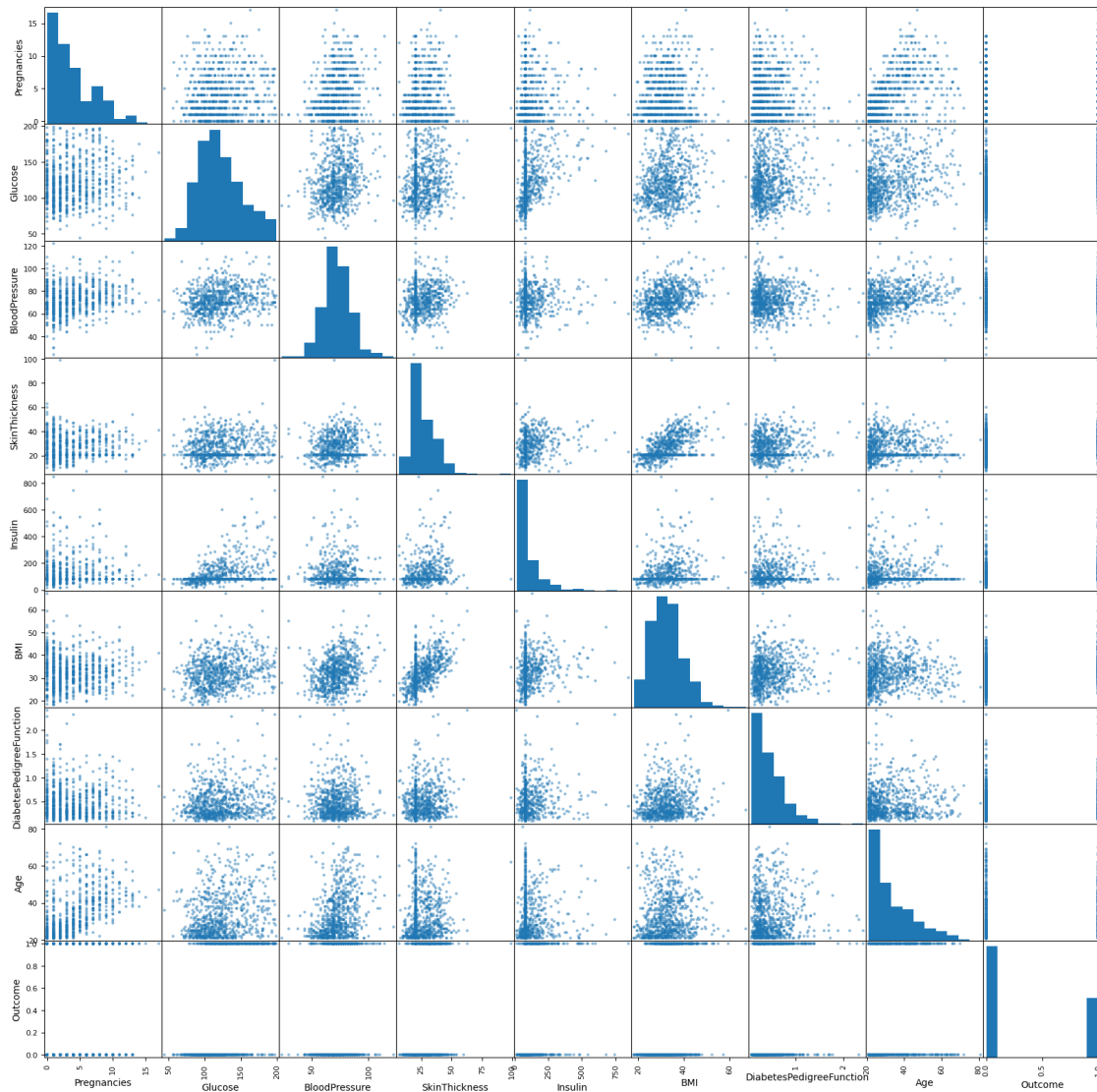
```
[29]: from pandas.plotting import scatter_matrix
      scatter_matrix(df, figsize=(20, 20))
```

```
[29]: array([[<Axes: xlabel='Pregnancies', ylabel='Pregnancies'>,
              <Axes: xlabel='Glucose', ylabel='Pregnancies'>,
              <Axes: xlabel='BloodPressure', ylabel='Pregnancies'>,
              <Axes: xlabel='SkinThickness', ylabel='Pregnancies'>,
              <Axes: xlabel='Insulin', ylabel='Pregnancies'>,
              <Axes: xlabel='BMI', ylabel='Pregnancies'>,
              <Axes: xlabel='DiabetesPedigreeFunction', ylabel='Pregnancies'>,
              <Axes: xlabel='Age', ylabel='Pregnancies'>,
              <Axes: xlabel='Outcome', ylabel='Outcome'>])
```

```

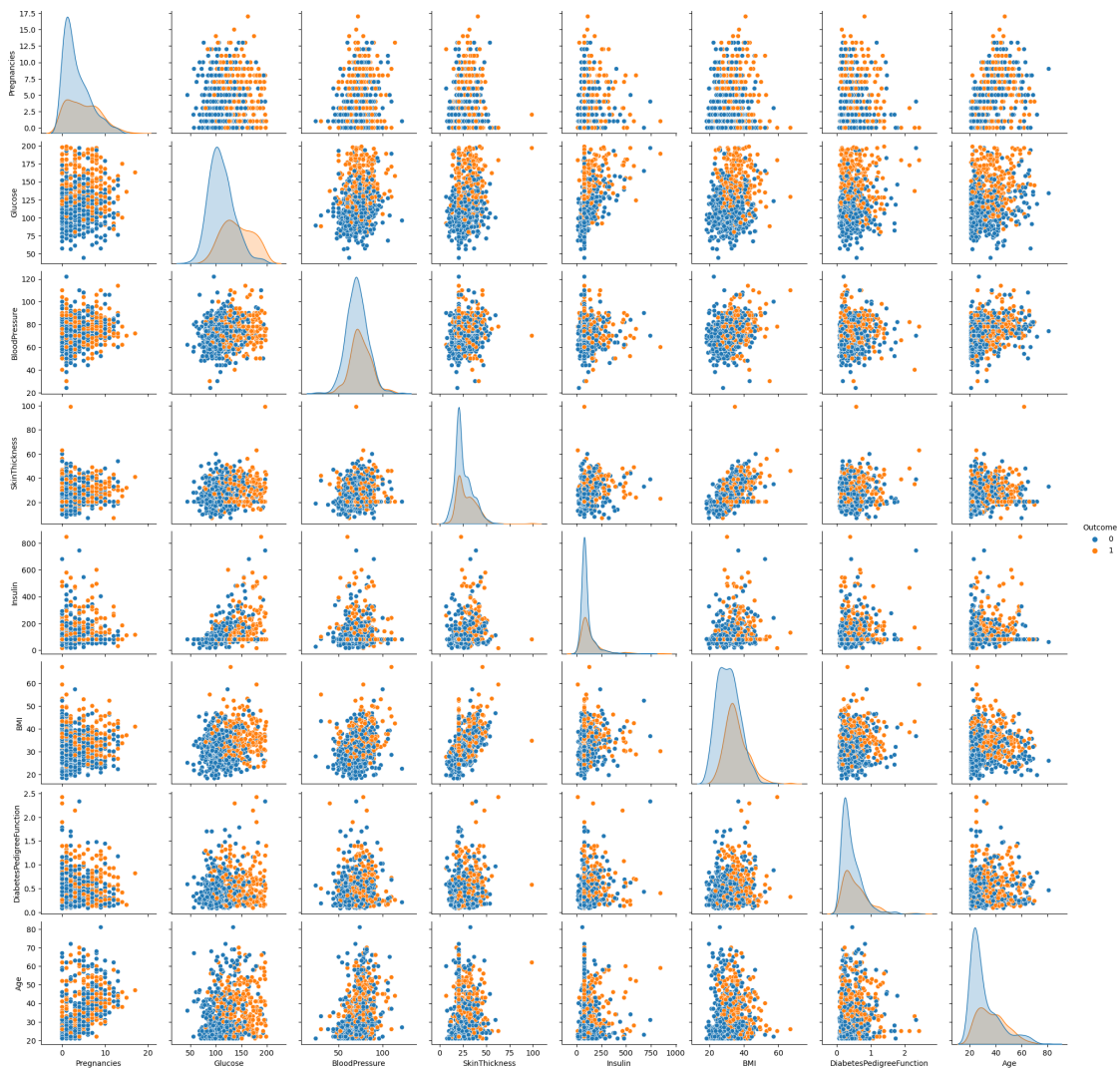
    <Axes: xlabel='Outcome', ylabel='Pregnancies'>],
[<Axes: xlabel='Pregnancies', ylabel='Glucose'>,
 <Axes: xlabel='Glucose', ylabel='Glucose'>,
 <Axes: xlabel='BloodPressure', ylabel='Glucose'>,
 <Axes: xlabel='SkinThickness', ylabel='Glucose'>,
 <Axes: xlabel='Insulin', ylabel='Glucose'>,
 <Axes: xlabel='BMI', ylabel='Glucose'>,
 <Axes: xlabel='DiabetesPedigreeFunction', ylabel='Glucose'>,
 <Axes: xlabel='Age', ylabel='Glucose'>,
 <Axes: xlabel='Outcome', ylabel='Glucose'>],
[<Axes: xlabel='Pregnancies', ylabel='BloodPressure'>,
 <Axes: xlabel='Glucose', ylabel='BloodPressure'>,
 <Axes: xlabel='BloodPressure', ylabel='BloodPressure'>,
 <Axes: xlabel='SkinThickness', ylabel='BloodPressure'>,
 <Axes: xlabel='Insulin', ylabel='BloodPressure'>,
 <Axes: xlabel='BMI', ylabel='BloodPressure'>,
 <Axes: xlabel='DiabetesPedigreeFunction', ylabel='BloodPressure'>,
 <Axes: xlabel='Age', ylabel='BloodPressure'>,
 <Axes: xlabel='Outcome', ylabel='BloodPressure'>],
[<Axes: xlabel='Pregnancies', ylabel='SkinThickness'>,
 <Axes: xlabel='Glucose', ylabel='SkinThickness'>,
 <Axes: xlabel='BloodPressure', ylabel='SkinThickness'>,
 <Axes: xlabel='SkinThickness', ylabel='SkinThickness'>,
 <Axes: xlabel='Insulin', ylabel='SkinThickness'>,
 <Axes: xlabel='BMI', ylabel='SkinThickness'>,
 <Axes: xlabel='DiabetesPedigreeFunction', ylabel='SkinThickness'>,
 <Axes: xlabel='Age', ylabel='SkinThickness'>,
 <Axes: xlabel='Outcome', ylabel='SkinThickness'>],
[<Axes: xlabel='Pregnancies', ylabel='Insulin'>,
 <Axes: xlabel='Glucose', ylabel='Insulin'>,
 <Axes: xlabel='BloodPressure', ylabel='Insulin'>,
 <Axes: xlabel='SkinThickness', ylabel='Insulin'>,
 <Axes: xlabel='Insulin', ylabel='Insulin'>,
 <Axes: xlabel='BMI', ylabel='Insulin'>,
 <Axes: xlabel='DiabetesPedigreeFunction', ylabel='Insulin'>,
 <Axes: xlabel='Age', ylabel='Insulin'>,
 <Axes: xlabel='Outcome', ylabel='Insulin'>],
[<Axes: xlabel='Pregnancies', ylabel='BMI'>,
 <Axes: xlabel='Glucose', ylabel='BMI'>,
 <Axes: xlabel='BloodPressure', ylabel='BMI'>,
 <Axes: xlabel='SkinThickness', ylabel='BMI'>,
 <Axes: xlabel='Insulin', ylabel='BMI'>,
 <Axes: xlabel='BMI', ylabel='BMI'>,
 <Axes: xlabel='DiabetesPedigreeFunction', ylabel='BMI'>,
 <Axes: xlabel='Age', ylabel='BMI'>,
 <Axes: xlabel='Outcome', ylabel='BMI'>],
[<Axes: xlabel='Pregnancies', ylabel='DiabetesPedigreeFunction'>,

```



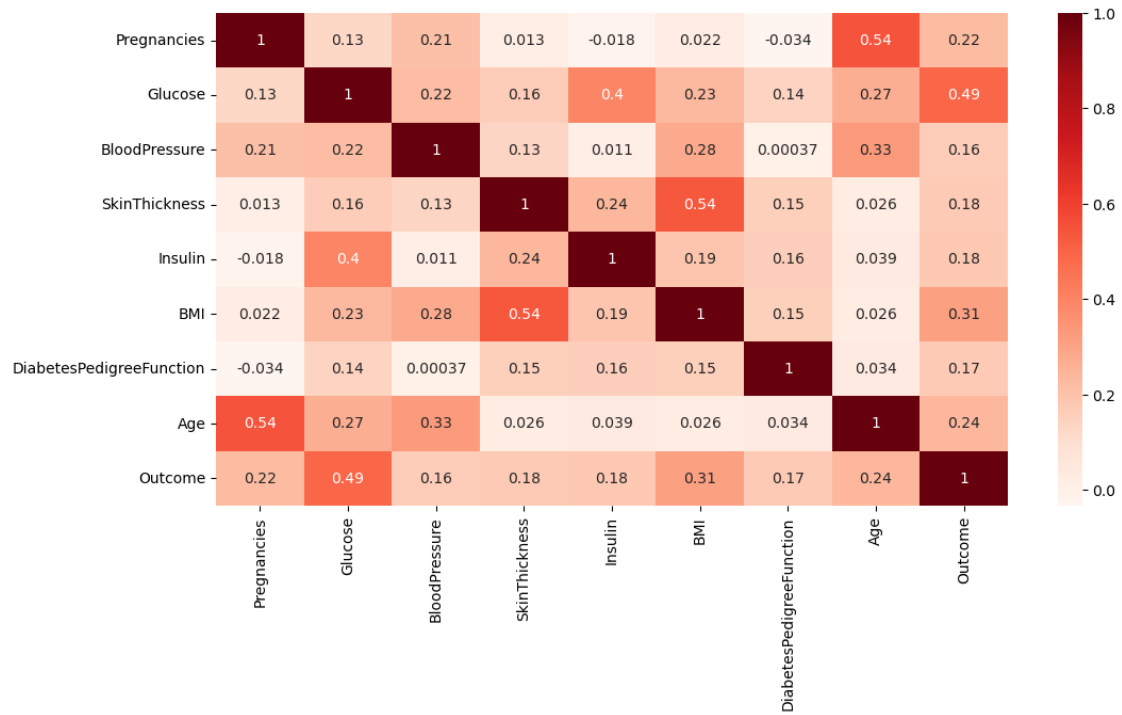
1.4.3 Pair plot:

```
[30]: sns.pairplot(data=df, hue='Outcome')
plt.show()
```



```
[31]: plt.figure(figsize=(12, 6))
sns.heatmap(df.corr(), annot=True, cmap='Reds')
plt.plot()
# Creating a heatmap of the correlation matrix for the columns in the DataFrame ↵
↵data
```

[31]: []



```
[32]: mean = df['Outcome'].mean()
      # Calculating the mean value of the 'Outcome' column in the DataFrame data
      mean
      # Displaying the calculated mean value
```

```
[32]: 0.3489583333333333
```

1.5 Split the DataFrame into X and y

```
[33]: target_name='Outcome'

      y=df[target_name]

      X= df.drop(target_name, axis=1)
```

```
[34]: X.head()
```

```
[34]:   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI  \
0           6    148.0         72.0      35.000000    79.799479  33.6
1           1     85.0         66.0      29.000000    79.799479  26.6
2           8    183.0         64.0      20.536458    79.799479  23.3
3           1     89.0         66.0      23.000000    94.000000  28.1
4           0    137.0         40.0      35.000000   168.000000  43.1
```

	DiabetesPedigreeFunction	Age
0	0.627	50
1	0.351	31
2	0.672	32
3	0.167	21
4	2.288	33

```
[35]: y.head()
```

```
[35]: 0    1
      1    0
      2    1
      3    0
      4    1
      Name: Outcome, dtype: int64
```

1.5.1 Future Scalling

```
[36]: # Standard Scaler:
      from sklearn.preprocessing import StandardScaler
      scaler = StandardScaler()
      scaler.fit(X)
      SSX = scaler.transform(X)
```

```
[37]: 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)
```

```
[38]: X_train.shape, y_train.shape
```

```
[38]: ((614, 8), (614,))
```

```
[39]: X_test.shape, y_test.shape
```

```
[39]: ((154, 8), (154,))
```

2 Classification Algorithms:

2.1 Logistic Regression:

```
[40]: from sklearn.linear_model import LogisticRegression
      lr = LogisticRegression(solver='liblinear', multi_class='ovr')
      lr.fit(X_train, y_train)
```

```
[40]: LogisticRegression(multi_class='ovr', solver='liblinear')
```

2.2 Decision Tree:

```
[41]: from sklearn.tree import DecisionTreeClassifier
      dt=DecisionTreeClassifier()
      dt.fit(X_train, y_train)
```

```
[41]: DecisionTreeClassifier()
```

3 Making prediction:

Logistic Regression:

```
[42]: X_test.shape
```

```
[42]: (154, 8)
```

```
[43]: lr_pred=lr.predict(X_test)
```

```
[44]: lr_pred.shape
```

```
[44]: (154,)
```

Decision Tree:

```
[45]: dt_pred=dt.predict(X_test)
```

```
[46]: dt_pred.shape
```

```
[46]: (154,)
```

4 Model Evaluation for Logistic Regression:

Train Score and Test Score

```
[47]: # For Logistic Regression:
      from sklearn.metrics import accuracy_score
      print("Train Accuracy of Logistic Regression: ", lr.score(X_train, y_train)*100)
      print("Accuracy (Test) Score of Logistic Regression: ", lr.score(X_test, y_test)*100)
      print("Accuracy Score of Logistic Regression: ", accuracy_score(y_test, lr_pred)*100)
```

Train Accuracy of Logistic Regression: 77.36156351791531

Accuracy (Test) Score of Logistic Regression: 77.27272727272727

Accuracy Score of Logistic Regression: 77.27272727272727

```
[48]: # For Decesion Tree:
print("Train Accuracy of Decesion Tree: ", dt.score(X_train, y_train)*100)
print("Accuracy (Test) Score of Decesion Tree: ", dt.score(X_test, y_test)*100)
print("Accuracy Score of Decesion Tree: ", accuracy_score(y_test, dt_pred)*100)
```

Train Accuracy of Decesion Tree: 100.0
 Accuracy (Test) Score of Decesion Tree: 80.51948051948052
 Accuracy Score of Decesion Tree: 80.51948051948052

5 Confusion Matrix

- *Confusion Matrix of “Logistic Regression”*

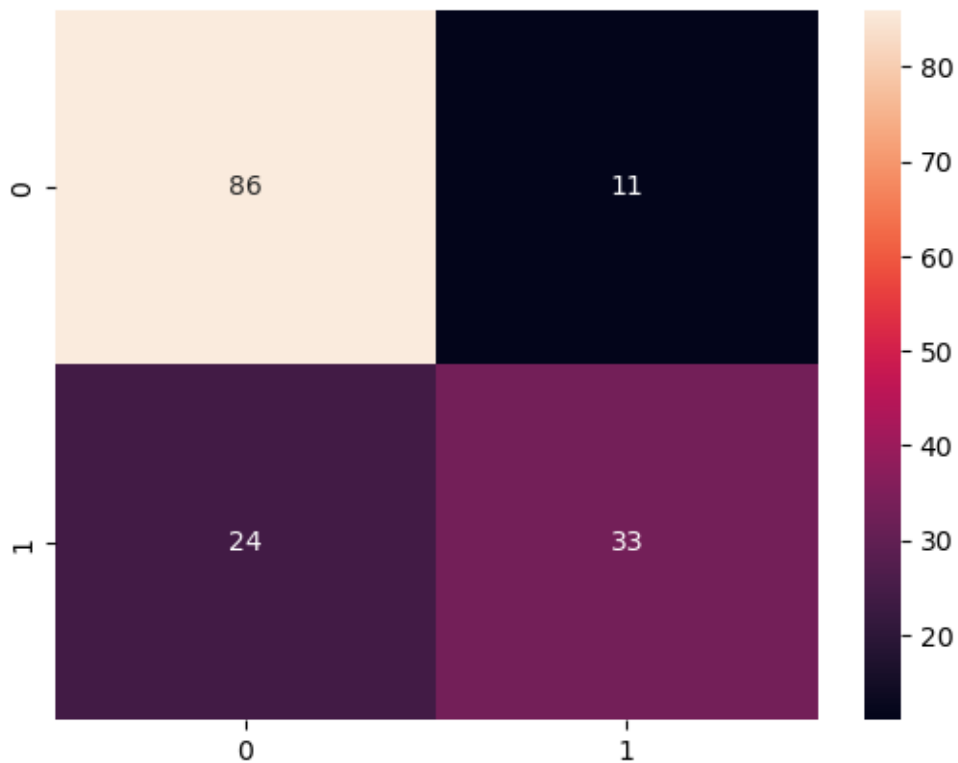
```
[49]: from sklearn.metrics import classification_report, confusion_matrix

cm = confusion_matrix(y_test, lr_pred)
cm
```

```
[49]: array([[86, 11],
          [24, 33]])
```

```
[50]: sns.heatmap(confusion_matrix(y_test, lr_pred), annot=True, fmt="d")
```

```
[50]: <Axes: >
```




```
[51]: TN =cm[0, 0]
      FP =cm[0,1]
      FN = cm[1,0]
      TP  = cm[1,1]
```

```
[52]: TN, FP, FN, TP
```

```
[52]: (86, 11, 24, 33)
```

```
[53]: from sklearn.metrics import classification_report, confusion_matrix
      from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
      cm = confusion_matrix(y_test, lr_pred)

      print('TN - True Negative {}'.format(cm[0,0]))
      print('FP - False Positive {}'.format(cm[0,1]))
      print('FN - False Negative {}'.format(cm[1,0]))
      print('TP - True Positive {}'.format(cm[1,1]))
      print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0], cm[1,1]]), np.
        ↳sum(cm))*100))
      print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1], cm[1,0]]), np.
        ↳np.sum(cm))*100))
```

```
TN - True Negative 86
FP - False Positive 11
FN - False Negative 24
TP - True Positive 33
Accuracy Rate: 77.27272727272727
Misclassification Rate: 22.727272727272727
```

```
[54]: 77.27272727272727+22.727272727272727
```

```
[54]: 100.0
```

```
[55]: import matplotlib.pyplot as plt
      import numpy as np

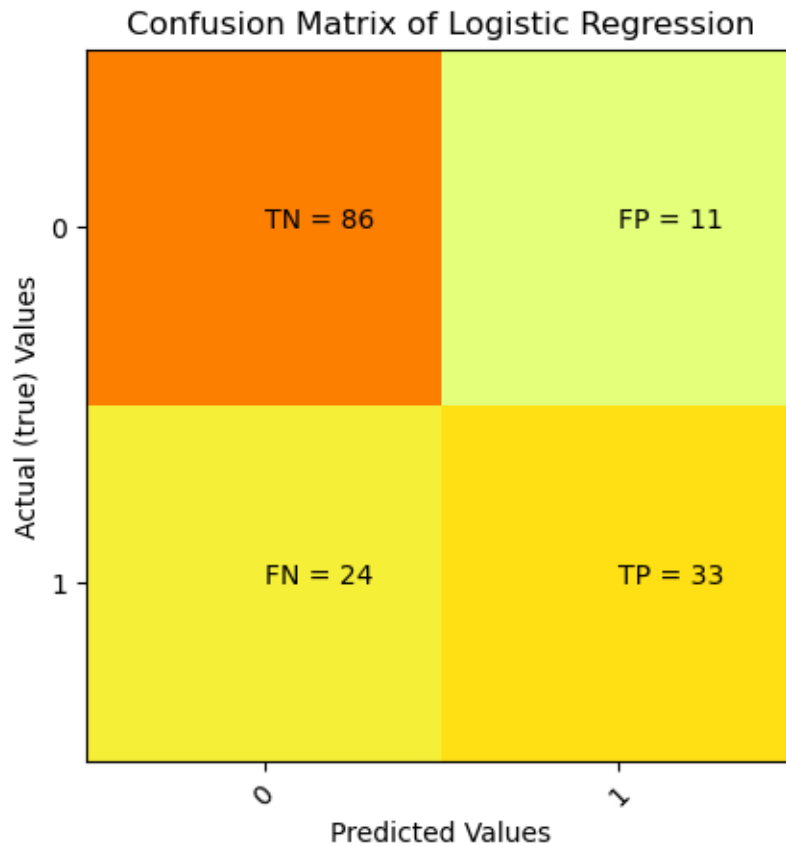
      plt.clf()
      plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
      classNames = ['0', '1']
      plt.title('Confusion Matrix of Logistic Regression')
      plt.ylabel('Actual (true) Values')
      plt.xlabel('Predicted Values')
      tick_marks = np.arange(len(classNames))
      plt.xticks(tick_marks, classNames, rotation=45)
      plt.yticks(tick_marks, classNames)
```

```

s = [['TN', 'FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j, i, str(s[i][j]) + " = " + str(cm[i][j]))

plt.show()

```



```
[56]: pd.crosstab(y_test, lr_pred, margins=False)
```

```

[56]: col_0    0    1
Outcome
0         86   11
1         24   33

```

```
[57]: pd.crosstab(y_test, lr_pred, margins=True)
```

```

[57]: col_0    0    1  All
Outcome
0         86   11   97
1         24   33   57

```

All 110 44 154

```
[58]: pd.crosstab(y_test, lr_pred, rownames=['Actual values'], colnames=['Predicted_
      ↪values'], margins=True)
```

```
[58]: Predicted values    0    1  All
      Actual values
      0                86   11   97
      1                24   33   57
      All              110   44  154
```

5.0.1 Precision:

PPV- positive Predictive Value

Precision = True Positive/True Positive + False Positive

Precision = TP/TP+FP

```
[59]: TP, FP
```

```
[59]: (33, 11)
```

```
[60]: Precision = TP/(TP+FP)
      Precision
```

```
[60]: 0.75
```

```
[61]: 33/(33+11)
```

```
[61]: 0.75
```

```
[62]: # precision Score:

      precision_score = TP/float(TP+FP)*100
      print('Precision Score: {0:0.4f}'.format(precision_score))
```

Precision Score: 75.0000

```
[63]: from sklearn.metrics import precision_score
      print("Precision Score is: ", precision_score(y_test, lr_pred)*100)
      print("Micro Average Precision Score is: ", precision_score(y_test, lr_pred,
      ↪average='micro')*100)
      print("Macro Average Precision Score is: ", precision_score(y_test, lr_pred,
      ↪average='macro')*100)
      print("Weighted Average Precision Score is: ", precision_score(y_test, lr_pred,
      ↪average='weighted')*100)
      print("precision Score on Non Weighted score is: ", precision_score(y_test,
      ↪lr_pred, average=None)*100)
```

```
[75]: from sklearn.metrics import f1_score
print('F1_Score of Macro: ', f1_score(y_test, lr_pred)*100)
```

F1_Score of Macro: 65.34653465346535

```
[76]: print("Micro Average f1 Score is: ", f1_score(y_test, lr_pred,
↪average='micro')*100)
print("Macro Average f1 Score is: ", f1_score(y_test, lr_pred,
↪average='macro')*100)
print("Weighted Average f1 Score is: ", f1_score(y_test, lr_pred,
↪average='weighted')*100)
print("f1 Score on Non Weighted score is: ", f1_score(y_test, lr_pred,
↪average=None)*100)
```

Micro Average f1 Score is: 77.27272727272727

Macro Average f1 Score is: 74.21916104653944

Weighted Average f1 Score is: 76.52373933045479

f1 Score on Non Weighted score is: [83.09178744 65.34653465]

5.3 Classification Report of Logistic Regression:

```
[77]: from sklearn.metrics import classification_report
print('Classification Report of Logistic Regression: \n',
↪classification_report(y_test, lr_pred, digits=4))
```

Classification Report of Logistic Regression:

	precision	recall	f1-score	support
0	0.7818	0.8866	0.8309	97
1	0.7500	0.5789	0.6535	57
accuracy			0.7727	154
macro avg	0.7659	0.7328	0.7422	154
weighted avg	0.7700	0.7727	0.7652	154

5.4 ROC Curve& ROC AUC

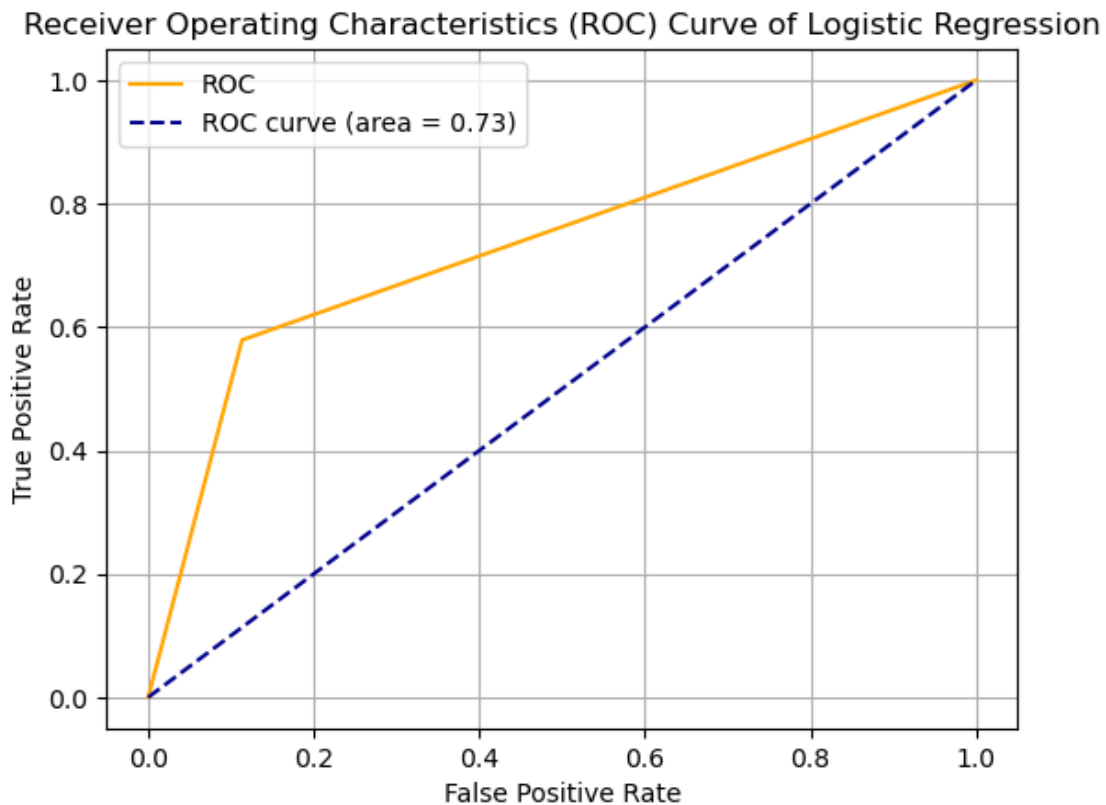
```
[78]: # Area under Curve:
auc= roc_auc_score(y_test, lr_pred)
print("ROC AUC SCORE of logistic Regression is ", auc)
```

ROC AUC SCORE of logistic Regression is 0.7327726532826913

```
[79]: from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_test, lr_pred)
```

```
plt.plot(fpr, tpr, color='orange', label="ROC")
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='ROC curve_
↳(area = %0.2f)' % auc(fpr, tpr))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristics (ROC) Curve of Logistic_
↳Regression")
plt.legend()
plt.grid()
plt.show()
```



5.5 Confusion Matrix:

- Confusion matrix of “Decision Tree”

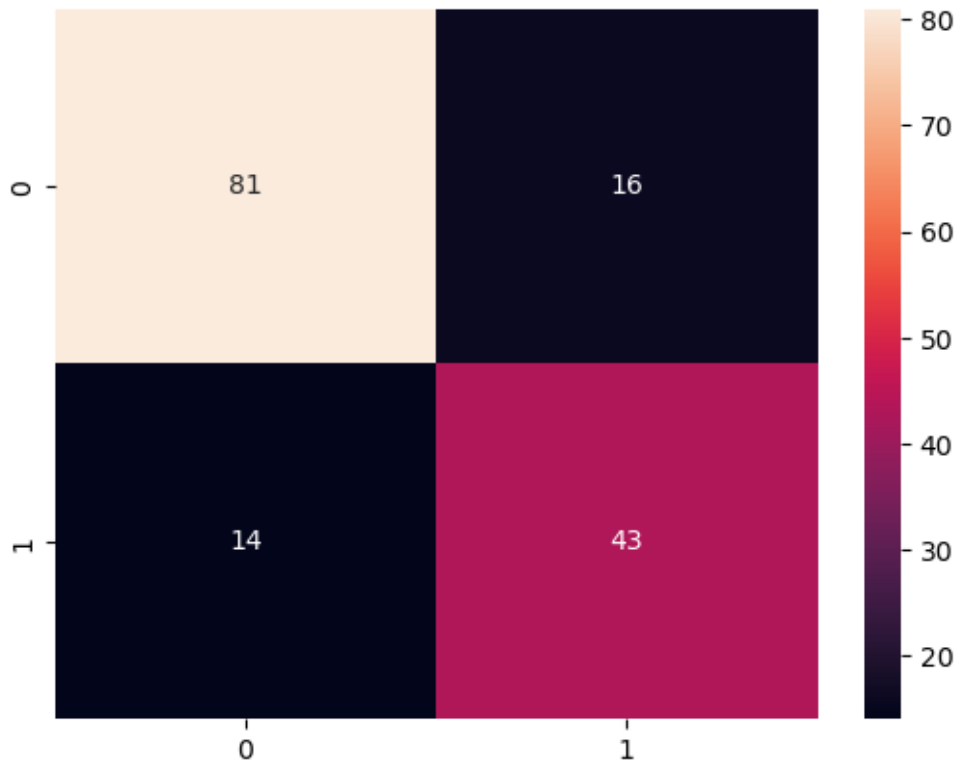
```
[80]: from sklearn.metrics import classification_report, confusion_matrix

cm = confusion_matrix(y_test, dt_pred)
cm
```

```
[80]: array([[81, 16],
           [14, 43]])
```

```
[81]: sns.heatmap(confusion_matrix(y_test, dt_pred), annot=True, fmt="d")
```

```
[81]: <Axes: >
```



```
[82]: TN =cm[0, 0]
FP =cm[0,1]
FN = cm[1,0]
TP = cm[1,1]
```

```
[83]: TN, FP, FN, TP
```

```
[83]: (81, 16, 14, 43)
```

```
[84]: from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
cm = confusion_matrix(y_test, dt_pred)

print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
```

```

print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0], cm[1,1]]), np.
    ↳sum(cm))*100))
print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1], cm[1,0]]),
    ↳np.sum(cm))*100))

```

TN - True Negative 81
 FP - False Positive 16
 FN - False Negative 14
 TP - True Positive 43
 Accuracy Rate: 80.51948051948052
 Misclassification Rate: 19.480519480519483

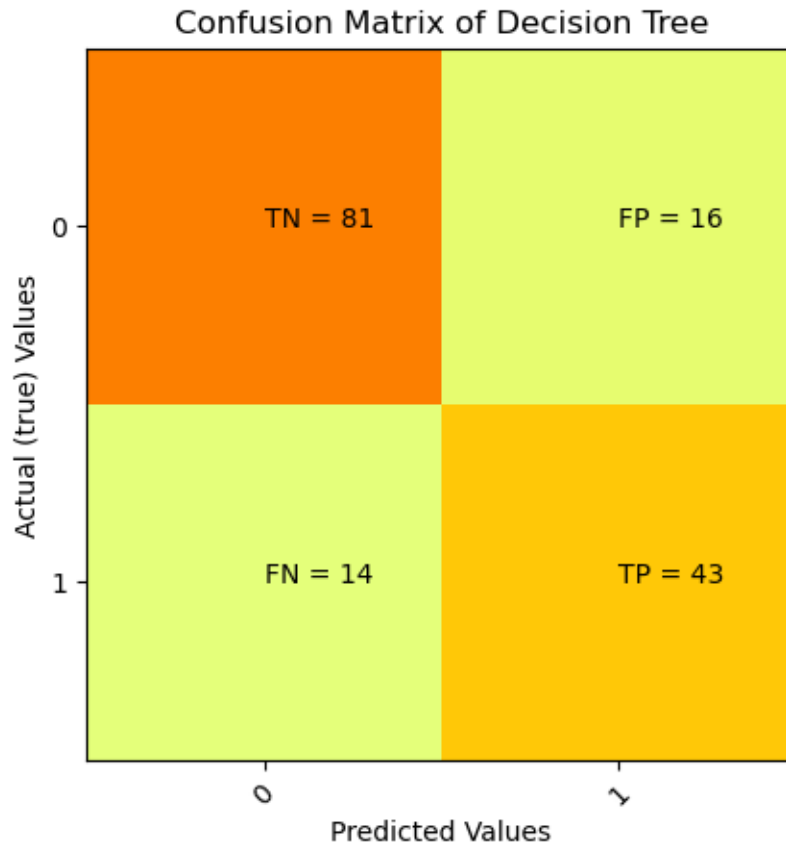
```

[85]: import matplotlib.pyplot as plt
import numpy as np

plt.clf()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Confusion Matrix of Decision Tree')
plt.ylabel('Actual (true) Values')
plt.xlabel('Predicted Values')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN', 'FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j, i, str(s[i][j]) + " = " + str(cm[i][j]))

plt.show()

```



5.6 Precision:

```
[86]: # precision Score:

precision_score = TP/float(TP+FP)*100
print('Precision Score: {0:0.4f}'.format(precision_score))
```

Precision Score: 72.8814

```
[87]: from sklearn.metrics import precision_score

print("Precision Score is:", precision_score(y_test, dt_pred) * 100)
print("Micro Average Precision Score is:", precision_score(y_test, dt_pred,
    ↳average='micro') * 100)
print("Macro Average Precision Score is:", precision_score(y_test, dt_pred,
    ↳average='macro') * 100)
print("Weighted Average Precision Score is:", precision_score(y_test, dt_pred,
    ↳average='weighted') * 100)
```



```
print("Precision Score on Non Weighted score is:", precision_score(y_test, dt_pred, average=None) * 100)
```

Precision Score is: 72.88135593220339
Micro Average Precision Score is: 80.51948051948052
Macro Average Precision Score is: 79.07225691347011
Weighted Average Precision Score is: 80.68028314237056
Precision Score on Non Weighted score is: [85.26315789 72.88135593]

5.7 Recall:

```
[88]: recall_score = TP / float(TP + FN) * 100  
print('recall_score', recall_score)
```

recall_score 75.43859649122807

```
[89]: from sklearn.metrics import recall_score  
print('Recall or Sensitivity_Score: ', recall_score(y_test, dt_pred) * 100)
```

Recall or Sensitivity_Score: 75.43859649122807

```
[90]: print("recall Score is: ", recall_score(y_test, dt_pred) * 100)  
print("Micro Average recall Score is: ", recall_score(y_test, dt_pred, average='micro') * 100)  
print("Macro Average recall Score is: ", recall_score(y_test, dt_pred, average='macro') * 100)  
print("Weighted Average recall Score is: ", recall_score(y_test, dt_pred, average='weighted') * 100)  
print("recall Score on Non Weighted score is: ", recall_score(y_test, dt_pred, average=None) * 100)
```

recall Score is: 75.43859649122807
Micro Average recall Score is: 80.51948051948052
Macro Average recall Score is: 79.47187556520167
Weighted Average recall Score is: 80.51948051948052
recall Score on Non Weighted score is: [83.50515464 75.43859649]

5.8 FPR

```
[91]: FPR = FP / float(FP + TN) * 100  
print('False Positive Rate: {:.4f}'.format(FPR))
```

False Positive Rate: 16.4948

5.9 Specificity:

```
[92]: specificity = TN / (TN + FP) * 100
      print('Specificity : {0:0.4f}'.format(specificity))
```

Specificity : 83.5052

```
[93]: from sklearn.metrics import f1_score
      print('F1_Score of Macro: ', f1_score(y_test, dt_pred) * 100)
```

F1_Score of Macro: 74.13793103448276

```
[94]: print("Micro Average f1 Score is: ", f1_score(y_test, dt_pred,
      ↪ average='micro') * 100)
      print("Macro Average f1 Score is: ", f1_score(y_test, dt_pred,
      ↪ average='macro') * 100)
      print("Weighted Average f1 Score is: ", f1_score(y_test, dt_pred,
      ↪ average='weighted') * 100)
      print("f1 Score on Non Weighted score is: ", f1_score(y_test, dt_pred,
      ↪ average=None) * 100)
```

Micro Average f1 Score is: 80.51948051948051

Macro Average f1 Score is: 79.25646551724138

Weighted Average f1 Score is: 80.58595499328258

f1 Score on Non Weighted score is: [84.375 74.13793103]

5.10 Classification Report of Decision Tree:

```
[95]: from sklearn.metrics import classification_report
      print('Classification Report of Decision Tree: \n',
      ↪ classification_report(y_test, dt_pred, digits=4))
```

Classification Report of Decision Tree:

	precision	recall	f1-score	support
0	0.8526	0.8351	0.8438	97
1	0.7288	0.7544	0.7414	57
accuracy			0.8052	154
macro avg	0.7907	0.7947	0.7926	154
weighted avg	0.8068	0.8052	0.8059	154

```

[1]: 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.preprocessing import StandardScaler
from sklearn import svm
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

RED = "\033[91m"
GREEN = "\033[92m"
YELLOW = "\033[93m"
BLUE = "\033[94m"
RESET = "\033[0m"

df = pd.read_csv("/kaggle/input/diabetes-data-set/diabetes.csv")

# DATA CLEANING
print(BLUE + "\nDATA CLEANING" + RESET)
# --- Check for missing values
missing_values = df.isnull().sum()
print(GREEN + "Missing Values : " + RESET)
print(missing_values)
# --- Handle missing values
mean_fill = df.fillna(df.mean())
df.fillna(mean_fill, inplace=True)
# --- Check for duplicate values
duplicate_values = df.duplicated().sum()
print(GREEN + "Duplicate Values : " + RESET)
print(duplicate_values)
# --- Drop duplicate values
df.drop_duplicates(inplace=True)

# DATA ANALYSIS
print(BLUE + "\nDATA ANALYSIS" + RESET)

```

```

# --- Summary Statistics
summary_stats = df.describe()
print(GREEN + "Summary Statistics : " + RESET)
print(summary_stats)
# --- Class Distribution
class_distribution = df["Outcome"].value_counts()
print(GREEN + "Class Distribution : " + RESET)
print(class_distribution)

# Support Vector Machine Modelling
print(BLUE + "\nMODELLING" + RESET)
X = df.drop("Outcome", axis=1)
y = df["Outcome"]
# --- Splitting 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
)
# --- Standardize Features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# --- init and train SVM model
model = svm.SVC(kernel="linear")
model.fit(X_train, y_train)
# --- Predict on test data
y_pred = model.predict(X_test)
# --- Evaluate model performance
accuracy = model.score(X_test, y_test)
print(GREEN + "Model Accuracy : " + RESET)
print(accuracy)
# --- Classification Report and Confusion Matrix
print(GREEN + "Classification Report : " + RESET)
print(classification_report(y_test, y_pred))
print(GREEN + "Confusion Matrix : " + RESET)
cm = ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
sns.heatmap(cm.confusion_matrix, annot=True, cmap="Blues")
plt.show()
print("Displayed")

# SAVING THE FILE
df.to_csv("/kaggle/working/cleaned_diabetes.csv", index=False)
print(BLUE + "\nDATA SAVING" + RESET)
print(GREEN + "Data Cleaned and Saved !" + RESET)
print("\n")

```

DATA CLEANING

Missing Values :

Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	0

dtype: int64

Duplicate Values :

0

DATA ANALYSIS

Summary Statistics :

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin \
count	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479
std	3.369578	31.972618	19.355807	15.952218	115.244002
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000
75%	6.000000	140.250000	80.000000	32.000000	127.250000
max	17.000000	199.000000	122.000000	99.000000	846.000000

	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000
mean	31.992578	0.471876	33.240885	0.348958
std	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.078000	21.000000	0.000000
25%	27.300000	0.243750	24.000000	0.000000
50%	32.000000	0.372500	29.000000	0.000000
75%	36.600000	0.626250	41.000000	1.000000
max	67.100000	2.420000	81.000000	1.000000

Class Distribution :

Outcome

0 500

1 268

Name: count, dtype: int64

MODELLING

Model Accuracy :

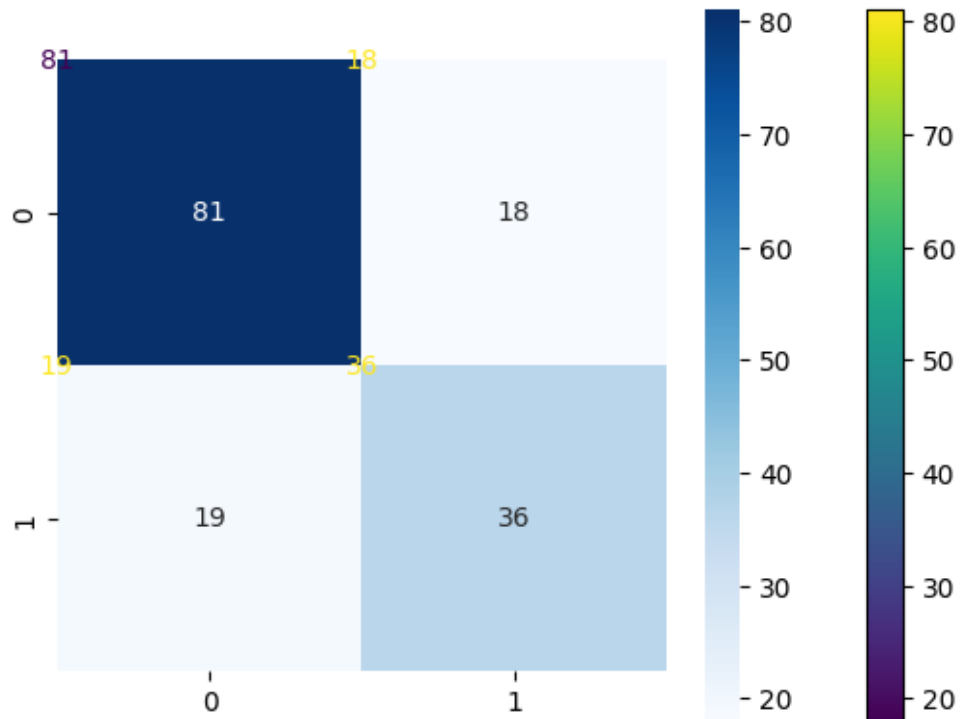
0.7597402597402597

Classification Report :

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.81	0.82	0.81	99
1	0.67	0.65	0.66	55
accuracy			0.76	154
macro avg	0.74	0.74	0.74	154
weighted avg	0.76	0.76	0.76	154

Confusion Matrix :



Displayed

DATA SAVING

Data Cleaned and Saved !

Conclusion:

In summary, the AI-Powered Diabetes Prediction System offers a data-driven solution for early risk assessment and personalized diabetes prevention. It integrates diverse medical data, applies rigorous data preprocessing, and develops accurate predictive models. By prioritizing data privacy, regulatory compliance, and continuous learning, this system has the potential to significantly improve public health. It underscores the transformative impact of AI in healthcare, empowering individuals and healthcare professionals to take proactive steps towards mitigating the risks of diabetes and its associated complications, ultimately enhancing both the quality of life and healthcare efficiency.