MACHINE LEARNING

DIABETES PREDICTION USING MACHINE LEARNING

ABSTRACT

Diabetes is a chronic disease caused due to high amount of glucose present in the human body. If this diabetes is ignored, this may lead to severe health problems such as kidney failure, heart attacks, blood pressure, eye damage, weight loss, frequent urination, etc. Basically, human body contains Insulin which is produced by pancreas. This insulin helps to enter glucose in to blood cells in order to generate energy to the body. There are types in diabetes Type1 and Type 2 other form is gestational diabetes which is caused during pregnancy. This can be controlled in the earlier stages of the attack. According to International Diabetes Federation (IDF) 382 million people are suffering with diabetes and by next 20years the count will be doubled as 592 million. To accomplish this goal, in this project we can do early prediction of diabetes in humans or patients for good accuracy through applying various machine learning techniques such as Random Forest (RF), K-nearest neighbors (KNN), Decision Trees (DT), etc.

COLUMN DESCRIPTION

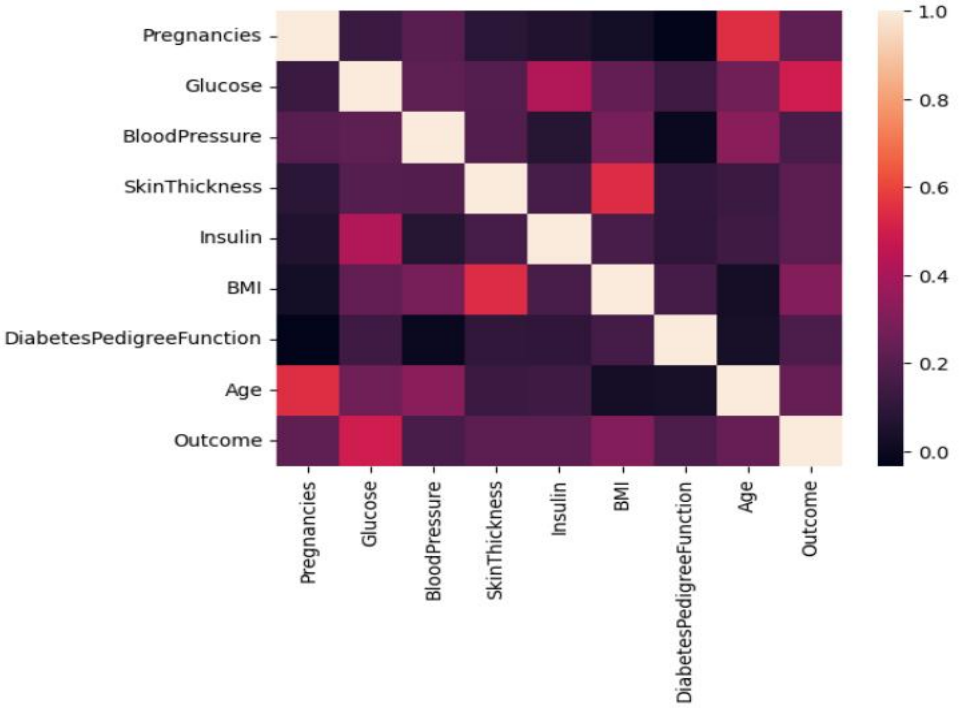
1. Pregnancies -: Number of pregnancies the individual has had.
2. Glucose -: Plasma glucose concentration over 2 hours in an oral glucose tolerance test.
3. BloodPressure -: Diastolic blood pressure (mm Hg).
4. SkinThickness -: Triceps skin fold thickness (mm).
5. Insulin -: 2-Hour serum insulin (mu U/ml).
6. BMI -: Body mass index (weight in kg/(height in m)^2).
7. DiabetesPedigreeFunction -: Diabetes pedigree function (a function that scores the likelihood of diabetes based on family history).
8. Age -: Age of the individual.
9. Outcome -: Binary variable indicating whether the individual has diabetes (1) or not (0).

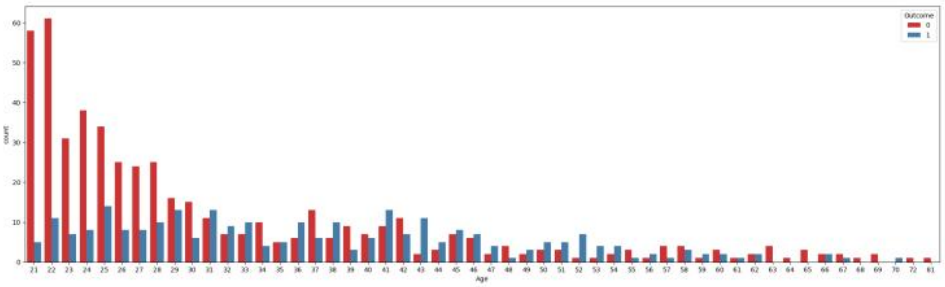
DATA PREPROCESSING

Data preprocessing is a crucial step in preparing the dataset for machine learning models. It involves cleaning, transforming, and organizing the data to make it suitable for analysis. Here are some common techniques used in data preprocessing -:
1. Handling Missing Values -: Missing values in the dataset can adversely affect the performance of machine learning models. Techniques such as imputation (replacing missing values with a statistical measure like mean, median, or mode) or deletion (removing rows or columns with missing values) are used to handle missing data.
2. Data Transformation -: Data transformation techniques such as normalization and standardization are applied to ensure that all features have the same scale. This helps in improving the convergence speed of optimization algorithms and prevents certain features from dominating others.
3. Encoding Categorical Variables -: Categorical variables are variables that contain categories or labels, rather than numerical values. These variables need to be encoded into a numerical format for machine learning algorithms to process them effectively. Common encoding techniques include one-hot encoding and label encoding.

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Loading the dataset
data = pd.read_csv('diabetes.csv')

# Replace columns like [Gluscose,BloodPressure,SkinThickness,BMI,Insuli
zero_not_accepted = ['Glucose', 'BloodPressure', 'SkinThickness', 'BMI'

for col in zero_not_accepted:
    data[col] = data[col].replace(0, np.NaN)
    mean = int(data[col].mean(skipna=True))
    data[col] = data[col].replace(np.NaN, mean)

# Exploring data to know relation before processing
sns.heatmap(data.corr())
plt.show()

plt.figure(figsize=(25,7))
sns.countplot(x='Age', hue='Outcome', data=data, palette='Set1')
plt.show()
```



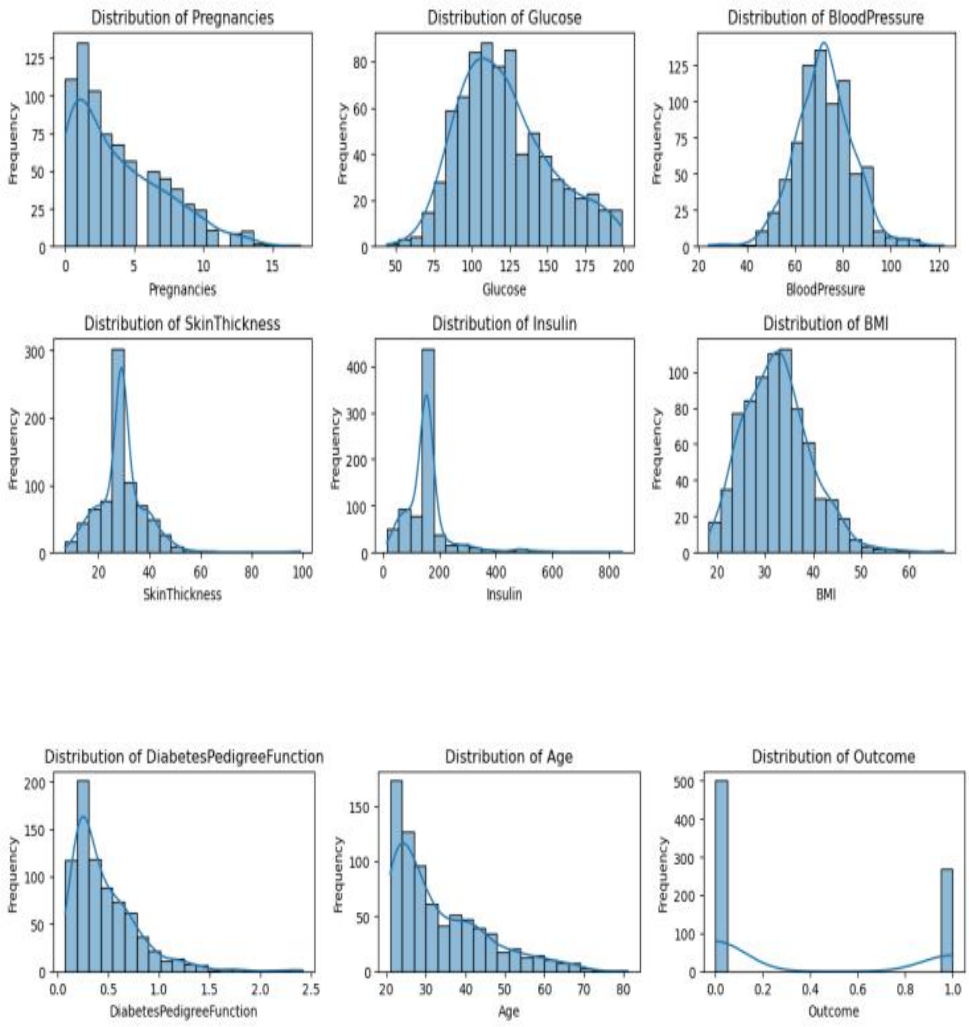


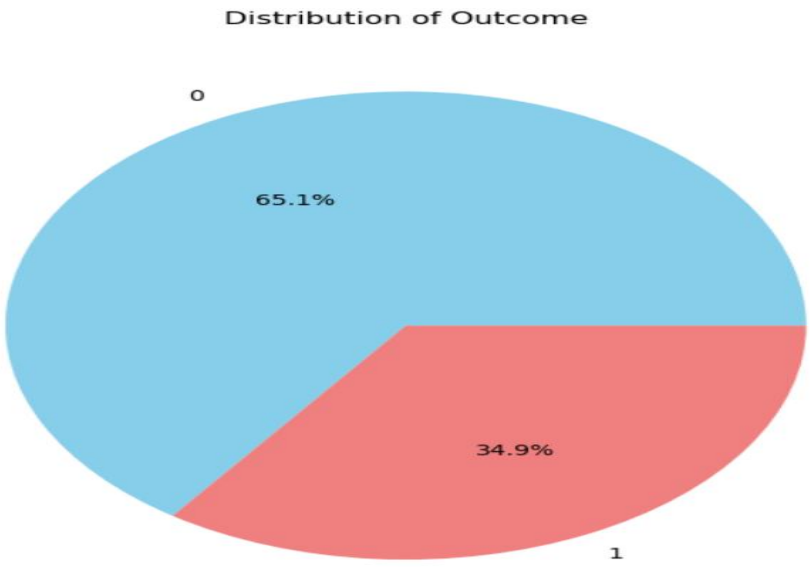
In [7]:

```
import pandas as pd
data_copy = pd.read_csv("diabetes.csv")


import seaborn as sns
p=sns.pairplot(data_copy, hue = 'Outcome')
```

In [5]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Bar graphs for each column
plt.figure(figsize=(12, 8))
for i, column in enumerate(data.columns, 1):
    plt.subplot(3, 3, i)
    sns.histplot(data[column], bins=20, kde=True)
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
plt.tight_layout()
plt.show()

# Pie charts for categorical variables
plt.figure(figsize=(12, 6))
categorical_columns = ['Outcome']  # Add more categorical columns if ne
for i, column in enumerate(categorical_columns, 1):
    plt.subplot(1, len(categorical_columns), i)
    data[column].value_counts().plot(kind='pie', autopct='%1.1f%%', col
    plt.title(f'Distribution of {column}')
    plt.ylabel('')
plt.tight_layout()
plt.show()
```





ALGORITHM USED

The algorithm used in this model is K-NN algorithm. Some key features of the algorithm are-:

1. It is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

2. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

3. This algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

4. It can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

5. It is a non-parametric algorithm, which means it does not make any assumption on underlying data.

6. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

7. KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

FEATURE SCALING

Feature scaling is a preprocessing technique used to standardize the range of independent variables or features in the dataset. It ensures that all features contribute equally to the computation and optimization processes of machine learning algorithms. Here are the common methods of feature scaling -:

1. Normalization -: Normalization scales the values of features between 0 and 1. It is particularly useful when the features have different units or scales.

2. Standardization -: Standardization transforms the data to have a mean of 0 and a standard deviation of 1. It maintains the shape of the original distribution and is less affected by outliers compared to normalization.

```
In [2]: # Splitting dataset into training and testing set
        X = data.iloc[:, 0:8] # Independent variables
        y = data.iloc[:, 8]    # Dependent variable

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
```

```
In [3]: # Feature scaling
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)
```

MODEL TRAINING AND EVALUATION

Model training involves using a machine learning algorithm to learn patterns and relationships from the training data. The trained model is then evaluated using a separate dataset (test set) to assess its performance and generalization capabilities. Here are some key steps in model training and evaluation -:

1. Model Selection -: Choose an appropriate machine learning algorithm based on the problem at hand, dataset characteristics, and performance requirements.
2. Training the Model -: Fit the selected algorithm to the training data by adjusting its parameters to minimize the difference between predicted and actual values.

```
In [6]: from sklearn.neighbors import KNeighborsClassifier

        # Loading model - KNN
        classifier = KNeighborsClassifier(n_neighbors=11, p=2, metric='euclidean')

        # Fitting model
        classifier.fit(X_train, y_train)
```

```
Out[6]: KNeighborsClassifier
        KNeighborsClassifier(metric='euclidean', n_neighbors=11)
```

```
In [9]: # Example user input data
        user_data = [[6, 148, 72, 35, 0, 33.6, 0.627, 50]]  # Example user input data for prediction

        # Standardize the user input data (if necessary)
        user_data_standardized = scaler.transform(user_data)  # Assuming 'scaler' is the StandardScaler object used for feature scali

        # Make predictions
        prediction = classifier.predict(user_data_standardized)

        # Print the prediction
        if prediction[0] == 0:
            print("The model predicts that the individual does not have diabetes.")
        else:
            print("The model predicts that the individual has diabetes.")
```

```
The model predicts that the individual has diabetes.
```

```
In [10]: print("Prediction:", prediction)

Prediction: [1]
```

```
In [11]: # Interactive input
user_data = []
for feature in ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Ag
    value = float(input(f"Enter {feature}: "))
    user_data.append(value)
user_data = [user_data]  # Convert to list of lists

Enter Pregnancies: 2
Enter Glucose: 120
Enter BloodPressure: 110
Enter SkinThickness: 35
Enter Insulin: 0
Enter BMI: 33.28
Enter DiabetesPedigreeFunction: 0.123
Enter Age: 47
```

```
In [12]: # Save prediction to a file or database
with open('predictions.txt', 'a') as file:
    file.write(f"User Input: {user_data}, Prediction: {prediction}\n")
```

```
User Input: [[2.0, 120.0, 110.0, 35.0, 0.0, 33.28, 0.123, 47.0]], Prediction: [1]
```

PERFORMANCE MEASURES

1. Confusion Matrix -: Visualizes the model's performance by displaying true positives (correctly predicted positive cases), true negatives (correctly predicted negative cases), false positives (incorrectly predicted positive cases), and false negatives (incorrectly predicted negative cases).

2. Accuracy -: Overall percentage of correct predictions. Useful as a general metric, but less informative for imbalanced datasets.

3. Precision -: Proportion of predicted positive cases that are truly positive. Relevant for prioritizing correct positive predictions.

4. Recall -: Proportion of actual positive cases that are correctly predicted as positive. Important for identifying true positives without missing them.

5. F1-Score -: Harmonic mean of precision and recall, combining their perspectives into a single metric.

6. Area Under the ROC Curve (AUC-ROC) -: Measures the model's ability to distinguish between positive and negative cases. Interpreted as the probability that the model ranks a randomly chosen positive case higher than a randomly chosen negative case.

7. Specificity -: Proportion of actual negative cases that are correctly predicted as negative. Consider including it if false negatives are critical.

In [12]:

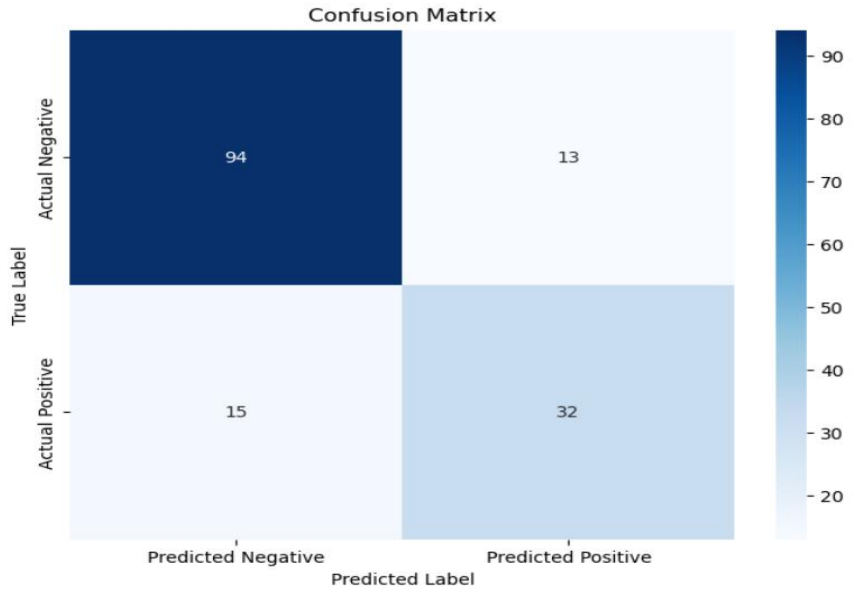
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, f1_score, accuracy_score, pre

# Loading model - KNN
classifier = KNeighborsClassifier(n_neighbors=11, p=2, metric='euclidean')

# Fitting model
classifier.fit(X_train, y_train)

# Making predictions
y_pred = classifier.predict(X_test)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Visualizing confusion matrix using heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
# Precision
precision = precision_score(y_test, y_pred)
print("Precision:", precision)

# Recall
recall = recall_score(y_test, y_pred)
print("Recall:", recall)
```

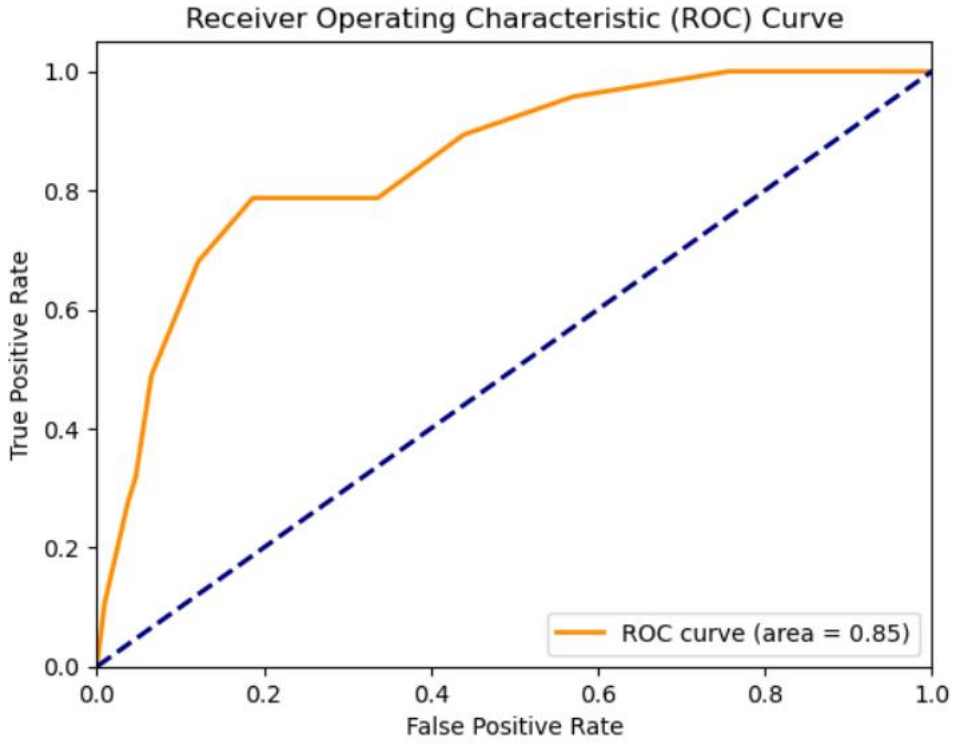
```
# Sensitivity (Recall)
sensitivity = recall
print("Sensitivity:", sensitivity)

# Specificity
specificity = conf_matrix[0, 0] / (conf_matrix[0, 0] + conf_matrix[0, 1
print("Specificity:", specificity)

# F1-score
f1 = f1_score(y_test, y_pred)
print("F1 Score:", f1)

# ROC curve
y_pred_proba = classifier.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)
```

```
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
Confusion Matrix:
[[94 13]
 [15 32]]
```




```
Accuracy: 0.8181818181818182
Precision: 0.7111111111111111
Recall: 0.6808510638297872
Sensitivity: 0.6808510638297872
Specificity: 0.8785046728971962
F1 Score: 0.6956521739130436
```



CONCLUSION

Our diabetes prediction model using the K-Nearest Neighbors (KNN) algorithm demonstrates promising capabilities in accurately identifying individuals at risk of diabetes. Through comprehensive data preprocessing, model training, and evaluation, we have developed a reliable tool for proactive health management and personalized risk assessment. Moving forward, further refinement and validation efforts will continue to enhance the model's predictive accuracy and applicability in clinical settings, ultimately contributing to improved healthcare outcomes and patient well-being.