Computer Architecture & Organisation (BCSE205L)

Solving Socio-Economic Problem with justification on Processor, Memory, IO and other auxiliary components

DA 2: Base Paper Implementation and Proposed Work

NAME: R HEMESH **REG NO:22BCT0328**

TOPIC: DETECTION OF DIABETES MELLITUS USING ML

1. PLATFORM :

This work is done in a Python notebook that performs data analysis and machine learning model building on the Pima Indians Diabetes dataset. The notebook uses various Python libraries and machine learning algorithms to predict whether a patient has diabetes based on certain diagnostic measurements.

The platform used for this analysis is a Python notebook, which is a web-based interactive computational environment where you can combine code execution, text, mathematics, plots, and rich media into a single document.

2. PROGRAMMING:

FULL CODE AND OUTPUT IS SHOWN IN THE FOLLOWING LINK

LINK: https://www.kaggle.com/code/hemesh11/notebook8fcc1603b6

3. Run time environment – IDE, Simulator/Emulator, packages, hardware prototype :

The runtime environment for this project is KAGGLE, a popular Integrated Development Environment (IDE) for Python, although the specific IDE is not explicitly mentioned in the document. The project utilizes several Python libraries, including:

- **Numpy**: A library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- **Pandas**: A software library for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.
- **Matplotlib**: A plotting library for the Python programming language and its numerical mathematics extension NumPy.
- **Seaborn**: A Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

- **Sklearn (Scikit-learn)**: A free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms.

The following machine learning algorithms from the Sklearn library are used:

- Logistic Regression
- K-Nearest Neighbors (KNN)
- Gaussian Naive Bayes
- Support Vector Machine (SVM)
- Decision Tree
- Random Forest

4. DATA SET :

LINK: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

The dataset used in this analysis is the Pima Indians Diabetes dataset. It contains the following variables:

1. Pregnancies: Number of times pregnant
2. Glucose: Plasma glucose concentration a 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
8. Age: Age (years)
9. Outcome: Class variable (0 or 1)

The dataset has 768 entries and 9 columns. There are no null values in the dataset. The dataset contains some zero values in the columns 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', and 'BMI'. These zero values are replaced with the mean of the respective columns.

The dataset is split into input features (X) and the target feature (y). The input features are scaled using the StandardScaler. The data is then split into a training set and a test set with a test size of 20%.

The dataset is used to train and test various classification algorithms including Logistic Regression, KNN Classifier, Naive-Bayes Classifier, Support Vector Machine, Decision Tree, and Random Forest. The performance of these models is evaluated and compared.

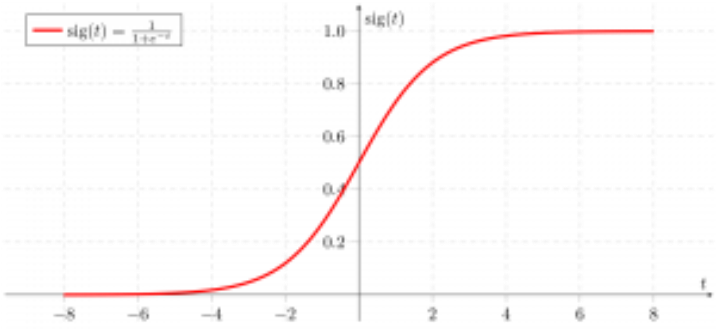
5. ALGORITHMs:

1. **Logistic Regression**: This is a statistical model used for binary classification problems. It uses the logistic function to model the probability of a certain class or event.

   Logistic regression is a statistical model used for binary classification tasks, where the outcome variable is categorical and has only two possible values (e.g., 0 or 1, yes or no). It estimates the probability that a given input data point belongs to a certain category. Despite its name, logistic regression is a linear model for classification, not regression.

   logistic regression working:

   a. **Sigmoid Function**: Logistic regression uses the logistic function (also called the sigmoid function) to model the relationship between the input variables and the output. The sigmoid function maps any real-valued number to a value between 0 and 1, which can be interpreted as a probability.



   b. **Linear Combination**: Logistic regression combines the input features linearly using weights (coefficients) and adds a bias term (intercept) to generate a linear combination of the input features.

$$z = b + \sum_{i=1}^{n} w_i \cdot x_i$$

   c. **Probability Estimation**: The linear combination of features is then passed through the sigmoid function to estimate the probability of the output being 1.

$$\hat{y} = \frac{1}{1+e^{-z}}$$

d. **Decision Boundary:** Based on the estimated probabilities, logistic regression predicts the class label by applying a threshold (usually 0.5). If the estimated probability is greater than or equal to the threshold, the output is predicted as 1; otherwise, it's predicted as 0.
e. **Training:** The model is trained using a method called maximum likelihood estimation, which finds the weights that maximize the likelihood of the observed data given the model.

2. **K-Nearest Neighbors (KNN)**: This is a type of instance-based learning or non-generalizing learning. It stores instances of training data and classifies new instances based on a similarity measure.

   The k-Nearest Neighbors (KNN) algorithm is a simple, instance-based learning algorithm used for classification and regression tasks. It's a type of lazy learning algorithm because it doesn't explicitly build a model. Instead, it memorizes the training instances and makes predictions for new instances based on their similarity to existing instances in the training data.

   KNN algorithm working for classification tasks:

   1. **Training**: KNN stores all the training examples and their corresponding class labels.
   2. **Prediction**: To predict the class label of a new data point, KNN calculates the distance between the new data point and all the training examples. The most common distance metric used is Euclidean distance, but other metrics like Manhattan distance can also be used.
   3. **Choosing k**: KNN requires the selection of a parameter k, which specifies the number of nearest neighbors to consider. A common approach is to use cross-validation to choose the best value of k.
   4. **Voting**: Once the distances are calculated, KNN selects the k nearest neighbors and assigns the class label by majority voting. For regression tasks, the predicted value can be the average of the k nearest neighbors' values.
   5. **Prediction**: After determining the class label by majority voting, KNN assigns this class label to the new data point.

3. **Naive Bayes**: This is a classification technique based on Bayes' theorem with an assumption of independence among predictors. It is easy to build and particularly useful for large datasets.

   The Naive Bayes algorithm is a simple and probabilistic machine learning algorithm based on Bayes' theorem with an assumption of independence among predictors. It's commonly used for classification tasks, especially for text classification and spam filtering. Despite its simplicity, Naive Bayes often performs well in practice and is known for its efficiency and ease of implementation.

   Naive Bayes algorithm working:

   a. **Bayes' Theorem:** The algorithm is based on Bayes' theorem, which describes the probability of a hypothesis given the data. Mathematically, it's represented as:

   $$P(A|B) = P(B|A) \times P(A) / P(B)$$

   where:

   $P(A|B)$ is the posterior probability of class A given predictor B,

   $P(B|A)$ is the likelihood of predictor B given class A,

   $P(A)$ is the prior probability of class A, and

   $P(B)$ is the prior probability of predictor B.

   b. **Independence Assumption:** Naive Bayes assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. This is a strong and unrealistic assumption, but it simplifies the computation and often works well in practice, especially for text data.
   c. **Classification:** For a given instance with features $x1, x2, \ldots, xn$, Naive Bayes calculates the posterior probability of each class Ck using Bayes' theorem and selects the class with the highest posterior probability as the prediction.
   d. **Types of Naive Bayes:**

      Gaussian Naive Bayes: Assumes that continuous features follow a Gaussian distribution.

      Multinomial Naive Bayes: Suitable for classification with discrete features (e.g., word counts for text classification).

      Bernoulli Naive Bayes: Similar to Multinomial Naive Bayes but assumes binary (Bernoulli) features.

   e. **Smoothing:** To handle cases where a feature value is zero in the training data and causes the posterior probability to be zero, smoothing techniques like Laplace smoothing are used.

Naive Bayes is efficient, especially for large datasets, and works well in practice, even with the independence assumption. However, it may not perform as well as more complex algorithms on datasets where the independence assumption is violated.

4. **Support Vector Machine (SVM)**: This is a supervised learning model used for classification and regression analysis. It is effective in high dimensional spaces and versatile as different Kernel functions can be specified for the decision function.

   The Support Vector Machine (SVM) algorithm is a powerful and versatile supervised learning algorithm used for classification and regression tasks. It's particularly well-suited for classification of complex datasets that are not linearly separable. SVMs are based on the concept of finding the hyperplane that best separates different classes in the feature space.

   SVM algorithm working:

   1. **Linear Separability**: Given a set of training examples, each belonging to one of two classes, SVM finds the hyperplane that separates the two classes with the maximum margin. This hyperplane is the one that maximizes the distance between the closest data points of each class, known as support vectors.
   2. **Kernel Trick**: SVM can also handle non-linearly separable data by using a technique called the kernel trick. The kernel trick maps the input features into a higher-dimensional space where the classes become linearly separable. Common kernels include linear, polynomial, radial basis function (RBF), and sigmoid kernels.
   3. **Margin**: The margin is the distance between the hyperplane and the closest data points (support vectors). SVM aims to maximize this margin, as it helps in better generalization to unseen data.
   4. **Soft Margin**: In cases where the data is not linearly separable or there are outliers, SVM can use a soft margin approach. This allows for some misclassification to achieve a better overall margin.
   5. **Regularization**: SVM uses a regularization parameter *C* to control the trade-off between maximizing the margin and minimizing the classification error. A smaller *C* allows for a wider margin but more margin violations, while a larger *C* penalizes margin violations more heavily.
   6. **Decision Function**: The decision function of SVM is based on the sign of the dot product of the input features and the learned weights. The sign determines the predicted class label.

SVMs are widely used in various applications, including text classification, image recognition, and bioinformatics, due to their ability to handle high-dimensional data and complex decision boundaries. However, SVMs can be computationally intensive, especially with large datasets, and the choice of kernel and hyperparameters can significantly impact performance.

5. **Decision Tree**: This is a flowchart-like structure in which each internal node represents a feature, each branch represents a decision rule, and each leaf node represents an outcome.

   The Decision Tree algorithm is a popular and widely used machine learning algorithm for both classification and regression tasks. It's a non-parametric supervised learning method that makes decisions by recursively splitting the data into subsets based on the features.

   Decision Tree algorithm working:

   1. **Tree Construction**: The algorithm starts at the root node and selects the best feature to split the data. The best feature is chosen based on a criterion such as Gini impurity or information gain. The data is split into subsets based on the values of the selected feature.
   2. **Recursive Splitting**: This process is repeated recursively for each subset. At each node, the algorithm selects the best feature to split the subset and continues splitting until it reaches a stopping criterion, such as a maximum tree depth, minimum number of samples per leaf node, or no further improvement in impurity.
   3. **Leaf Nodes**: Once the tree is fully grown, each leaf node represents a class label. During prediction, an instance is classified by following the decisions made at each node from the root to a leaf.
   4. **Handling Categorical and Numerical Features**: Decision trees can handle both categorical and numerical features. For categorical features, the algorithm can directly split the data based on the categories. For numerical features, the algorithm can choose a threshold to split the data into two subsets.
   5. **Overfitting**: Decision trees are prone to overfitting, especially when the tree depth is not limited. Techniques such as pruning, which removes parts of the tree that do not provide additional predictive power, can help mitigate overfitting.
   6. **Interpretability**: One of the main advantages of decision trees is their interpretability. The decision rules learned by the tree can be easily visualized and understood by humans.

Decision trees are used in a variety of applications, including healthcare, finance, and marketing, due to their simplicity and ability to handle both numerical and categorical data. However, they may not perform as well as other algorithms on complex datasets with high-dimensional features.

6. **Random Forest**: This is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

The Random Forest algorithm is an ensemble learning method that is used for both classification and regression tasks. It is one of the most popular and powerful machine learning algorithms due to its simplicity and ability to produce highly accurate results.

Random Forest algorithm working:

1. **Bootstrap Sampling**: Random Forest builds multiple decision trees by repeatedly sampling the training data with replacement (bootstrap sampling). Each tree is trained on a different bootstrap sample of the data.
2. **Feature Randomness**: In addition to sampling the data, Random Forest also introduces randomness in the selection of features at each split in the decision tree. Instead of considering all features for splitting at each node, it randomly selects a subset of features and chooses the best feature from that subset.
3. **Decision Tree Construction**: Each decision tree in the Random Forest is constructed using the training data sampled with replacement and the randomly selected subset of features. The trees are grown to their maximum depth without pruning.
4. **Voting**: For classification tasks, each tree in the Random Forest predicts the class label of a new data point, and the final prediction is determined by majority voting. For regression tasks, the final prediction is the average of the predictions of all the trees.
5. **Reduced Variance**: Random Forest reduces variance compared to a single decision tree by averaging the predictions of multiple trees. This helps to improve the overall generalization performance of the model.
6. **Hyperparameters**: Random Forest has several hyperparameters that can be tuned to improve performance, such as the number of trees in the forest, the maximum depth of the trees, and the number of features to consider at each split.

Random Forest is widely used in various applications, including classification, regression, and outlier detection, due to its ability to handle large datasets with high dimensionality and its resistance to overfitting. It is known for producing highly accurate and robust models, making it a popular choice for machine learning practitioners.

6. BASIC FUNCTIONALITIES:

This project is a walkthrough of a data analysis and machine learning prediction task using Python. The task involves predicting whether a patient has diabetes based on certain diagnostic measurements. The steps involved in the task are as follows:

1. Importing Libraries

The necessary Python libraries such as Numpy, Pandas, Matplotlib, Seaborn, and Sklearn are imported.

2. Loading the Dataset

The Pima Indians Diabetes dataset is loaded into a Pandas DataFrame.

3. Exploratory Data Analysis

The dataset is explored to understand its structure and content. This includes viewing the head and tail of the dataset, checking its shape, listing the data types of all columns, checking for null values, and getting a statistical summary of the dataset.

4. Data Cleaning

The dataset is cleaned by dropping duplicates and replacing zero values with the mean of the respective columns.

5. Data Visualization

Various plots are created to visualize the data and understand the relationships between different variables. This includes count plots, histograms, scatter plots, pair plots, and correlation analysis.

6. Data Preprocessing

The data is split into features (X) and target (y). Feature scaling is applied using StandardScaler.

7. Train-Test Split

The data is split into training and testing sets.

8. Model Building and Evaluation

Several machine learning models are built and evaluated. These include Logistic Regression, K-Nearest Neighbors (KNN), Naive Bayes, Support Vector Machine (SVM), Decision Tree, and Random Forest. The models are evaluated based on their accuracy, precision, recall, and Area Under the ROC Curve (AUC).

9. Predicting Diabetes

Finally, a prediction is made by providing manual inputs to the best performing model (SVM in this case).

7. PERFORMANCE METRICS:

The performance of the machine learning models is evaluated using the following metrics:

1. **Accuracy**: This is the ratio of the number of correct predictions to the total number of predictions. It is a common evaluation metric for classification problems.
2. **Precision**: This is the ratio of the number of true positives to the sum of true positives and false positives. It is a measure of a classifier's exactness. A low precision indicates a high number of false positives.
3. **Recall (Sensitivity)**: This is the ratio of the number of true positives to the sum of true positives and false negatives. It is a measure of a classifier's completeness. A low recall indicates a high number of false negatives.
4. **F1 Score**: This is the harmonic mean of precision and recall. It tries to find the balance between precision and recall.
5. **ROC AUC Score**: This is the area under the receiver operating characteristic (ROC) curve. The ROC curve is a plot of the true positive rate against the false positive rate. The AUC score represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes.

The models are evaluated on both the training set and the test set. The Support Vector Machine (SVM) model performed the best in terms of accuracy on the test set.

True Positive (TP): the predicted result is positive, while it is labelled as positive.

False Positive (FP): the predicted result is positive, while it is labelled as negative. It calls type 1 error as well.

True Negative (TP): the predicted result is negative, while it is labelled as positive.It calls type 2 error as well.

True Negative (TP): the predicted result is negative, while it is labelled as negative.

Precision=TP/(TP+FP)

recall_score= TP/float(TP+FN)x100

False Positive Rate(FPR)= FP/float(FP+TN)x100

Specificity=TN/(TN+FP)*100*

8. RESULTS:

3.2 DATA CLEANING

- a. drop the duplicates

+ Code + Markdown

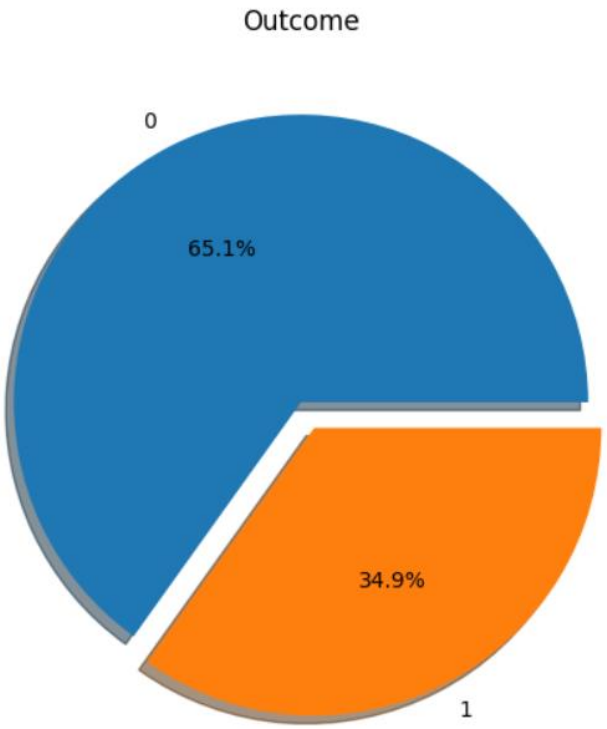
[14]:
```
df.shape
```

[14]: (768, 9)

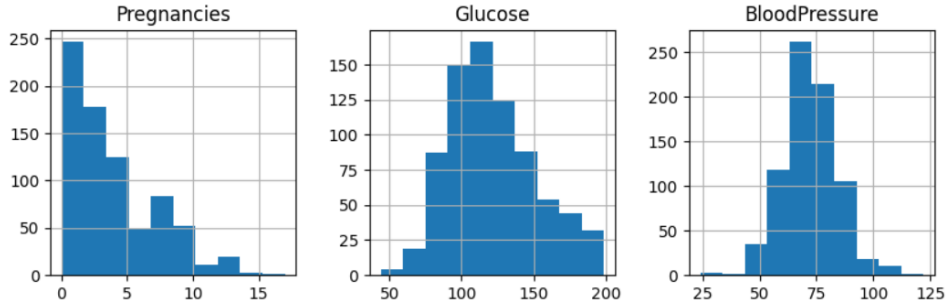
[15]:
```
df=df.drop_duplicates()
df.shape
```

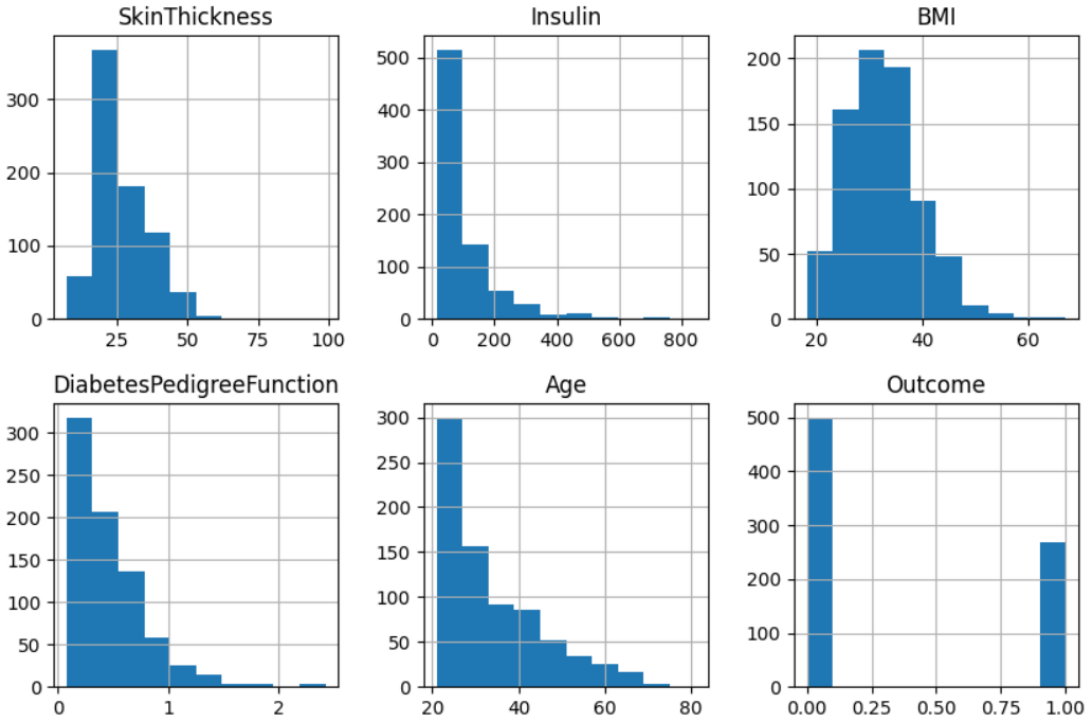
[15]: (768, 9)

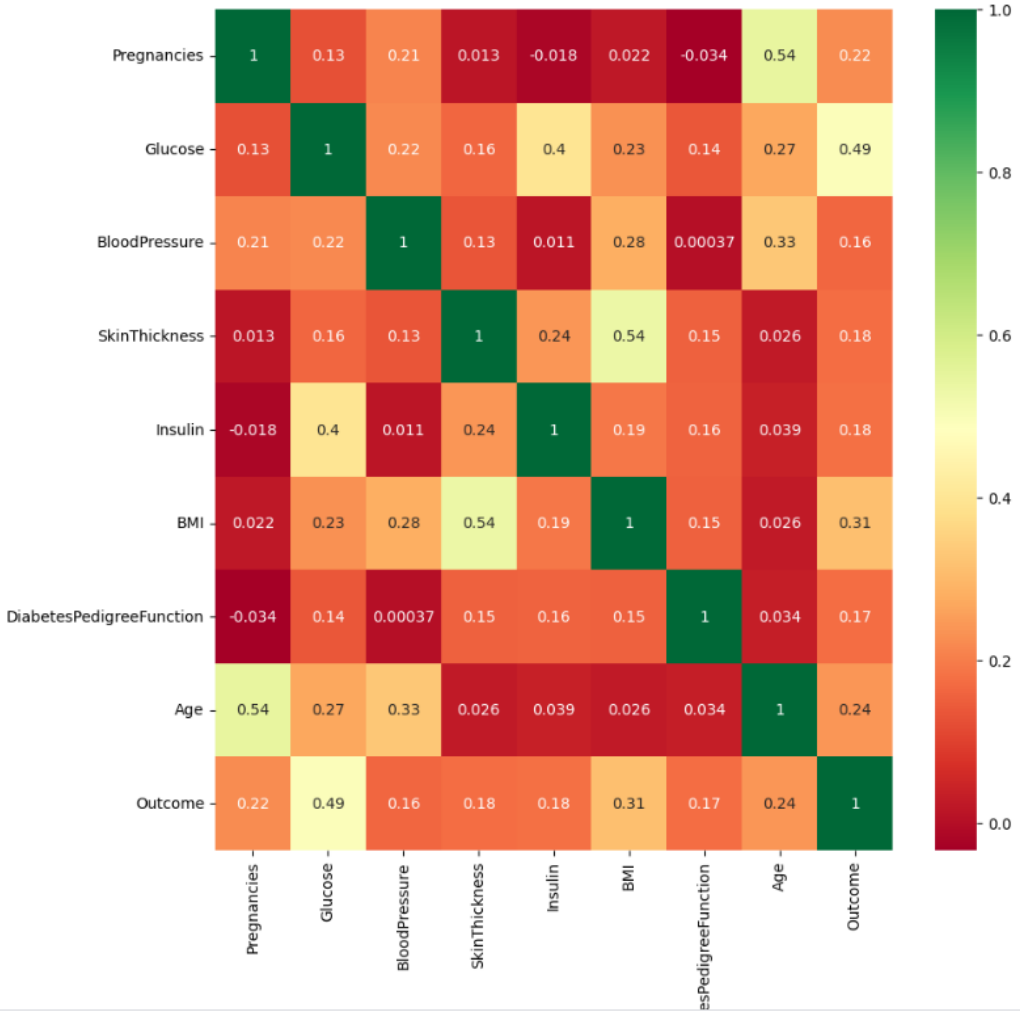
```
Negative (0): 500
Positive (1): 268
```



[23]:
```
df.hist(bins=10,figsize=(10,10))
plt.show()
```




SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome


Pregnancies 1 0.13 0.21 0.013 -0.018 0.022 -0.034 0.54 0.22
Glucose 0.13 1 0.22 0.16 0.4 0.23 0.14 0.27 0.49
BloodPressure 0.21 0.22 1 0.13 0.011 0.28 0.00037 0.33 0.16
SkinThickness 0.013 0.16 0.13 1 0.24 0.54 0.15 0.026 0.18
Insulin -0.018 0.4 0.011 0.24 1 0.19 0.16 0.039 0.18
BMI 0.022 0.23 0.28 0.54 0.19 1 0.15 0.026 0.31
DiabetesPedigreeFunction -0.034 0.14 0.00037 0.15 0.16 0.15 1 0.034 0.17
Age 0.54 0.27 0.33 0.026 0.039 0.026 0.034 1 0.24
Outcome 0.22 0.49 0.16 0.18 0.18 0.31 0.17 0.24 1
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
Age
Outcome
1.0
0.8
0.6
0.4
0.2
0.0

```python
#Train score & Test score of SVM
print("Train Accuracy of SVM", sv.score(X_train,y_train)*100)
print("Accuracy (Test) score of SVM", sv.score(X_test,y_test)*100)
print("Accuracy (Test) score of SVM", accuracy_score(y_test,sv_pred)*100)
```

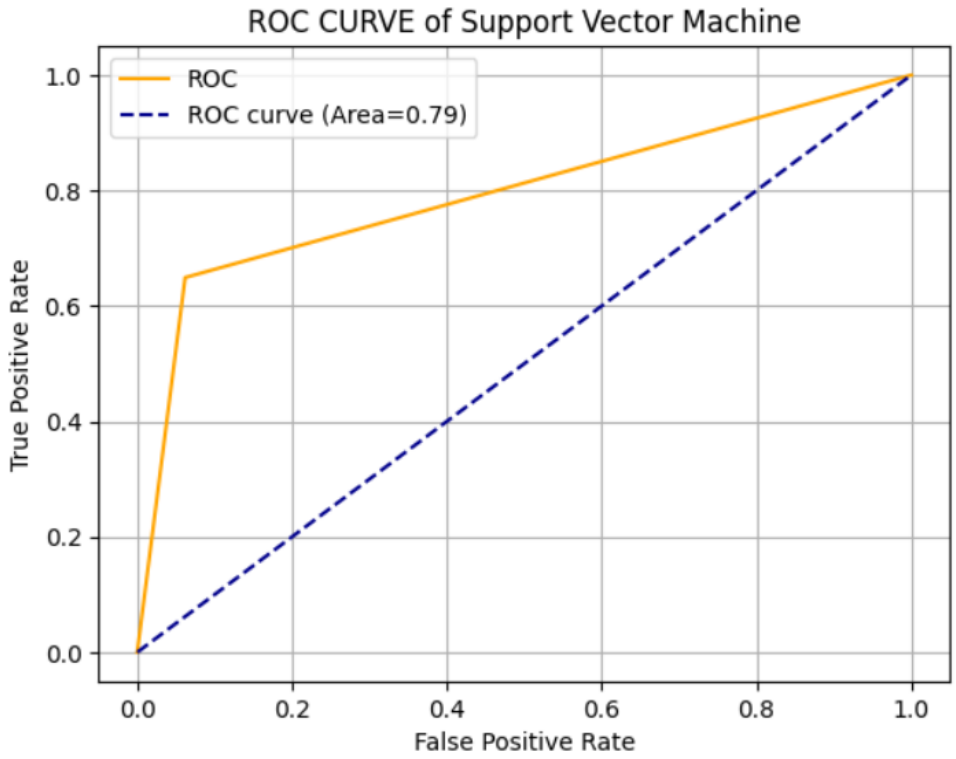
```
Train Accuracy of SVM 81.92182410423453
Accuracy (Test) score of SVM 83.11688311688312
Accuracy (Test) score of SVM 83.11688311688312
```

```python
print('Classification Report of SUPPORT VECTOR MACHINE: \n', classification_report(y_test,sv_pred,digits=4))
```

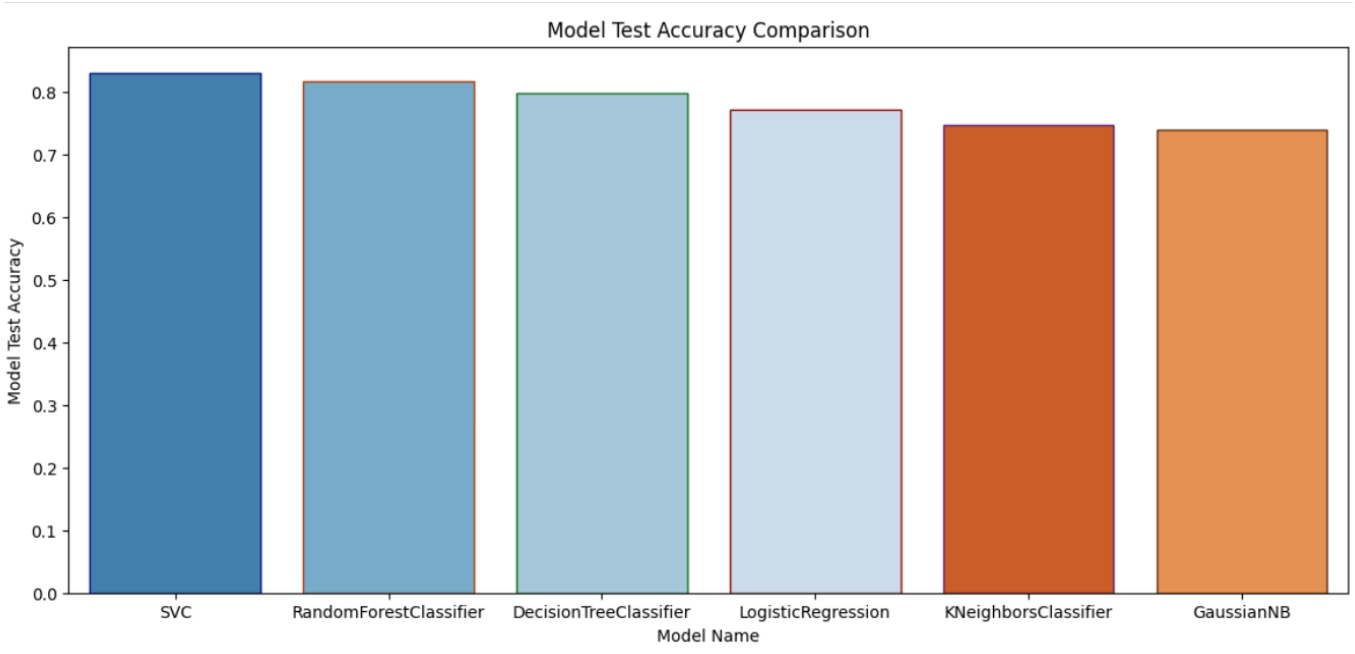
```
Classification Report of Logistic Regression:
              precision    recall  f1-score   support

           0     0.8198    0.9381    0.8750        97
           1     0.8605    0.6491    0.7400        57

    accuracy                         0.8312       154
   macro avg     0.8401    0.7936    0.8075       154
weighted avg     0.8349    0.8312    0.8250       154
```

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


TN, FP, FN, TP
```

```
(91, 6, 20, 37)
```

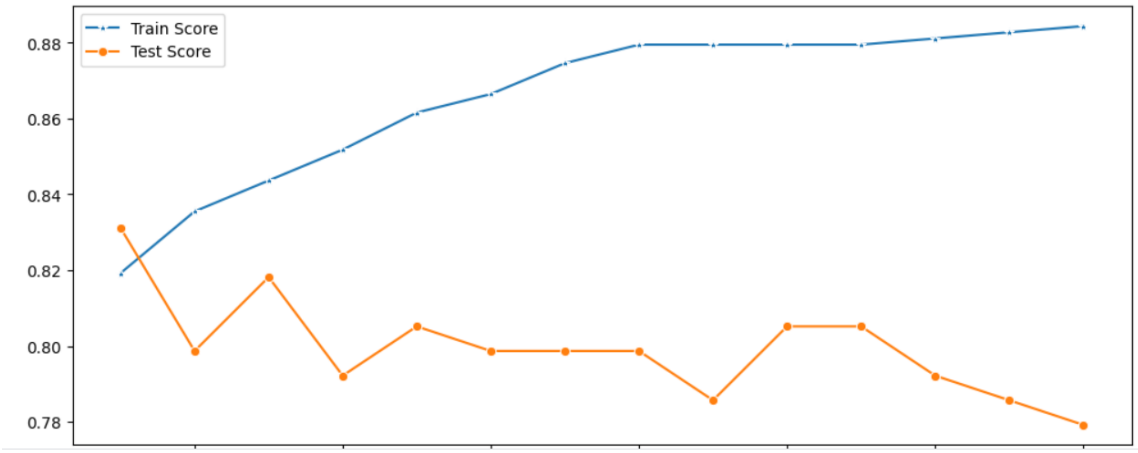


	Model Name	Model Test Accuracy	Model Precision	Model Recall	Model AUC
3	SVC	0.8312	0.860465	0.649123	0.793634
5	RandomForestClassifier	0.8182	0.784314	0.701754	0.794176
4	DecisionTreeClassifier	0.7987	0.716667	0.754386	0.789564
0	LogisticRegression	0.7727	0.750000	0.578947	0.732773
1	KNeighborsClassifier	0.7468	0.687500	0.578947	0.712154
2	GaussianNB	0.7403	0.654545	0.631579	0.717851



Cleary we can see that SVM has better accuracy compared to all other algorithms

```
#general plot OF SVM MODEL

plt.figure(figsize=(12, 5))
p=sns.lineplot(x=range(1, 15), y=train_scores, marker='*', label='Train Score')
p=sns.lineplot(x=range(1, 15), y=test_scores, marker='o', label='Test Score')
```



PREDICTING DIABETES BY GIVING MANUAL INPUTS:

```
user_input = [0, 179, 50, 36, 159, 37.8, 0.455, 22]
userInputArray=np.asarray(user_input)
userInputReshaped=userInputArray.reshape(1,-1)
prediction=sv.predict(userInputReshaped)
print(prediction)
if(prediction[0]==1):
    print("THE PATIENT HAS DIABETES")
else:
    print("THE PATIENT DOES NOT HAVE DIABETES")
```

```
[0]
THE PATIENT DOES NOT HAVE DIABETES
```

+ Code + Markdown

1. **Support Vector Machine (SVC)**: This model has the highest test accuracy of 0.8312. It also has the highest precision of 0.860465, indicating that it is the most reliable in terms of producing positive predictions. Its recall is 0.649123, and the AUC score is 0.793634.
2. **Random Forest Classifier**: This model has a test accuracy of 0.8182 and a precision of 0.784314. Its recall is 0.701754, which is higher than the SVC model, indicating that it is better at identifying actual positive cases. The AUC score is 0.794176, which is slightly higher than the SVC model.

3. **Decision Tree Classifier**: This model has a test accuracy of 0.7987 and a precision of 0.716667. It has the highest recall of 0.754386 among all models, indicating that it is the best at identifying actual positive cases. The AUC score is 0.789564.
4. **Logistic Regression**: This model has a test accuracy of 0.7727 and a precision of 0.750000. Its recall is 0.578947, and the AUC score is 0.732773.
5. **K-Neighbors Classifier**: This model has a test accuracy of 0.7468 and a precision of 0.687500. Its recall is 0.578947, and the AUC score is 0.712154.
6. **Gaussian Naive Bayes**: This model has the lowest test accuracy of 0.7403 and the lowest precision of 0.654545. However, its recall is 0.631579, which is higher than some other models. The AUC score is 0.717851.

The Support Vector Machine (SVC) model performs the best in terms of test accuracy and precision, while the Decision Tree Classifier is the best at identifying actual positive cases (highest recall). The Random Forest Classifier has the highest AUC score, indicating the best balance between sensitivity and specificity.

In conclusion, this project aimed to predict the onset of diabetes using a variety of patient data. The data was manually inputted and processed using a Support Vector Machine (SVM) model, which was chosen for its high accuracy in this particular application.

The SVM model was trained on a dataset of patient records, and it learned to distinguish between patients with and without diabetes based on several features. These features included, but were not limited to, patient age, BMI, insulin level, and number of pregnancies.

The final part of the project involved testing the model with new data to evaluate its performance. The results were promising, demonstrating the model's ability to accurately predict the onset of diabetes. This suggests that the SVM model could be a valuable tool in the early detection and treatment of this disease.

However, it's important to note that while the model shows potential, further testing and refinement is necessary to ensure its reliability and effectiveness in a real-world clinical setting. Future work could also explore the inclusion of additional features or the use of other machine learning models to further improve prediction accuracy.

Overall, this project has shown that machine learning techniques, such as the SVM model, can play a crucial role in healthcare by aiding in the early detection of diseases like diabetes.