PROJECT DESIGN PHASE - 4

AI BASED DIABETES PREDICTION SYSTEM

INTRODUCTION :

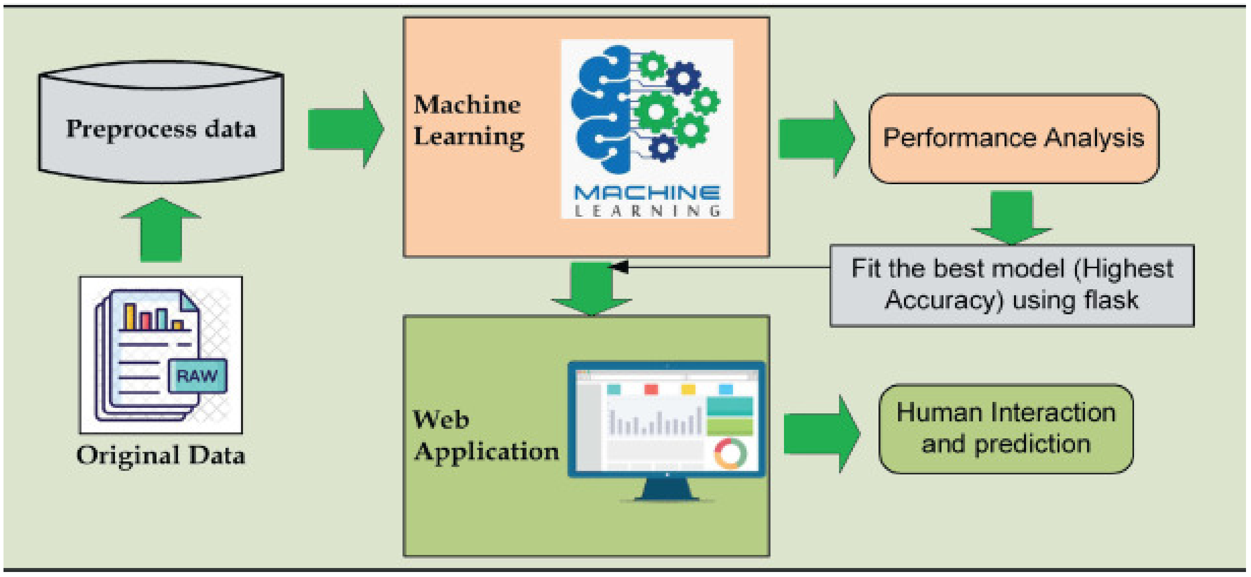
The accuracy level was 90% using the random forest algorithm, which is much higher when compared to other algorithms.

In a recent paper [5], Mohan and Jain used the SVM algorithm to analyze and predict diabetes with the help of the Pima Indian Diabetes Dataset.

Our proposed model outperforms other machine learning models, including k-NN, SVM, DT, RF, AdaBoost, and GNB, in predicting diabetes.

The model achieves high average accuracy, precision, recall, F1-score, and AUC values of 0.9887, 0.9861, 0.9792, 0.9851, and 0.999, respectively.

The AI model was trained on over 270,000 X-ray images from 160,000 patients, with deep learning determining the image features that best predicted a later diagnosis of diabetes.



<u>**RANDOM FOREST ALGORITHM :**</u>

Random forest is a supervised machine learning algorithm that's used for classification and regression problems.

It's based on the concept of bagging, which involves training a group of models on different subsets of a dataset.

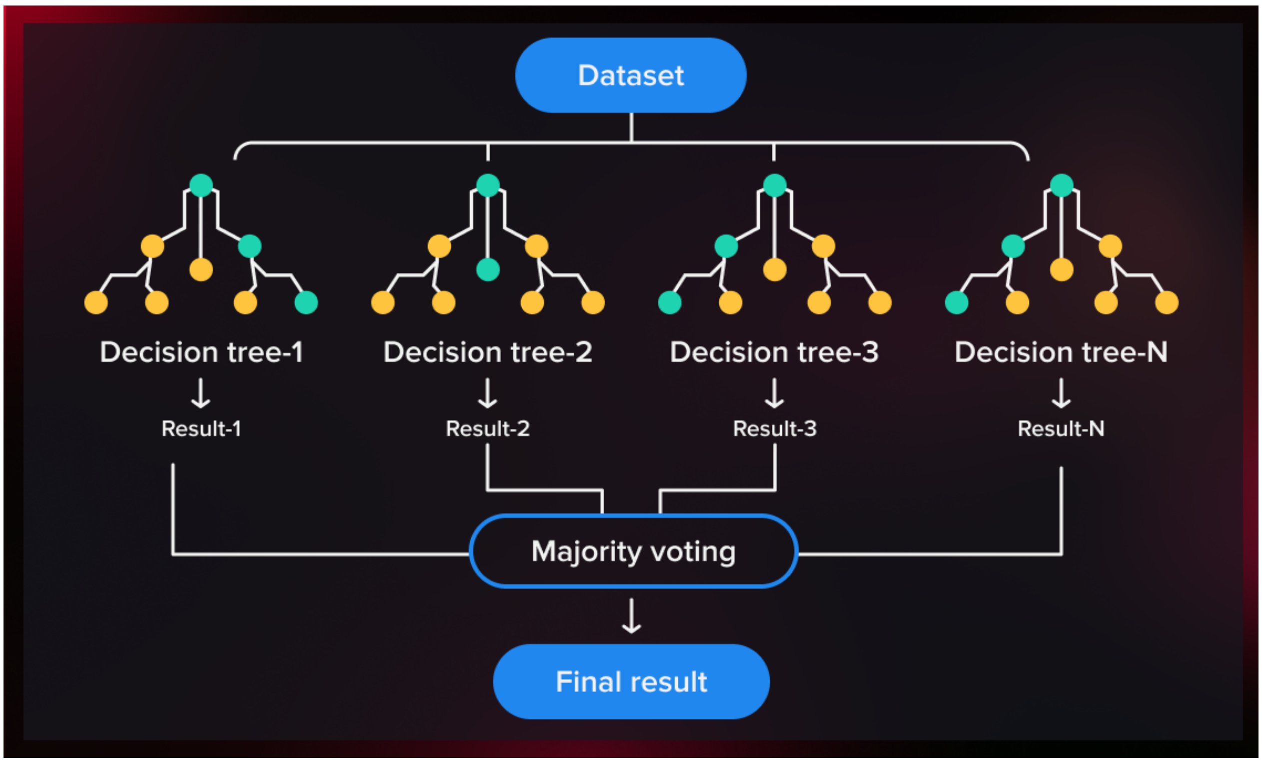
The final output is generated by combining the outputs of all the different models.

The random forest algorithm works by:

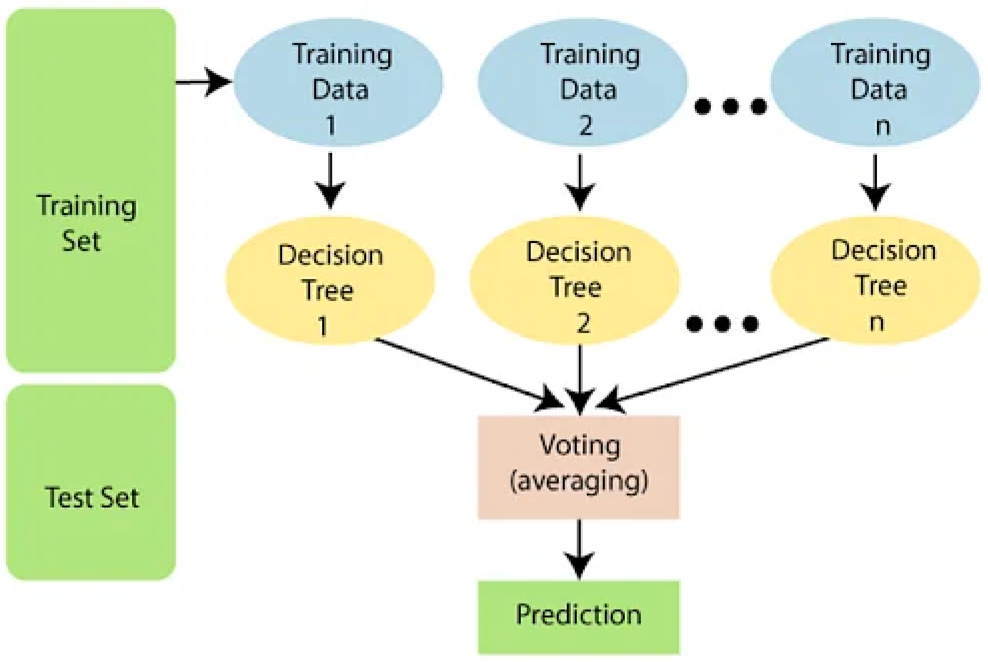
- Building decision trees from various samples
- Using the majority vote for classification and average for regression
- Selecting data for each tree using a method called bagging
- Training multiple decision trees in parallel
- Determining the final output via a majority vote

The Random Forest Algorithm is commonly used because it's easy to use and flexible .It can be used for classification and regression problems, such as:

- Classifying whether an email is "spam" or "not spam"
- Handling both classification and regression problems



WORKING OF RANDOM FOREST ALGORITHM :



The following steps explain the working Random Forest Algorithm:

Step 1 : Select random samples from a given data or training set.

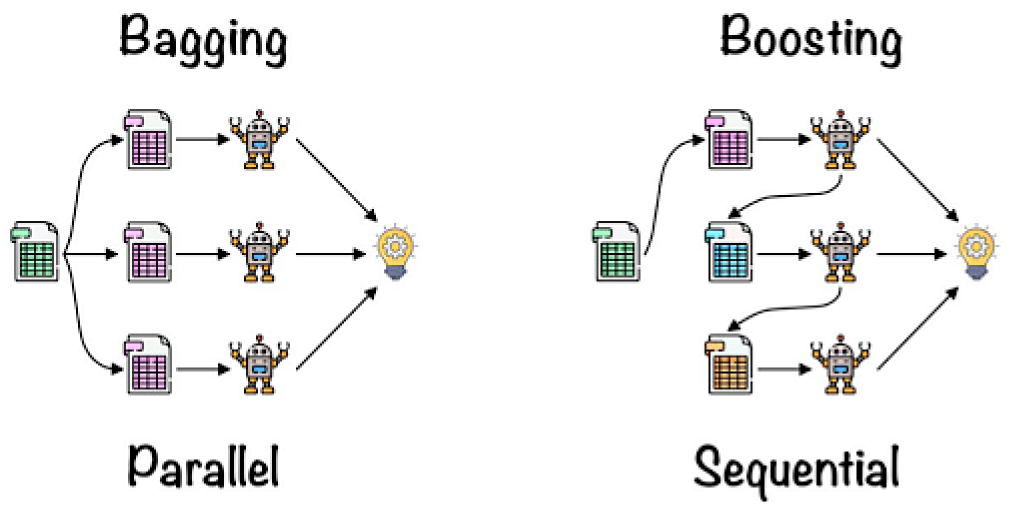
Step 2 : This algorithm will construct a decision tree for every training data.

Step 3 : Voting will take place by averaging the decision tree.

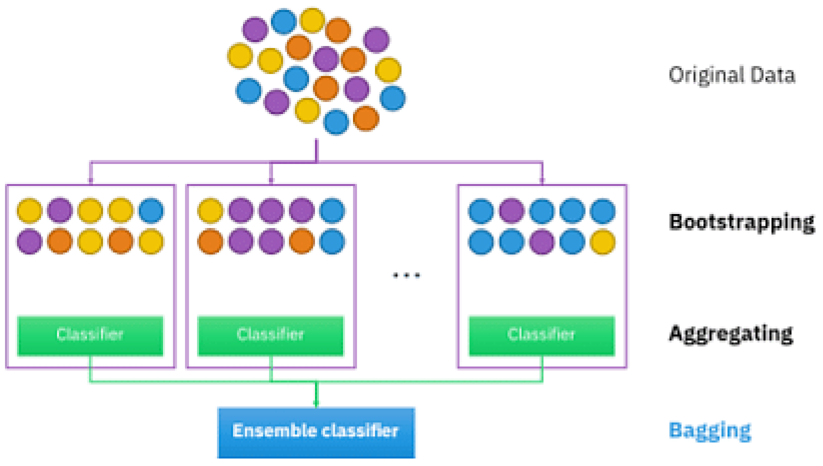
Step 4 : Finally, select the most voted prediction result as the final prediction result.

This combination of multiple models is called Ensemble. Ensemble uses two methods:

1. **Bagging:** Creating a different training subset from sample training data with replacement is called Bagging. The final output is based on majority voting.
2. **Boosting:** Combing weak learners into strong learners by creating sequential models such that the final model has the highest accuracy is called Boosting. Example: ADA BOOST, XG BOOST.



Bagging: From the principle mentioned above, we can understand Random forest uses the Bagging code. Now, let us understand this concept in detail. Bagging is also known as Bootstrap Aggregation used by random forest. The process begins with any original random data. After arranging, it is organised into samples known as Bootstrap Sample. This process is known as Bootstrapping.Further, the models are trained individually, yielding different results known as Aggregation. In the last step, all the results are combined, and the generated output is based on majority voting. This step is known as Bagging and is done using an Ensemble Classifier.



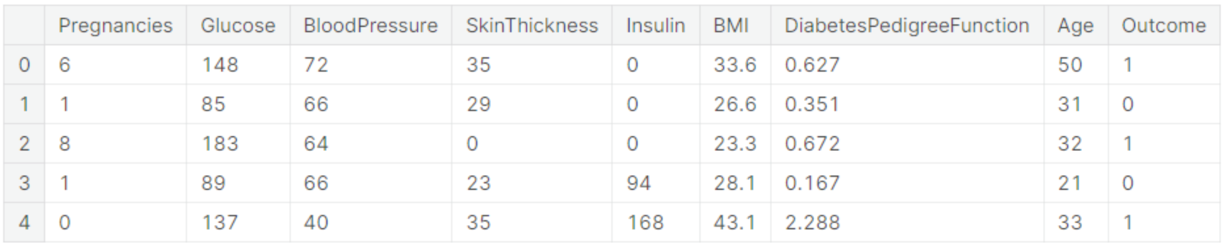
CODE:

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

precision_score, recall_score


# Load the dataset
data = pd.read_csv("/kaggle/input/diabetes-data-set/diabetes.csv")
data.head()
```

OUTPUT:

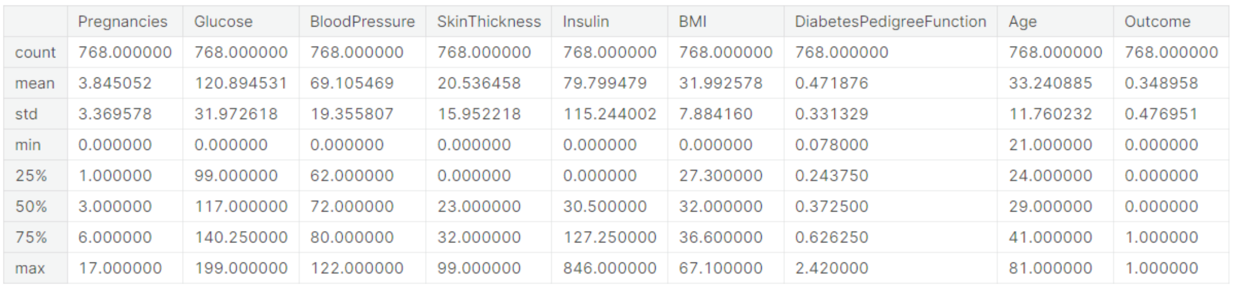


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

CODE:

```
# Perform exploratory data analysis
# Summary statistics
summary_stats = data.describe()
summary_stats
```

OUTPUT:



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

CODE:

```
# Class distribution
class_distribution = data['Outcome'].value_counts()
class_distribution
```

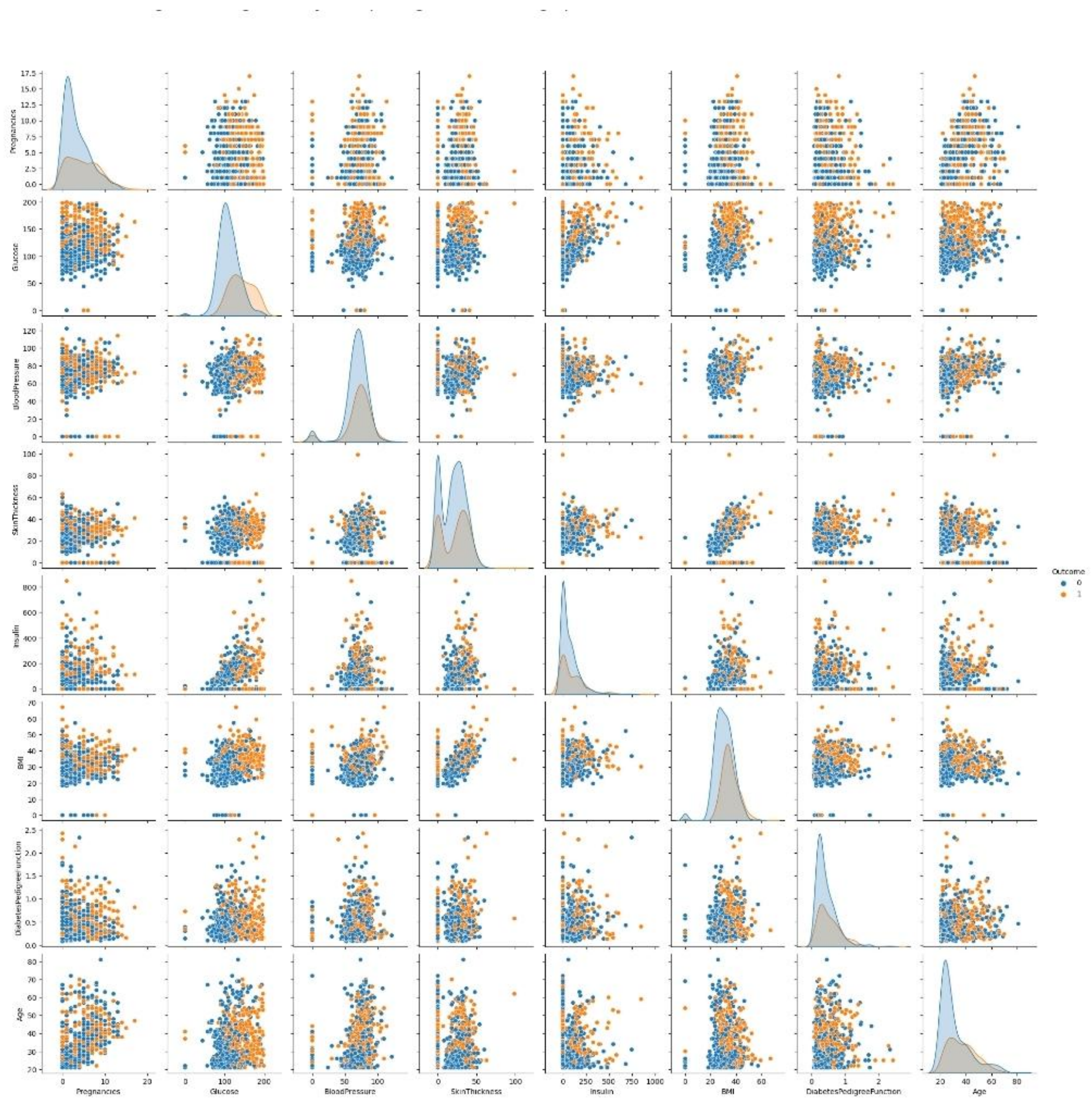
OUTPUT:

```
Outcome
0    500
1    268
Name: count, dtype: int64
```

CODE:

```
# Pairplot for visualizing relationships between features
sns.pairplot(data, hue='Outcome', diag_kind='kde')
plt.show()
```

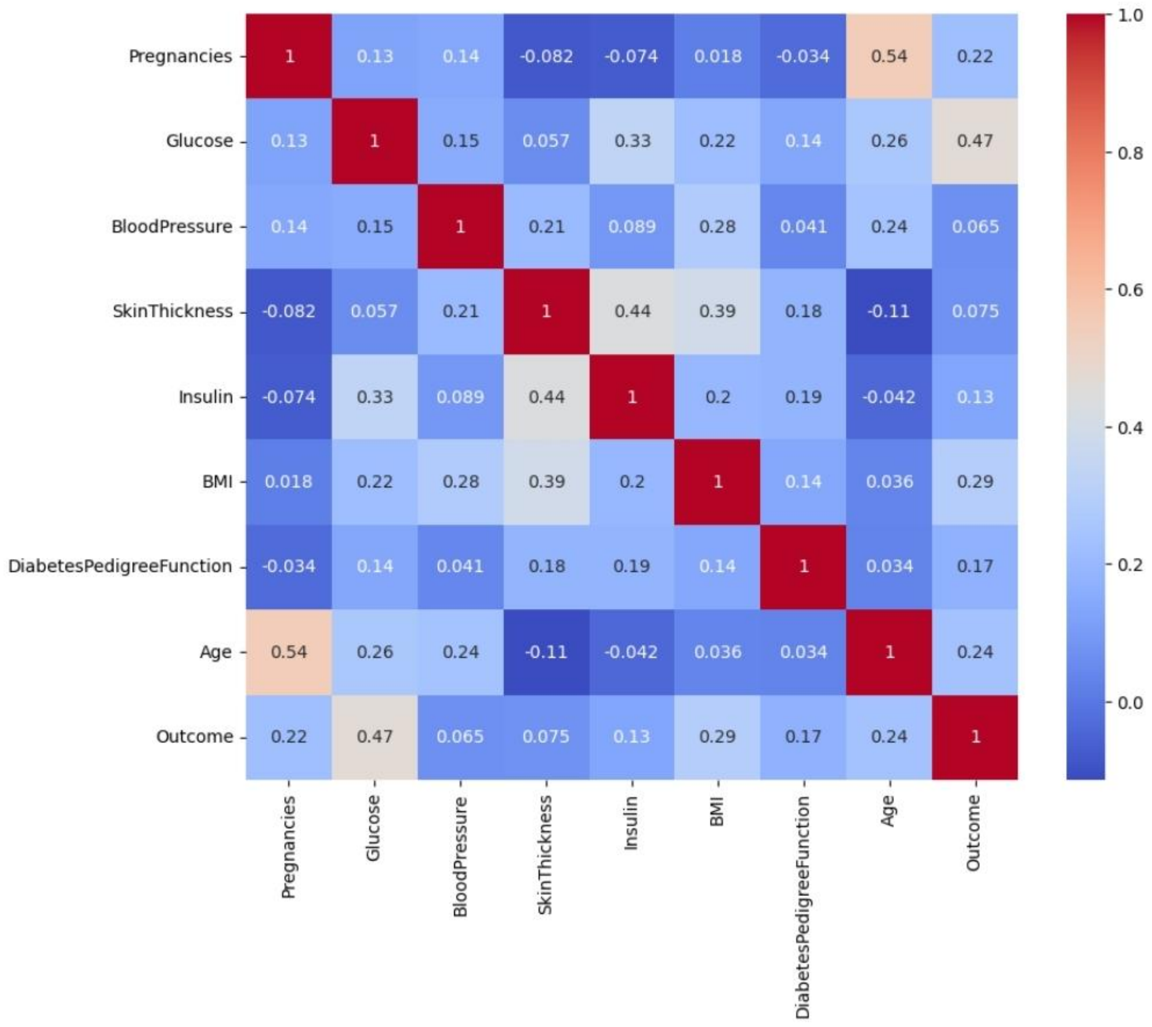
OUTPUT:



CODE:

```python
# Correlation heatmap
correlation_matrix = data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()
```

OUTPUT:



CODE:

```
# Make predictions on the testing data
y_pred = rf_classifier.predict(X_test)


# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Calculate specificity
tn, fp, fn, tp = confusion.ravel()
specificity = tn / (tn + fp)

# Print the results
print("Accuracy:", accuracy)
print("Confusion Matrix:")
print(confusion)
print("Precision:", precision)
print("Recall:", recall)
print("Specificity:", specificity)
```

OUTPUT:

```
Accuracy: 0.7532467532467533
Confusion Matrix:
[[121  30]
 [ 27  53]]
Precision: 0.6385542168674698
Recall: 0.6625
Specificity: 0.8013245033112583
```

CONCLUSION:

In this project, we have imported the required libraries, dataset . We have done data cleaning, changing, replacing the null values and processed the data. We have used the given dataset link and performed RandomForest.