# ARTIFICIALINTELLIGENCE

AI – BASED DIABETES PREDICTION SYSTEMS

**Phase 4**: Development part 02.

**Topic:** Building the project by selecting the machine learning algorithm and evaluating it's performance.

**Team Leader :**

Pandeeswaran C.K( pandeeswaranckit21@jkkmct.edu.in)

**Team Members :**

Krishnan.S( krishnansit21@jkkmct.edu.in)

Parthiban.M( parthibanmit21@jkkmct.edu.in)

Dinesh.M (dineshmit21@jkkmct.edu.in)

Naveen.S (naveensit21@jkkmct.edu.in)

# Diabetes Predictionusing MLAlgorithm

## RANDOM FOREST CLASSIFIER :-

Random forest is a commonly-used machine learning algorithm trademarked by Leo Breiman and Adele Cutler, which combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems.

## WHY RANDOM FOREST IS USED IN ML :-

- High accuracy
- Robustness
- Feature importance
- Versatility
- Scalability

## Necessary Steps :-

1.Selecting a machine learning algorithm.

2.Trainning the model.

3.Evaluating its performance.

### 1.Selecting a machine learning algorithm

- Random Forest Classifier ( )

**PROGRAM:-**

```
import pandas as pd
importnumpy as np
importmatplotlib.pyplot as plt
importseaborn as sns
fromsklearn.model_selection import train_test_split
fromsklearn.ensemble import RandomForestClassifier
```

fromsklearn.metrics import accuracy_score,confusion_matrx,precision_score,recall_score

#Load the dataset

data=pd.read_csv (“diabetes.csv”)

data.head ( )

**OUTPUT:-**

```
In [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.ensemble import RandomForestClassifier
        from sklearn.metrics import accuracy_score,confusion_matrix,precision_score,recall_score
        data=pd.read_csv("diabetes.csv")
        data.head()

        Matplotlib is building the font cache; this may take a moment.
```

Out[1]:

| | 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 |

#summary statistics

Summary_stats = data.describe ( )

Summary_stats

**OUTPUT:-**

```
In [2]: summary_stats=data.describe()
        summary_stats
```

Out[2]:

| | 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 |

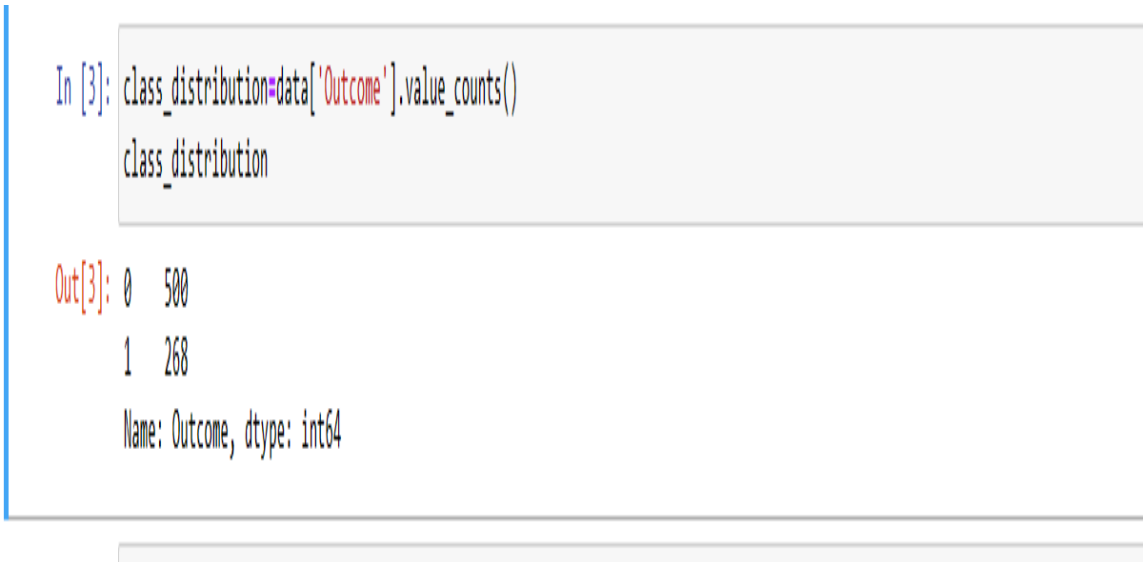
---

#class distribution

class_distribution = data [‘Outcome’].value_counts ( )

class_distribution

**OUTPUT:-**



```
In [3]: class_distribution=data['Outcome'].value_counts()
        class_distribution

Out[3]: 0   500
        1   268
        Name: Outcome, dtype: int64
```

#pairplot for visualizing relationships between features

Sns.pairplot ( data,hue='Outcome',diag_kind='kde' )

Plt.show ( )

## OUTPUT:-

```
1    268
Name: Outcome, dtype: int64
```

```
In [4]: sns.pairplot(data,hue='Outcome',diag_kind='kde')
plt.show()
```

#correlation heatmap

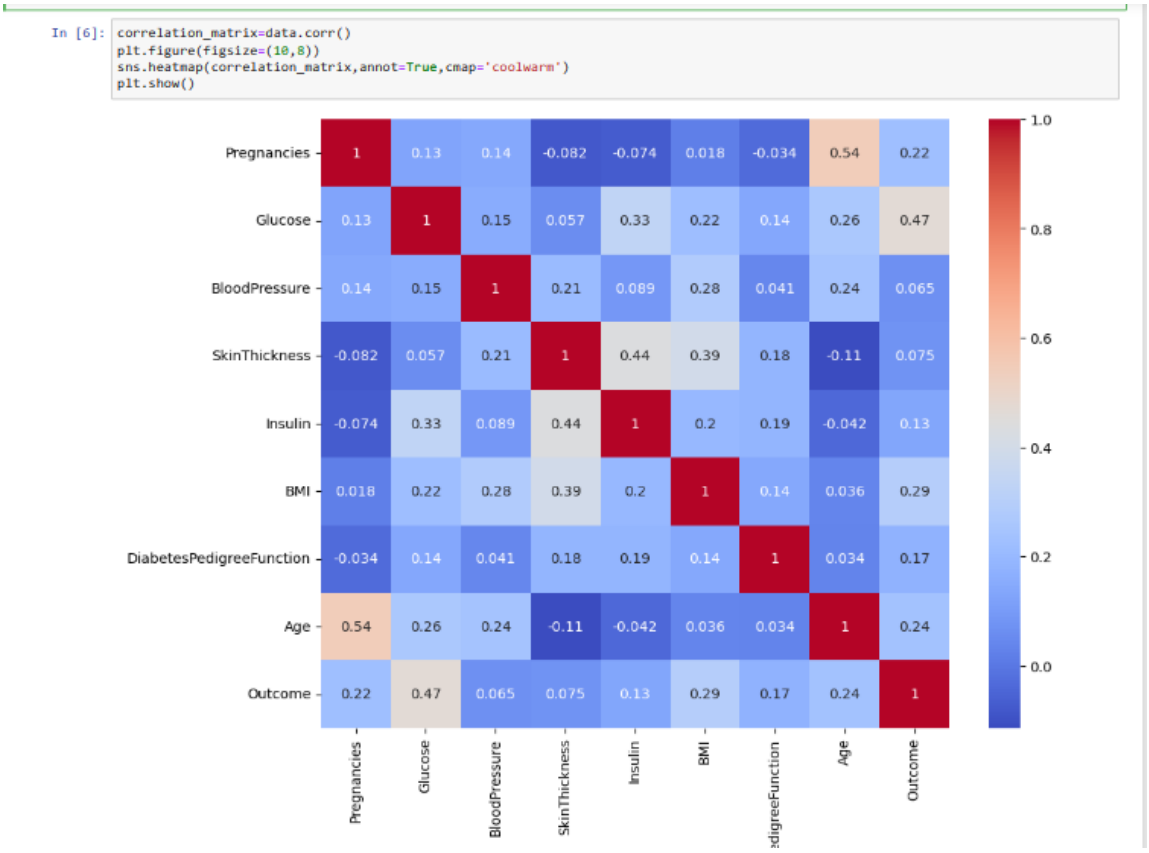
correlation_matrix = data.corr ( )

plt.figure( figsize=( 10,8 ) )

sns.heatmap( correlation_matrix,annot=True, cmap= ‘coolwam’ )

plt.show ( )

**OUTPUT:-**



#split the data into feature and target variables

X = data.drop( “Outcome” ,axis=1 )

y = data [‘Outcome’]

**2.Training the Model**

#split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.3,random_state=42 )

#create a Random Forest Classifier

rf_classifier= RandomForestClassifier ( n_estimators=100,random_state=42)

#train the classifier on the training data

rf_classifier.fit( X_train, y_train )

**OUTPUT:-**

```
In [12]: X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=42
         rf_classifier=RandomForestClassifier(n_estimators=100,random_state=42)

In [15]: rf_classifier.fit(X_train,y_train)

Out[15]: RandomForestClassifier(random_state=42)
```

#make predictions on the testing data

y_pred = rf_classifier.predict( X_test )

**3.Evaluate its performance**

#evaluate the model's performance

accuracy = accuracy_score( y_test,y_pred )

confusion = confusion_matrix ( y_test,y_pred )

```
precission = precission_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:-**

```
In [19]: print("Accuracy:",accuracy)
         print("Confusion Matrix:")
         print(confusion)
         print("Precision:",precision)
         print("Recall:",recall)
         print("Specificity:",specificity)

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

## Advantages:-

- It reduces overfitting in decision trees and helps to improve the accuracy
- It is flexible to both classification and regression problems
- It works well with both categorical and continuous values
- It automates missing values present in the data
- Normalising of data is not required as it uses a rule-based approach.

## Disadvantages:-

- It requires much computational power as well as resources as it builds numerous trees to combine their outputs.
- It also requires much time for training as it combines a lot of decision trees to determine the class.
- Due to the ensemble of decision trees, it also suffers interpretability and fails to determine the significance of each variable.

## Random Forest Architecture