Breast Cancer Wisconsin (Diagnostic) Data Set are given.It is to Predict whether the cancer is benign or malignant from the given dataset by using different model

Data preprocessing

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sb
```

Importing the dataset

```
dataset = pd.read_csv('Breast Cancer Wisconsin (Diagnostic) Data Set.csv')
X = dataset.iloc[:, 2:31].values
y = dataset.iloc[:, 1].values
```

Analyzing the dataset

```
dataset.shape
```

```
(569, 33)
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
```

```
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

```
dataset.isnull().sum()
```

```
id                           0
diagnosis                    0
radius_mean                  0
texture_mean                 0
perimeter_mean               0
area_mean                    0
smoothness_mean              0
compactness_mean             0
concavity_mean               0
concave points_mean          0
symmetry_mean                0
fractal_dimension_mean       0
radius_se                    0
texture_se                   0
perimeter_se                 0
area_se                      0
smoothness_se                0
compactness_se               0
concavity_se                 0
concave points_se            0
symmetry_se                  0
fractal_dimension_se         0
radius_worst                 0
texture_worst                0
perimeter_worst              0
area_worst                   0
smoothness_worst             0
compactness_worst            0
concavity_worst              0
concave points_worst         0
symmetry_worst               0
fractal_dimension_worst      0
Unnamed: 32                569
dtype: int64
```

```
dataset.dtypes
```

```
id                           int64
diagnosis                   object
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             float64
concave points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              float64
compactness_se             float64
concavity_se               float64
concave points_se          float64
symmetry_se                float64
fractal_dimension_se       float64
radius_worst               float64
texture_worst              float64
perimeter_worst            float64
area_worst                 float64
smoothness_worst           float64
compactness_worst          float64
concavity_worst            float64
concave points_worst       float64
symmetry_worst             float64
fractal_dimension_worst    float64
Unnamed: 32                float64
dtype: object
```

```
dataset['diagnosis'].value_counts()
```

```
B    357
M    212
Name: diagnosis, dtype: int64
```

Removing the columns with all missing values

```
dataset=dataset.dropna(axis=1)
```

```
dataset.shape #for checking the number of column and rows
```

```
(569, 32)
```

Encoding Categorical Dataset

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
y=le.fit_transform(y)
```

```
print(y)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1
 0 1 0 1 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0
 0 0 0 0 0 0 1 1 1 0 1 1 0 0 0 1 1 0 1 0 1 1 0 1 1 0 0 1 0 0 1 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 1 1 1 0 1
 0 1 0 0 0 1 0 0 1 1 0 1 1 1 1 0 1 1 1 0 1 0 1 0 0 1 0 1 1 1 1 0 0 1 1 0 0
 0 1 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0
 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1
 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 0
 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 1 0 1 0 1 1
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0]
```

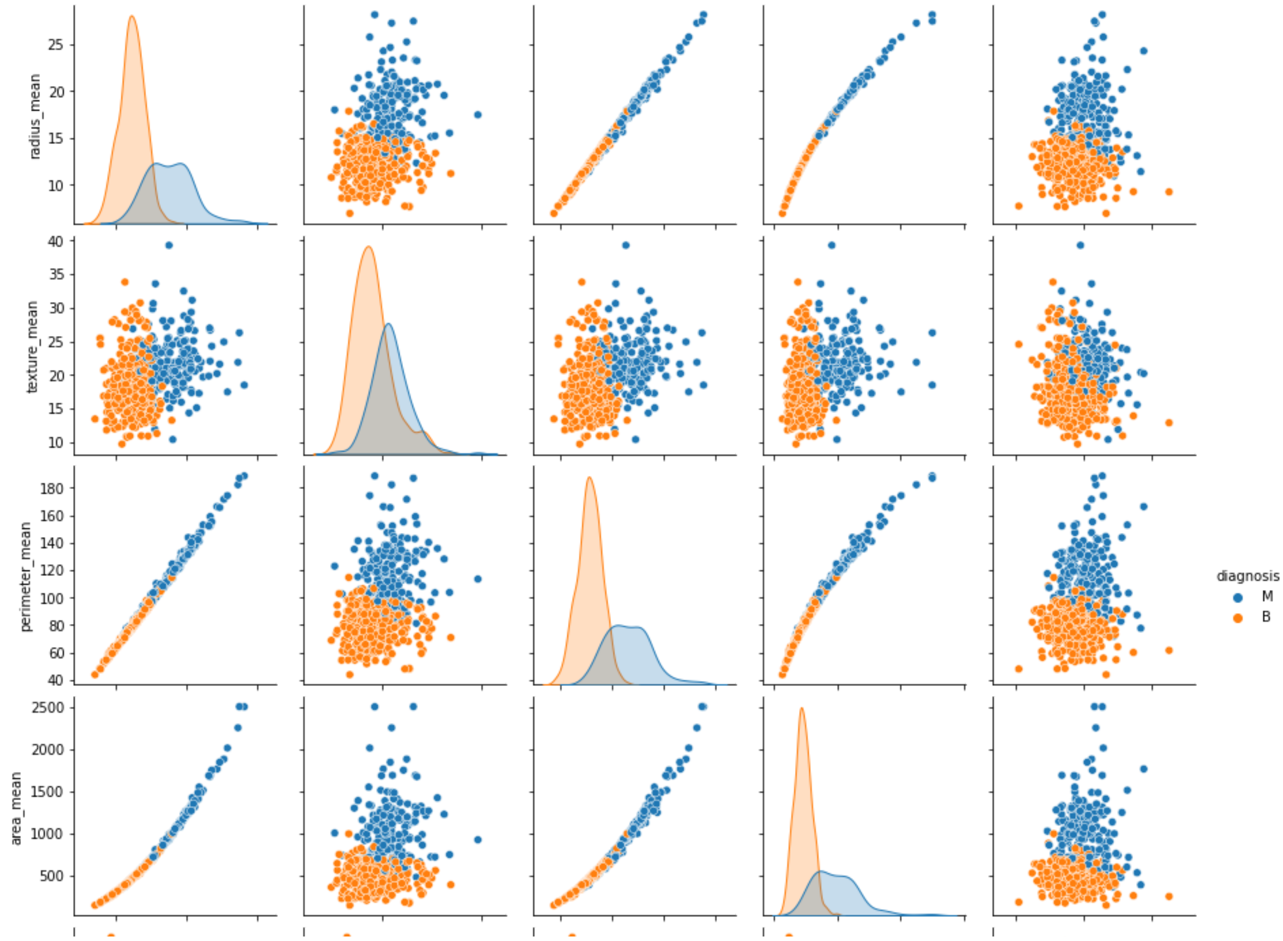
Splitting the dataset into training set and test set

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)
```

Paireise plots of the dataset

```
sb.pairplot(dataset.iloc[:,1:7],hue="diagnosis")
```

```
<seaborn.axisgrid.PairGrid at 0x7faf95d02210>
```



Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.fit_transform(X_test)
```

```
print(X_train)
```

```
[[-0.65079907 -0.43057322 -0.68024847 ... -0.69592933 -0.36433881
   0.32349851]
 [-0.82835341  0.15226547 -0.82773762 ... -1.29277423 -1.45036679
   0.62563098]
 [ 1.68277234  2.18977235  1.60009756 ...  0.26255563  0.72504581
  -0.51329768]
 ...
 [-1.33114223 -0.22172269 -1.3242844  ... -0.78274313 -0.98806491
  -0.69995543]
 [-1.25110186 -0.24600763 -1.28700242 ... -1.36015587 -1.75887319
  -1.56206114]
 [-0.74662205  1.14066273 -0.72203706 ...  0.47201917 -0.2860679
  -1.24094654]]
```

```
print(X_test)
```

```
[[-0.1839902   0.22170989 -0.11761404 ...  0.97465513  1.40089716
```

```
  1.16977773]
 [-0.23927557  1.20953909 -0.30776593 ... -0.59768168 -0.79588429
  -0.81775175]
 [-0.00358531 -0.79326895 -0.07782455 ... -0.92095006 -0.46102846
  -1.35426278]
 ...
 [-0.49242436 -1.50124802 -0.52388569 ... -0.42800809 -0.0848268
   0.34236625]
 [-0.14616337 -1.77900972 -0.14818913 ... -0.82451961 -0.58355147
  -0.35440132]
 [ 1.61714893 -0.27324893  1.6440133  ...  1.69566211  1.69773906
   1.27080903]]
```

Training the different models on the training dataset for classification of Benign(B) vs Malignant(M)

(1) Training the Random Forest classification Model on the training dataset

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 20,criterion='entropy', random_state =
classifier.fit(X_train, y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=20, random_state=0)
```

Predicting the test set result

```
y_pred=classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[0 0]
[0 0]
[0 0]
[1 1]
[1 1]
[0 0]
[1 1]
[0 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[0 0]
```

```
 [0 0]
 [1 1]
 [1 1]
 [0 0]
 [1 1]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [1 1]]
```

Confusion matrix for Random Forest and accuracy checking with goodness of fitting

```
from sklearn.metrics import confusion_matrix,accuracy_score,r2_score
cm=confusion_matrix(y_test,y_pred)
print(cm)
print(f"the accuracy of the prediction is :{accuracy_score(y_test,y_pred)}")
print(f"the r_square value is:{r2_score(y_test,y_pred)}")
```

```
[[87  3]
 [ 1 52]]
the accuracy of the prediction is :0.972027972027972
the r_square value is:0.880083857442348
```

(2) Training the Logistic Regression Model on training dataset

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
```

```
classifier.fit(X_train, y_train)
```

```
LogisticRegression(random_state=0)
```

Predicting the test set result

```
y_pred=classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[0 0]
[0 0]
[0 0]
[1 1]
[1 1]
[0 0]
[0 1]
[1 0]
[1 1]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[0 0]
[0 1]
[0 1]
[0 0]
[1 1]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[1 1]
[0 0]
[1 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
```

```
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [1 1]]
```

Confusion matrix for Logistic Regression and accuracy checking with goodness of fitting

```
from sklearn.metrics import confusion_matrix,accuracy_score,r2_score
cm=confusion_matrix(y_test,y_pred)
print(cm)
print(f"the accuracy of the prediction is :{accuracy_score(y_test,y_pred)}")
print(f"the r_square value is:{r2_score(y_test,y_pred)}")
```

```
[[86  4]
 [ 3 50]]
the accuracy of the prediction is :0.951048951048951
the r_square value is:0.790146750524109
```

(3) Training the Decision Tree Classification Model on training dataset

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy',random_state = 0)
classifier.fit(X_train, y_train)
```

```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

Predicting the test set result

```
y_pred=classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 0]
 [0 0]
 [0 0]
 [1 0]
```

```
 [1 0]
 [1 1]
 [0 0]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
```

Confusion matrix for Decision Tree Classification and accuracy checking with goodness of fitting

```
from sklearn.metrics import confusion_matrix,accuracy_score,r2_score
cm=confusion_matrix(y_test,y_pred)
print(cm)
print(f"the accuracy of the prediction is :{accuracy_score(y_test,y_pred)}")
print(f"the r_square value is:{r2_score(y_test,y_pred)}")
```

```
[[83  7]
 [ 2 51]]
```

```
the accuracy of the prediction is :0.9370629370629371
the r_square value is:0.730188679245283
```