CSE382, Data Mining and Business Intelligence

Project title: Prediction for type of breast cancer

Team Members:

Mohamed Ibrahim Elsayed Ibrahim Barakat	**20p8449**
Ahmed Hany Shaker Gheith	**19p3738**
Mohamed Ayman Abbas Zaki	**19p9201**
Donia Sameh	**20p3424**
Ibrahim Ahmed Hassan	**16p3062**

Contents

List of figures

1. Introduction

This project analyzes the dataset on breast cancer to make predictions for breast cancer diagnoses using a variety of machine learning approaches to extract insights from the dataset, it starts with some data preprocessing steps such as handling missing values, outlier detection, feature scaling, and visualization. It then proceeds to model training and evaluation. Also, K-Means clustering is used to explore potential cluster structures within the data.

2. Data preprocessing:

In this section we started by checking for missing values ensuring that the dataset is clean and generated descriptive statistics of the data frame to understand the distribution and range of values in the dataset. Then we removed the unnecessary columns to reduce dimensionality and focus the analysis on relevant features. Finally, encoding categorical values to convert them into numerical format.

Screenshots of some boxplots:

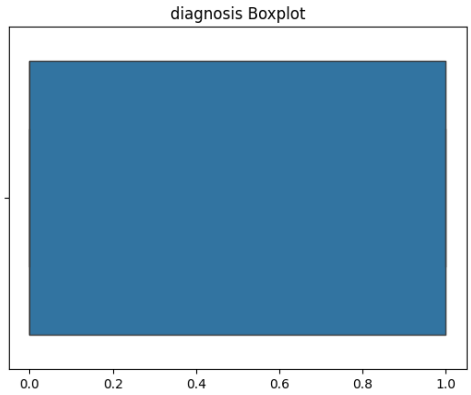

Figure 1 diagnosis boxplot

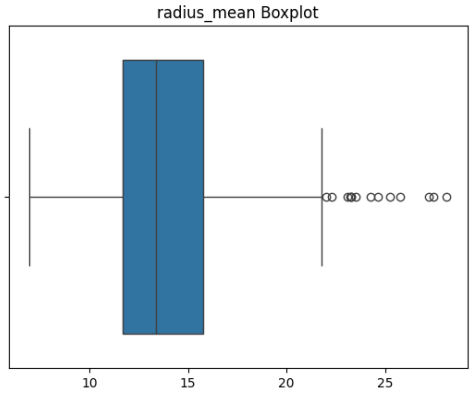

Figure 2 radius_mean boxplot

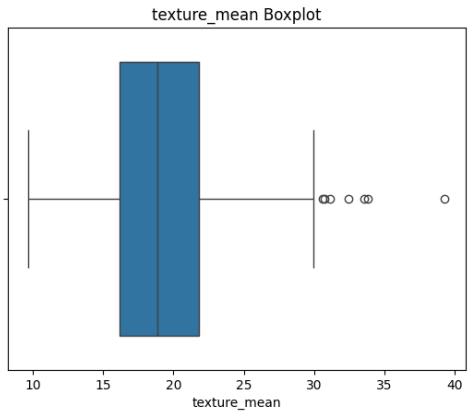

Figure 3 texture_mean boxplot

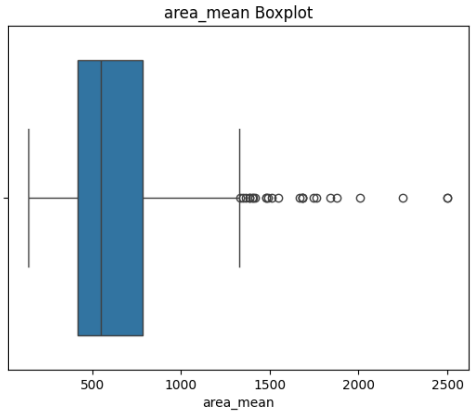

Figure 4 area_mean boxplot

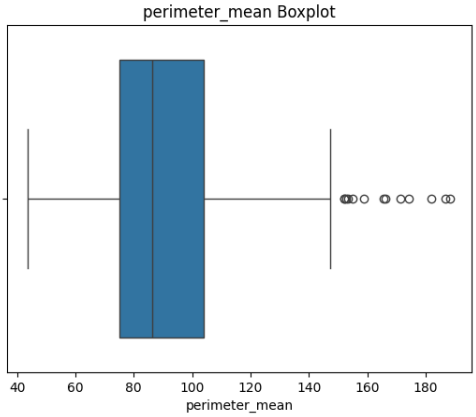

Figure 5 perimeter_mean boxplot

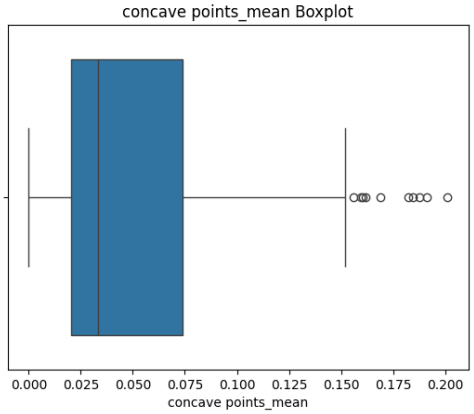

Figure 6 concave points_mean boxplot

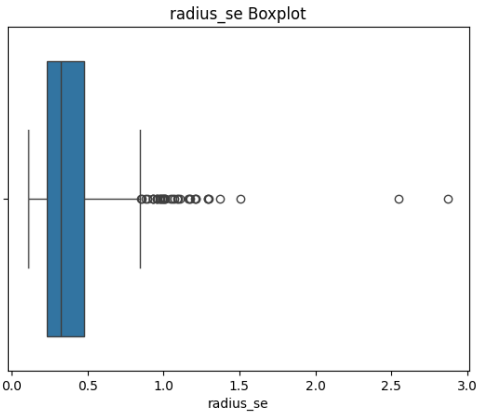

Figure 7 radius_se boxplot

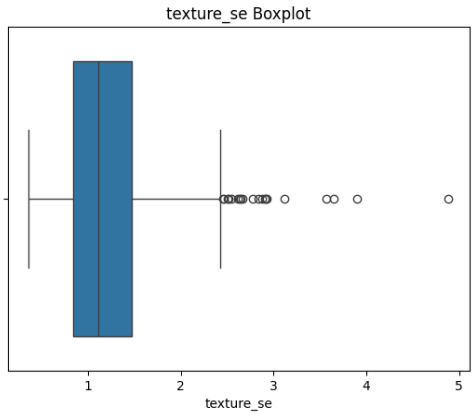

Figure 8 texture_se boxplot

3. Outlier Detection and handling

In this part of the code we began to identify outlier features by creating a list of features excluding the target variable (‘diagnosis’) and any specific column (‘concave points_worst’) that might not require outlier handling, then we calculated the IQR for each feature and computing the lower boundary and the upper boundary, finally replacing the outliers beyond the upper and lower boundaries with the boundary values, Boxplots are created for each feature again to visualize the data distribution after outlier removal.

Screenshots of boxplots after outlier handling:

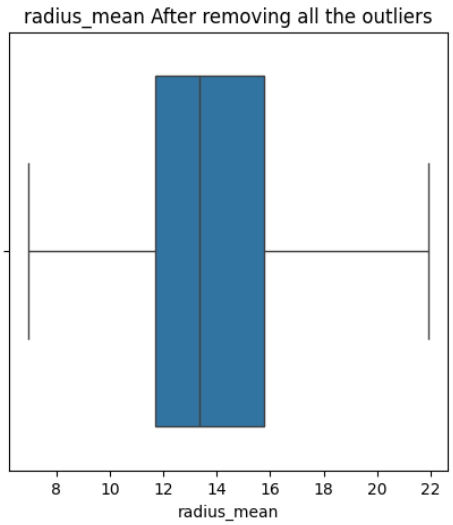

Figure 9 radius_mean without outliers

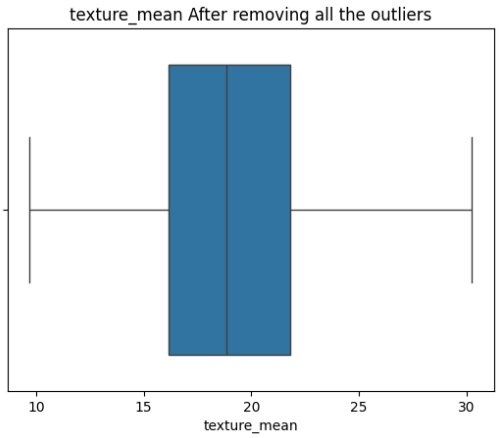

Figure 10 texture-mean without outliers

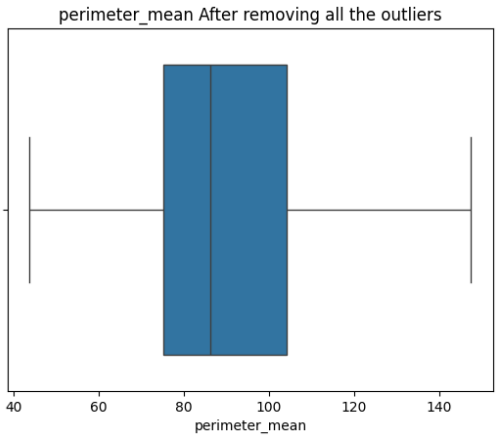

Figure 11 perimeter_mean without outliers

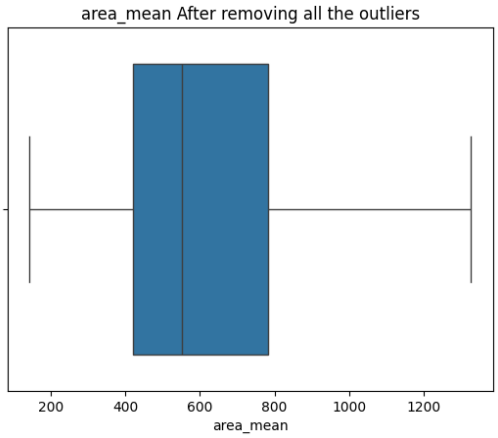

Figure 12 area_mean without outliers

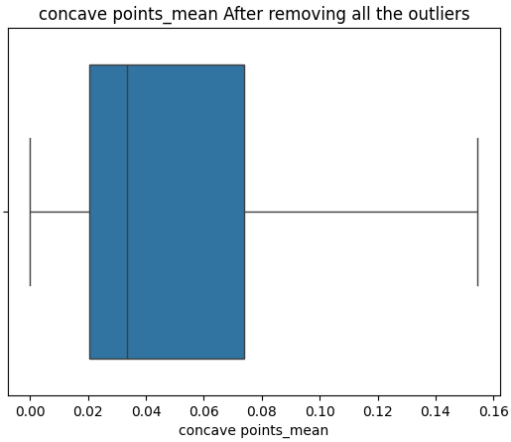

Figure 14 concave points_mean without outliers

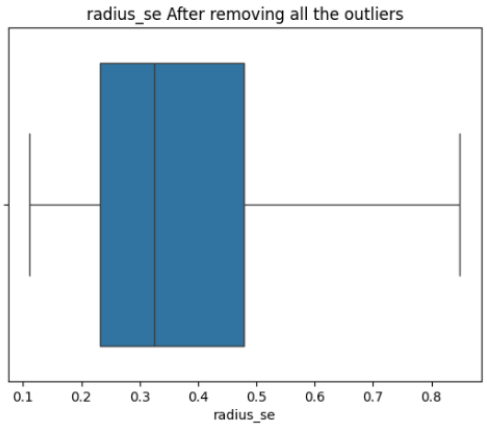

Figure 13 radius_se without outliers

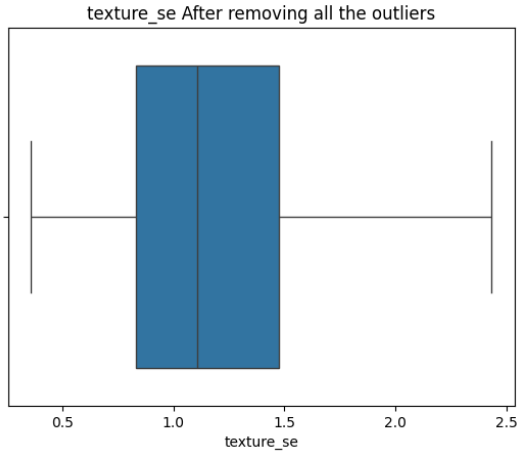

Figure 15 texture_se without outliers

4. Feature selection and visualization

After selecting columns with numerical values and generating a pairwise scatterplots for a random sample of 10 rows from the Data Frame, visualizing the relationship between numerical values in the dataset. Each plot shows the relationship between two variables and the diagonal shows the distribution of each variable. And then we grouped the data frame by the diagnosis column, calculating the mean value for each feature for each group, then created feature matrix ‘X’ by dropping the target variable 'diagnosis' from the Data Frame. The resulting Data Frame contains all the features used for model training. And created the target vector 'y' by selecting the 'diagnosis' column from the Data Frame. It contains the labels indicating the diagnosis (e.g., 'M' for malignant, 'B' for benign). Finally
initialized a StandardScaler object from scikit-learn, which is used to standardize features by removing the mean and scaling to unit variance and applied the StandardScaler to standardize the feature matrix 'X'. It transforms the values of each feature so that they have a mean of 0 and a standard deviation of 1.

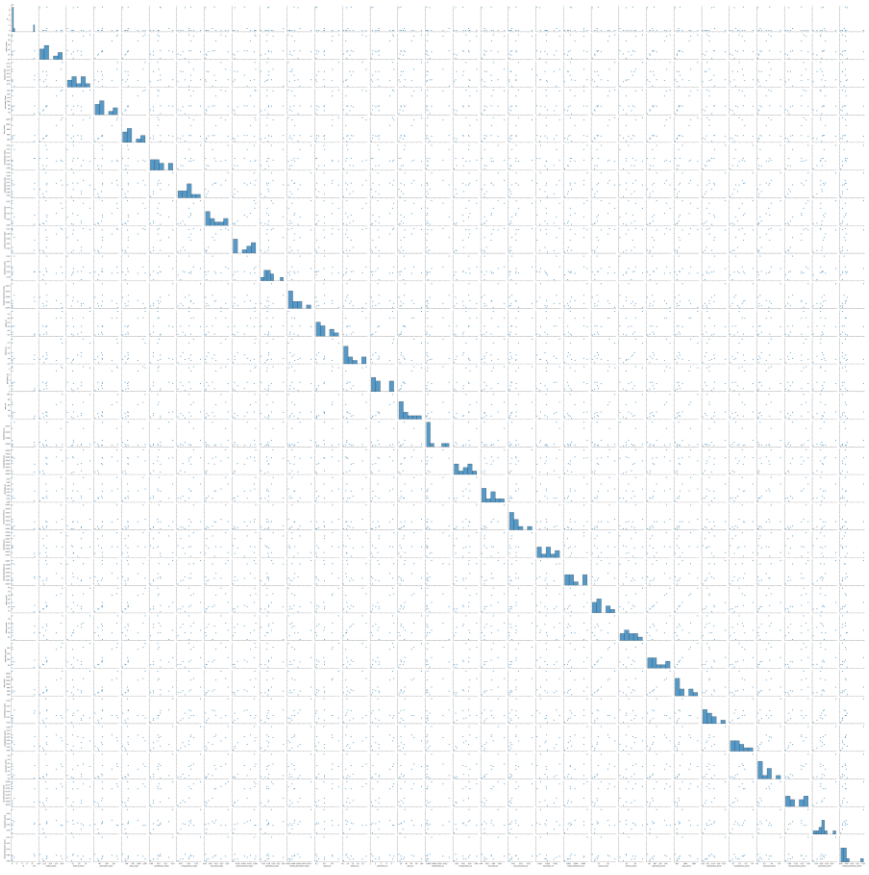

Figure 16

Checking correlation between the attributes with each other:

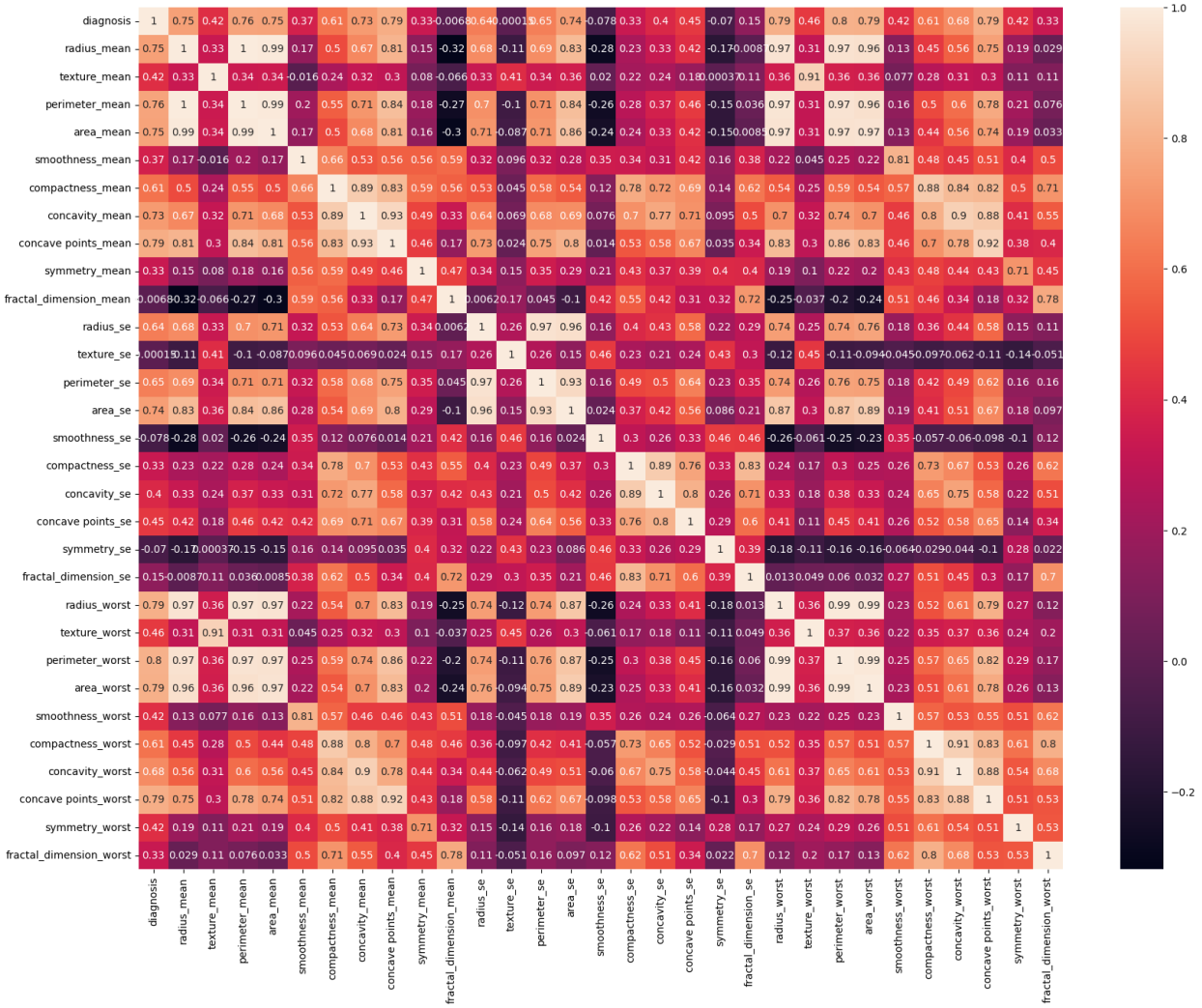

Figure 17 correlation of attributes

5. Scaling with standard scaler

Steps:

- Grouping data by diagnosis and calculating mean
- Feature selection
- Target variable separation
- Feature scaling using standard scaler.

6. Model training.

Splitting the dataset into training and testing sets. 80% of the data is used for training (train size=0.8), and the remaining 20% is used for testing. Random state=50 ensures reproducibility by fixing the random seed.

logistic regression model:

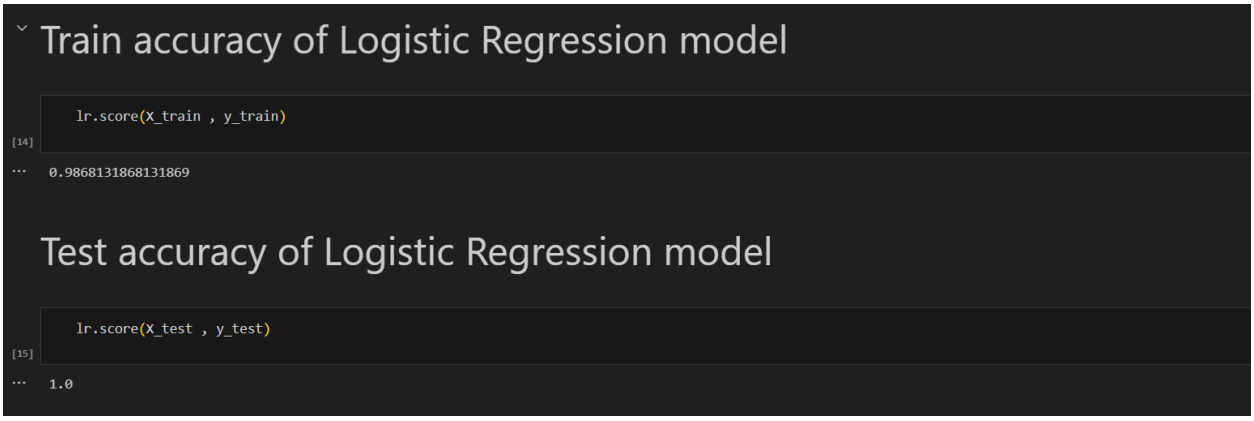

Figure 18 logistic regression model

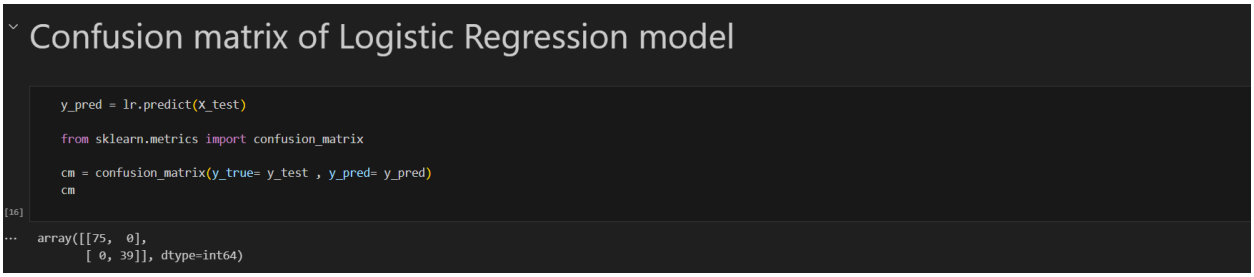

Figure 19 logistic reg_confusion matrix

KNN Model:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

for i in range(1, 6):
    KNN = KNeighborsClassifier(n_neighbors=i)

    KNN.fit(X_train, y_train)

    y_pred = KNN.predict(X_test)

    print(f"Accuracy with k={i}: ", accuracy_score(y_test, y_pred))
    print(f"Confusion Matrix with k={i}:\n", confusion_matrix(y_test, y_pred))
```

[17] Python

```
Accuracy with k=1:  0.9649122807017544
Confusion Matrix with k=1:
 [[71  4]
 [ 0 39]]
Accuracy with k=2:  0.9473684210526315
Confusion Matrix with k=2:
 [[72  3]
 [ 3 36]]
Accuracy with k=3:  0.956140350877193
Confusion Matrix with k=3:
 [[72  3]
 [ 2 37]]
Accuracy with k=4:  0.9473684210526315
Confusion Matrix with k=4:
 [[72  3]
 [ 3 36]]
Accuracy with k=5:  0.9473684210526315
Confusion Matrix with k=5:
 [[71  4]
```

Figure 20 KNN Model

Decision Tree model:

```
print("Decision Tree")
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

DT = DecisionTreeClassifier()

DT = DT.fit(X_train, y_train)

y_pred = DT.predict(X_test)

print("Accuracy: ", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

[18]

```
Decision Tree
Accuracy:  0.9210526315789473
Confusion Matrix:
 [[69  6]
 [ 3 36]]
```

Figure 21 decision tree model

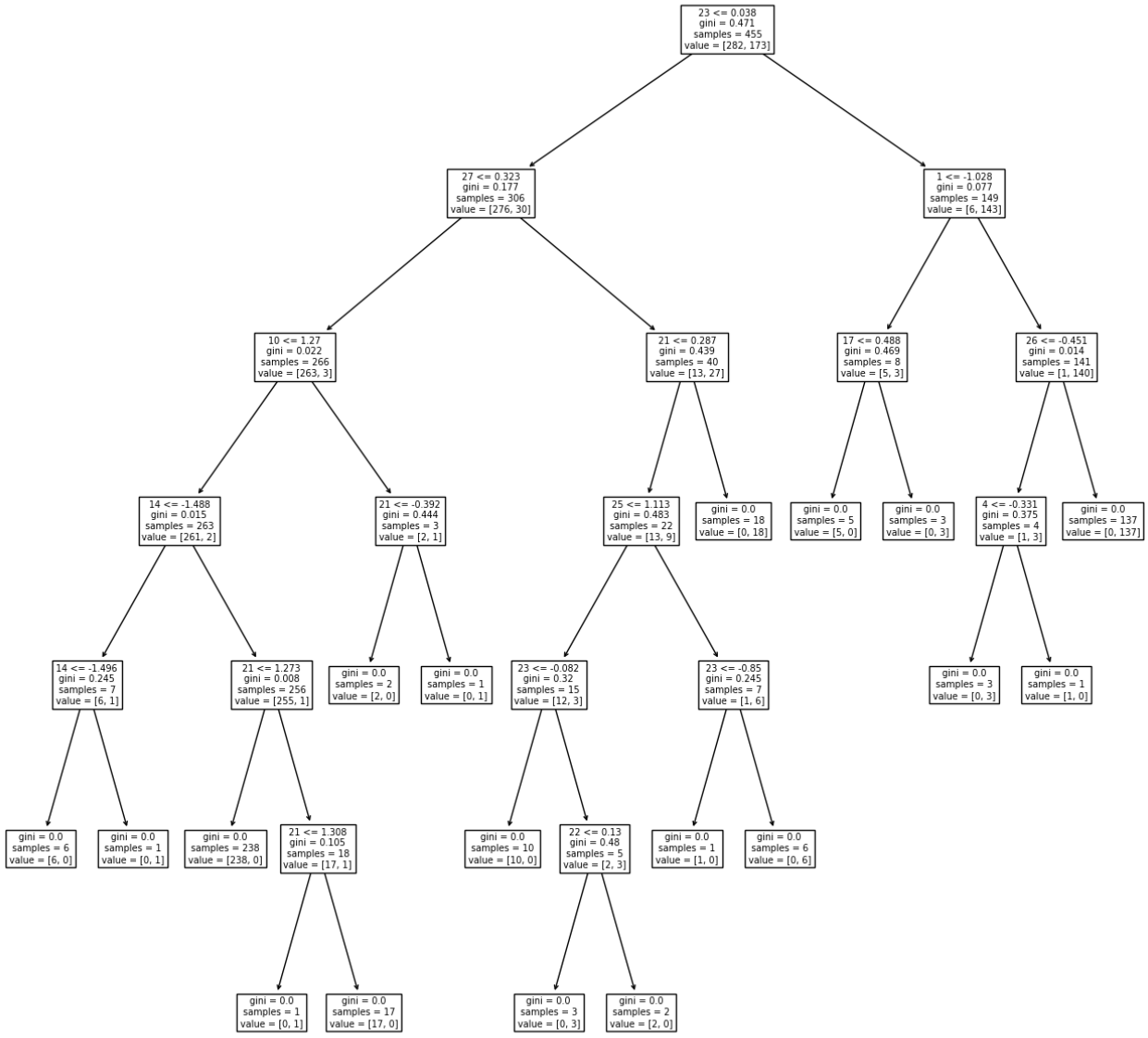

Figure 22 decision tree

7. Histograms and Subplots

Plotting Histograms for some columns:

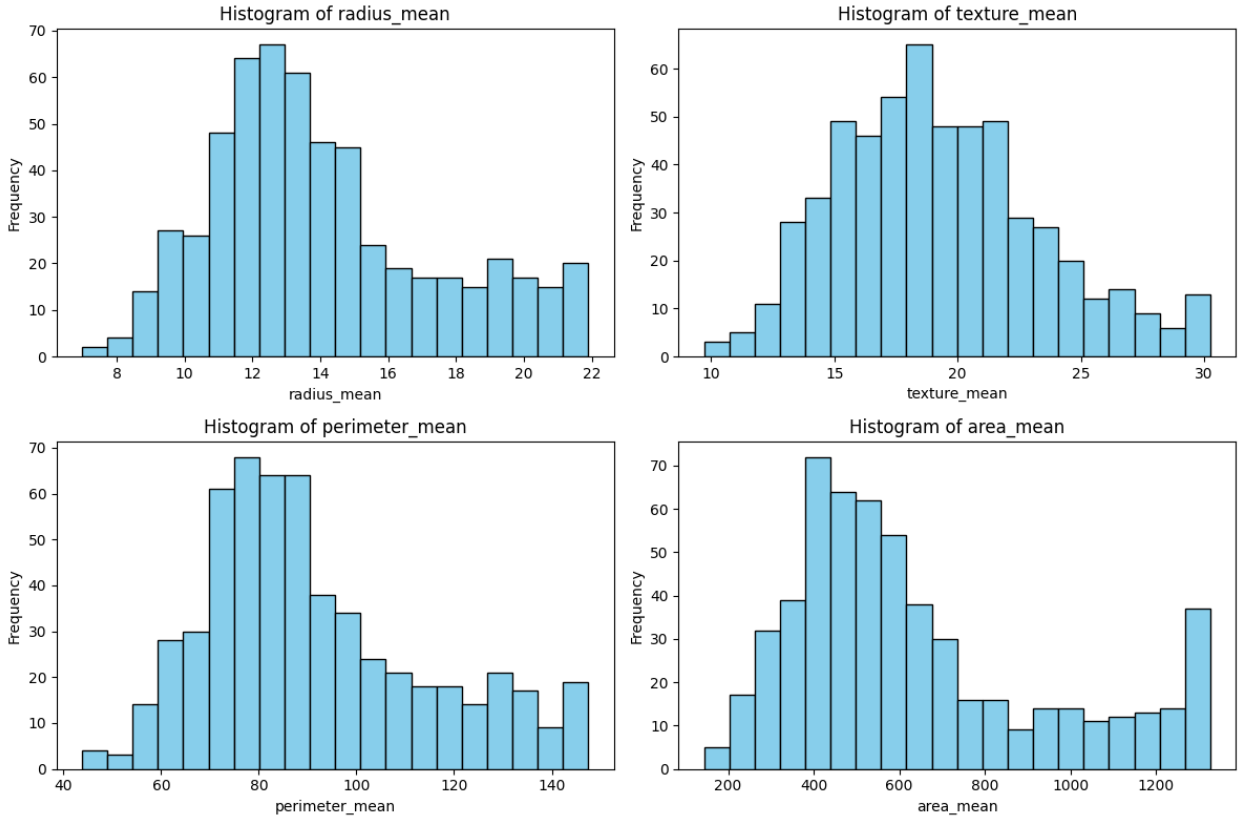

Figure 23 Histograms

Subplots without boxplots inside it:

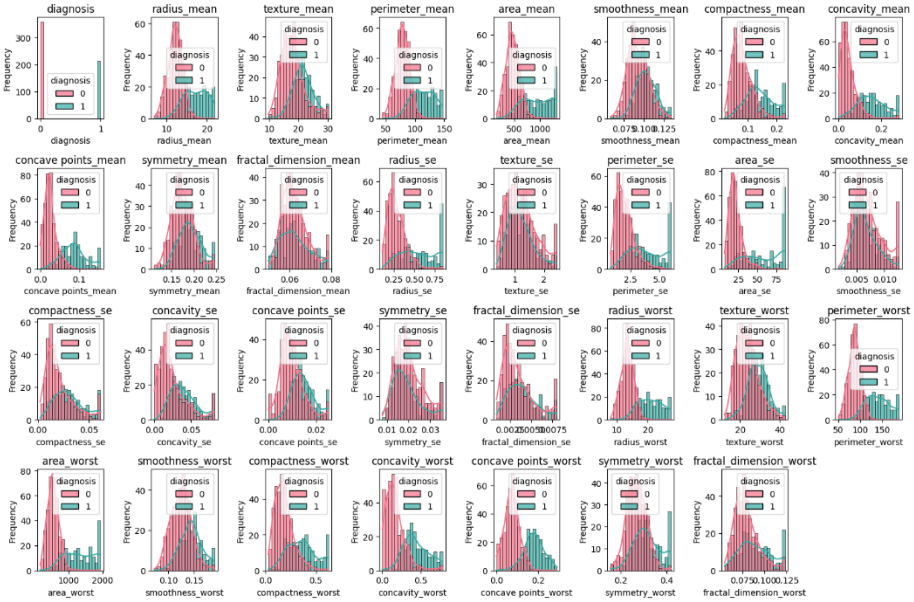

Figure 24 Subplots without boxplots

Subplots using boxplots inside it:

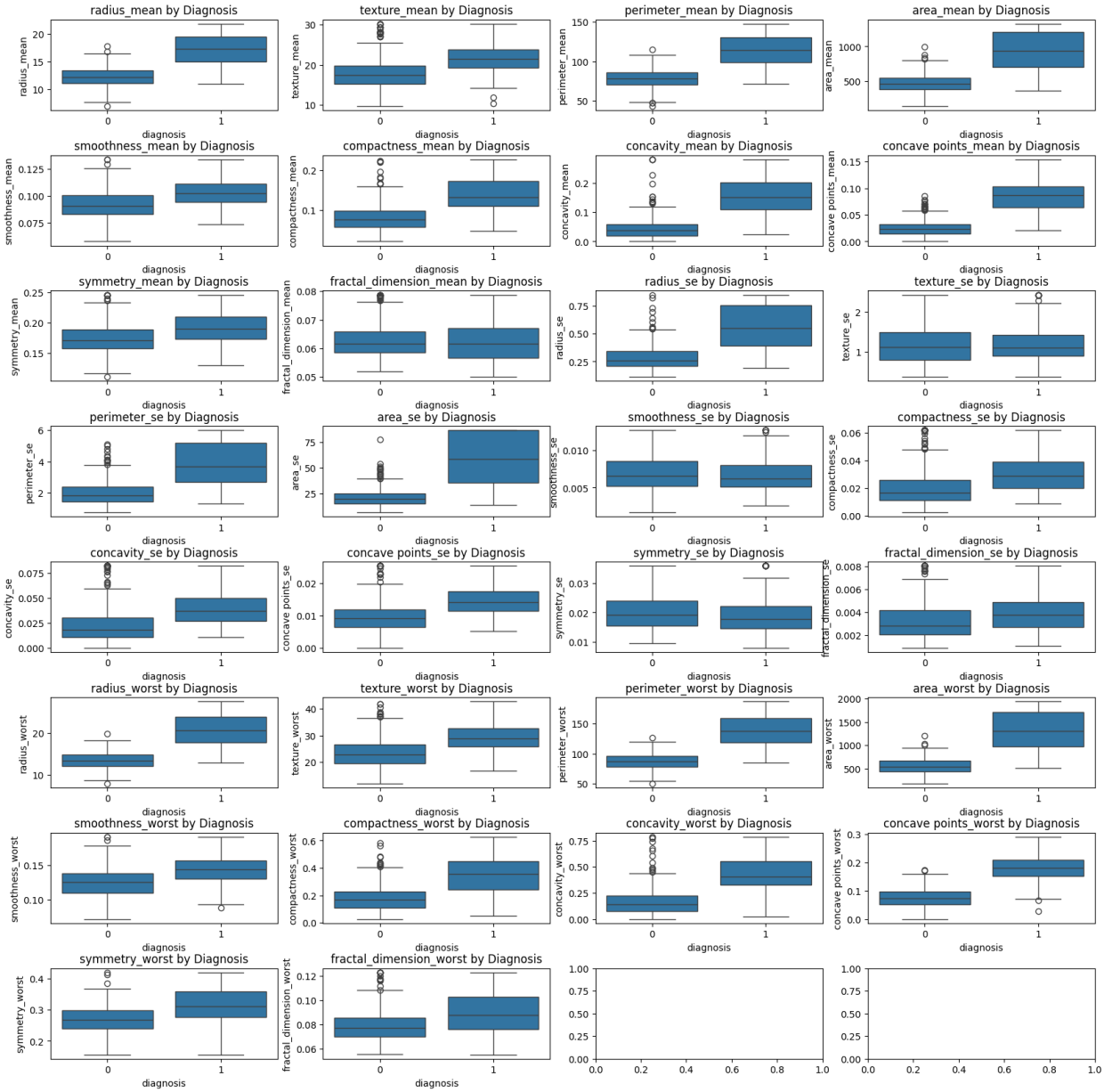

Figure 25 subplots with boxplots

8. Scaling Features with MinMaxScaler

We standardized the features using MinMaxScaler after dropping the target variable 'diagnosis'. To ensure that all features are on a similar scale, which is beneficial for certain machine learning algorithms that are sensitive to feature scaling.

```
[25]
scaler2 = MinMaxScaler()

X_new = df.drop('diagnosis',axis= 1)

X_new = scaler2.fit_transform(X_new)
```

```
[26]
X_new
```

```
array([[0.73791809, 0.03262722, 0.76187262, ..., 0.91202749, 1.        ,
        0.93953215],
       [0.91085193, 0.39250061, 0.85926426, ..., 0.63917526, 0.45117076,
        0.49992644],
       [0.85186675, 0.56196737, 0.83130032, ..., 0.83505155, 0.77974491,
        0.47874062],
       ...,
       [0.64474831, 0.89457025, 0.62205294, ..., 0.48728522, 0.24861984,
        0.34073856],
       [0.91286279, 0.95544193, 0.92869196, ..., 0.91065292, 0.96021321,
        1.        ],
       [0.0522153 , 0.72218164, 0.0398245 , ..., 0.        , 0.49723967,
        0.22583493]])
```

9. Hyper Parameters tunning

Now importing the models again to make a dictionary with a lot of models and hyper parameters and then testing algorithms with MinMaxScaler using GridSearch (with and without feature scaling) to detect the optimum hyper parameter in each model then choose the best accuracy between the models.

```python
from sklearn.model_selection import GridSearchCV
import pandas as pd
scores = []

for model_name, mp in model_params.items():
    clf =  GridSearchCV(mp['model'], mp['params'], cv=5, return_train_score=False)
    clf.fit(X_new,y)
    scores.append({
        'model': model_name,
        'best_score': clf.best_score_,
        'best_params': clf.best_params_
    })

df_gridsearch_scaled = pd.DataFrame(scores,columns=['model','best_score','best_params'])
df_gridsearch_scaled
```

Python

Algorithms without scaling

```
from sklearn.model_selection import GridSearchCV
import pandas as pd
scores = []

for model_name, mp in model_params.items():
    clf =  GridSearchCV(mp['model'], mp['params'], cv=5, return_train_score=False)
    clf.fit(X_new_without_scaling,y)
    scores.append({
        'model': model_name,
        'best_score': clf.best_score_,
        'best_params': clf.best_params_
    })

df_gridsearch_without_scaling = pd.DataFrame(scores,columns=['model','best_score','best_params'])
df_gridsearch_without_scaling
```

[29] Python

10. K-Means clustering

In this section we performed KMeans clustering to determine the optimal number of clusters using the Elbow Method, It calculates the sum of squared errors (SSE) for different numbers of clusters (k) and plots the SSE against the number of clusters, Aso applying KMeans clustering with k=3 clusters and assigning cluster labels to the data.

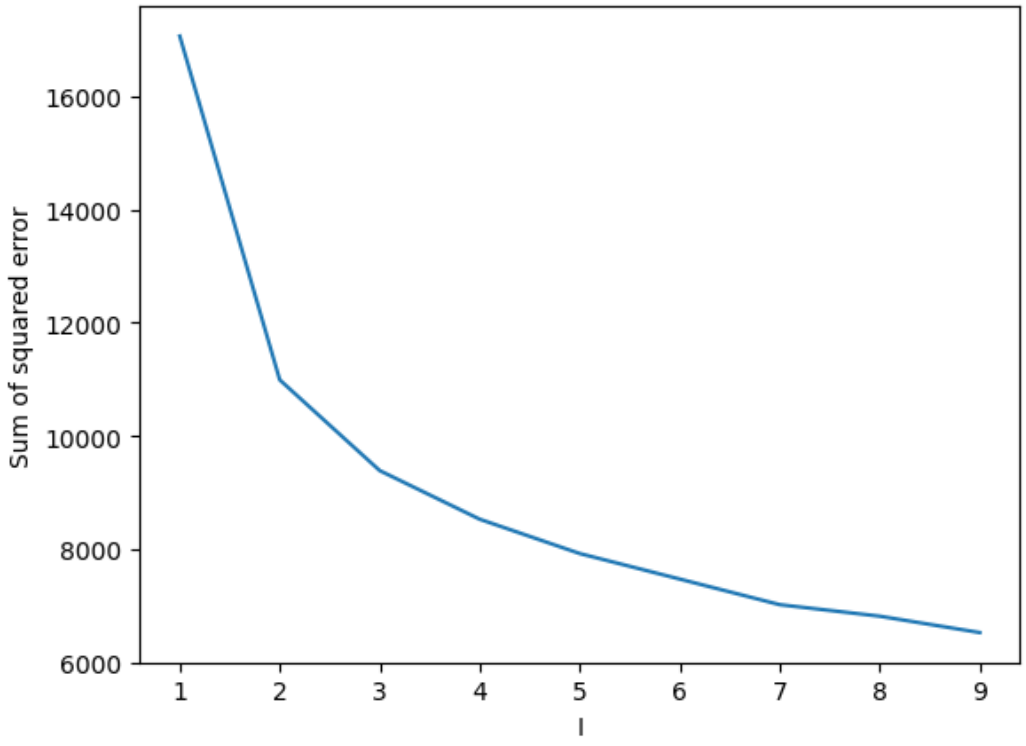

Figure 26 SSE

```
km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(X,y)
y_predicted
```

[32]

```
array([1, 1, 1, 2, 1, 2, 1, 2, 2, 2, 0, 2, 1, 0, 2, 2, 0, 2, 1, 0, 0, 0,
       2, 1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 1, 2, 1, 2, 0, 0, 2, 0, 2, 1, 2,
       0, 1, 0, 2, 0, 0, 0, 0, 0, 1, 0, 0, 1, 2, 0, 0, 0, 0, 2, 0, 2, 2,
       0, 0, 2, 0, 1, 2, 1, 0, 0, 1, 0, 1, 1, 0, 0, 2, 1, 1, 0, 1, 2, 1,
       0, 2, 0, 0, 0, 0, 2, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 2, 2, 0, 1, 0,
       0, 2, 2, 2, 0, 0, 0, 2, 2, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 2, 1, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 2, 2, 2, 0, 0, 0, 2, 2, 0,
       0, 0, 1, 0, 0, 0, 2, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 2, 0, 0, 0,
       2, 2, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 2, 2, 0, 2, 1,
       1, 2, 0, 1, 1, 2, 0, 0, 0, 0, 2, 0, 1, 0, 1, 1, 2, 2, 2, 0, 1, 1,
       0, 0, 0, 2, 0, 0, 0, 0, 0, 2, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
       2, 0, 1, 0, 0, 2, 0, 0, 1, 0, 1, 1, 1, 2, 1, 2, 1, 2, 1, 0, 1, 0,
       1, 1, 0, 0, 0, 2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 2, 0, 0,
       2, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 2, 1, 0, 1, 0, 0, 0, 0, 1, 2,
       2, 2, 0, 0, 0, 1, 0, 1, 0, 1, 2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2,
       1, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 2, 0, 1, 1,
       0, 0, 2, 0, 0, 2, 2, 0, 0, 2, 0, 0, 0, 0, 2, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 2, 2, 1, 1, 0, 2, 0, 0, 0, 0,
       2, 1, 0, 0, 0, 0, 1, 0, 0, 1, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 2, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0,
       0, 2, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 2, 0, 1, 1, 0, 2, 0, 1, 2, 2,
       0, 2, 0, 2, 0, 0, 2, 0, 0, 0, 1, 1, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0,
       2, 0, 0, 0, 0, 1, 0, 1, 2, 2, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 1, 1, 1, 1, 0])
```

Figure 27 Data clustering