**Data Preparation/Feature Engineering**

**1. Overview**

The project focuses on predicting the benign or malignant nature of breast cancer masses using the Breast Cancer Wisconsin (Diagnostic) Data Set. Beginning with data collection, the dataset underwent comprehensive cleaning operations, including the removal of irrelevant columns, encoding categorical variables, logarithmic transformation, and Z-score normalization. Exploratory data analysis revealed normal distribution patterns and identified feature correlations, guiding subsequent decisions in the presence of outliers. Feature engineering incorporated scaling for improved model performance, and dimensionality reduction techniques like Principal Component Analysis (PCA) were considered. Data transformation included encoding categorical variables and standardizing features. The project's holistic approach integrates data exploration, cleaning, feature engineering, and transformation, laying the groundwork for robust machine learning model development with potential applications in medical diagnosis and treatment strategies.

**2. Data Collection**

The data source that will be used is Breast Cancer Wisconsin (Diagnostic) Data Set. This Dataset is tailored for predicting the benign or malignant nature of breast cancer masses. It encompasses features derived from digitized fine needle aspirates (FNAs) images, detailing characteristics of cell nuclei present in the images.

**3. Data Cleaning**

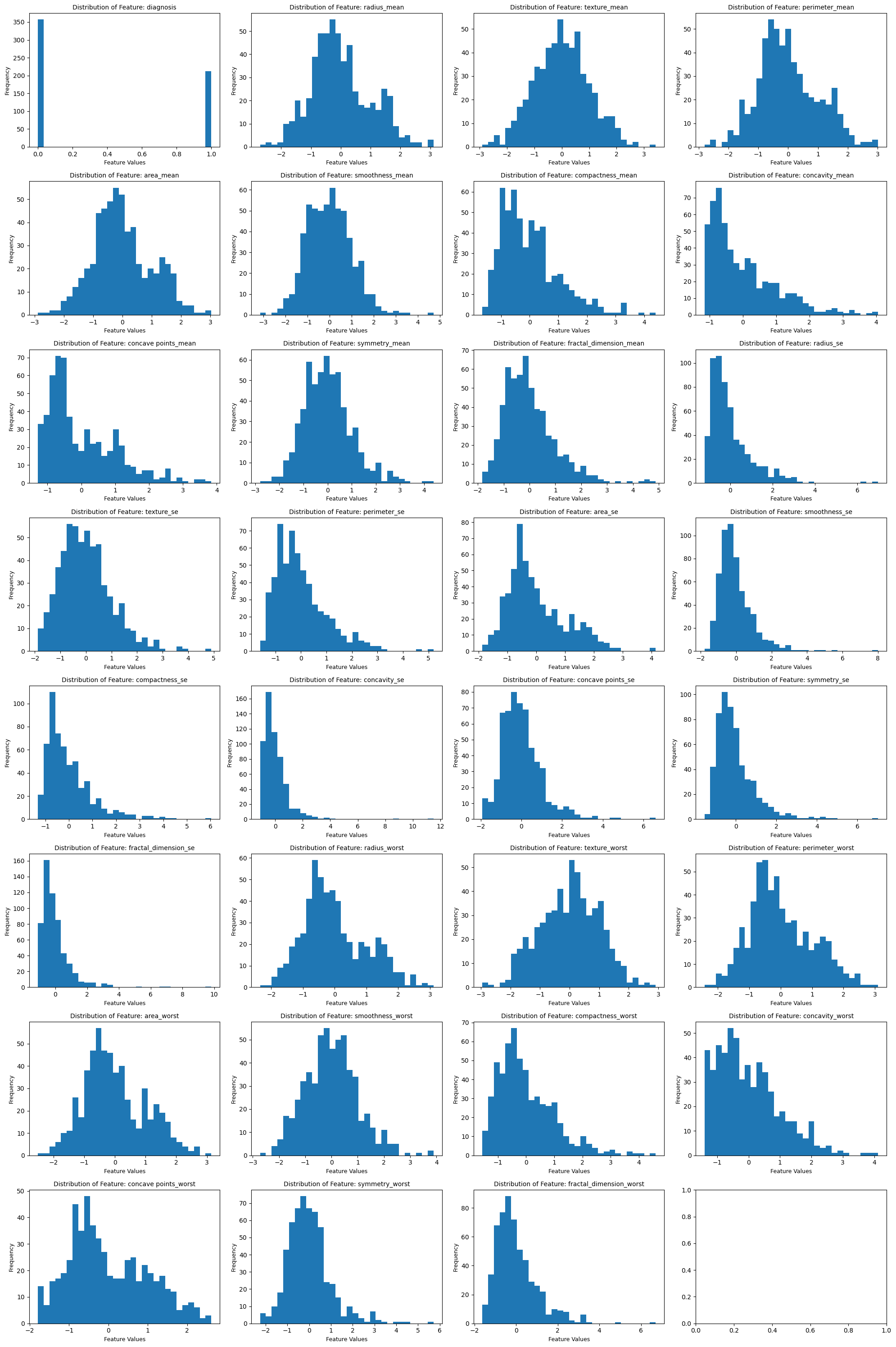
The chosen dataset doesn’t have any missing values or duplicated rows, so the operations to deal with them were spared. The set of operations taken place to conduct data cleaning were as follows:

1. Dropping irrelevant columns (Removing unwanted features)
2. Encoding Categorical Variables: Malignant as 1 and Benign as 0.
3. Logarithmic Transformation
4. Z-Score Normalization

**4. Exploratory Data Analysis (EDA)**

Our dataset does not have null values and except the target feature, all other features are numerical values. There are not any duplicates in the dataset as well.

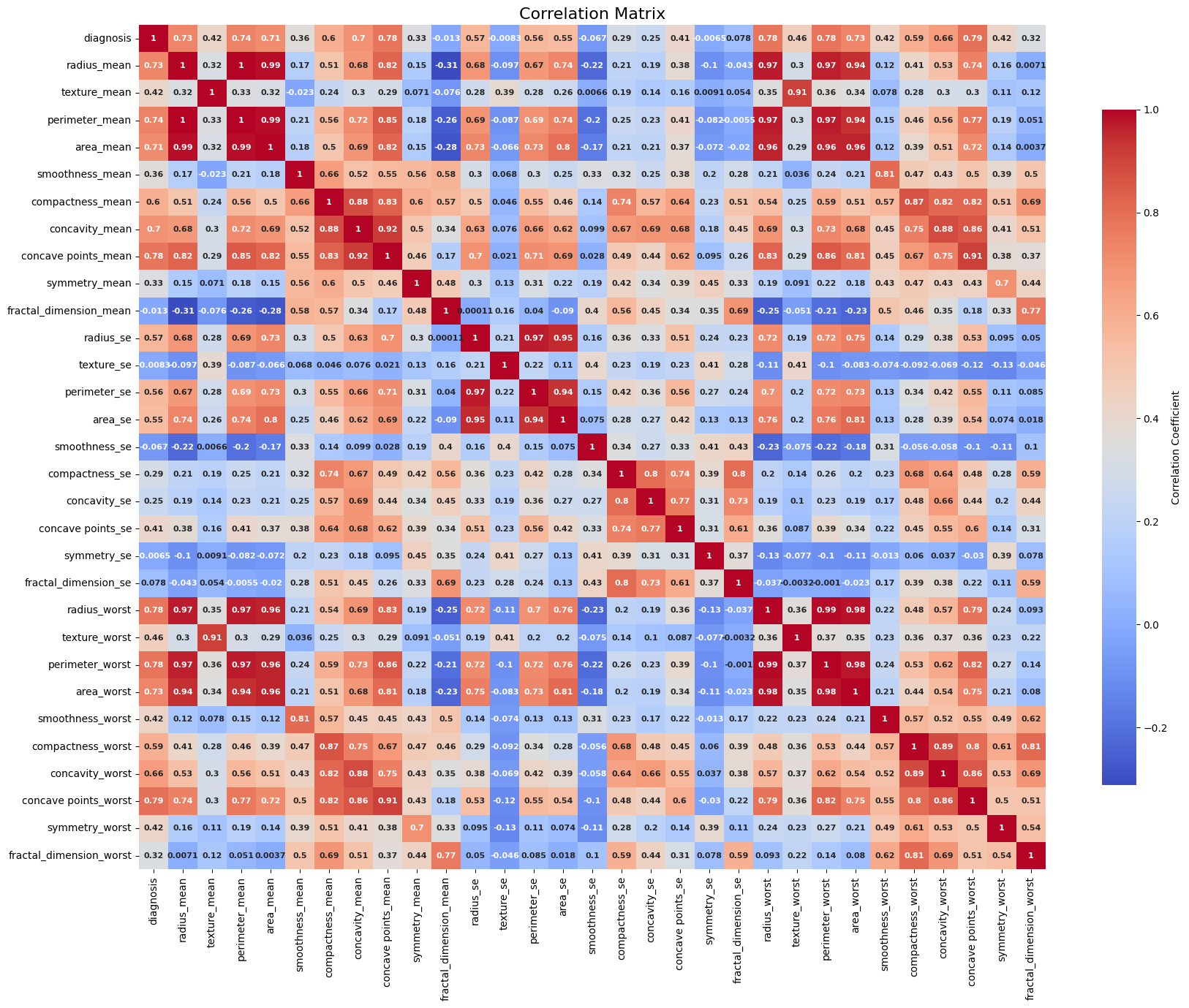
Form our EDA, we have found out that most our dataset’s features are fairly normally distributed.



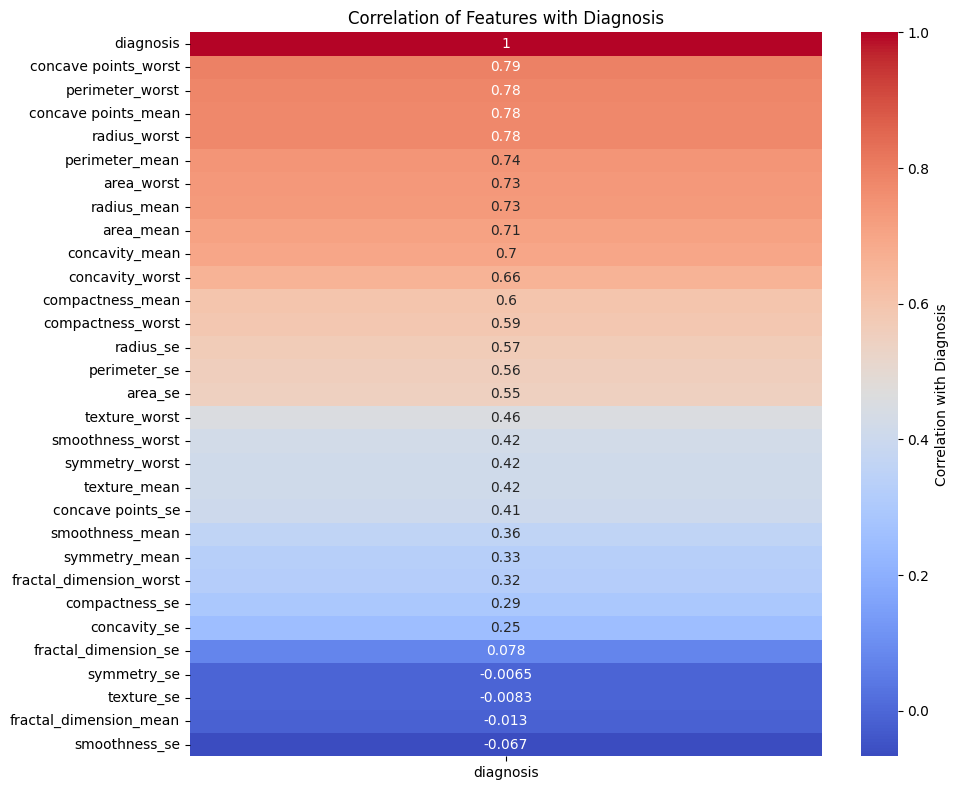
**Figure 1:** Distribution of features of the dataset.

Also, we have found out that there are features that are highly correlated, which can be reduced by PCA.

Correlation analysis is a valuable step in understanding relationships between variables in the dataset. It helps identify how strongly pairs of variables are related to each other and the direction of their relationship. Examining correlations in the context of breast cancer prediction is vital. It allows us to uncover relationships between specific variable (see Figure 2) and the diagnosis (malignant or benign) (see Figure 3). By identifying strong correlations, we pinpoint influential variable, aiding in variable (feature) selection and precise model development. These insights shed light on critical factors that may indicate malignancy, guiding better-informed decisions for medical diagnosis and treatment strategies.



**Figure 2:** Correlation of between features.

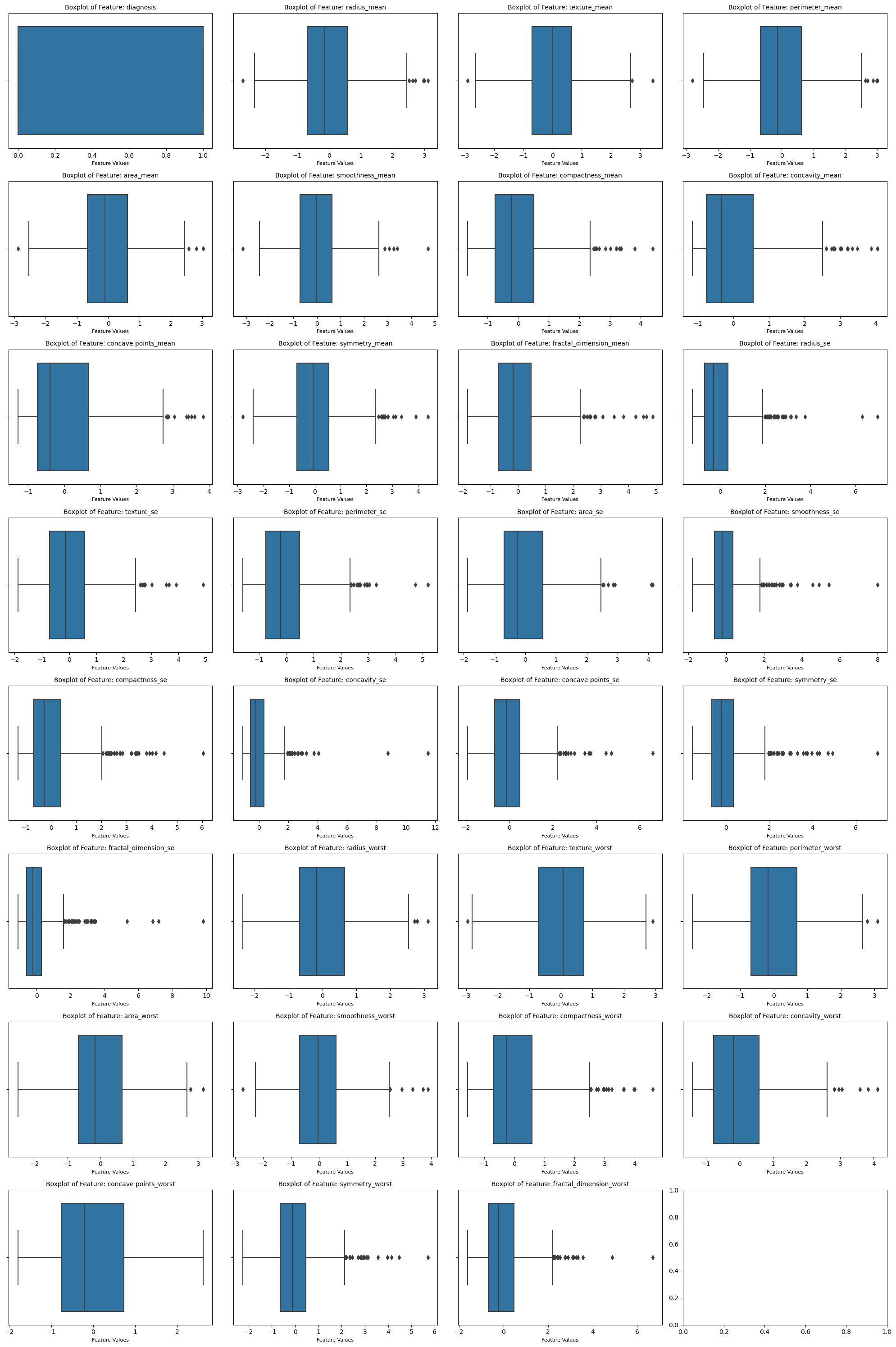


**Figure 3:** Correlation of the features to the diagnosis feature.

In Figure 3, the correlation of the variables to the diagnosis variable is highlighted using heatmap. With color going from a blue to red, correlation degree is represented with color going from blue (low correlation) to red (higher correlation).

Although we applied logarithmic transformation in the data cleaning step to help dealing with outliers in our dataset, there still seems to be outliers in most of the features as well.

The next thing we could do to handle this would be replacing those values with their respective medians, effectively making them contribute nothing. But, since the dataset is small, we decided not to follow through.

 **Figure 4:** Outliers in the dataset.

**5. Feature Engineering**

In data cleaning phase, we performed feature scaling to yield better performance (Figure 1). As we have mentioned before, we could perform feature extraction to reduce the dimensionality of our dataset by using the PCA (Principal Component Analysis) technique. Reducing dimensions in a dataset serves several purposes beyond just improving computational performance. While dimensionality reduction techniques like Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE) can indeed lead to more efficient model training, there are other important reasons to consider dimensionality reduction such as, better Visualization, Overfitting Prevention, Collinearity Reduction, Noise Reduction, Computational and Memory Efficiency, Curse of Dimensionality (where the volume of the data increases exponentially with the number of features), Interpretability.

In PCA, the cumulative explained variance represents the proportion of the total variance in the original data that is captured by the retained principal components. The choice of a "good" cumulative explained variance in Principal Component Analysis (PCA) depends on the specific goals of the analysis and the trade-off willing to be made between dimensionality reduction and information retention. A common approach is to set a threshold for the cumulative explained variance and choose the number of principal components that exceed that threshold. A rough guideline would be as follows:

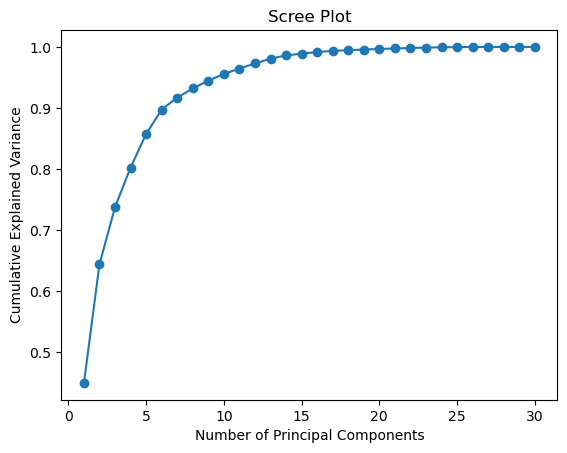
* For High Variance Retention: 95% or more
* For Balanced Trade-Off: 80% to 90%
* For Aggressive Dimensionality Reduction: 70% or less

In our dataset, the relation between the number of components and the information retained is shown in Figure 6.

The following table shows the number of principal component and the information retained.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0.449 | 0.644 | 0.737 | 0.802 | 0.857 | 0.897 | 0.917 | 0.932 | 0.945 | 0.956 |
|  |  |  |  |  |  |  |  |  |  |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 0.964 | 0.973 | 0.981 | 0.986 | 0.989 | 0.992 | 0.993 | 0.995 | 0.996 | 0.997 |
|  |  |  |  |  |  |  |  |  |  |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 0.997 | 0.998 | 0.999 | 0.999 | 1 | 1 | 1 | 1 | 1 | 1 |

We decided to pick **11** components as we want to retain as much variance/information as possible while keeping the number as low as possible.

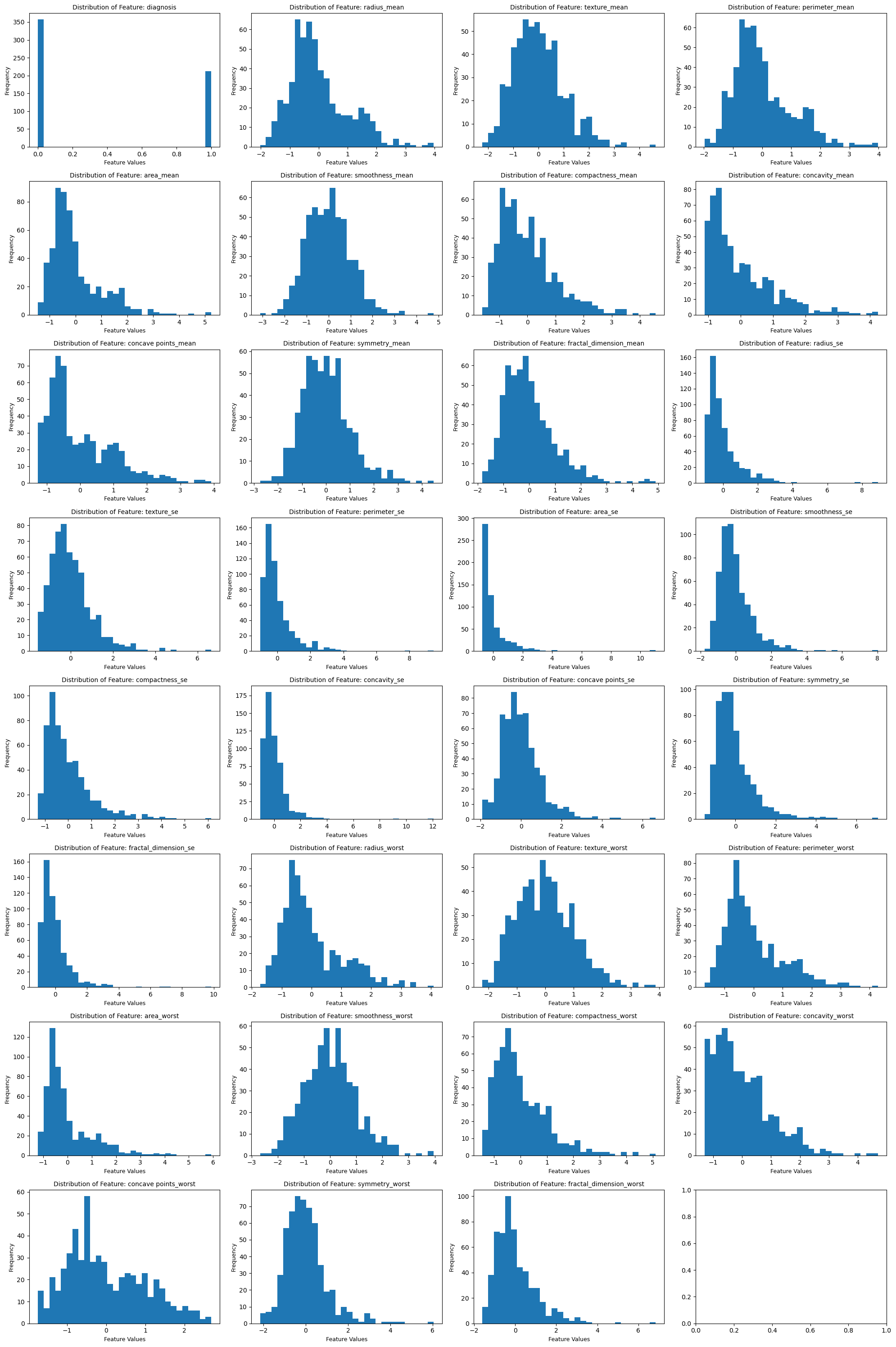


**Figure 6:** Relationship between the number of Principal Components and CEV (Information Retained).

**6. Data Transformation**

* **Encoding performed on the dataset using Label Encoding.**

Within our dataset, the target attribute ‘diagnosis’ stands out from the rest. Unlike other data points in the dataset that are numerical, the 'diagnosis' attribute contains categorical information that describes whether a case is identified as benign (‘B’) or malignant (‘M’). Since non-numerical data cannot undergo mathematical operations directly, they need to be transformed. The figure below illustrates the result of Assigning numerical labels to categories (0 for benign, 1 for malignant).



**Figure 7:** Historical visualization of the diagnosis attribute of the dataset after encoding.

Data Transformation such as Z-score normalization and logarithmic transformations had taken place in previous steps (Figure 6).

Z-score normalization, also known as standardization, is a data transformation technique used to scale features so that they have a mean of zero and a standard deviation of one. It involves transforming numerical data in such a way that the distribution will have a mean of 0 and a standard deviation of 1. Z-score normalization is a useful technique in standardizing features to a common scale, facilitating fair comparisons among different features and aiding the performance of various machine learning algorithms.

**Model Exploration**

**1. Model Selection**

We decided to explore the following models:

* Decision Trees
* Logistic Regression
* KNN

**1. Decision Trees:**

**Rationale:**

* Decision Trees are suitable for both classification and regression tasks.
* They are capable of capturing nonlinear relationships and interactions between features.
* Decision Trees are interpretable, making them useful for understanding decision-making processes.

**Strengths:**

* **Interpretability:** Decision Trees provide a transparent and easy-to-understand decision-making structure.
* **Handles Mixed Data Types:** Can handle both numerical and categorical features.
* **Feature Importance:** Easily identifies important features.

**Weaknesses:**

* **Overfitting:** Decision Trees can be prone to overfitting, especially on noisy data.
* **Instability:** Small changes in the data can lead to different tree structures.

**2. Logistic Regression:**

**Rationale:**

* Logistic Regression is a widely used model for binary classification tasks.
* It provides probability estimates, making it interpretable and suitable for ranking predictions.

**Strengths:**

* **Interpretability:** Logistic Regression coefficients represent the impact of each feature on the log-odds of the response.
* **Efficiency:** Training and prediction are computationally efficient.
* **Probabilistic Predictions:** Provides probability estimates.

**Weaknesses:**

* **Assumption of Linearity:** Logistic Regression assumes a linear relationship between features and the log-odds.
* **Limited Expressiveness:** May struggle with highly nonlinear relationships.

**3. Random Forests:**

**Rationale:**

* Random Forests are an ensemble learning method that builds multiple Decision Trees and combines their predictions.
* The rationale behind using Random Forests is to improve the predictive performance, generalization, and robustness compared to individual Decision Trees.
* By aggregating predictions from multiple trees, Random Forests aim to reduce overfitting and enhance model accuracy.

**Strengths:**

* **High Accuracy:** Random Forests often provide higher accuracy compared to individual Decision Trees, especially on complex datasets.
* **Reduction in Overfitting:** The ensemble nature of Random Forests helps reduce overfitting, making them more robust to noise and outliers.
* **Feature Importance:** Random Forests can provide a measure of feature importance, indicating which features contribute the most to the model's predictions.
* **Versatility:** Suitable for both classification and regression tasks, making them versatile for a wide range of applications.
* **Handles Missing Values:** Random Forests can handle missing values in the dataset without requiring imputation.
* **Robustness:** Robust to outliers and can handle unbalanced datasets.

**Weaknesses:**

* **Less Interpretable:** The ensemble of trees makes Random Forests less interpretable compared to individual Decision Trees.
* **Computational Resources:** Training and predicting with Random Forests can be computationally expensive, especially for large datasets.
* **Memory Usage:** Random Forests can consume significant memory, particularly as the number of trees increases.
* **Not Suitable for Real-Time Applications:** Random Forests may not be suitable for real-time applications where low-latency predictions are critical.
* **Black-Box Nature:** The ensemble nature of Random Forests can make them more challenging to interpret compared to simpler models like Logistic Regression.

**Considerations:**

* **Hyperparameter Tuning:** The performance of Random Forests can be influenced by hyperparameters such as the number of trees, depth of trees, and the number of features considered at each split. Tuning these hyperparameters is crucial for optimal performance.
* **Randomness:** The use of randomness in feature selection and data bootstrapping contributes to the diversity of the ensemble, improving generalization.
* **Ensemble Advantage:** The strength of Random Forests lies in the diversity of the constituent trees. If individual trees make errors on certain instances, the ensemble can still provide accurate predictions.

**3. K-Nearest Neighbors (KNN):**

**Rationale:**

* KNN is a non-parametric, instance-based algorithm suitable for both classification and regression.
* It is simple to understand and implement, requiring no training phase.

**Strengths:**

* **Nonlinearity:** KNN can model complex relationships without assuming linearity.
* **No Training Phase:** KNN is an instance-based method.

**Weaknesses:**

* **Computational Complexity:** Prediction time can be slow, especially for large datasets.
* **Sensitivity to Noise and Outliers:** KNN can be sensitive to noisy or irrelevant features.

**5. Artificial Neural Networks (ANN):**

**Rationale:**

* ANN is a powerful model capable of learning complex relationships in data.
* Well-suited for tasks with large amounts of data and intricate patterns.

**Strengths:**

* **Nonlinearity:** ANNs can capture highly nonlinear relationships.
* **Representation Learning:** Automatically learns hierarchical representations of features.
* **Versatility:** Suitable for a wide range of tasks, including classification and regression.

**Weaknesses:**

* **Complexity:** Training and tuning ANNs can be computationally intensive.
* **Requires Sufficient Data:** May require a large amount of data to avoid overfitting.
* **Interpretability:** ANNs are often considered black-box models, making interpretation challenging.

Each of these models were trained with the normal dataset and also with 11 principal components. The accuracy of the models varied between 97% and 99%. The two contenders for the top spot were the ANN and the Random Forest models. We decided to select the Random Forest model as our dataset is too small for ANN hence, it is prone to overfitting and also it has outliers in a lot of its features.

**2. Model Training**

The model which we selected, Random Forest, was trained with 100 estimators. The dataset split was 75/25, meaning 75% of the data was used to train and 25% was used to test the model. The classification report for our model is as follows:

Random Forest Accuracy: 0.993006993006993

Classification Report:

precision recall f1-score support

0 0.99 1.00 0.99 89

1 1.00 0.98 0.99 54

accuracy 0.99 143

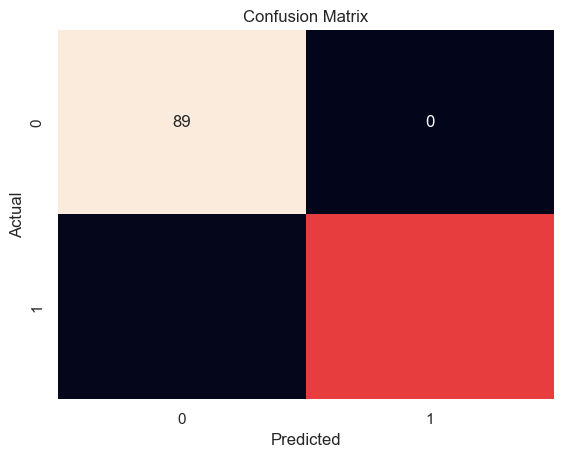
macro avg 0.99 0.99 0.99 143

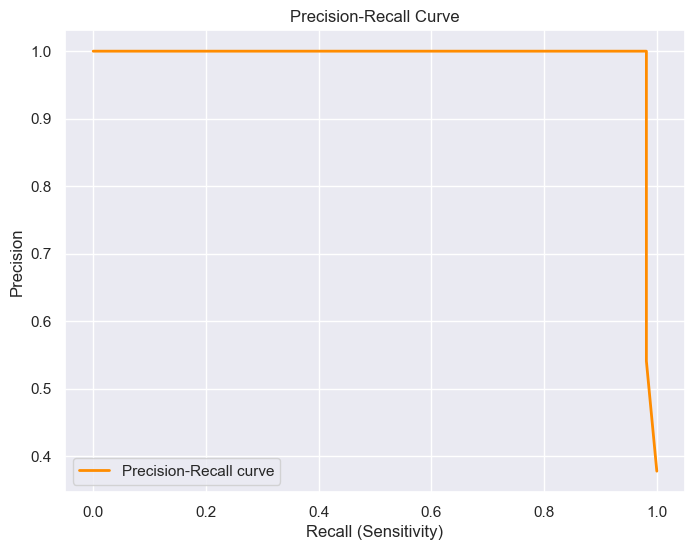
weighted avg 0.99 0.99 0.99 143

**3. Model Evaluation**

The visualizations for model evaluation are as follows:

* Confusion Matrix:



* ROC Curve:  
  
* Precision-Recall Curve:  
  

**4. Code Implementation**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

import seaborn as sns

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

# Reading the dataset

df = pd.read\_csv("data.csv")

# Data preparation

df.drop(columns=["id","Unnamed: 32"], inplace=True)

# Label Encoding

diagnosis\_mapping = {"M": 1, "B":0}

df["diagnosis"] = df["diagnosis"].map(diagnosis\_mapping)

# Scaling/Normalizing Data

scaler = StandardScaler()   # z-score normalization

training\_df = df.copy(deep=True)

training\_df.drop("diagnosis", inplace=True, axis=1)

training\_df\_cols = training\_df.columns

training\_df = np.log1p(1+training\_df)  # log-transformation

training\_df = scaler.fit\_transform(training\_df)

training\_df = pd.DataFrame(training\_df, columns=training\_df\_cols)

training\_df.insert(0, "diagnosis", df.diagnosis)

# Since we ended up not using components extracted from PCA, the code

# for performing PCA is not included

# Preparing data for training

cols = list(training\_df.columns)

cols.remove("diagnosis")

y = np.array(training\_df["diagnosis"]).reshape(-1,1)

X = np.array(training\_df[cols]).astype(float)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=.25, random\_state=123)

# Model Selection and Training: Random Forest

# Random Forest Classifier

random\_forest\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

random\_forest\_model.fit(X\_train, y\_train)

y\_pred\_random\_forest = random\_forest\_model.predict(X\_test)

accuracy\_random\_forest = accuracy\_score(y\_test, y\_pred\_random\_forest)

print("Random Forest Accuracy:", accuracy\_random\_forest)

print("Classification Report:")

print(classification\_report(y\_test, y\_pred\_random\_forest))