Cancer Data Classification: Investigating the Impact of the Implementation Platform on Machine Learning Algorithms Performance

By

Adedayo Seun Olowolayemo

SID: 100072314

(MSc Data Intelligence-2022/23)

A Thesis Submitted to The Department of Computing, School of Engineering, Technology and Design, Canterbury Christ Church University, United Kingdom

In partial fulfilment of the requirement for the degree of Master of Science

August 2023

Supervisors

Dr. Amina Souag **Prof. Konstantinos Sirlantzis**

Abstract:

Cancer is a global menace affecting millions of people worldwide and responsible for approximately one in every 6 deaths reported in 2020 according to the World Health Organization (WHO) [1]with Breast Cancer being one of the leading causes. Delays in diagnosis have been identified as one of the major bottlenecks in bringing down the high morbidity. This has led to Scientists having to use Machine Learning (ML) techniques in finding ways to reduce the diagnosis time.

In a bid to proffer a viable solution to the problem, many researchers have carried out different experiments demonstrating the development of hypothetical model capable of performing cancer diagnosis in a shorter time frame. Though very promising achievement have been recorded, but in several cases the results are different even when same dataset was used, the cause of which has not been examined hence the need to know if inherent features of the platform affect model's performance or should scientist bother about the choice of implementation platform when developing a model especially for use in healthcare industry.

This study aims to investigate whether the choice of platform for developing a machine learning model can lead to variations in the performance of predictive models. It uses the publicly available WDBC dataset from the UCI data repository to train four algorithms: Logistic Regression (LR), Decision Tree (DT), Random Forest (RF), and Gradient Boosted Trees (GB) on two different platforms, Python SciKit-Learn and Knime Analytics. The algorithms' performance was then assessed and compared. To ensure a fair basis for comparison, both platforms were operated in their default configurations. The results showed that LR achieved the highest accuracy of 92.11% in Knime, while in Python, GB achieved the highest accuracy at 97.37%. In general, all algorithms demonstrated higher accuracy in Python compared to Knime.

The significance of platform selection when implementing machine learning models cannot be underplayed. as can be deduced from the variations observed in the algorithms performances suggesting platform features plays important role in shaping results. It is therefore necessary to carefully consider the platform to use in building predictive models as this is one decision that could substantially influence model efficacy and, ultimately, patient outcomes.

[1] https://www.who.int/news-room/fact-sheets/detail/cancer

Acknowledgement

I want to seize this chance to convey my heartfelt thanks to those who have provided substantial support and guidance throughout my academic journey and the entire lifecycle of my MSc research project; their contributions have played a vital role in the successful completion of this endeavor.

My sincere gratitude goes to my supervisors Dr. Amina Souag and Prof. Konstantinos Sirlantzis for their invaluable guidance, unwavering encouragement, and profound research insights. Their expertise and mentorship provided me with clear direction and significantly enriched the quality of this research. It was great to have drawn from their wealth of very rich experience. Also thanking Dr Abdullahi Ahmed, Dr Scott Turner and all lecturers at Canterbury Christ Church University, Computing Department for their impactful teaching, mentorship, and dedication to fostering a conducive learning environment.

I would like to acknowledge the contributions of the current PhD students (Mohammed Al-alawi and Merlin Kasirajan) who provided valuable feedback and constructive criticism on my research work. Their insights helped refine my ideas and strengthen the overall quality of this study.

Lastly, I wish to express my gratitude to all my family members for their encouragement and support most importantly my spouse, Oluwatobi Olowolayemo, for her help and motivation in diverse ways all through. Her unwavering belief in my abilities was a driving force behind my perseverance during the study period. I also recognize some of my fellow course mates who I held hands together with and mutually benefitted from in ways large and small throughout this academic journey.

This research would not have been possible without the collective support, guidance, and encouragement of these individuals. I am truly fortunate to have had such remarkable mentors and well-wishers by my side.

Table of Contents

Chapter 1: Introduction

The human body is composed of trillions of cells (*NCI*, 2021) that serve as the starting point for cancer development resulting from abnormal cell cycle. During the normal cell cycle, the body cells grow, die and new cells become but due to genetic and environmental factors that interfere with this process, these cells begin to proliferate out of control, resulting in the formation of a tumour like malignant (cancerous) and benign (non-cancerous). Malignant tumours demonstrated in Figure 1 as cancer cells are masses of cells that grow out of control and spread into surrounding tissues with the potential to cause serious health problems and, as a result, become cancerous (Kavitha *et al.*, 2023). Benign tumours are cells that do not invade nearby tissues and do not spread to other body parts, rather they grow slowly and remain localised to their site of origin (Aamir *et al.*, 2022).

There are many types of cancer already identified but the few of the most common and deadly ones are breast cancer, lung cancer, colorectal cancer, lymphoma, kidney cancer, skin cancer, prostate cancer and brain cancer which causes about 10 million deaths every year globally.

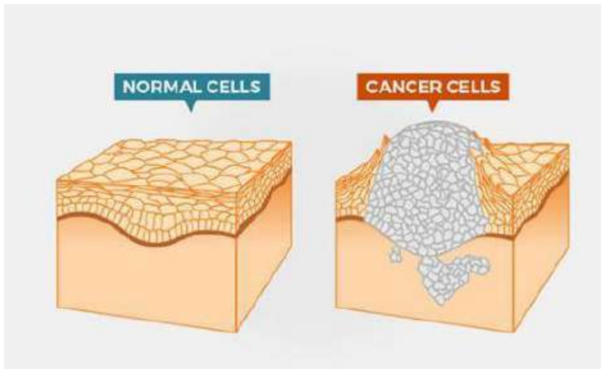


Figure 1-Pictorial view of a Normal Cell and a Cancer Cell *(*NCI, *2021)*

1.1 Background and Motivation

There are many research studies on the vast amount of work done by Machine Learning (ML) experts developing models to carry out tasks like classification, prediction, anomaly detection and regression. Researchers also assess the performances of models to understand how well they solve the problem as intended by evaluating their efficiencies in terms of various metrics.

In the healthcare sector for example, Artificial Intelligence has shown the potential to enhance and expedite the capabilities of medical professionals in radiotherapy planning, enabling them to complete the process 13 times more quickly (Oktay et al., 2020).

Developing models and assessing their performances are done on implementation platforms like Python Scikit-learn, PyTorch, Keras, Tensorflow, Knime and Weca. These have distinct characteristics and functionality features which could affect the performances of the algorithms and the choice of algorithms to adopt and deployed to solve specific problem, hence the need to critically look at the potential impact the different platforms can have on the efficiency outlook of the models.

1.2 Research Objectives

A core aspect of the research involves evaluating the potential of Machine Learning (ML) algorithms, a subset of Artificial Intelligence, as a viable solution for enhancing cancer diagnosis and prognosis. ML algorithms have shown promise in various fields, and this study seeks to determine their applicability and effectiveness in the context of cancer diagnosis, where early detection is critical for successful treatment. The main research question (RQ) that will be tackled in this study is:

RQ: Does the choice of the implementation platform affect the performance of different machine learning algorithms with Cancer data classification?

A comparative evaluation of some commonly explored ML classification algorithms, such as Logistic Regression (LR), Decision Trees (DT), Random Forest (RF), and Gradient Boosting (GB), will be conducted. The analysis will include assessment of the performance of these algorithms using key metrics such as accuracy, precision, recall, F1-Score, and Area Under Curve (AUC) on the two different implementation platforms (Python Scikit-learn and Knime analytics).

1.3 Scope and Limitations

This work will develop commonly used classification ML algorithms; LR, DT, RF, SVM, GB and train them on Wisconsin Breast Cancer datasets after which their performances based on how they were able to learn and classify accurately will be assessed. Also, the accuracy, recall, precision, and F1-Score of the algorithms would be evaluated comparatively when they are implemented on Python and Kinme Analytics platforms. This study was done without tunning the hyperparameters of the algorithms hence are, where possible, operated at their default.

Chapter 2: Literature Review

2.1 Introduction

Cancer is a global health issue with a about 2.3 million in the category of women diagnosed with breast cancer and about 683,000 deaths in 2020 only (ATEŞ and BİLGİN, 2021a).

The nature of this disease is complex and multifaceted as it affects various parts of the body, such as the breast, kidneys, brain, lungs, prostate, ovaries, and skin, and has become a major concern for both healthcare professionals and patients. Scientists have made considerable progress in understanding, diagnosing, and developing conventional medical treatments to treat or manage cancer, but all these efforts take a considerable amount of time. The delay in cancer diagnosis however renders a cancer patient's condition becomes more severe and, in many cases, irreparable leading to several death cases as exemplified in Figure 2. Consequently, scientist have been investing significant resources to finding a quicker approach to early and effective diagnosis and one of the areas enormously researched is how artificial intelligence which has proven to be immensely useful in several other industries can equally be employed in bringing down the time of diagnosis delay.

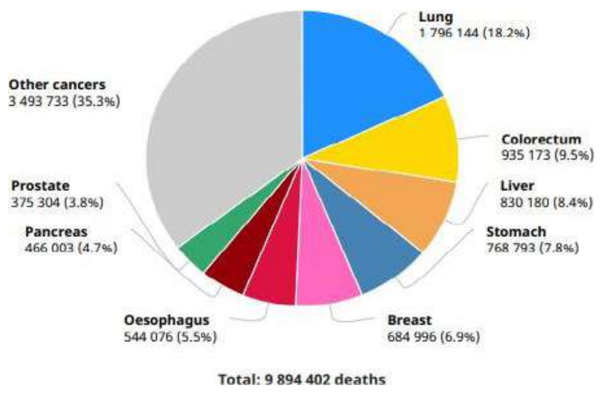


Figure 2-Cancer mortality statistics in 2020 across all ages and sex- (All cancers exclusive of non-melanoma skin cancer)

(Sung et al.*, 2021)*

2.2 Machine Learning in Medical Research

Machine learning (ML), a subset of artificial intelligence that involves the development of algorithms and models enabling systems to learn patterns from data and improve their performance over time has emerged as a pivotal tool in medical research, revolutionizing diagnostic, prognostic, and treatment strategies. The integration of advanced computational techniques has facilitated the extraction of meaningful insights from complex medical data, enhancing clinical decision-making and patient outcomes.

From image analysis for precise disease detection (Jasti et al., 2022) to predictive modeling of patient responses to therapies (Kong et al., 2022), machine learning algorithms have exhibited remarkable potential in transforming healthcare practices.

Supervised learning, an advanced computing technique, has three primary branches: supervised, unsupervised, and reinforcement learning. Supervised learning trains models with labeled data to make predictions, while unsupervised learning discovers patterns in unlabeled data. Reinforcement learning teaches agents to make decisions based on environmental feedback, like rewards or penalties. This study, however, focused on training supervised algorithms with labeled WDBC datasets and using them to predict unseen portions of the dataset.

2.3 Classification Algorithms in Cancer Studies

Researchers have explored and reported the use of various supervised Machine Learning (ML) algorithms in nearly all areas of human existence including health and medical fields. Some previous studies reviewed are briefly discussed below.

2.3.1 Algorithm Selection

Logistic Regression, a linear model is a powerful predictive analysis tool that is especially useful for binary classification (Zhu, Idemudia and Feng, 2019). (Rahman et al., 2019) examined six ML algorithms for predicting Chronic Liver Disease (CLD) and linear regression (LR) algorithm was found to be the most effective in predicting CLD based on the selected features. (Zhu, Idemudia and Feng, 2019) experimented with improved LR in the classification of binary variable and one or more independent variables to predict diabetes. (Ebrahim, Sedky and Mesbah, 2023), (Shafique et al., 2023), (Uddin et al., 2023) and countless other scientist have performed experiments regarding machine learning development involving or mainly testing LR that performs classification tasks by modulating the probability with the nonlinear (logistic) function sigmoid, which modifies any real-valued input to a range between (0 and 1) demonstrated in Figure 3.

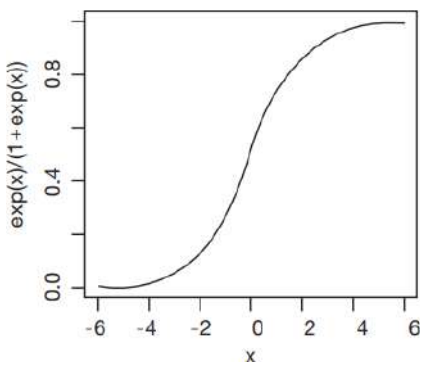


Figure 3-Graph of logistic curve where α=0 and β=1 (Park, 2013)

Likewise, Tree based algorithms including DT, RF and GBoost are widely researched with the intent of harnessing their strengths particularly in performing classification tasks. Decision Trees serve as foundational structures, offering transparency and interpretability by partitioning feature spaces into hierarchical branches thereby excelling in capturing non-linear relationships and feature interactions, enabling straightforward visualization of decision-making processes (Figure 4). Moving beyond individual trees, Random Forests amalgamate multiple Decision Trees through ensemble techniques, averting overfitting and amplifying predictive accuracy (Minnoor and Baths, 2023). By combining varied perspectives from individual trees, Random Forests provide robust generalization and robustness to noisy data. By extension, Gradient Boosting algorithms, a more advanced incarnation, embrace iterative refinement to enhance predictive performance and in particular, Gradient Boosting Trees such as XGBoost employ sequential tree fitting to target the residuals of prior iterations, systematically improving model predictions. These algorithms excel in modeling complex relationships, accommodating non-linearities, and excelling in predictive accuracy across domains (Li et al., 2019) (Wan, 2019).

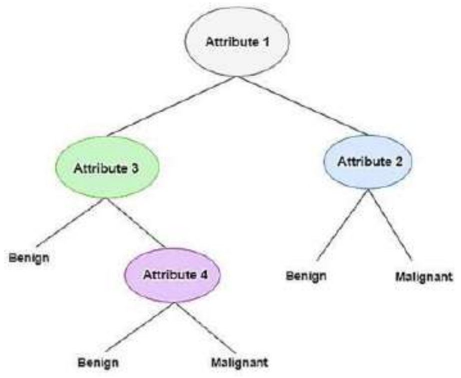


Figure 4- A two level decision tree sample *(Ghennioui* et al.*, 2019)*

2.3.2 Train-Test Split

In terms of model selection, (Ebrahim, Sedky and Mesbah, 2023) explored five classification and three deep learning algorithms including DT, LR, K-Nearest Neighbour (KNN), SVM, Recurrent Neural Networks (RNN) and Ensemble with Train/Test split of 70:30 and 90:10 of which DT and Ensemble proved higher accuracy both before and after feature selection. DT however did not perform as much in Kidney Cancer Lung Metastasis prediction by (Yi et al., 2023) trained with 52,222 data from SEER database and 492 hospital patient data with Train/Test split of 70:30 performing at accuracy of 82% significantly lower than in other studies compared.

2.3.3 Performance Metrics

When models are being developed to answer research questions or to be deployed to provide solutions to specific problems, the reliance on the model can only be justified if it is efficient in truly doing what it was developed for. Algorithm efficiencies are assessed through diverse statistical means viz-a-viz its accuracy and other deduced measures from its confusion matrix like the precision, recall, and F1-score. The performance metrics, as can be seen in most of the literature reviewed, are commonly assessed to justify the efficiencies of the models and this is evidenced in Table 1.

Confusion Matrix as illustrated in Figure 5 is a tabular representation that summarizes the model's predictions against the actual outcomes. It is particularly useful for calculating precision, recall, and F1-score..

- True Positive (TP): Instances correctly predicted as positive.
- True Negative (TN): Instances correctly predicted as negative.
- False Positive (FP): Instances incorrectly predicted as positive.
- False Negative (FN): Instances incorrectly predicted as negative.

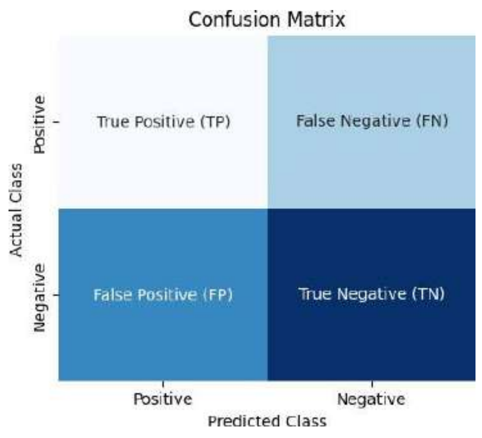


Figure 5- Illustrative Confusion Matrix Table

The computation of the performance metrics is shown below using the confusion matrix:

Equation 1- □□□□□□□□□ = □□/(□□ + □□)

Equation 2- □□□□□□ = □□/(□□ + □□)

Equation 3- □1 □□□□□ = 2 * (□□□□□□□□□ * □□□□□□)/(□□□□□□□□□ + □□□□□□)

Equation 4 □□□□□□□□ = (□□ + □□)/(□□ + □□ + □□ + □□)

- Accuracy

Accuracy measures the ratio of correctly predicted instances to the total number of instances in the dataset. It provides a general overview of how well the model predicts across all classes.

High accuracy might indicate good overall performance, but it is insufficient for imbalanced datasets (Cui et al., 2023). Though a common metric, it may be misleading when used to measure a model exposed to an imbalanced dataset. It is, however, useful when dealing with a balanced class distribution.

- Precision

Precision is the proportion of true positive predictions (correctly predicted positive instances) out of all instances predicted as positive by the model Equation 1.

Precision focuses on how accurate the positive predictions are. That is, when the precision score is high, the chances of the positive prediction to be correct is equally high (Bailly et al., 2022). It is crucial in applications where false positives are costly. High precision is important when false positives are costly, like in medical diagnoses.

- Recall

Recall, also known as (sensitivity) is the measure of the proportion of true positive predictions out of all actual positive instances (Boukhatem, Youssef and Nassif, 2022), Equation 2. This is an important metric in the medical field where the cost of missing positive instances is high for example if a malignant tumour is wrongly classified as benign suggesting that a rather dangerous condition is not so much a problem, this is a very costly error. High recall indicates that the model effectively captures most of the positive instances.

- F1-Score

F1-score balances precision and recall thereby providing a balanced measure of a model's performance. It considers both the false positives and false negatives; hence, it is an appropriate metric when a class imbalance exists in the dataset.

2.4 Dataset

Data is a principal factor in machine learning research; therefore, its availability, type, quality, and quantity of data have significant impact on being able to develop a model, the effectiveness and generalization of a model, and fundamentally shaping model's predictive performance and real-world utility.

The Wisconsin Breast Cancer dataset (WDBC) has been heavily explored by scientists and stands as a compelling testament to the profound influence of machine learning within the realm of healthcare. Its significance becomes particularly evident in the context of binary tumour classification, specifically in discerning between benign and malignant tumours—a pivotal task for the timely detection of cancer and the formulation of effective treatment strategies.

While numerous recent studies like (Omondiagbe, Veeramani and Sidhu, 2019), (Shafique et al., 2023), (Uddin et al., 2023) have utilized open-source datasets like the WDBC, which typically comprises fewer than 600 records and 30 features available from sources such as UCI, Kaggle, and TCGA, (Ebrahim, Sedky and Mesbah, 2023) diverged by utilizing a considerably larger dataset Obtained from the National Cancer Institute (NIH) in the USA, this dataset contained approximately 1.7 million records and 210 features, which were subsequently refined through feature selection to yield 70,079 records and 107 relevant breast cancer-related features for the specific classification task of categorizing tumors as malignant or benign. It is important to note that the dataset's size should be weighed against its quality; a smaller dataset, characterized by precision, representativeness, and cleanliness, can surpass a larger dataset replete with noise or extraneous information. This disparity in performance is evident in the algorithm accuracy achieved using the open-source datasets often employed in the field proving to have higher accuracies at 99.12%, 99.67% and 100% compared to Ebrahim et al. model with a lower accuracy of 98.7%.

2.5 Implementation Platforms Overview

Based on studies reviewed so far, none was seen to have previously worked on assessment of implementation platform impact on algorithms performance. While Knime Analytics2 promises a friendly user interface that allows the construction of complex workflows and is a no-code Data Science tool with "strong compatibility with various art tools, including Weka, Matlab, ImageJ, R, and Python" (Althubaity et al., 2023). (Shamu et al., 2022) in their work to study Cancer incidence among people living with HIV in Zimbabwe used Knime entirely for all stages of machine learning research from data collection to model development and performance assessment. Notwithstanding, Python has emerged as a preeminent programming language in the domain of machine learning due to its expansive ecosystem, ease of use, and powerful libraries particularly SciKit-Learn as much of the literatures including Liu and Maheshet al., showed this tool was mostly used. In either case, the two platforms have their advocates, and both are receiving huge appraisals by scientists using them.

2.6 Conclusion

In this chapter, we reviewed the current literature related to the use of Machine Learning in Medical research and Cancer classification. The review demonstrates that research in this area has been highly active over the last five years in implementing different ML classification algorithms with various sources of Cancer Data.

The review also confirms that there is a gap. To the best of our knowledge, there is no research that addressed specifically the *impact of the implementation platform on machine learning algorithms performance in Cancer classification*. This problem will be the focus of our contribution and will be explored and explained in detail in the next chapter.

Table 1- Comparative review of other studies that used Machine Learning Techniques in Cancer Research

Author	Year	Data Source	Records/Features	Train/Test Split	Feature Engineering Technique	Implementation Platform	Algorithm Type	Validation Type	Model Accuracy	Other Performance Metrics
(Ebrahim, Sedky and Mesbah, 2023)	2023	National Cancer Institute (NIH), USA	70,079/107	70:30 & 90:10	-	Python	DT, LR, SVM, LD, ET, PNN	K-Fold (K=10)	98.7%	Recall, Precision, F1 Score
(Shafique et al., 2023)	2023	Kaggle	569 /30	75:25	PCA, SVD, Chi-square	-	RF, SVM, GBM, LR, MLP, KNN	K-Fold (K=10)	100%	-
(Uddin et al., 2023)	2023	UCI	569 /30	70:30	PCA, Standard Scalar	Python	SVM, RF, KNN, NB, DT, LR, AB, GB, MLP, NCC, VC[2]	K-Fold (K=10)	98.7%	Recall, Precision, F1 Score, AUC, ROC
(Zhang et al., 2022)	2022	TCGA	604	-	-	R & Python	RF, SVM, libD3C	K-Fold (K=10)	99.67%	Recall, Precision, F1 Score, ROC
(Aamir et al., 2022)	2022	UCI	569/26	80:20 & 70:30	Z-Score Normalization	Python & Tensor Flow	RF, GB, SVM, ANN, MLP	K-Fold (K=5)	99.12%	-
(Yi et al., 2023)	2023	SEER& Southwest Hospital, China.	52,714	70:30	SEERStat Software		LR, XGB, RF, SVM, ANN, DT	K-Fold (K=10)		Recall, Precision, F1 Score, AUC

[2] VC- Voting Classifier

Mahesh *et al*., 2022	2022	Kaggle	143/10	70:30	SMOTE	Python	NB, AltDT, RedEPT, RF	-	98.20%	TTMB, F1 Score
(ATEŞ and BİLGİN, 2021a)	2021	Kaggle	569/30	70:30	-	Knime	NB, DT, MLP	-	96.5%	Sensitivity, Specificity, Precision, F1 Score
(Minnoor and Baths, 2023)	2023	UCI	569/30	80:20	Pearson Correlation, RFE	-	RF, SVM, DT, MLP, KNN	K-Fold (K=10)	100%	Recall, Precision, F1 Score, ROC-AUC
(Ara, Das and Dey, 2021)	2021	UCI	569/30	75:25	-	-	SVM, LR, KNN, DT, NB, RF	-	96.5%	-
(Liu, 2018)	2018	UCI	569/30	75:25	-	Python	LR	-	96.5%	-

Chapter 3: Investigating the Impact of the Implementation Platform on Machine Learning Algorithms Performance – Methodology and Results.

3.1 Introduction

To address the main research question, the research methodology presented in Figure 6 has been followed. The methodology for this study involves a series of systematic steps aimed at performing a comparative analysis of machine learning algorithms using the WDBC dataset[3] and two implementations

Platforms in the following sequence: Data Collection, Data Exploration, Feature Engineering, and feature selection were performed with a "combination of filtering and random forest selection techniques to select important features" (et al., 2021). Lastly, before model development, the data was split into Train-Test Split: 80% training, 20 % test.

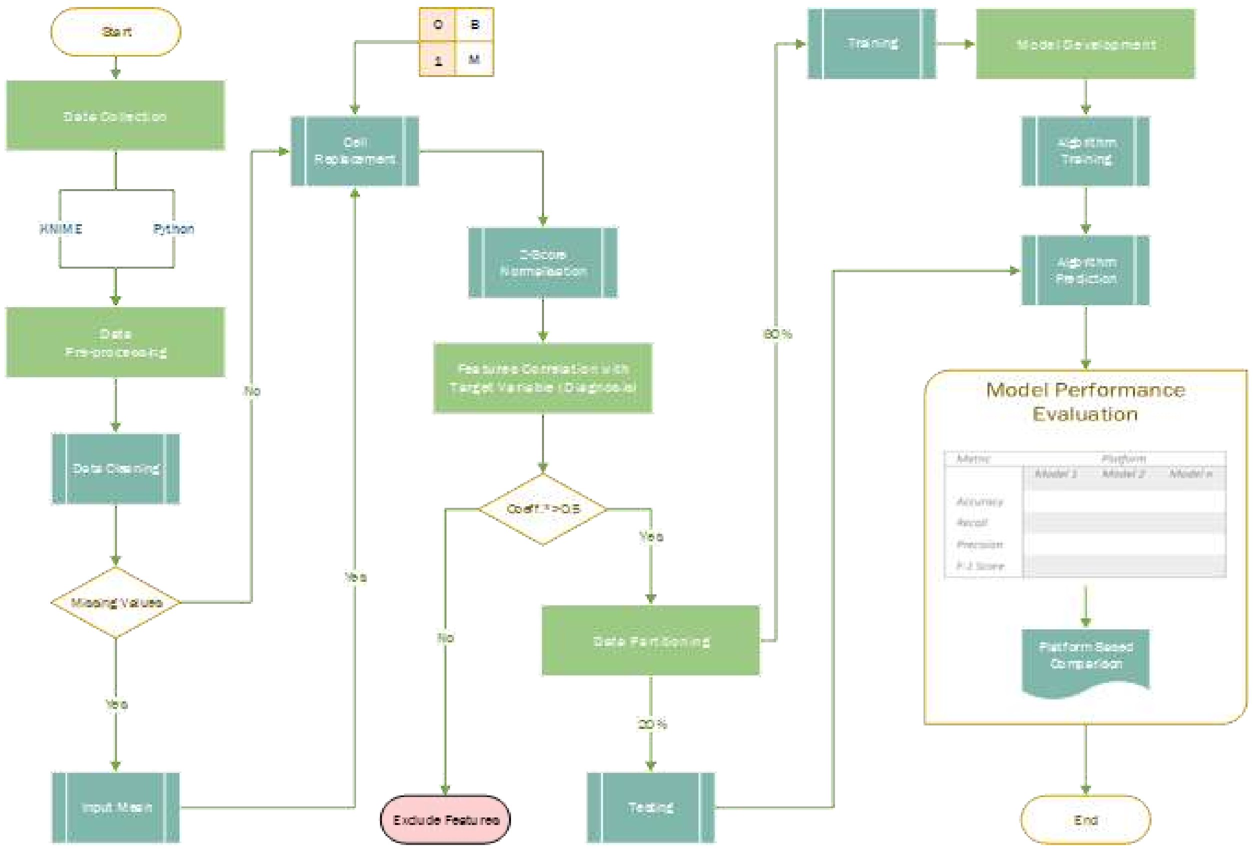
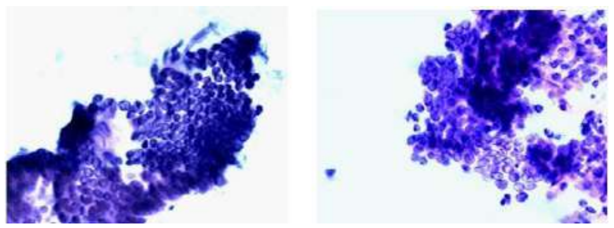


Figure 6-Research Methodology Flowchart

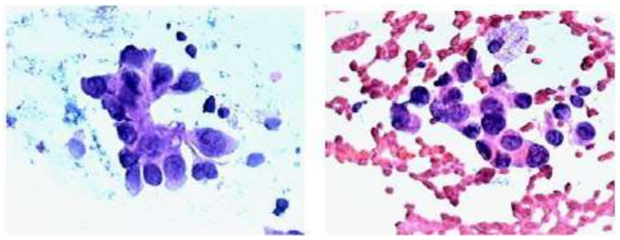
3 https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic

3.2 Data Collection and Preprocessing

The Wisconsin Diagnostic Breast Cancer (WDBC) a dataset widely used in machine learning research due to its real-world relevance, binary classification task, and data quality was obtained from a medical research study and made publicly available on UCI (open source) Machine Learning repository in comma-separated values (csv) format was acquired. The dataset is multi-dimensional, comprising 569 occurrences and 30 attributes, which were extracted from a digitized image of a Breast Mass Fine Needle Aspiration (FNA) specimen (Wolberg, et al., 1995). FNA- a medical procedure in which a fine needle is used to remove small amounts of breast tissue or breast fluid from an area that has been suspected of being cancerous. WDBC attributes include "Diagnosis" which contains categorical values; "M" for Malignant Cell and "B" for Benign Cell used for classifying breast cancer as either malignant or benign as well as features such as "radius_mean", "texture_mean", "perimeter_mean" among others various measurements from cell nuclei present in biopsy images (Index of /math-prog/cpo-dataset/machine-learn/cancer/cancer_images, 1994).



(a)



(b)

Figure 7 - (a) Benign cancer cell sample images, (b) Malignant cancer cell sample images (ATEŞ and BİLGİN, 2021)

3.2 Data Preprocessing

For a global view of the dataset's structure, the data was saved into python dataframe and named "breast" after which the code *breast.info ()* was passed to give output Figure 7 showing there is no null value in the data.

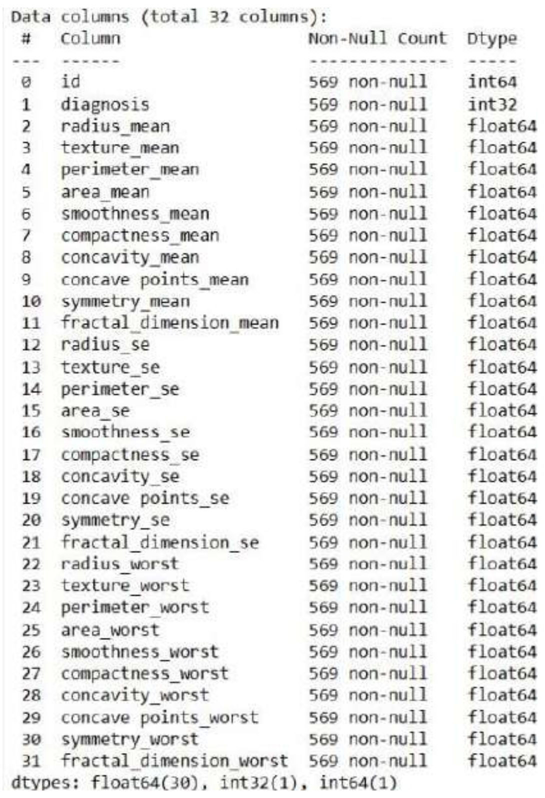

```
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    int32
 2   radius_mean              569 non-null    float64
 3   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
dtypes: float64(30), int32(1), int64(1)
```

Figure 8- Dataset exploration showing features, data type and test for null values.

Handling Outliers

Further, to test if there is (are) outliers able to disproportionately influence the training process resulting in decreased generalization of the model, box plotting of the data was done which revealed there are outliers as seen in Figure 8, and this was treated by using Capping method, a technique used by (Feng, Cai and Xin, 2023) which upon definition of both the lower Q1 and upper quantile Q3, assigns the value of the lower whisker of the box plot to datapoints with value smaller than the lower whisker while datapoint values higher than the upper whiskers were assigned values of the upper whiskers.

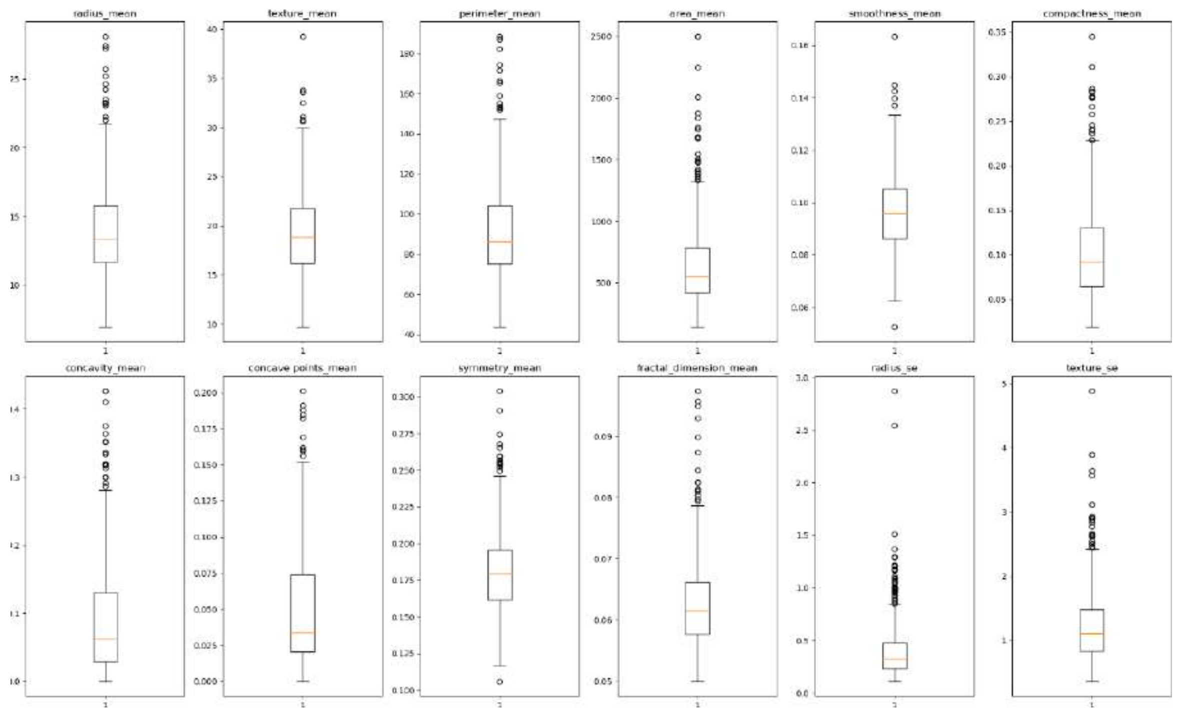


Figure 9- Box plot of some of the features showing outliers.

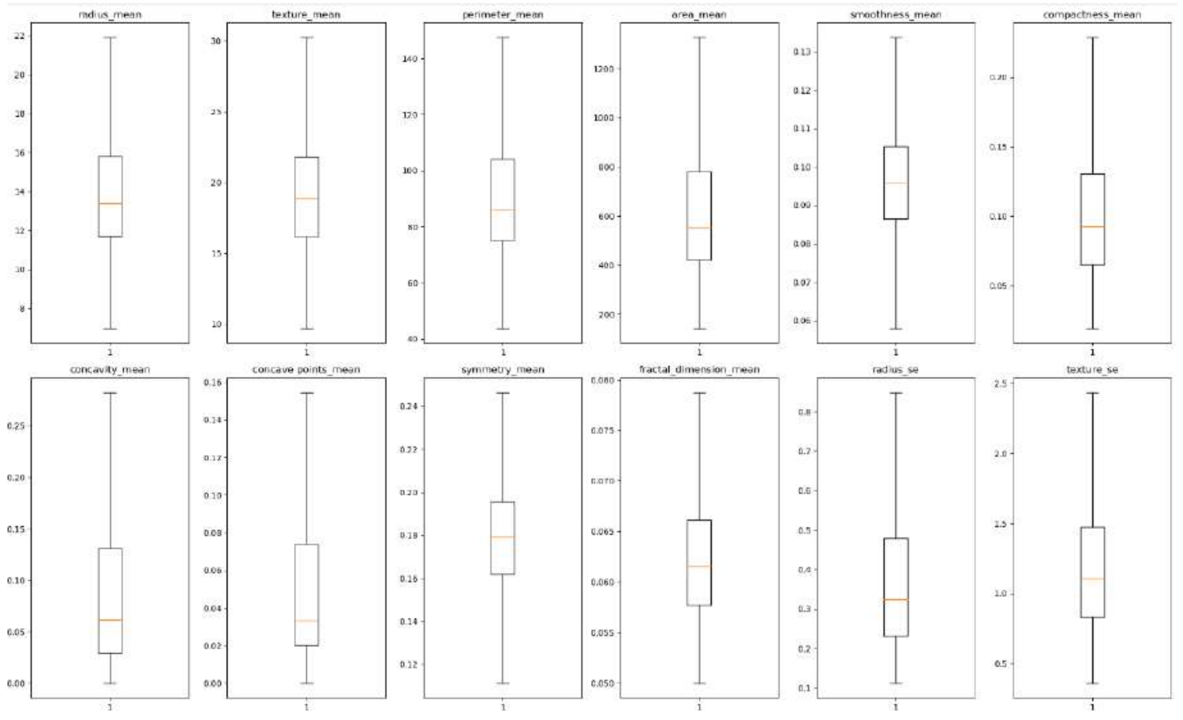
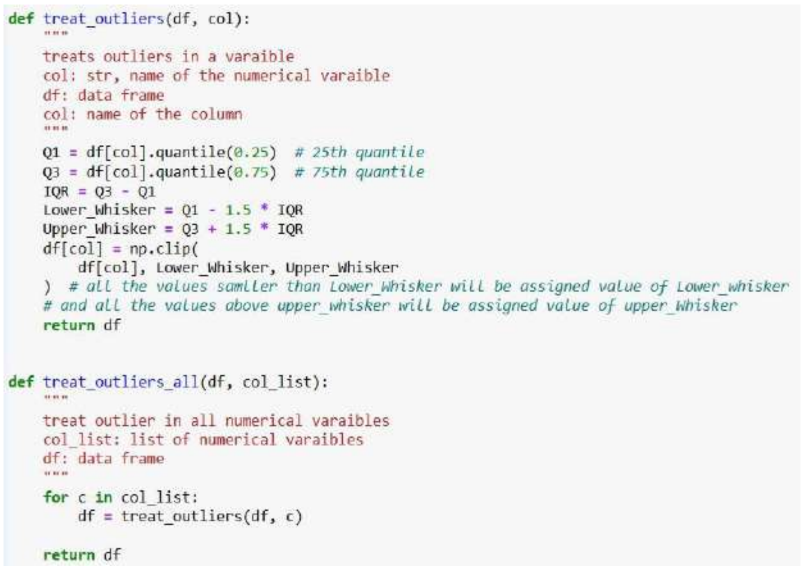


Figure 10-Box plot of some of the features after treating the outliers.

Capping of the outlier was achieved by defining a python function as shown below.



```
def treat_outliers(df, col):
    """
    treats outliers in a varaible
    col: str, name of the numerical varaible
    df: data frame
    col: name of the column
    """
    Q1 = df[col].quantile(0.25)  # 25th quantile
    Q3 = df[col].quantile(0.75)  # 75th quantile
    IQR = Q3 - Q1
    Lower_Whisker = Q1 - 1.5 * IQR
    Upper_Whisker = Q3 + 1.5 * IQR
    df[col] = np.clip(
        df[col], Lower_Whisker, Upper_Whisker
    )  # all the values samller than Lower_Whisker will be assigned value of Lower_whisker
    # and all the values above upper_whisker will be assigned value of upper_Whisker
    return df


def treat_outliers_all(df, col_list):
    """
    treat outlier in all numerical varaibles
    col_list: list of numerical varaibles
    df: data frame
    """
    for c in col_list:
        df = treat_outliers(df, c)

    return df
```

Figure 11-Python code defining function to treat outliers using the cap method.

Normalization

Z-Score Normalization (Standardization) technique was for feature scaling to transform the data such that values of each feature has a mean of 0 and standard deviation of 1 (Sumin et al., 2022) (Pagan, Zarlis and Candra, 2023).

Scaling the data points to same scale is an essential preprocessing step that plays a pivotal role in facilitating fair and accurate learning across various machine learning algorithms. The scale of features directly impacts the behavior and performance of algorithms, influencing how they interpret and process data. When features are not on the same scale, algorithms may assign disproportionate importance to features with larger magnitudes, inadvertently biasing the results. Doing this will ensure that each feature contributes equally to the learning process, preventing any single feature from dominating model decisions.

The equation below represents the computation formula for z-score standardization.

Equation 5 $□ = \frac{□−□}{□}$ *(*Statology*, 2023)*

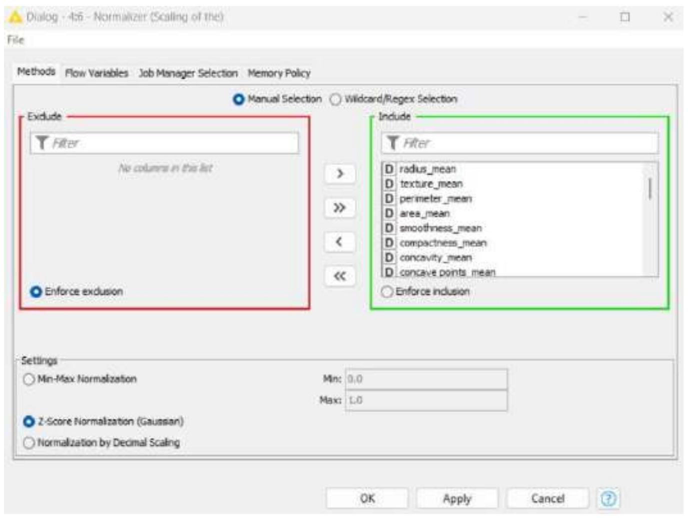
where z is the scaled value of the feature,

x is the original value of the feature,

μ is the mean value of the feature, and

σ is the standard deviation of the feature.

From the SciKit-Learn module, scikit-learn-preprocessing, z-score is accessible through StandardScalar Class while on Knime z-score option was selected as in Figure 12.


Dialog - 4:6 - Normalizer (Scaling of the)

Methods | Flow Variables | Job Manager Selection | Memory Policy

[✓ Manual Selection] Wildcard/Regex Selection

Exclude: No columns in this list — [✓ Enforce exclusion]

Include:
- D radius_mean
- D texture_mean
- D perimeter_mean
- D area_mean
- D smoothness_mean
- D compactness_mean
- D concavity_mean
- D concave points_mean

Enforce inclusion

Settings
- Min-Max Normalization — Min: 0.0 Max: 1.0
- [✓ Z-Score Normalization (Gaussian)]
- Normalization by Decimal Scaling

Figure 12-Knime Analytics Platform Node settings dialog showing the selection of Z-Score option for standardizing the WDBC

3.3 Exploratory Data Analysis

An analysis of the data showed it is grouped into two classes where Benign tumours made up 62.7% of the total instances while the cancerous tumour, malignant class comprise 37.3%.

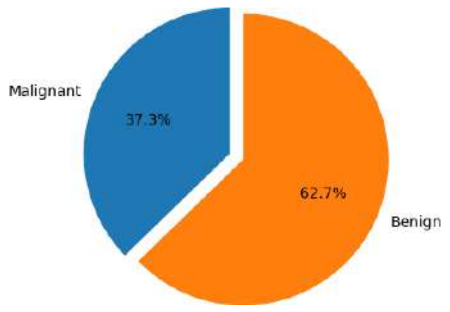


Figure 13- Pie chart showing percentage composition of the class labels as M-malignant and B-benign.

Following this, correlation analysis was performed to assess the relationship between each of the features shown in the heatmap below. This is an important step preceding feature selection as it enables researchers to have a view of features that are independently related to the target variable.

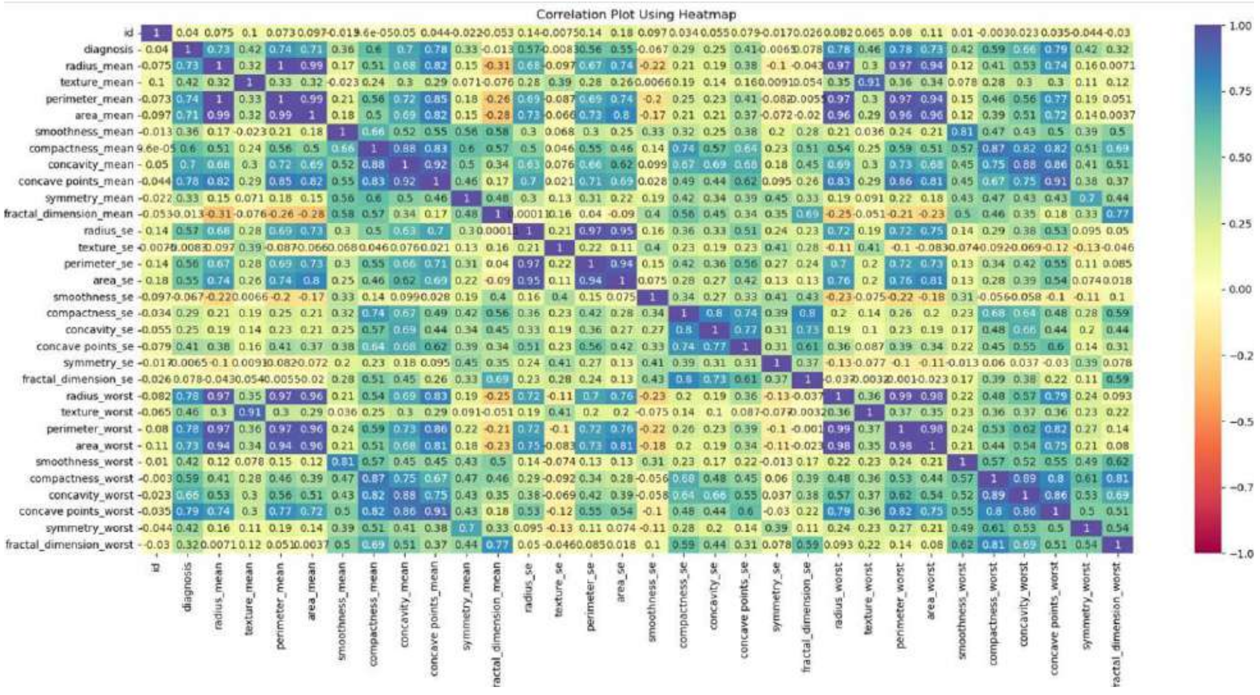


Figure 14-Heatmap of WDBC features correlations.

Also, a granular look at the relationship between individual features was carried out to understand the resultant effect that the change in one feature would have on the other and identify if any two independent features are strongly correlated which can indicate they would provide similar information consequently adding less value to the model for effective prediction.

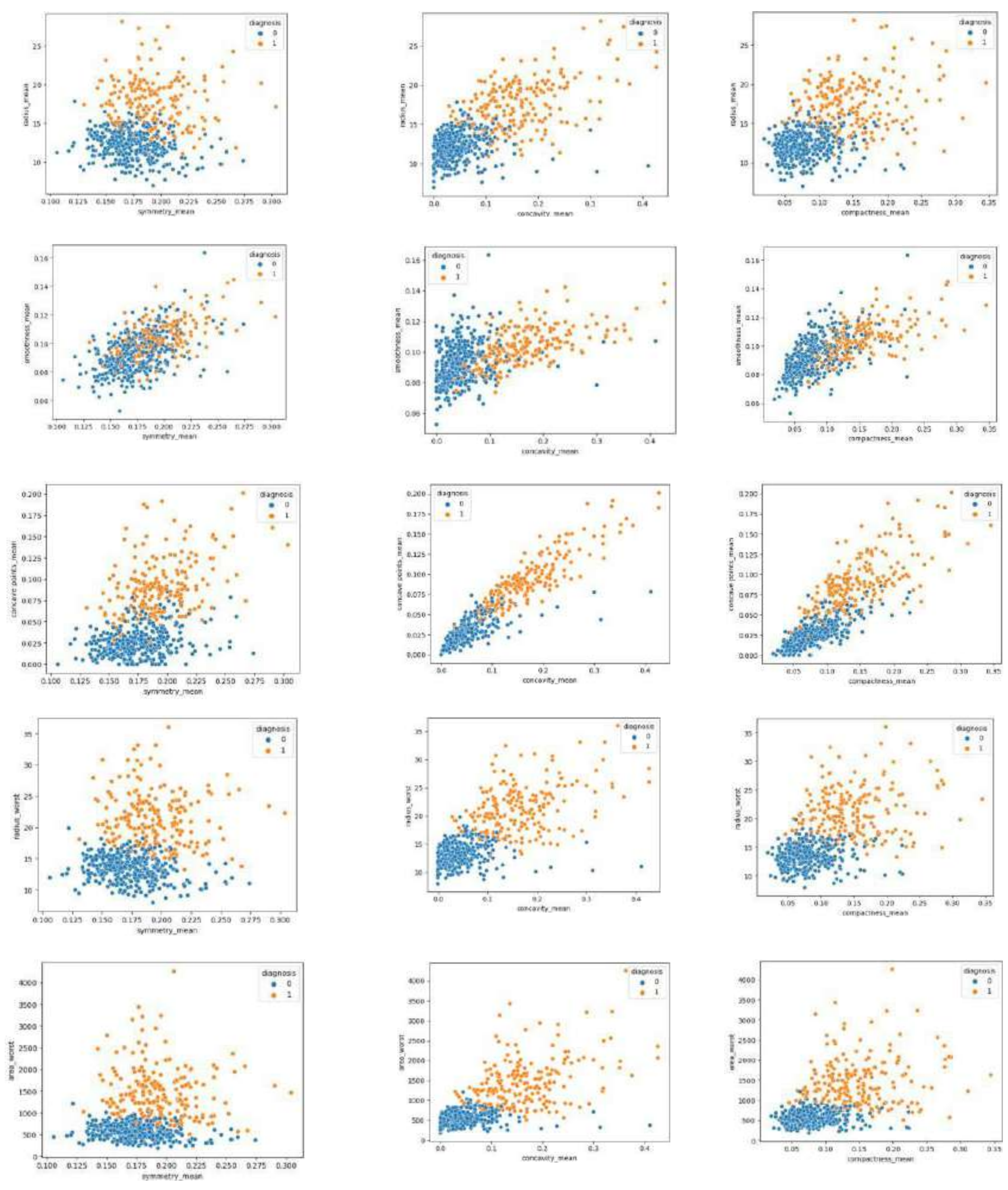


Figure 15-Scatterplot of relationship between some of the features

3.4 Feature Engineering

Selecting the important features is another critical step in model development research (Cai *et al.*, 2018) and this was done in this work by firstly applying the Filter Method of features selection (Faisal, Zamzami and Sutarman, 2020) (Ara, Das and Dey, 2021) to evaluate the dataset features according to their correlation ranking scores in relation to the target variable. Features with coefficient ≤ 0.5 were eliminated, while features with correlation score above this threshold were included in the selection. (Pudjihartono *et al.*, 2022). To do this, the features correlation coefficients were assessed using a two-sided p-value Rank correlation between the independent features and the target variable – "diagnosis" Output revealed that three attributes; *symmetry_se*, *smoothness_se* and *fractal_dimension_mean* have negative correlation while other 27 attributes are positively correlated with correlation values range between 0.02 and 0.80.

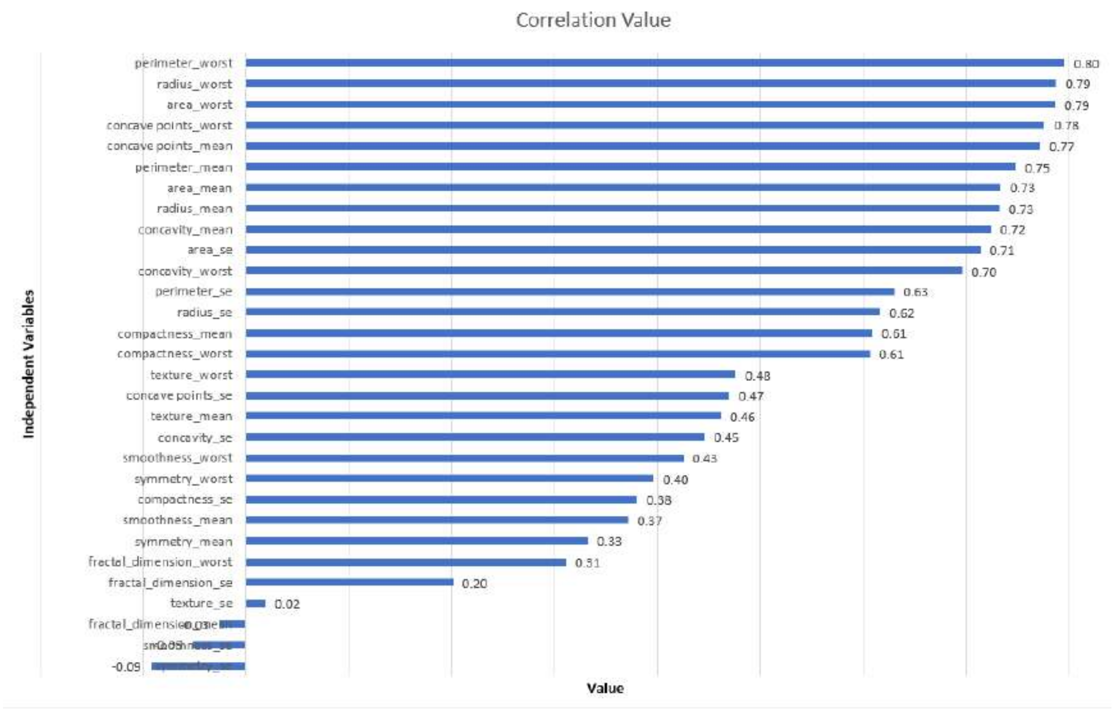


Figure 16-Chart of the correlation values of all the independent features with the target variable

Out of the 30 features, 15 features having correlation coefficient above 0.5 were selected for further steps.

Secondly, Tree-Based Method was used to confirm the features arrived at by the filtering method. The technique offers distinct advantages over other feature selection techniques. By ranking features based on their contribution to model performance, tree-based methods strike a balance between interpretability, computational efficiency, and capturing both linear and non-linear relationships, making them versatile and advantageous in various machine learning scenarios. Hence Random Forest Classifier, being a tree-based model with inherent ability to perform feature selection was used and the result is shown Figure 17 below.

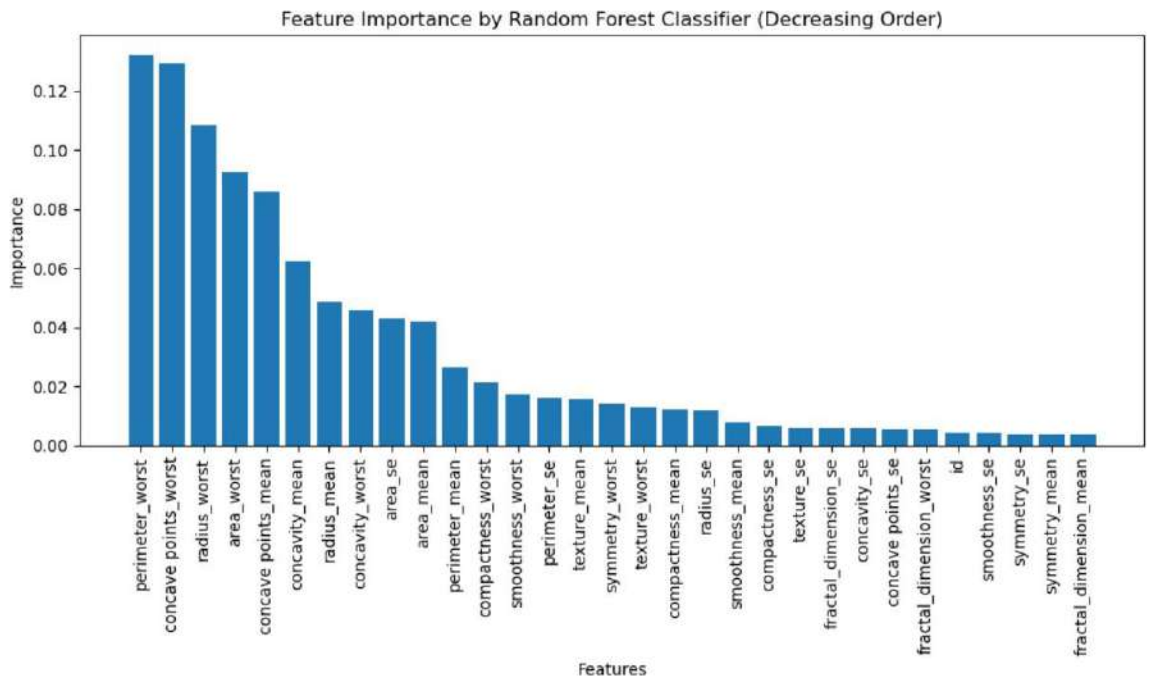


Figure 17-Bar plot of the features showing their importance.

3.5 Model Selection and Implementation

3.5.1 Model Selection

Four popular Supervised ML classification algorithms were selected for experimentation based on their popularity among researchers who have used them in various work as deduced from different literatures reviewed and the algorithms unique quality in principle.

Logistic Regression (LR)- a common algorithm used in various research specifically studies involving binary classification for its interpretability and low complexity was chosen as one of the algorithms to test for same reason of being a linear algorithm that is computationally efficient and less subjected to overfitting.

DT, RF and GB are all Tree Based algorithms also tested in this experiment were chosen based on their principle involving a recursive partitioning process aiming to identify the best feature and a split point where the class label separates most efficiently.

3.5.2 Implementation Python Scikit-Learn & Knime Analytics

Experiment was performed on Knime Analytics Platform Version 4.7.6 and Python version 3.11.4 (Jupyterlab). As much as possible, the algorithms hyperparameters were operated at default except in circumstances where the default settings did return no output, or the output was not reasonable based on general knowledge.

Train-Test Split

A train-test split of 80:20 was used. This holdout validation method which involves allocating 80% of the dataset for training purposes, enabling the algorithms to learn patterns and relationships from the data, and reserving the remaining 20% for testing, thereby evaluating the models' generalization ability to unseen instances.

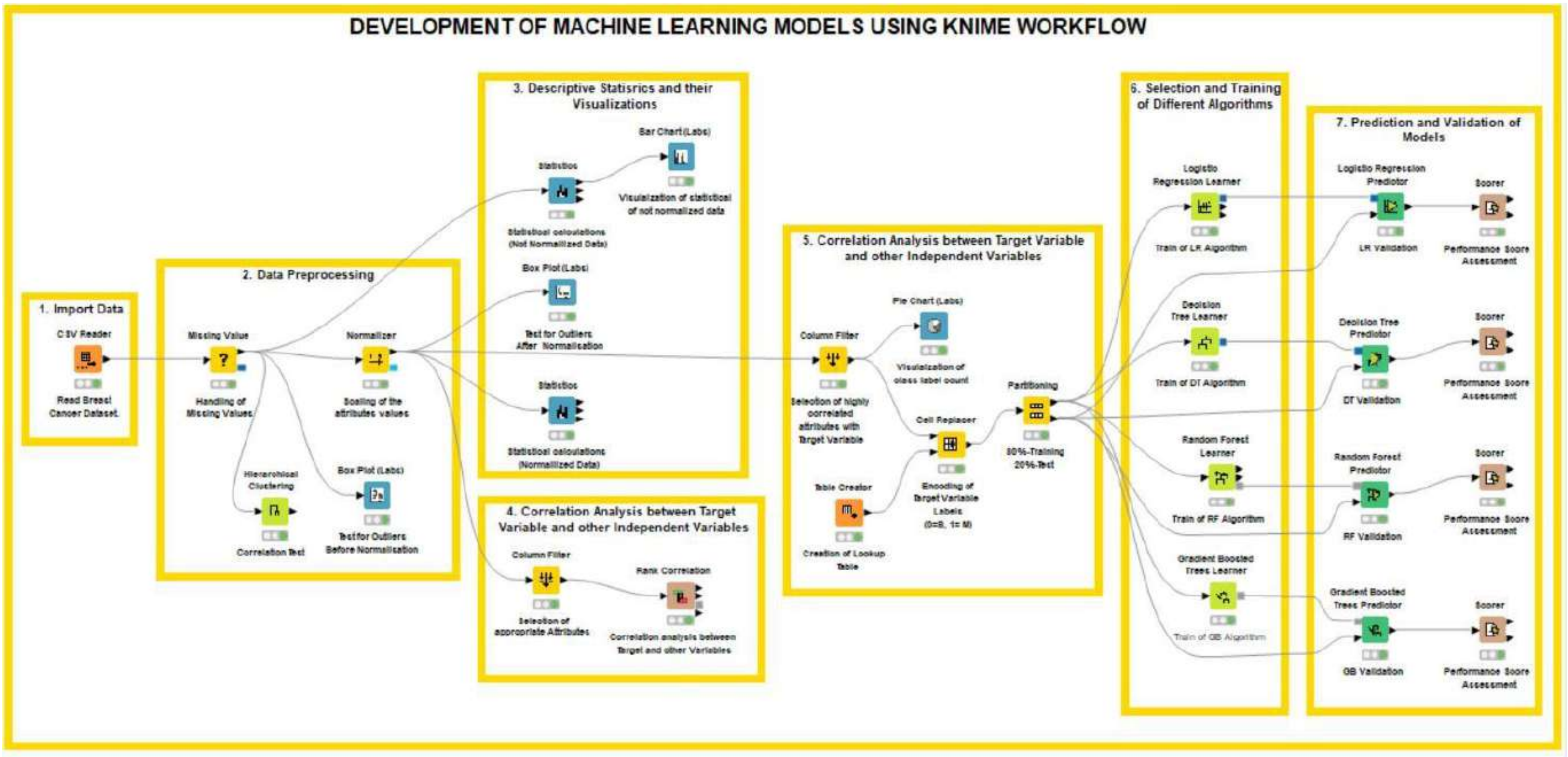

DEVELOPMENT OF MACHINE LEARNING MODELS USING KNIME WORKFLOW
1. Import Data
CSV Reader
Read Breast Cancer Dataset.
2. Data Preprocessing
Missing Value
Handling of Missing Values
Normalizer
Scaling of the attributes values
Hierarchical Clustering
Correlation Test
Box Plot (Labs)
Test for Outliers Before Normalisation
3. Descriptive Statisrics and their Visualizations
Bar Chart (Labs)
Visualization of statistical of not normalized data
Statistics
Statistical calculations (Not Normalized Data)
Box Plot (Labs)
Test for Outliers After Normalisation
Statistics
Statistical calculations (Normalized Data)
4. Correlation Analysis between Target Variable and other Independent Variables
Column Filter
Selection of appropriate Attributes
Rank Correlation
Correlation analysis between Target and other Variables
5. Correlation Analysis between Target Variable and other Independent Variables
Column Filter
Selection of highly correlated attributes with Target Variable
Pie Chart (Labs)
Visualization of class label count
Cell Replacer
Encoding of Target Variable Labels (0=B, 1= M)
Table Creator
Creation of Lookup Table
Partitioning
80%-Training 20%-Test
6. Selection and Training of Different Algorithms
Logistic Regression Learner
Train of LR Algorithm
Decision Tree Learner
Train of DT Algorithm
Random Forest Learner
Train of RF Algorithm
Gradient Boosted Trees Learner
Train of GB Algorithm
7. Prediction and Validation of Models
Logistic Regression Predictor
LR Validation
Decision Tree Predictor
DT Validation
Random Forest Predictor
RF Validation
Gradient Boosted Trees Predictor
GB Validation
Scorer
Performance Score Assessment
Scorer
Performance Score Assessment
Scorer
Performance Score Assessment
Scorer
Performance Score Assessment

Figure 18-Knime Workflow of the ML model development.

Chapter 4: Results and Discussion

This section presents the results of the experiment conducted using the various machine learning algorithms, implemented in both Knime and Python's SciKit-Learn followed by the evaluation of key performance metrics including Accuracy, Recall, Precision, and F1 Score, which are critical measures for assessing the quality of a classification model.

4.1 Results

Table 2 shows the algorithms performance when implemented on SciKit-Learn while Table 3 and Table 4 shows results obtained from Knime and comparative assessment of algorithms on the two platforms respectively.

Table 2-Table of Algorithms Performances on SciKit-Learn

Algorithm	Accuracy	Recall	Precision	F1-Score
LR	0.956	0.929	0.951	0.940
DT	0.965	0.952	0.952	0.952
RF	0.947	0.976	0.891	0.932
GB	0.974	0.976	0.953	0.965

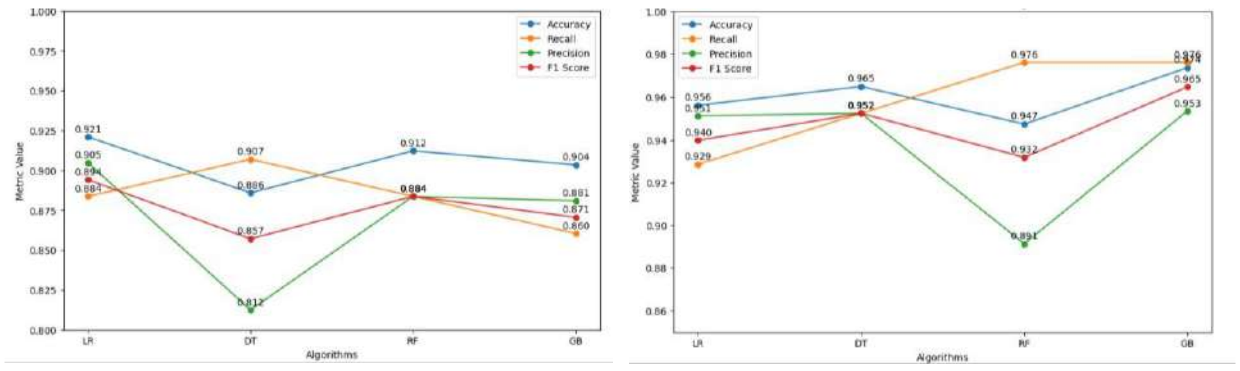
Table 3-Table of Algorithms Performances on Knime

Algorithm	Accuracy	Recall	Precision	F1-Score
LR	0.921	0.884	0.905	0.894
DT	0.886	0.907	0.813	0.857
RF	0.912	0.884	0.884	0.884
GB	0.904	0.861	0.881	0.871

Table 4-Comparative assessment of model performance on the two platforms

Algorithm	Tool	Accuracy	Recall	Precision	F1-Score
LR	SciKit-Learn	0.956	0.929	0.951	0.940
	Knime	0.921	0.884	0.905	0.894
DT	SciKit-Learn	0.965	0.952	0.952	0.952
	Knime	0.886	0.907	0.813	0.857
RF	SciKit-Learn	0.947	0.976	0.891	0.932
	Knime	0.912	0.884	0.884	0.884
GB	SciKit-Learn	0.974	0.976	0.953	0.965
	Knime	0.904	0.861	0.881	0.871

Figure *19* is the visualization of results collected and presented in tables 2 and 3 helping to also see the pattern of metrics for the model on both tools.



Knime Python

Figure 19- Platform based overview of all assessed performances for each of the algorithms, (a) Knime and (b) Python.

Confusion matrix comprising the True Positive, True Negative, False Positive and False negatives values from the experiment is shown in Table 5. Values in the table were used in computing and the recall, precision, and F1 Score.

Table 5- Platform based Confusion Matrix of the algorithms.

Knime

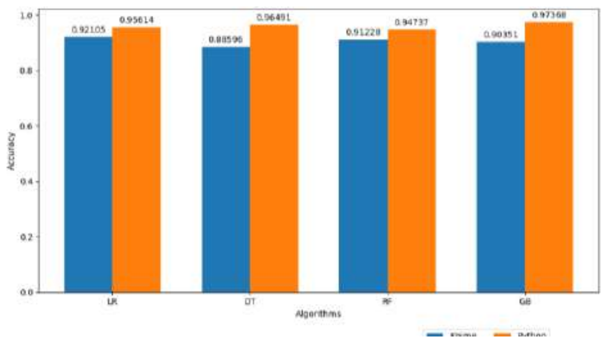
	Logistic Regression		**Decision Tree**	
	Negative	Positive	Negative	Positive
Negative	67	4	62	9
Positive	5	38	4	39

	Random Forest		**Gradient Boosting**	
	Negative	Positive	Negative	Positive
Negative	66	5	66	5
Positive	5	38	6	37

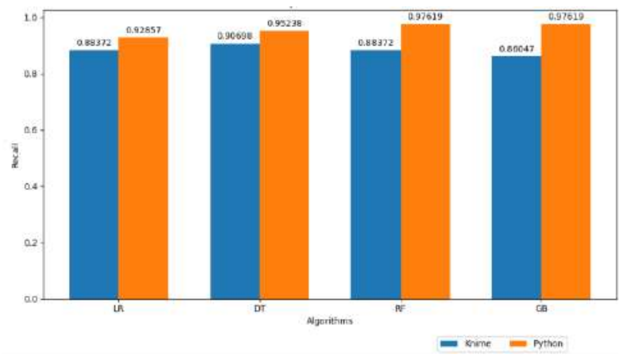
SciKit-Learn

	Logistic Regression		**Decision Tree**	
	Negative	Positive	Negative	Positive
Negative	70	2	70	2
Positive	3	39	2	40

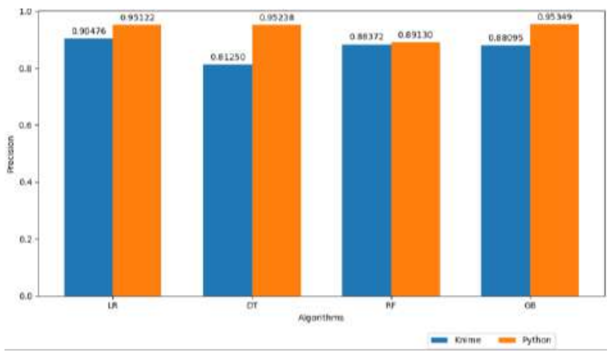
	Random Forest		**Gradient Boosting**	
	Negative	Positive	Negative	Positive
Negative	67	5	70	2
Positive	1	41	1	41



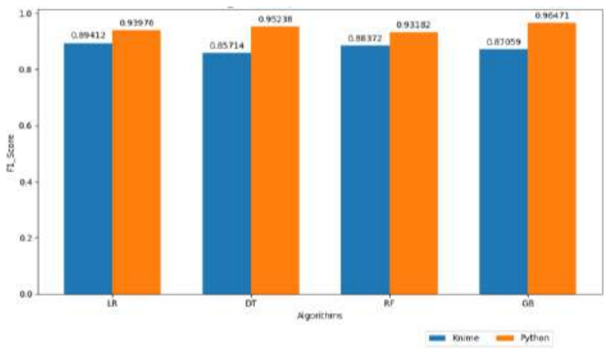
(a)



(b)



(c)



(d)

Figure 20-Comparison of all the algorithms performance on both platforms. *(a) Accuracy (b)Recall, (c) Precision, and (d) F1-Score*

4.2 Discussion

In the KNIME Analytics platform, the LR algorithm achieved an Accuracy of 0.92105, with Recall, Precision, and F1 Score of 0.88372, 0.90476, and 0.89412, respectively signifying that the model correctly classified approximately 92.11% of the instances. In comparison, the DT algorithm demonstrated a slightly lower Accuracy of 0.88596, yet it exhibited higher Recall (0.90698) and F1 Score (0.85714), suggesting that it is proficient in capturing true positive instances while maintaining a balance between precision and recall, although its Precision score was 0.81250, indicating a relatively lower ability to avoid false positives. The RF algorithm on the other hand achieved an Accuracy of 0.91228, almost on par with LR. It yielded Recall, Precision, and F1 Score of 0.88372, 0.88372, and 0.88372, respectively, presenting consistent performance across the metrics. The GB algorithm, similar to DT, secured an Accuracy of 0.90351, while it demonstrated a Recall of 0.86047, Precision of 0.88095, and F1 Score of 0.87059 reflecting a balanced trade-off between sensitivity and precision, critical in medical diagnosis scenarios.

Whereas on Python platform, the LR model exhibited superior performance, with an Accuracy of 0.95614. This signifies a substantial enhancement in predictive accuracy when compared to its counterpart in KNIME Analytics. The Recall 0.92857, Precision 0.95122, and F1 Score 0.93976 further validate the model's proficiency in correctly classifying instances. DT and RF algorithms also displayed an improvement in their performance in the Python environment, with Accuracy values of 0.96491 and 0.94737, respectively. Moreover, the Recall, Precision, and F1 Score values for these models witnessed an increase, thereby strengthening their overall predictive capabilities. The GB showcased remarkable performance, attaining an Accuracy of 0.97368, a significant improvement compared to its counterpart in KNIME corroborating its impressive performance with Recall, Precision, and F1 Score values of 0.97619, 0.95349, and 0.96471, respectively, making it a standout choice in terms of all metrics.

The comparative analysis of these algorithms across the two platforms underscores the intricate relationship between algorithm choice, implementation environment, and resultant performance metrics. While KNIME Analytics rendered reliable results, Python emerged as the platform offering enhanced predictive accuracy across the board. Notably, the GB algorithm stood out in Python, exhibiting remarkable performance, which is highly relevant in medical contexts where accurate classification holds paramount importance. These findings underscore the necessity of carefully considering both algorithm selection and platform for optimal performance in predictive modeling endeavors.

Additionally, confusion matrix of the models was evaluated on their ability to predict both the 'Positive' and 'Negative' classes, and the calculated metrics offer valuable insights into their proficiency. The matrices reveal that while models generally perform well, some algorithms, such as DT and GB, consistently exhibit a higher number of True Positives emphasizing the accurate prediction of positive cases, crucial in medical contexts to minimize the risk of false negatives.

Comparing the KNIME and SciKit-Learn platforms, a pattern emerges. Generally, the SciKit-Learn platform showcases slightly better performance metrics, particularly in terms of True Positives and True Negatives. This disparity suggests that the SciKit-Learn implementation may have certain advantages in terms of predictive accuracy and class separation.

4.3 Significance of Findings

The observed variations emphasize the importance of algorithm selection and platform choice in machine learning model development, especially for medical applications such as breast cancer classification. The results provide valuable insights for practitioners and researchers seeking optimal algorithm-platform combinations for similar tasks. Overall, this comparative analysis contributes to the understanding of performances of the algorithms tested and their implications in the medical field, offering a foundation for further research and improvements in breast cancer classification systems.

Chapter 5: Conclusion and Future Work

In the following sections, summary inference on this study is done followed by brief commentary on its contributions to the field of Data Science, specifically, ML, with a wrap on the limitations and future work identified.

5.1 Summary of Research

This comparative experiment was performed to explore the possibility of the assumption that the selection of a machine learning platform might introduce variations in the performance of predictive models by using WDBC dataset on four classification algorithms for both training and inference phases on Python SciKit-Learn and Knime Analytics.

Results collected showed metrics for the algorithms were higher in Python than Knime. Whereas LR showed highest accuracy in Knime the highest recall of 0.90698 was seen in DT, this is different to the behavior of the algorithms in Python where, highest recall was recorded in RF and GB with both returning same recall value of 0.97619 suggesting an algorithm may be better if the focus is on assessing high recall a metric that is particularly important in cancer diagnosis. This however is not a verdict on the efficacy of either tools as a robust and reliable tool for ML model development, rather an investigation as to whether there would be differences based on their respective architecture.

5.2 Contributions to the Field

The study points to the fact that when developing ML projects where success or reliability on outcome of the work is dependent on algorithm's performance metrics, a test on more than one platform is advised for fair evaluation and justifiable acceptability.

5.3 Limitations and Challenges

The size of dataset used is modest relative to many other data commonly used in real world. Larger datasets might likely present significantly different results and behaviors generally. Also, due to time constraints, the effects of the size of this data on the platform performance was not explored.

In addition, the study used only one dataset for evaluation, this might limit the generalizability of the findings to other datasets possibly having different characteristics.

5.4 Future Directions for Research

Future work may involve looking to discern the inherent factors which may be responsible for the observed differences by carefully studying the tools architecture.

Another area that can be expanded on is researching the behaviors of other algorithms that have proved to be powerful classifiers also, such as Support Vector Machine (SVM) and Multi-Layer Perceptron (MLP) on the platforms while similar work can be performed on platforms including R and Weka.

Lastly, while keeping research goal same, multiple datasets may be put to test. We have started to explore proteomic dataset (See Appendix I) with aim of using the model developed to predict renal cancer and test evaluate the performance of the algorithms on these platforms.

References

Aamir, S. et al. (2022) ‘Predicting Breast Cancer Leveraging Supervised Machine Learning Techniques’, Computational and Mathematical Methods in Medicine, 2022. Available at: https://doi.org/10.1155/2022/5869529.

Althubaity, D.A.D. et al. (2023) ‘Automated Lung Cancer Segmentation in Tissue Micro Array Analysis Histopathological Images Using a Prototype of Computer-Assisted Diagnosis’, Journal of Personalized Medicine, 13(3). Available at: https://doi.org/10.3390/jpm13030388.

Ara, S., Das, A. and Dey, A. (2021) ‘Malignant and Benign Breast Cancer Classification using Machine Learning Algorithms’, in 2021 International Conference on Artificial Intelligence, ICAI 2021. Institute of Electrical and Electronics Engineers Inc., pp. 97–101. Available at: https://doi.org/10.1109/ICAI52203.2021.9445249.

ATEŞ, İ. and BİLGİN, T.T. (2021a) ‘The Investigation of the Success of Different Machine Learning Methods in Breast Cancer Diagnosis’. Available at: https://doi.org/10.18521/ktd.912462.

ATEŞ, İ. and BİLGİN, T.T. (2021b) ‘The Investigation of the Success of Different Machine Learning Methods in Breast Cancer Diagnosis’, Konuralp Tıp Dergisi, 13(2), pp. 347–356. Available at: https://doi.org/10.18521/ktd.912462.

Bailly, A. et al. (2022) ‘Effects of dataset size and interactions on the prediction performance of logistic regression and deep learning models’, Computer Methods and Programs in Biomedicine, 213. Available at: https://doi.org/10.1016/j.cmpb.2021.106504.

Boukhatem, C., Youssef, H.Y. and Nassif, A.B. (2022) ‘Heart Disease Prediction Using Machine Learning’, in 2022 Advances in Science and Engineering Technology International Conferences, ASET 2022. Institute of Electrical and Electronics Engineers Inc. Available at: https://doi.org/10.1109/ASET53988.2022.9734880.

Cai, J. et al. (2018) ‘Feature selection in machine learning: A new perspective’, Neurocomputing, 300, pp. 70–79. Available at: https://doi.org/10.1016/j.neucom.2017.11.077.

Cui, J. et al. (2023) ‘A novel multi-module integrated intrusion detection system for high-dimensional imbalanced data’, Applied Intelligence, 53(1), pp. 272–288. Available at: https://doi.org/10.1007/s10489-022-03361-2.

Ebrahim, M., Sedky, A.A.H. and Mesbah, S. (2023) ‘Accuracy Assessment of Machine Learning Algorithms Used to Predict Breast Cancer’, Data, 8(2). Available at: https://doi.org/10.3390/data8020035.

Faisal, M., Zamzami, E.M. and Sutarman (2020) ‘Comparative Analysis of Inter-Centroid K-Means Performance using Euclidean Distance, Canberra Distance and Manhattan Distance’, in Journal of Physics: Conference Series. Institute of Physics Publishing. Available at: https://doi.org/10.1088/1742-6596/1566/1/012112.

Feng, X., Cai, Y. and Xin, R. (2023) ‘Optimizing Diabetes Classication with a Machine Learning-Based Framework’. Available at: https://doi.org/10.21203/rs.3.rs-2866487/v1.

Ghennioui, H. et al. (2019) Machine Learning based System for Prediction of Breast Cancer Severity.

Index of /math-prog/cpo-dataset/machine-learn/cancer/cancer_images (1994). Available at: https://ftp.cs.wisc.edu/math-prog/cpo-dataset/machine-learn/cancer/cancer_images/ (Accessed: 22 August 2023).

Jasti, V.D.P. et al. (2022) ‘Computational Technique Based on Machine Learning and Image Processing for Medical Image Analysis of Breast Cancer Diagnosis’, Security and Communication Networks, 2022. Available at: https://doi.org/10.1155/2022/1918379.

Kavitha, R. et al. (2023) ‘Ant Colony Optimization-Enabled CNN Deep Learning Technique for Accurate Detection of Cervical Cancer’, BioMed Research International, 2023. Available at: https://doi.org/10.1155/2023/1742891.

Kong, J.H. et al. (2022) ‘Network-based machine learning approach to predict immunotherapy response in cancer patients’, Nature Communications, 13(1). Available at: https://doi.org/10.1038/s41467-022-31535-6.

Li, W. et al. (2019) ‘Gene Expression Value Prediction Based on XGBoost Algorithm’, Frontiers in Genetics, 10. Available at: https://doi.org/10.3389/fgene.2019.01077.

Liu, L. (2018) ‘Research on logistic regression algorithm of breast cancer diagnose data by machine learning’, in Proceedings - 2018 International Conference on Robots and Intelligent System, ICRIS 2018. Institute of Electrical and Electronics Engineers Inc., pp. 157–160. Available at: https://doi.org/10.1109/ICRIS.2018.00049.

Mahesh, T.R. et al. (2022) ‘Performance Analysis of XGBoost Ensemble Methods for Survivability with the Classification of Breast Cancer’, Journal of Sensors, 2022. Available at: https://doi.org/10.1155/2022/4649510.

Minnoor, M. and Baths, V. (2023) ‘Diagnosis of Breast Cancer Using Random Forests’, Procedia Computer Science, 218, pp. 429–437. Available at: https://doi.org/10.1016/j.procs.2023.01.025.

NCI (2021) National Cancer Institute. Available at: https://www.cancer.gov/about-cancer/ (Accessed: 18 July 2023).

Oktay, O. et al. (2020) ‘Evaluation of Deep Learning to Augment Image-Guided Radiotherapy for Head and Neck and Prostate Cancers’, JAMA Network Open [Preprint]. Available at: https://doi.org/10.1001/jamanetworkopen.2020.27426.

Omondiagbe, D.A., Veeramani, S. and Sidhu, A.S. (2019) ‘Machine Learning Classification Techniques for Breast Cancer Diagnosis’, in IOP Conference Series: Materials Science and Engineering. Institute of Physics Publishing. Available at: https://doi.org/10.1088/1757-899X/495/1/012033.

Pagan, M., Zarlis, M. and Candra, A. (2023) ‘Investigating the impact of data scaling on the k-nearest neighbor algorithm’, Computer Science and Information Technologies, 4(2), pp. 135–142. Available at: https://doi.org/10.11591/csit.v4i2.pp135-142.

Park, H.A. (2013) ‘An introduction to logistic regression: From basic concepts to interpretation with particular attention to nursing domain’, Journal of Korean Academy of Nursing, 43(2), pp. 154–164. Available at: https://doi.org/10.4040/jkan.2013.43.2.154.

Pudjihartono, N. et al. (2022) ‘A Review of Feature Selection Methods for Machine Learning-Based Disease Risk Prediction’, Frontiers in Bioinformatics, 2. Available at: https://doi.org/10.3389/fbinf.2022.927312.

Rahman, S. et al. (2019) ‘A Comparative Study On Liver Disease Prediction Using Supervised Machine Learning Algorithms’, INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH, 8(11). Available at: www.ijstr.org.

Shafique, R. et al. (2023) ‘Breast Cancer Prediction Using Fine Needle Aspiration Features and Upsampling with Supervised Machine Learning’, Cancers, 15(3). Available at: https://doi.org/10.3390/cancers15030681.

Shamu, T. et al. (2022) ‘Cancer incidence among people living with HIV in Zimbabwe: A record linkage study’, Cancer Reports, 5(10). Available at: https://doi.org/10.1002/cnr2.1597.

Statology (2023). Available at: https://www.statology.org/?s=min-max+normalization (Accessed: 14 August 2023).

Sumin et al. (2022) ‘The Impact of Z-Score Transformation Scaling on the Validity, Reliability, and Measurement Error of Instrument SATS-36’, Jurnal Pengukuran Psikologi dan Pendidikan Indonesia, 11(2), pp. 166–180. Available at: https://doi.org/10.15408/jp3i.v11i2.26591.

Sung, H. et al. (2021) ‘Global Cancer Statistics 2020: GLOBOCAN Estimates of Incidence and Mortality Worldwide for 36 Cancers in 185 Countries’, CA: A Cancer Journal for Clinicians, 71(3), pp. 209–249. Available at: https://doi.org/10.3322/caac.21660.

Tomar, G. et al. (2022) Liver Disease Prediction Using Machine Learning Classification Techniques.

Uddin, K.M.M. et al. (2023) ‘Machine learning-based diagnosis of breast cancer utilizing feature optimization technique’, Computer Methods and Programs in Biomedicine Update, 3. Available at: https://doi.org/10.1016/j.cmpbup.2023.100098.

Wan, X. (2019) ‘Influence of feature scaling on convergence of gradient iterative algorithm’, in Journal of Physics: Conference Series. Institute of Physics Publishing. Available at: https://doi.org/10.1088/1742-6596/1213/3/032021.

Zhu, C., Idemudia, C.U. and Feng, W. (2019) ‘Improved logistic regression model for diabetes prediction by integrating PCA and K-means techniques’, Informatics in Medicine Unlocked, 17. Available at: https://doi.org/10.1016/j.imu.2019.100179.

Appendices

(I) Proteomic Research Progress Report: Accessing Proteomic Dataset for Machine Learning Classification of Kidney Cancer

(II) Python SciKit-Learn Algorithm Implementation Code

Appendix I: Proteomic Research Progress Report

Cancer Classification: Investigating the Impact of the Implementation Platform on Machine Learning Algorithms Performance: Accessing Proteomic Dataset and Identification of Potential Features.

By

Adedayo Seun Olowolayemo

Supervisors:
Dr. Amina Souag
Prof. Konstatino Sirlantzis

1.0 Introduction

This report documents the progress made on the further work done in respect to the MSc project research "Cancer Classification: Investigating the Impact of the Implementation Platform on Machine Learning Algorithms Performance".

This phase of the research aims to evaluate comparatively the performance metrics of the algorithms comparatively when implemented using Knime Analytics and Python SciKit-Learn on machine learning model developed to predict Kidney Cancer using the Urine Proteome dataset available on ProteomeXchange website.

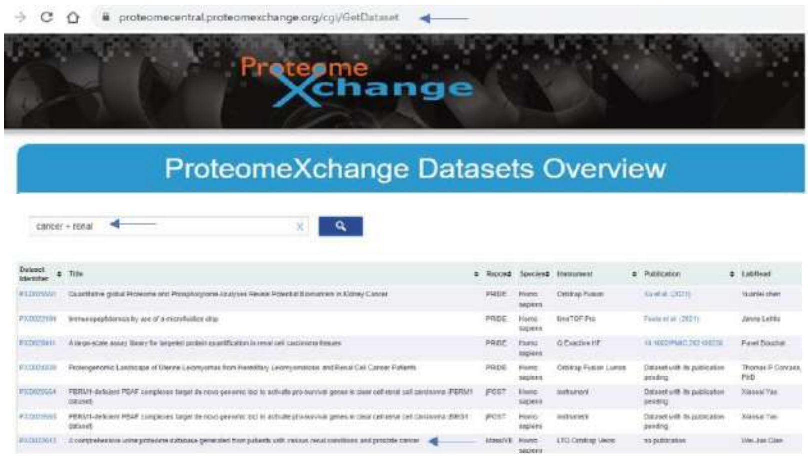
2.0 Urine Proteomic Data

The Urine cancer proteomic dataset accessible from Proteome Central ProteomeXchangeDatasets [1] will be explored with aim of understanding proteins that are upregulated or downregulated in Clear Cell Renal Cell Carcinoma (ccRCC) and are indicative of Kidney Cancer.

2.1 Accessing the Data

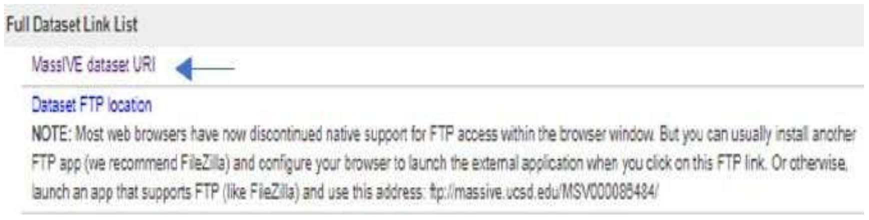
Proteomic datasets are large data generated by often large experiments (Domon & Aebersold, 2006) hence are mostly available in big sizes that requires specific download process. The process followed for the dataset to be used in this work is enumerated below step wisely.

1. The ProteomeXchange site was launched by clicking on URL https://proteomecentral.proteomexchange.org/cgi/GetDataset.
2. On the Search box, input search parameter in this case "*cancer + renal*". This will return a trimmed list of datasets matching the criteria.


proteomecentral.proteomexchange.org/cgi/GetDataset
ProteomeXchange
ProteomeXchange Datasets Overview
cancer + renal

Dataset Identifier	Title	Repos	Species	Instrument	Publication	LabHead
[illegible]	Quantitative global Proteome and Phosphoproteome Analyses Reveal Potential Biomarkers in Kidney Cancer	PRIDE	Homo sapiens	Orbitrap Fusion	[illegible] et al. (2021)	[illegible]
[illegible]	[illegible] by use of a microfluidics chip	PRIDE	Homo sapiens	timsTOF Pro	[illegible] et al. (2021)	[illegible]
[illegible]	A large-scale assay library for targeted protein quantification in renal cell carcinoma tissues	PRIDE	Homo sapiens	Q Exactive HF	[illegible]	Pavel Bouchal
[illegible]	Proteogenomic Landscape of Uterine Leiomyomas from Hereditary Leiomyomatosis and Renal Cell Cancer Patients	PRIDE	Homo sapiens	Orbitrap Fusion Lumos	Dataset with its publication pending	Thomas P Conrads, PhD
[illegible]	PBRM1-deficient PBAF complexes target de novo genomic loci to activate pro-survival genes in clear cell renal cell carcinoma (PBRM1 dataset)	jPOST	Homo sapiens	instrument	Dataset with its publication pending	Xiaosai Yao
[illegible]	PBRM1-deficient PBAF complexes target de novo genomic loci to activate pro-survival genes in clear cell renal cell carcinoma (BRD7 dataset)	jPOST	Homo sapiens	instrument	Dataset with its publication pending	Xiaosai Yao
[illegible]	A comprehensive urine proteome database generated from patients with various renal conditions and prostate cancer	MassIVE	Homo sapiens	LTQ Orbitrap Velos	no publication	[illegible]

3. The title labels describe the dataset, select choice data by clicking the Dataset Identifier "*A comprehensive urine proteome database generated from patients with various renal conditions and prostate cancer*."
   For example, the data PXD022612 (ProteomeXchange Dataset PXD022612) was chosen because it contains data of interest for the project.
4. Full dataset was then fetched via the URL from the repository (the Urine Proteome data, PXD022612) is in MaasIVE repository.



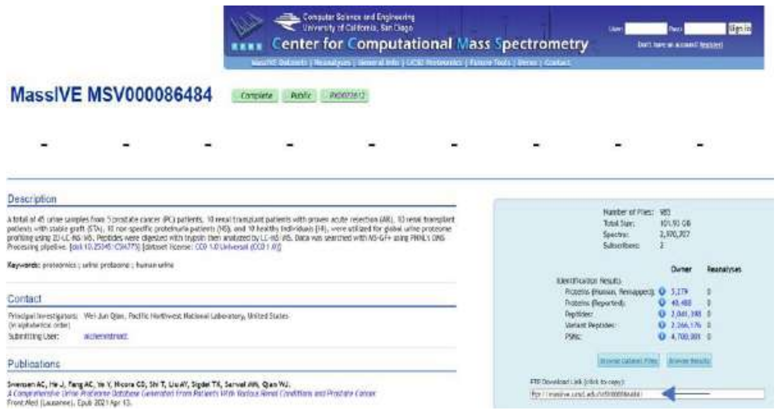
Full Dataset Link List

MassIVE dataset URI

Dataset FTP location

NOTE: Most web browsers have now discontinued native support for FTP access within the browser window. But you can usually install another FTP app (we recommend FileZilla) and configure your browser to launch the external application when you click on this FTP link. Or otherwise, launch an app that supports FTP (like FileZilla) and use this address: ftp://massive.ucsd.edu/MSV000086484/

5. The MassIVE page created by Centre for Computational Mass Spectrometry (CCMS) contains metadata and description of the dataset with information of Principal Investigators and related publications.
   It is important to register as a user on this site before further steps.


Computer Science and Engineering
University of California, San Diego
Center for Computational Mass Spectrometry
User:
Pass:
Sign in
Complete
Public
PXD022612
Browse Dataset Files
Browse Results

MassIVE MSV000086484

Description

A total of 45 urine samples from 5 prostate cancer (PC) patients, 10 renal transplant patients with proven acute rejection (AR), 10 renal transplant patients with stable graft (STA), 10 non-specific proteinuria patients (NS), and 10 healthy individuals (H), were utilized for global urine proteome profiling using 2D-LC-MS/MS. Peptides were digested with trypsin then analyzed by LC-MS/MS. Data was searched with MS-GF+ using PNNL's DMS Processing pipeline. [doi:10.25345/[illegible]] [dataset license: CC0 1.0 Universal (CC0 1.0)]

Keywords: proteomics ; urine proteome ; human urine

Contact

Principal Investigators: (in alphabetical order)	Wei-Jun Qian, Pacific Northwest National Laboratory, United States
Submitting User:	[illegible]

Publications

Svensen AC, He J, Fang AC, Ye Y, Nicora CD, Shi T, Liu AY, Sigdel TK, Sarwal MM, Qian WJ.
A Comprehensive Urine Proteome Database Generated From Patients With Various Renal Conditions and Prostate Cancer
Front Med (Lausanne). Epub 2021 Apr 13.

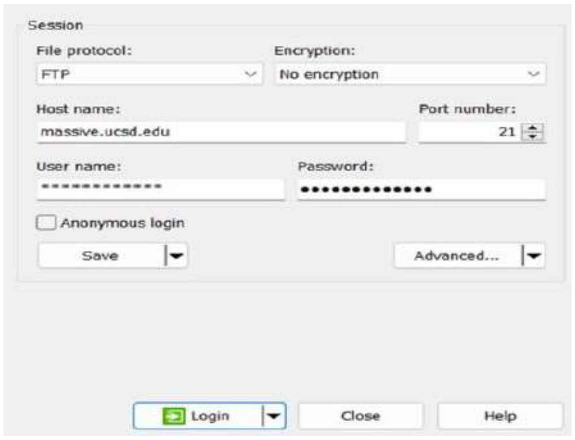
Number of Files:	985
Total Size:	101.93 GB
Spectra:	2,970,707
Subscribers:	2

	Owner	Reanalyses
Identification Results		
Proteins (Human, Remapped):	5,179	0
Proteins (Reported):	40,488	0
Peptides:	2,041,398	0
Variant Peptides:	2,266,176	0
PSMs:	4,700,001	0

FTP Download Link (click to copy):
ftp://massive.ucsd.edu/MSV000086484/

6. File Transfer Protocol (FTP) file is downloaded using specific File Manager applications such as Winscp and Filezilla. (For this dataset, Winscp version 6.1.1 was used).
   Download **Winscp** [2] from Google or through URL https://winscp.net/eng/download.php.

7. Launch Winscp, copy dataset URL from previous step and paste in the Host Name, following which username and password is entered to gain remote access.



Session

File protocol: FTP

Encryption: No encryption

Host name: massive.ucsd.edu

Port number: 21

User name: ************

Password: ••••••••••••••

8. Once access to dataset has been granted, the files would then be transferred to local computer storage or cloud where it will thereafter be used. The dataset has 985 files and Total File Size of 101.93 GB.

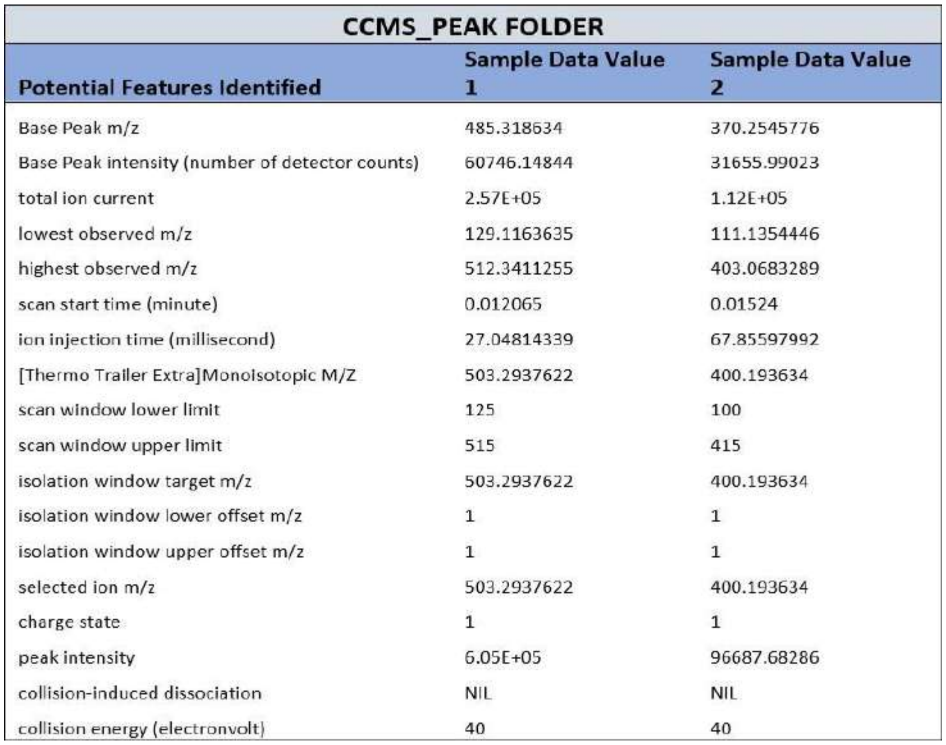
2.2 Understanding the Proteomic Dataset

The transferred files come in different formats including xml, mzml, txt formats saved in a few folders labelled as

- ccms_result
- ccms_peak
- ccms_parameters
- ccms_statistics
- peak
- sequence
- result
- search
- raw

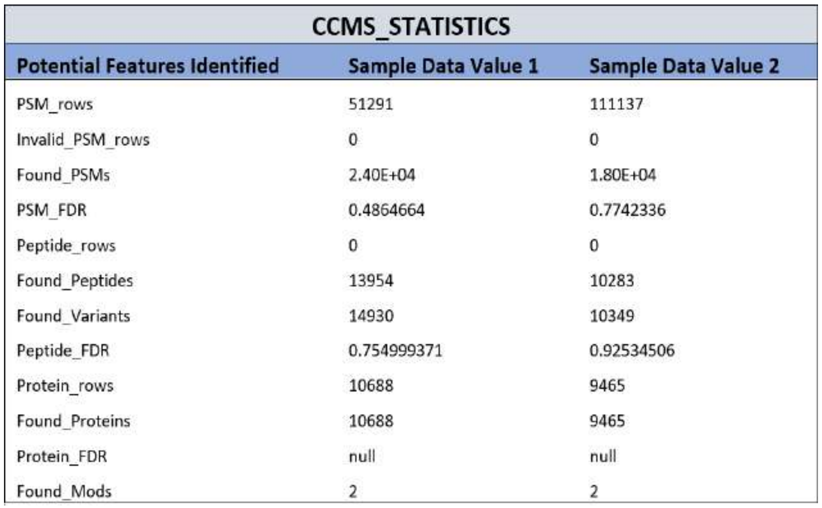
Table below shows some of the identified possible attributes from the datasets when viewed on Notepad or Visual Studio Code.

Table 1- Some of potential features identified in the ccms_peak folder of proteomic dataset PXD022612 (A comprehensive urine proteome database generated from patients with various renal conditions and prostate cancer).



CCMS_PEAK FOLDER		
Potential Features Identified	**Sample Data Value 1**	**Sample Data Value 2**
Base Peak m/z	485.318634	370.2545776
Base Peak intensity (number of detector counts)	60746.14844	31655.99023
total ion current	2.57E+05	1.12E+05
lowest observed m/z	129.1163635	111.1354446
highest observed m/z	512.3411255	403.0683289
scan start time (minute)	0.012065	0.01524
ion injection time (millisecond)	27.04814339	67.85597992
[Thermo Trailer Extra]Monoisotopic M/Z	503.2937622	400.193634
scan window lower limit	125	100
scan window upper limit	515	415
isolation window target m/z	503.2937622	400.193634
isolation window lower offset m/z	1	1
isolation window upper offset m/z	1	1
selected ion m/z	503.2937622	400.193634
charge state	1	1
peak intensity	6.05E+05	96687.68286
collision-induced dissociation	NIL	NIL
collision energy (electronvolt)	40	40

Table 2-- Some of potential features identified in the ccms_statistics folder of proteomic dataset PXD022612 (A comprehensive urine proteome database generated from patients with various renal conditions and prostate cancer).



CCMS_STATISTICS		
Potential Features Identified	**Sample Data Value 1**	**Sample Data Value 2**
PSM_rows	51291	111137
Invalid_PSM_rows	0	0
Found_PSMs	2.40E+04	1.80E+04
PSM_FDR	0.4864664	0.7742336
Peptide_rows	0	0
Found_Peptides	13954	10283
Found_Variants	14930	10349
Peptide_FDR	0.754999371	0.92534506
Protein_rows	10688	9465
Found_Proteins	10688	9465
Protein_FDR	null	null
Found_Mods	2	2

3.0 Conclusion

Having a clear understanding of the already identified potential features and others yet to be identified but are also contained in the dataset is important to proceed with further work using the dataset. This is work in progress (WIP) which may require additional resources in terms of time and some bioinformatics knowledge.

4.0 References

Domon, B., & Aebersold, R. (2006). Challenges and opportunities in proteomics data analysis. *Molecular and Cellular Proteomics*, *5*(10), 1921–1926. https://doi.org/10.1074/mcp.R600012-MCP200

ProteomeXchange Dataset PXD022612. (2021) Retrieved August 7, 2023, from https://proteomecentral.proteomexchange.org/cgi/GetDataset?ID=PXD022612

[1] https://proteomecentral.proteomexchange.org/cgi/GetDataset
[2] https://winscp.net/eng/download.php

Appendix II: Python SciKit-Learn Algorithm Implementation Code

In [1]:

```
# Import the relevant libraries

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.preprocessing import LabelEncoder

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
#from sklearn.ensemble import XGBClassifier

from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score,confusio
```

In [2]:

```
# Load the data

breast = pd.read_csv(r"C:¥Users¥Admin¥Desktop¥Breast Cancer For Project Work.csv")

# To see the top 5 rows in the data

breast.head()
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	0
2	84300903	M	19.69	21.25	130.00	1203.0	0
3	84348301	M	11.42	20.38	77.58	386.1	0
4	84358402	M	20.29	14.34	135.10	1297.0	0

5 rows × 32 columns

In [3]:

```
# To instantiate LabelEncoder

le = LabelEncoder()
breast['diagnosis'] = le.fit_transform(breast['diagnosis'])

print ("0 = Benign Cells")
print ("1 = Malignant Cells")
```

```
0 = Benign Cells
1 = Malignant Cells
```

In [4]:

```
# To see the top 5 rows in the data

breast.head()
```

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness
0	842302	1	17.99	10.38	122.80	1001.0	
1	842517	1	20.57	17.77	132.90	1326.0	0
2	84300903	1	19.69	21.25	130.00	1203.0	0
3	84348301	1	11.42	20.38	77.58	386.1	0
4	84358402	1	20.29	14.34	135.10	1297.0	0

5 rows × 32 columns

In [5]:

```

breast_counts = breast['diagnosis'].value_counts()
```

In [6]:

```
# To check the dimension of the data

breast.shape
```

Out[6]:

```
(569, 32)
```

In [7]:

```
print("diagnosis:")
print(breast_counts)
```

```
diagnosis:
0    357
1    212
```

```
Name: diagnosis, dtype: int64
```

In [ ]:

```

```

In [8]:

```
# To check the structure of the data

breast.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    int32  
 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
dtypes: float64(30), int32(1), int64(1)
memory usage: 140.2 KB
```

In [9]:

```
# To check the missing values

breast.isna().sum()
```

Out[9]:

```
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
dtype: int64
```

In [10]:

```
# To check the duplication

breast.duplicated().sum()
```

Out[10]:

```
0
```

In [11]:

```
# To randomly select any 5 rows in the data

breast.sample(5)
```

Out[11]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothne
282	89122	1	19.40	18.18	127.20	1145.0	
50	857343	0	11.76	21.60	74.72	427.9	
3	84348301	1	11.42	20.38	77.58	386.1	
435	908489	1	13.98	19.62	91.12	599.5	
131	8670	1	15.46	19.48	101.70	748.9	

5 rows × 32 columns

In [12]:

```
# To see the descriptive statistics

breast.describe().T
```

Out[12]:

	count	mean	std	min	25%	
id	569.0	3.037183e+07	1.250206e+08	8670.000000	869218.000000	906
diagnosis	569.0	3.725835e-01	4.839180e-01	0.000000	0.000000	
radius_mean	569.0	1.412729e+01	3.524049e+00	6.981000	11.700000	
texture_mean	569.0	1.928965e+01	4.301036e+00	9.710000	16.170000	
perimeter_mean	569.0	9.196903e+01	2.429898e+01	43.790000	75.170000	
area_mean	569.0	6.548891e+02	3.519141e+02	143.500000	420.300000	
smoothness_mean	569.0	9.636028e-02	1.406413e-02	0.052630	0.086370	
compactness_mean	569.0	1.043410e-01	5.281276e-02	0.019380	0.064920	
concavity_mean	569.0	8.879932e-02	7.971981e-02	0.000000	0.029560	
concave points_mean	569.0	4.891915e-02	3.880284e-02	0.000000	0.020310	
symmetry_mean	569.0	1.811619e-01	2.741428e-02	0.106000	0.161900	
fractal_dimension_mean	569.0	6.279761e-02	7.060363e-03	0.049960	0.057700	
radius_se	569.0	4.051721e-01	2.773127e-01	0.111500	0.232400	
texture_se	569.0	1.216853e+00	5.516484e-01	0.360200	0.833900	
perimeter_se	569.0	2.866059e+00	2.021855e+00	0.757000	1.606000	
area_se	569.0	4.033708e+01	4.549101e+01	6.802000	17.850000	
smoothness_se	569.0	7.040979e-03	3.002518e-03	0.001713	0.005169	
compactness_se	569.0	2.547814e-02	1.790818e-02	0.002252	0.013080	
concavity_se	569.0	3.189372e-02	3.018606e-02	0.000000	0.015090	
concave points_se	569.0	1.179614e-02	6.170285e-03	0.000000	0.007638	
symmetry_se	569.0	2.054230e-02	8.266372e-03	0.007882	0.015160	
fractal_dimension_se	569.0	3.794904e-03	2.646071e-03	0.000895	0.002248	
radius_worst	569.0	1.626919e+01	4.833242e+00	7.930000	13.010000	
texture_worst	569.0	2.567722e+01	6.146258e+00	12.020000	21.080000	
perimeter_worst	569.0	1.072612e+02	3.360254e+01	50.410000	84.110000	
area_worst	569.0	8.805831e+02	5.693570e+02	185.200000	515.300000	
smoothness_worst	569.0	1.323686e-01	2.283243e-02	0.071170	0.116600	
compactness_worst	569.0	2.542650e-01	1.573365e-01	0.027290	0.147200	
concavity_worst	569.0	2.721885e-01	2.086243e-01	0.000000	0.114500	
concave points_worst	569.0	1.146062e-01	6.573234e-02	0.000000	0.064930	
symmetry_worst	569.0	2.900756e-01	6.186747e-02	0.156500	0.250400	
fractal_dimension_worst	569.0	8.394582e-02	1.806127e-02	0.055040	0.071460	

In [13]:

```
# To know the correlation among the variables

breast_df = breast.corr()
breast_df
```

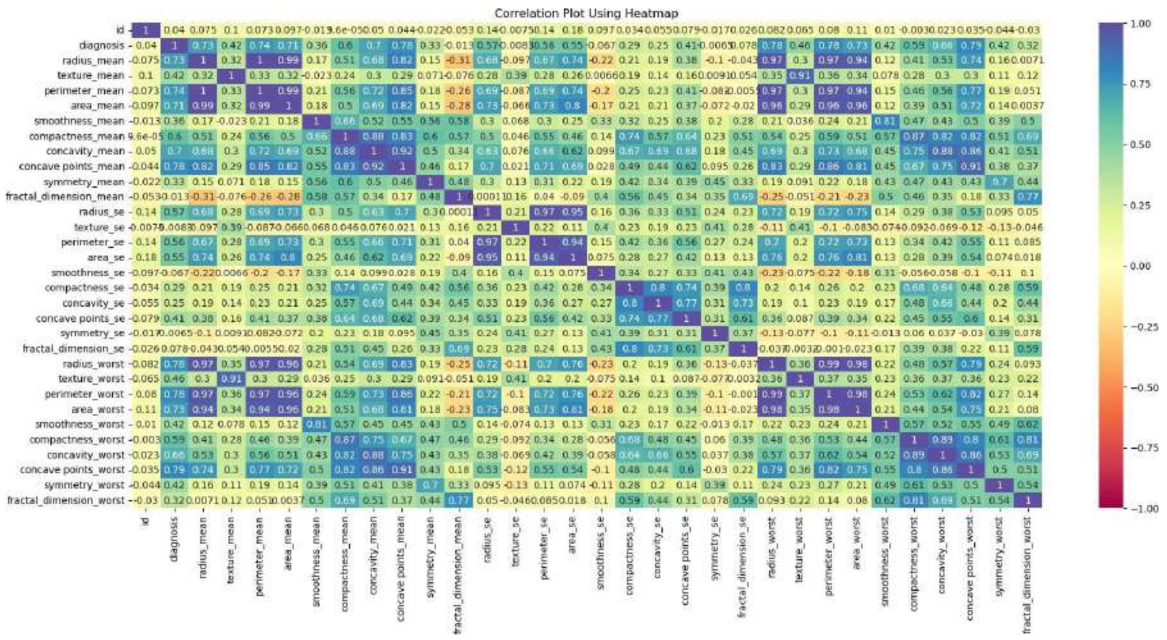
Out[13]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	a
id	1.000000	0.039769	0.074626	0.099770	0.073159	
diagnosis	0.039769	1.000000	0.730029	0.415185	0.742636	
radius_mean	0.074626	0.730029	1.000000	0.323782	0.997855	
texture_mean	0.099770	0.415185	0.323782	1.000000	0.329533	
perimeter_mean	0.073159	0.742636	0.997855	0.329533	1.000000	
area_mean	0.096893	0.708984	0.987357	0.321086	0.986507	
smoothness_mean	-0.012968	0.358560	0.170581	-0.023389	0.207278	
compactness_mean	0.000096	0.596534	0.506124	0.236702	0.556936	
concavity_mean	0.050080	0.696360	0.676764	0.302418	0.716136	
concave points_mean	0.044158	0.776614	0.822529	0.293464	0.850977	
symmetry_mean	-0.022114	0.330499	0.147741	0.071401	0.183027	
fractal_dimension_mean	-0.052511	-0.012838	-0.311631	-0.076437	-0.261477	
radius_se	0.143048	0.567134	0.679090	0.275869	0.691765	
texture_se	-0.007526	-0.008303	-0.097317	0.386358	-0.086761	
perimeter_se	0.137331	0.556141	0.674172	0.281673	0.693135	
area_se	0.177742	0.548236	0.735864	0.259845	0.744983	
smoothness_se	0.096781	-0.067016	-0.222600	0.006614	-0.202694	
compactness_se	0.033961	0.292999	0.206000	0.191975	0.250744	
concavity_se	0.055239	0.253730	0.194204	0.143293	0.228082	
concave points_se	0.078768	0.408042	0.376169	0.163851	0.407217	
symmetry_se	-0.017306	-0.006522	-0.104321	0.009127	-0.081629	
fractal_dimension_se	0.025725	0.077972	-0.042641	0.054458	-0.005523	
radius_worst	0.082405	0.776454	0.969539	0.352573	0.969476	
texture_worst	0.064720	0.456903	0.297008	0.912045	0.303038	
perimeter_worst	0.079986	0.782914	0.965137	0.358040	0.970387	
area_worst	0.107187	0.733825	0.941082	0.343546	0.941550	
smoothness_worst	0.010338	0.421465	0.119616	0.077503	0.150549	
compactness_worst	-0.002968	0.590998	0.413463	0.277830	0.455774	
concavity_worst	0.023203	0.659610	0.526911	0.301025	0.563879	
concave points_worst	0.035174	0.793566	0.744214	0.295316	0.771241	
symmetry_worst	-0.044224	0.416294	0.163953	0.105008	0.189115	
fractal_dimension_worst	-0.029866	0.323872	0.007066	0.119205	0.051019	

32 rows × 32 columns

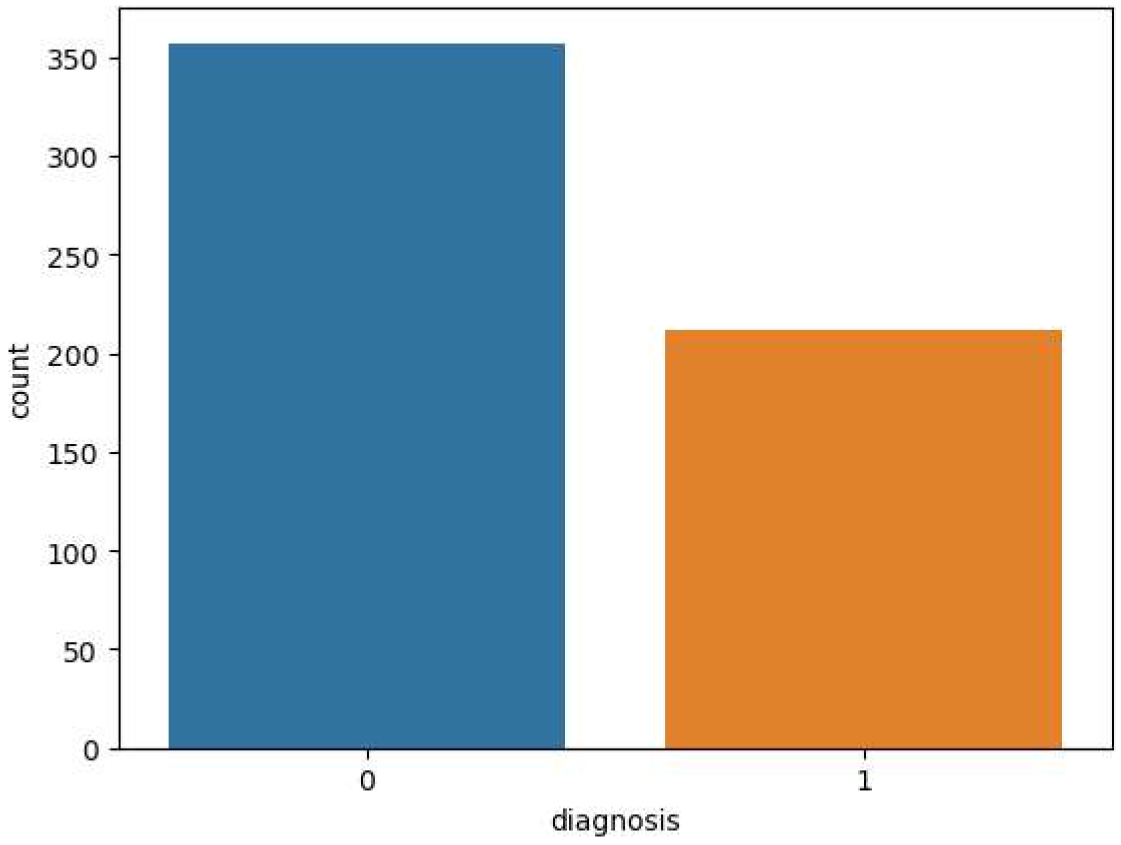
In [14]:

```
# To view the corrlation using heatmap

plt.figure(figsize=(21,9))
sns.heatmap(data= breast_df, annot=True, vmin =-1, vmax =1, cmap="Spectral")
plt.title("Correlation Plot Using Heatmap")
plt.show();
```



In [15]:

```
# count of M and B

sns.countplot(data = breast, x = "diagnosis");
```



In [16]:

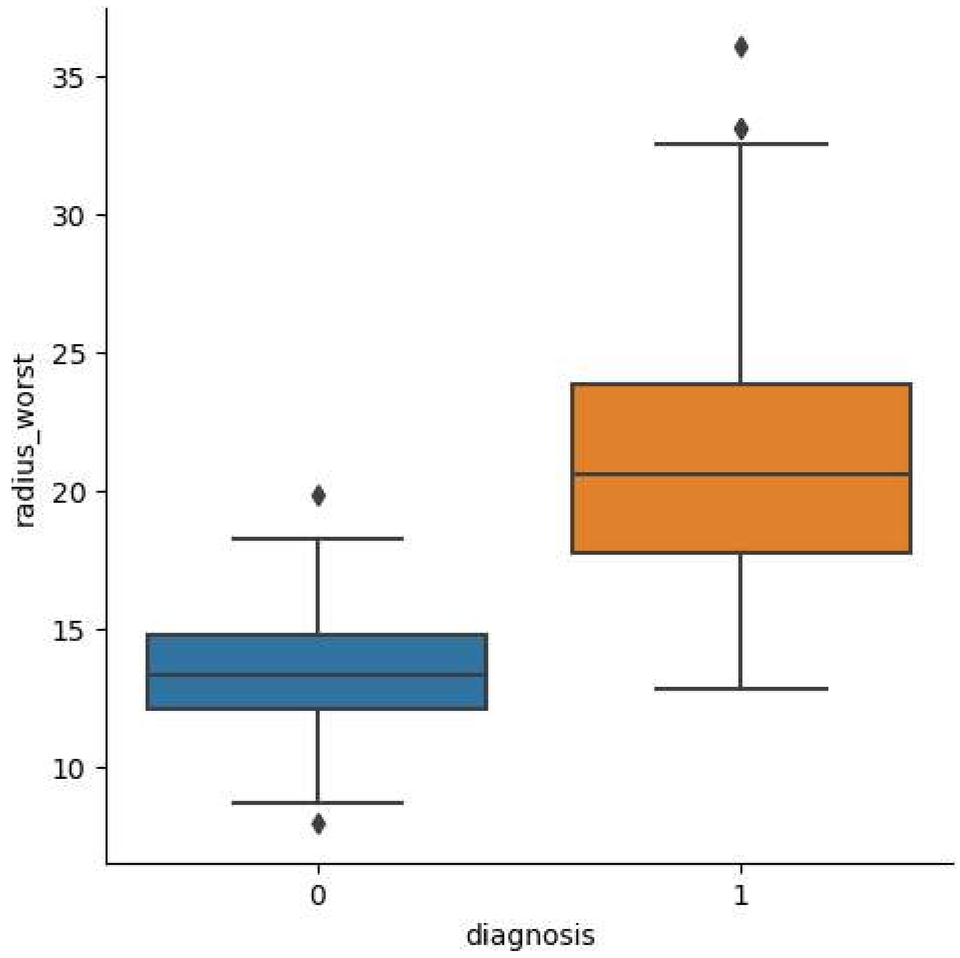
```
# To know the percentage of M and B

breast["diagnosis"].value_counts(normalize=True)
```

Out[16]:

```
0    0.627417
1    0.372583
Name: diagnosis, dtype: float64
```

In [17]:

```
sns.catplot(data = breast, x ="diagnosis", y = "radius_worst", kind= "box" );
#plt.xlabel("diagnosis")
#plt.ylabel("radius_mean")
```



In [18]:

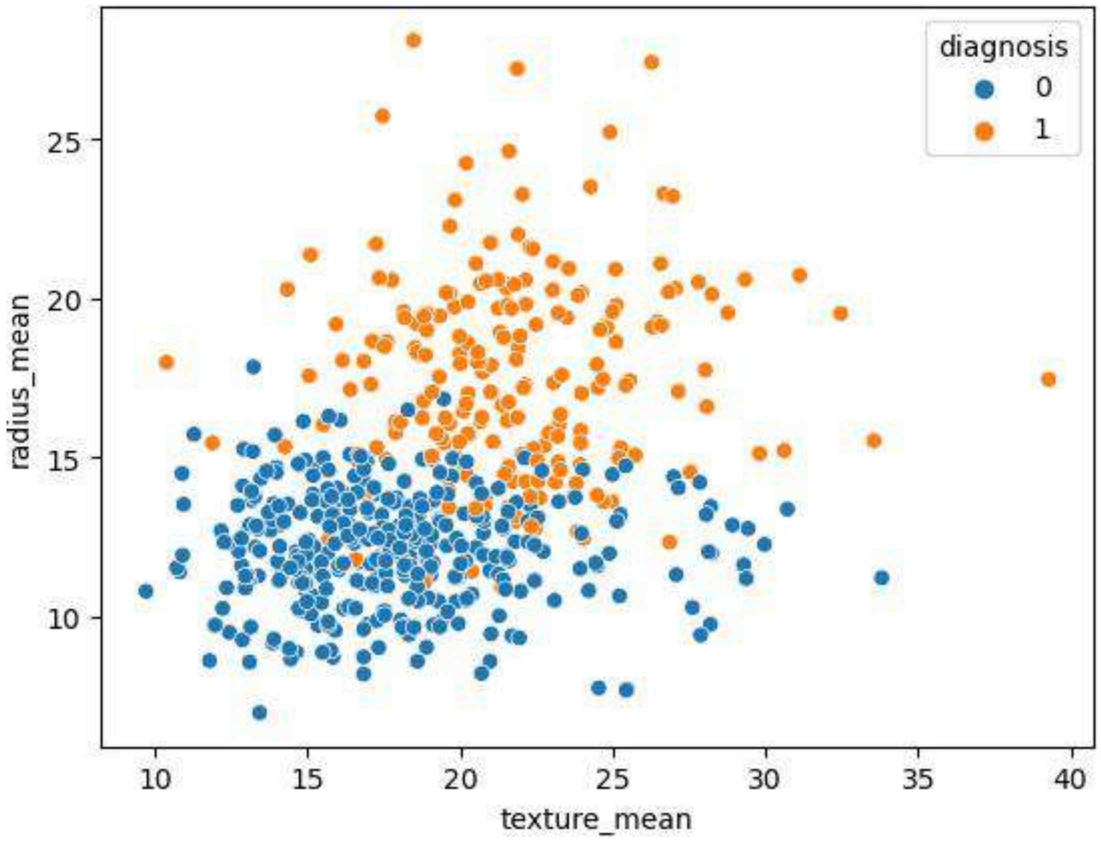
```
breast['diagnosis'].groupby([breast['radius_mean'], breast['radius_se'],
                             breast['radius_worst']]).max()
```

Out[18]:

```
radius_mean   radius_se   radius_worst
6.981         0.2241      7.930           0
7.691         0.2196      8.678           0
7.729         0.3777      9.077           0
7.760         0.3857      9.456           0
8.196         0.1563      8.964           0
                                         ..
25.220        0.8973      30.000          1
25.730        0.9948      33.130          1
27.220        0.8361      33.120          1
27.420        2.5470      36.040          1
28.110        2.8730      28.110          1
Name: diagnosis, Length: 569, dtype: int32
```

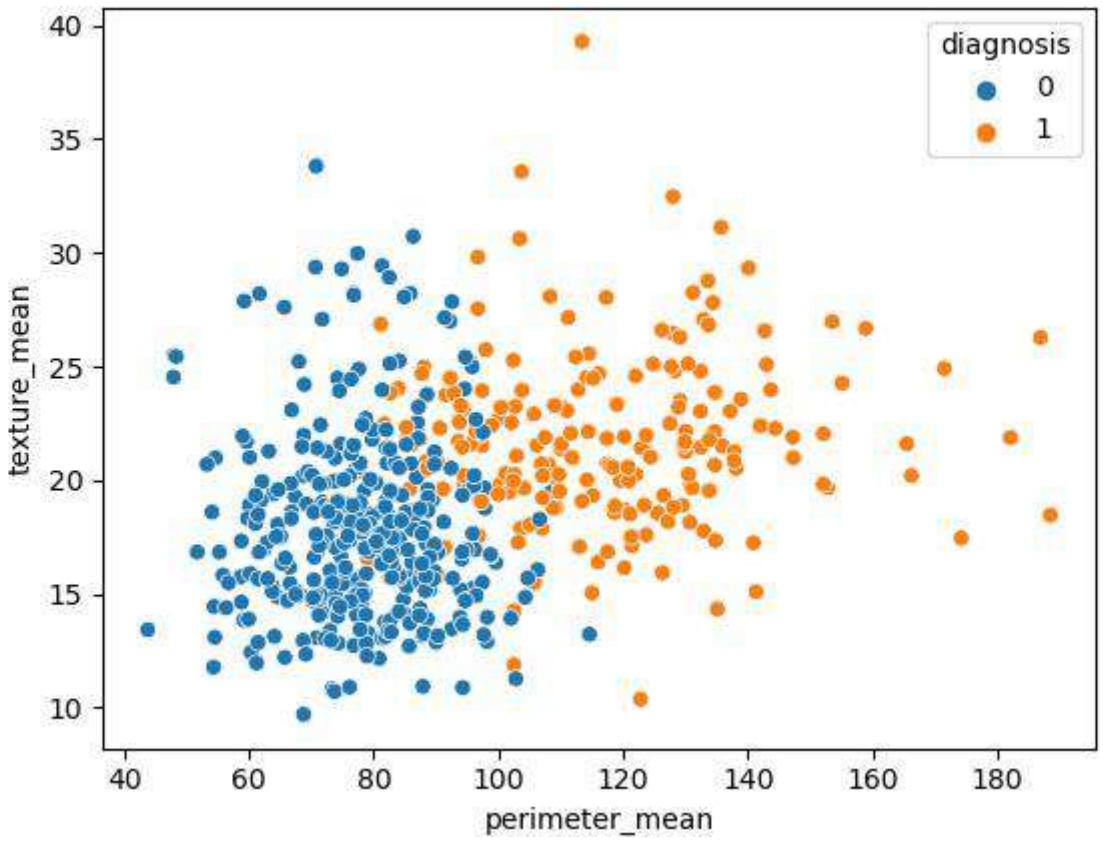
In [19]:

```
# Scatter plot of the target and the radius variable

sns.scatterplot(data = breast, y = "radius_mean", x = "texture_mean", hue= "diagnosis");
```



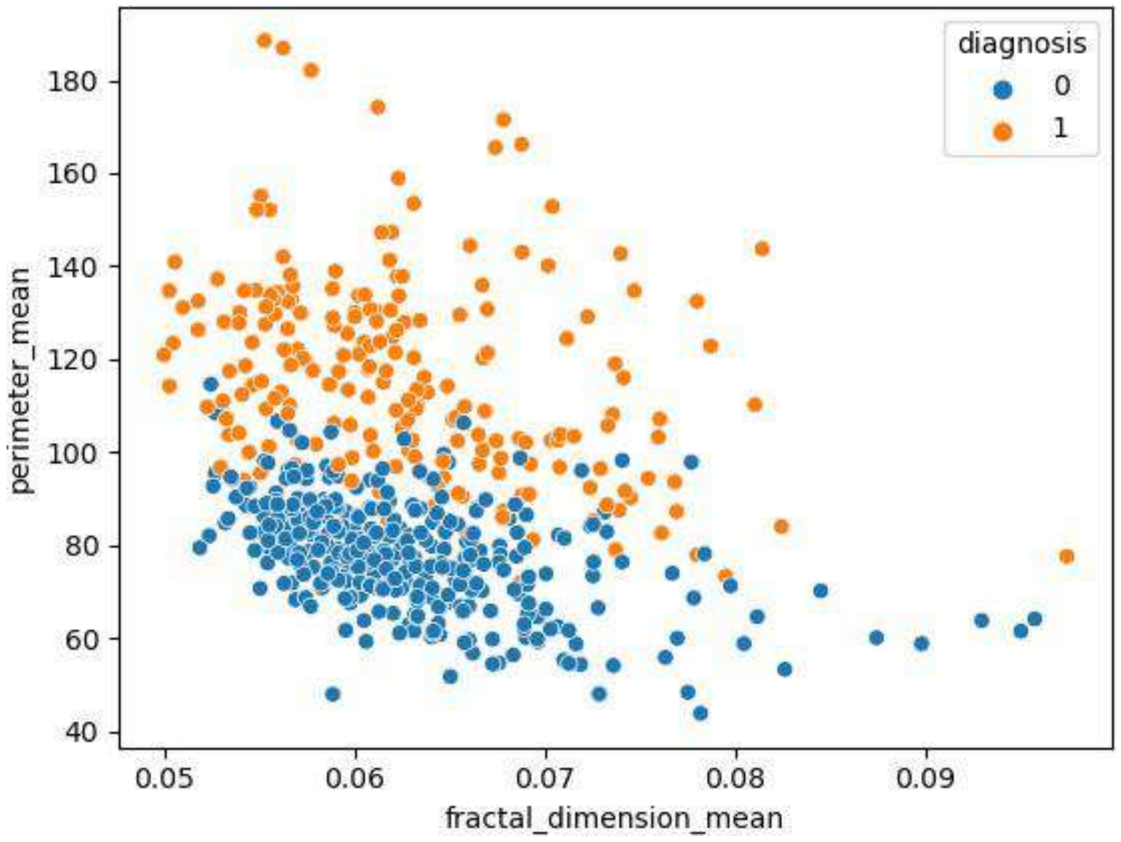
In [20]:

```
sns.scatterplot(data = breast, y = "texture_mean", x = "perimeter_mean", hue= "diagnosis"
```



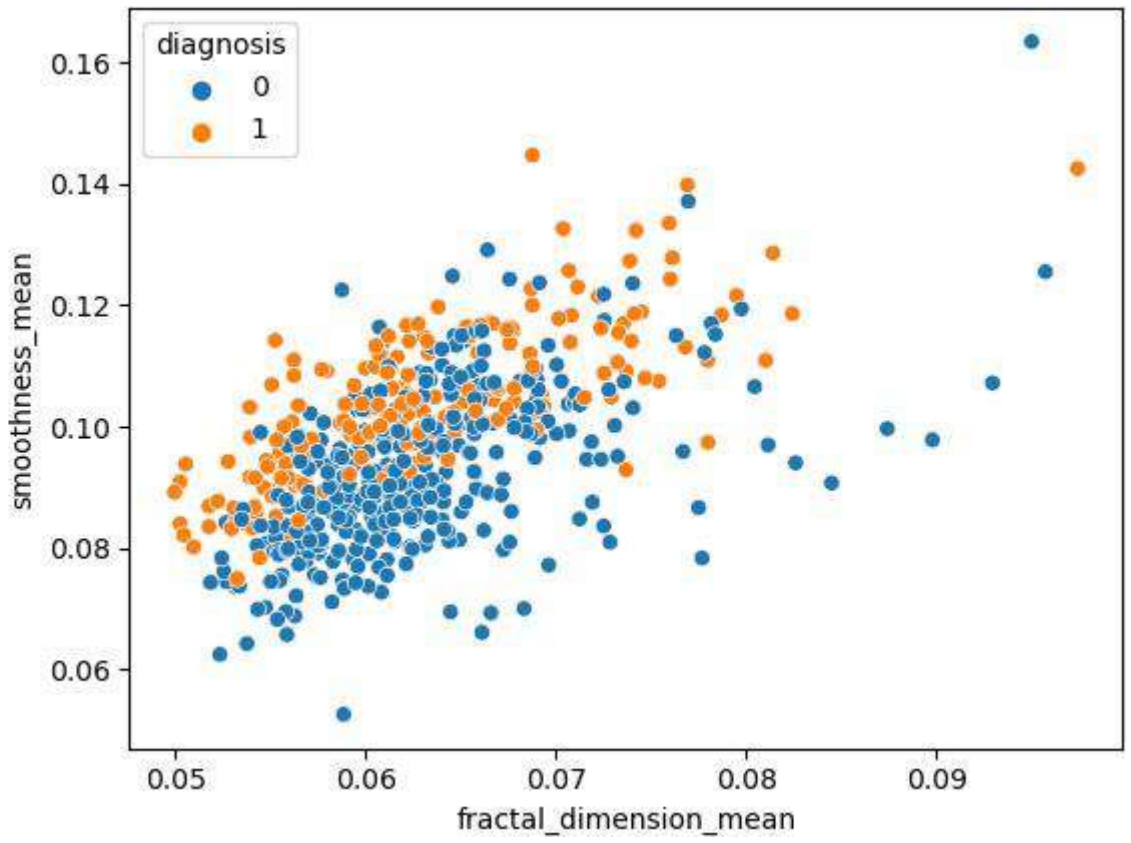
In [21]:

```
sns.scatterplot(data = breast, y = "perimeter_mean", x = "fractal_dimension_mean", hue= "
```



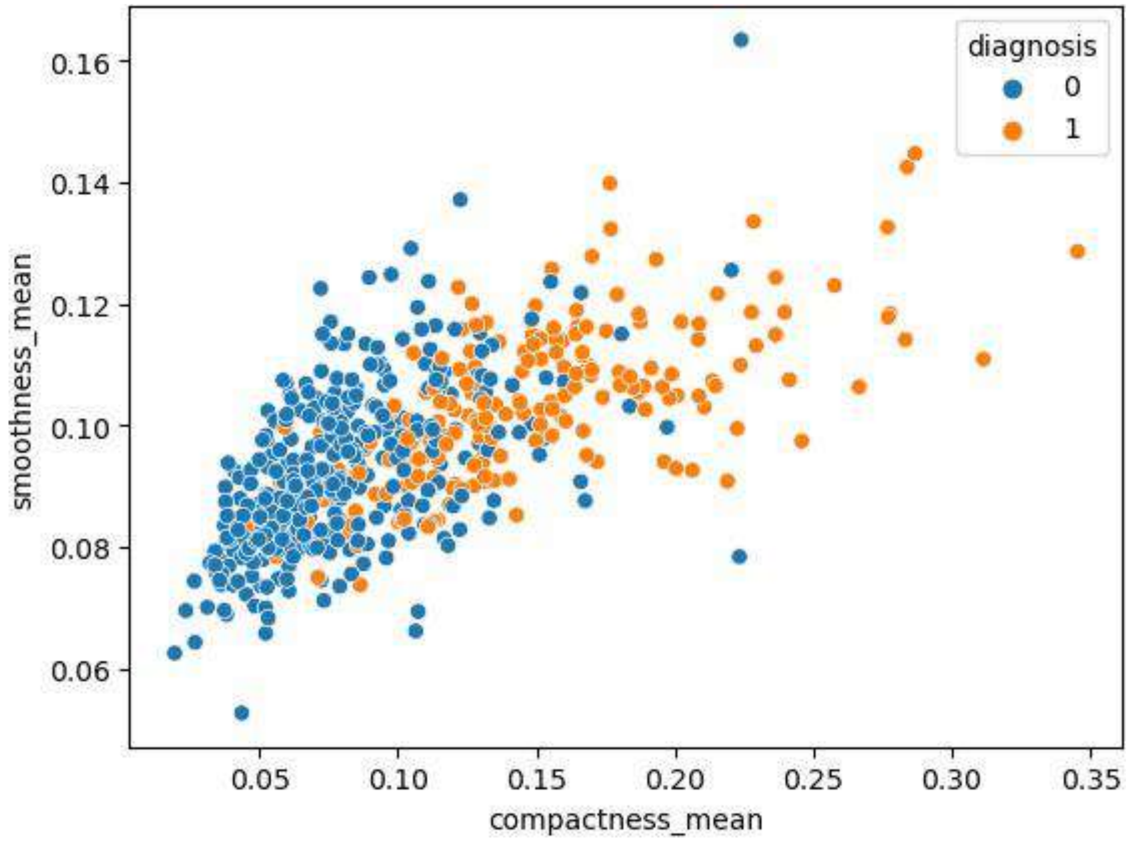
In [22]:

```
sns.scatterplot(data = breast, y = "smoothness_mean", x = "fractal_dimension_mean", hue="
```



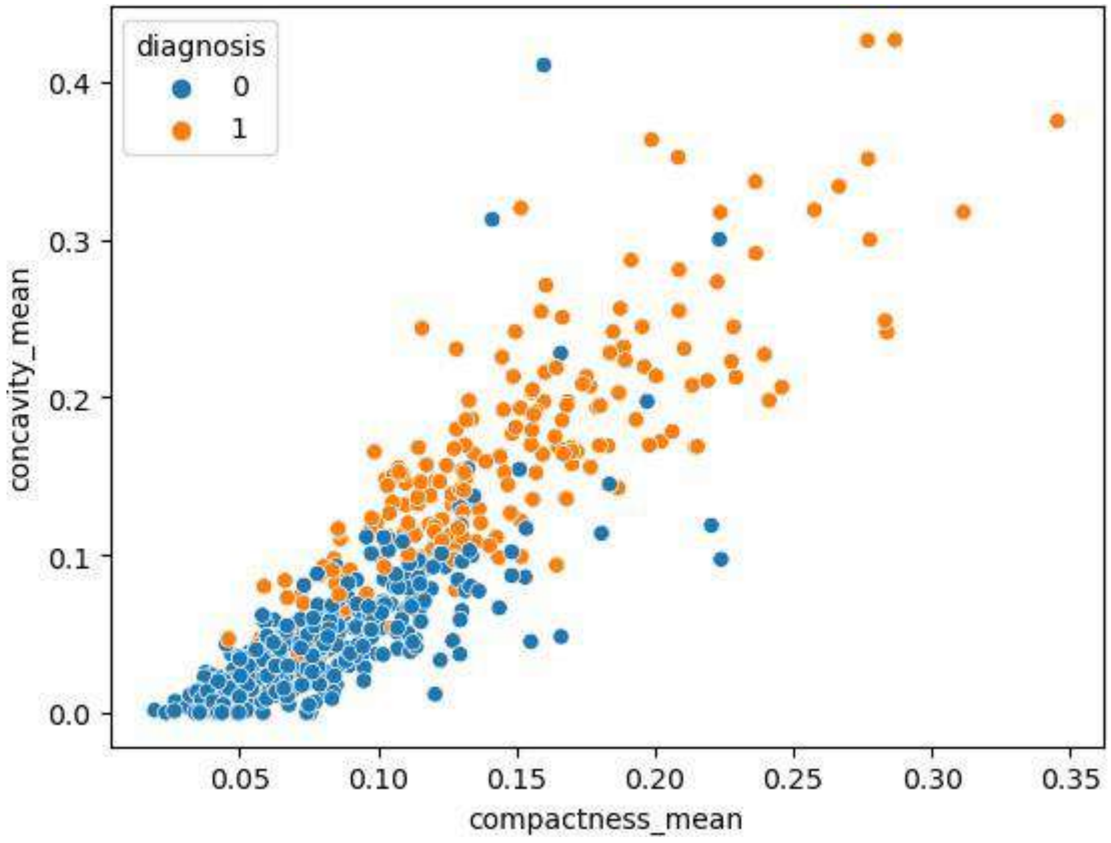
In [23]:

```
sns.scatterplot(data = breast, y = "smoothness_mean", x = "compactness_mean", hue="diagno
```



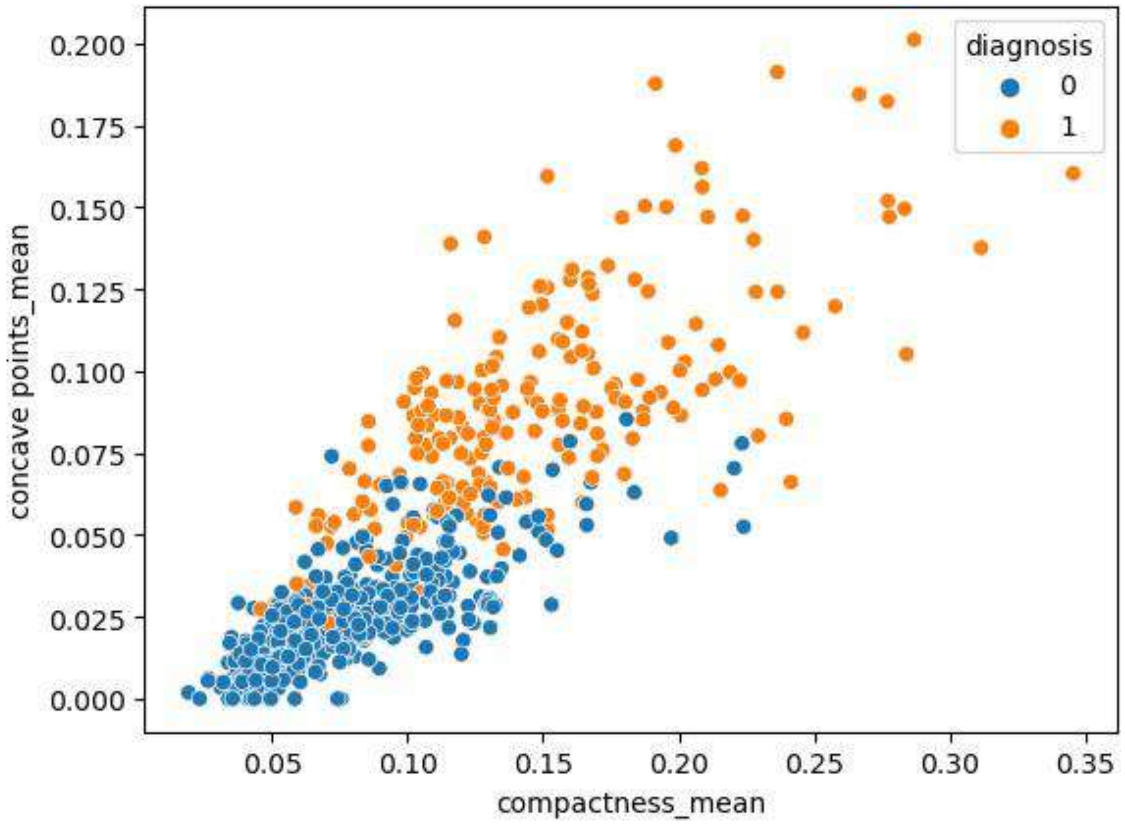
In [24]:

```
sns.scatterplot(data = breast, y = "concavity_mean", x = "compactness_mean", hue="diagnos
```



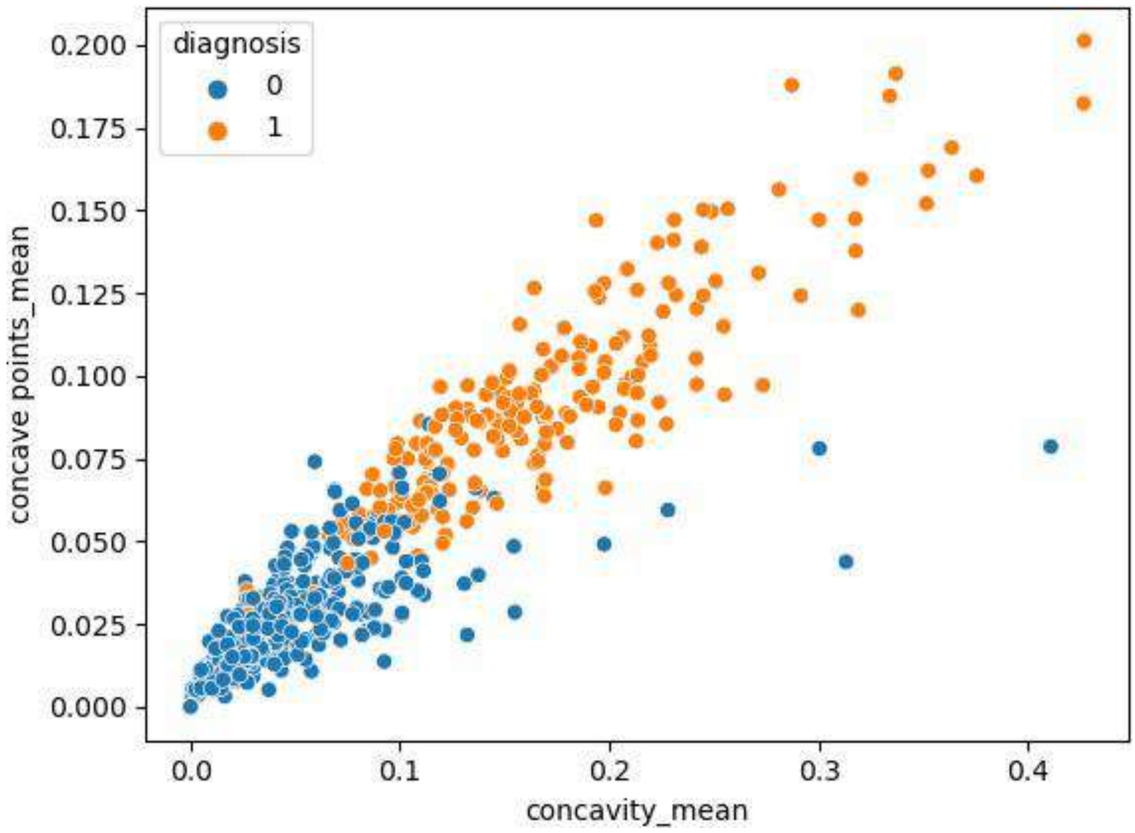
In [25]:

```
sns.scatterplot(data = breast, y = "concave points_mean", x = "compactness_mean", hue="di
```



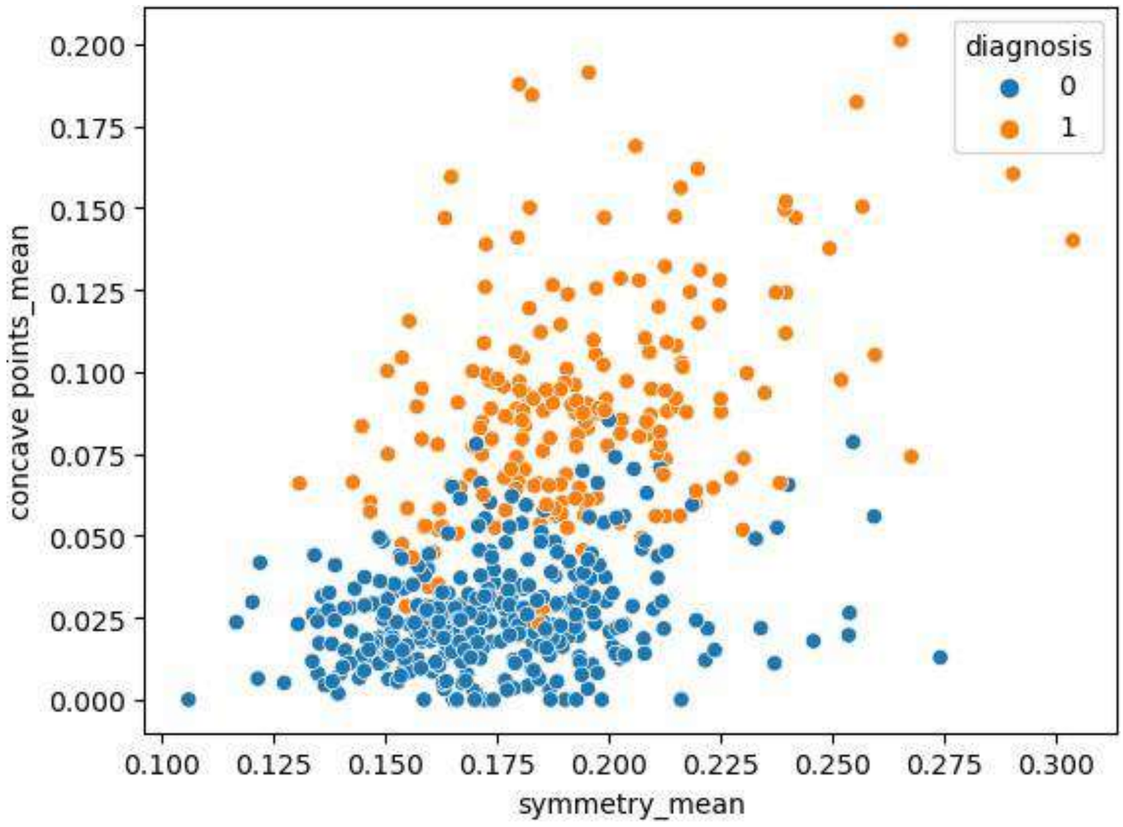
In [26]:

```
sns.scatterplot(data = breast, y = "concave points_mean", x = "concavity_mean", hue="diag
```



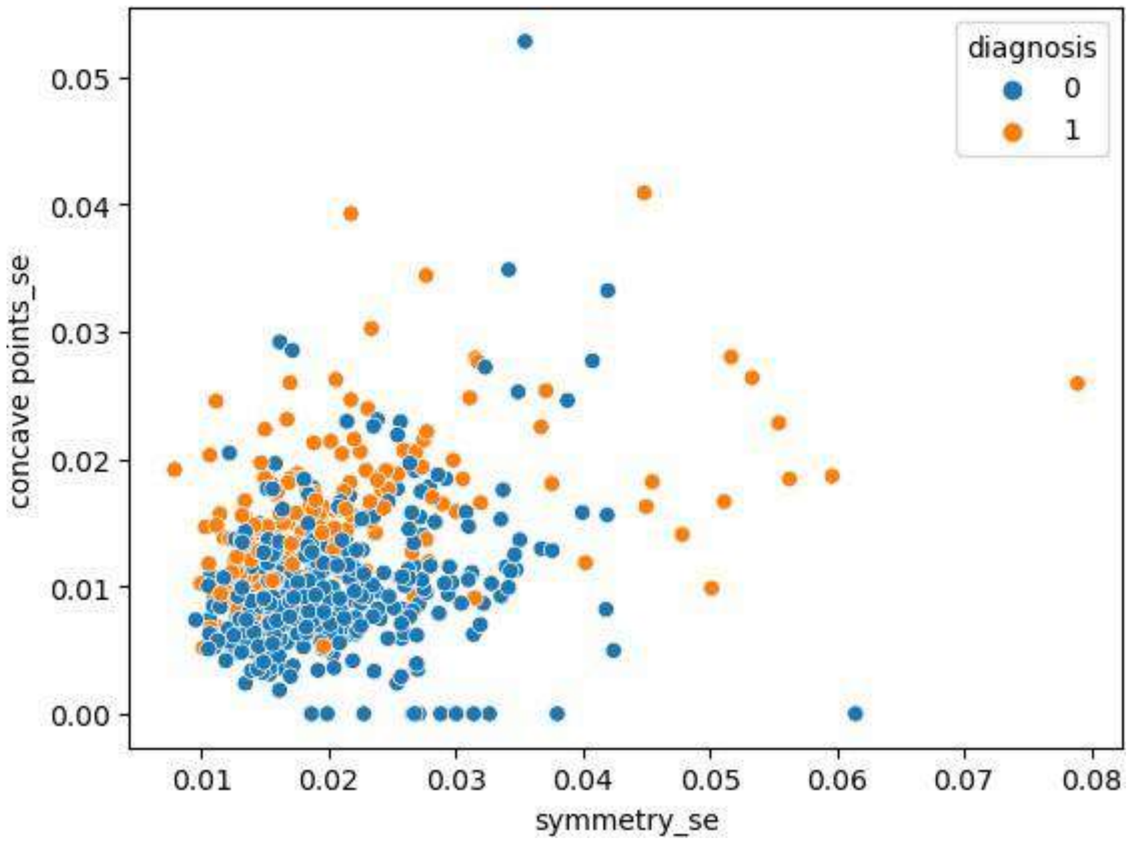
In [27]:

```
sns.scatterplot(data = breast, y = "concave points_mean", x = "symmetry_mean", hue="diagn
```



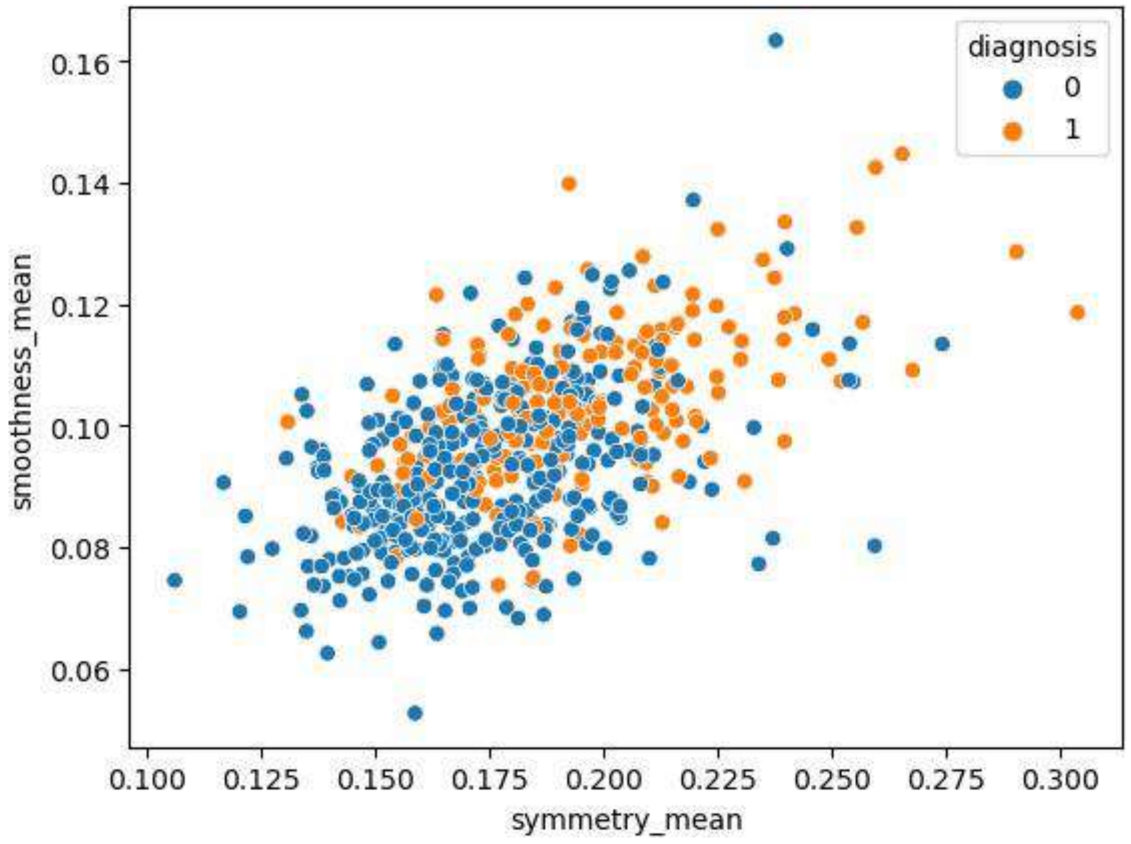
In [28]:

```
sns.scatterplot(data = breast, y = "concave points_se", x = "symmetry_se", hue="diagnosis
```



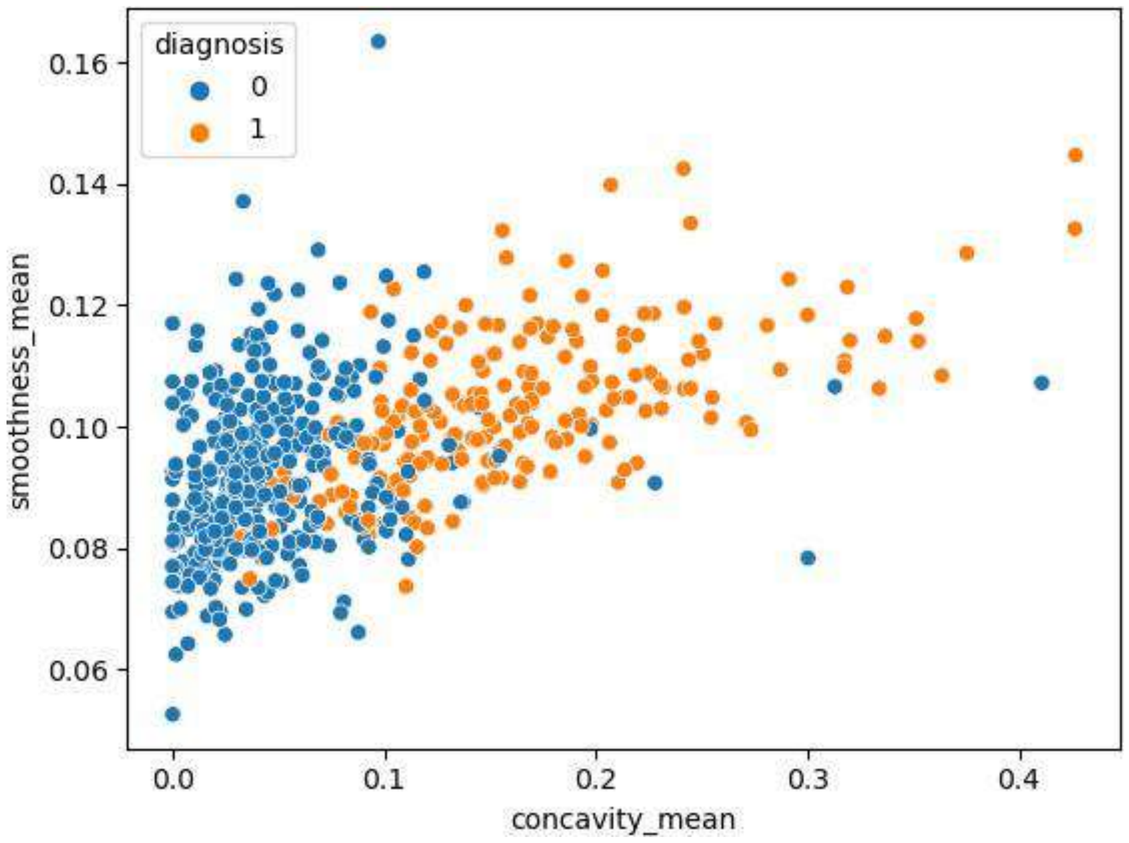
In [29]:

```
sns.scatterplot(data = breast, y = "smoothness_mean", x = "symmetry_mean", hue="diagnosis
```



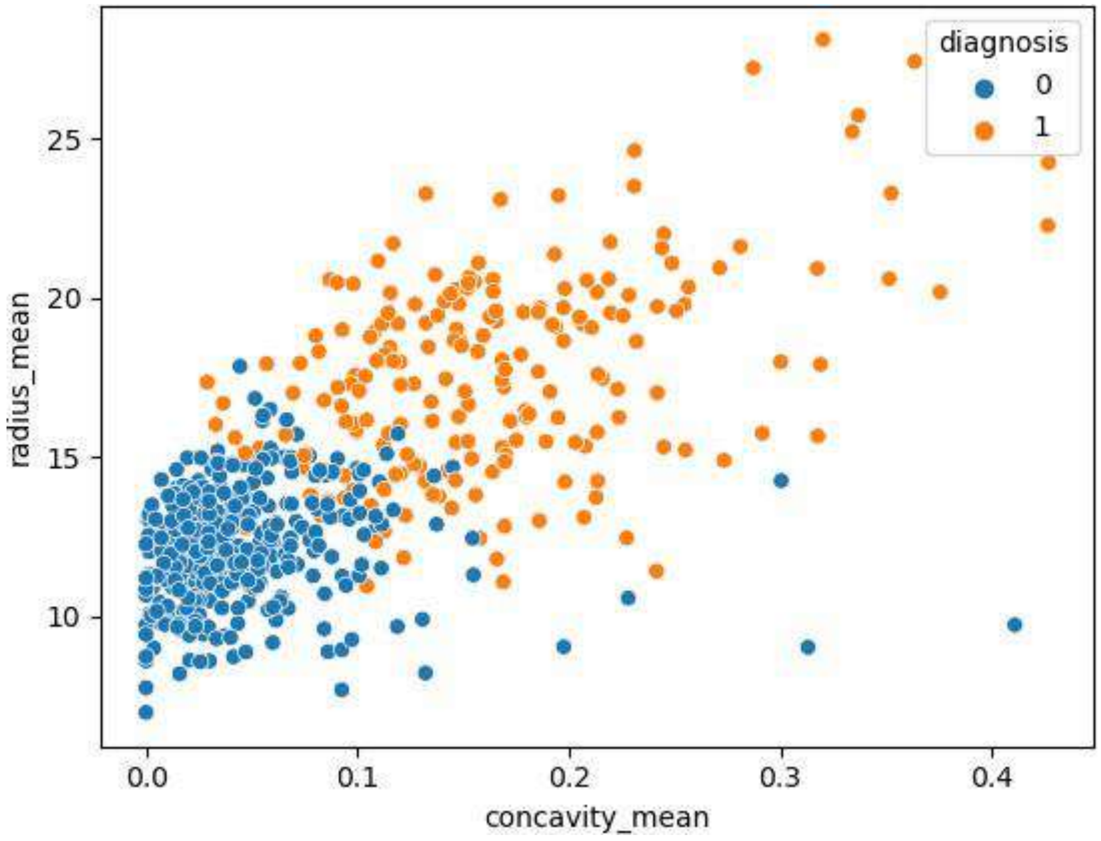
In [30]:

```
sns.scatterplot(data = breast, y = "smoothness_mean", x = "concavity_mean", hue="diagnosi
```



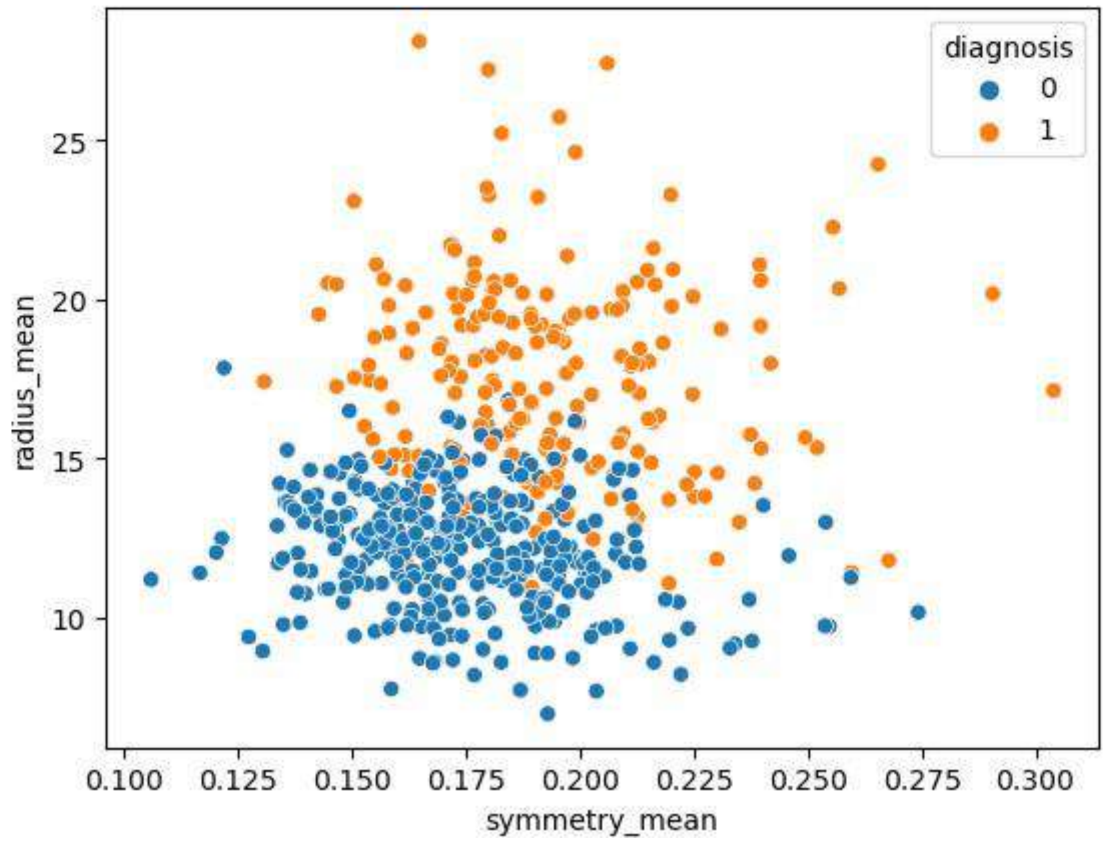
In [31]:

```
sns.scatterplot(data = breast, y = "radius_mean", x = "concavity_mean", hue="diagnosis");
```



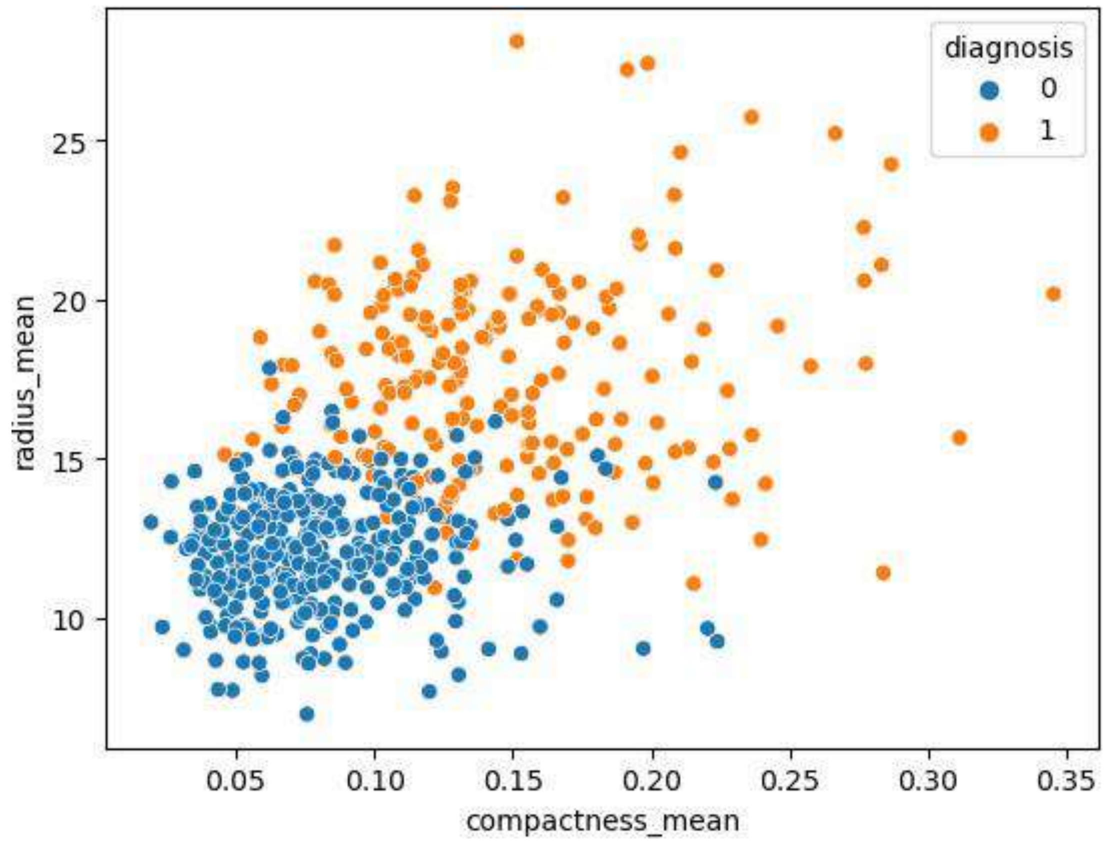
In [32]:

```
sns.scatterplot(data = breast, y = "radius_mean", x = "symmetry_mean", hue="diagnosis");
```



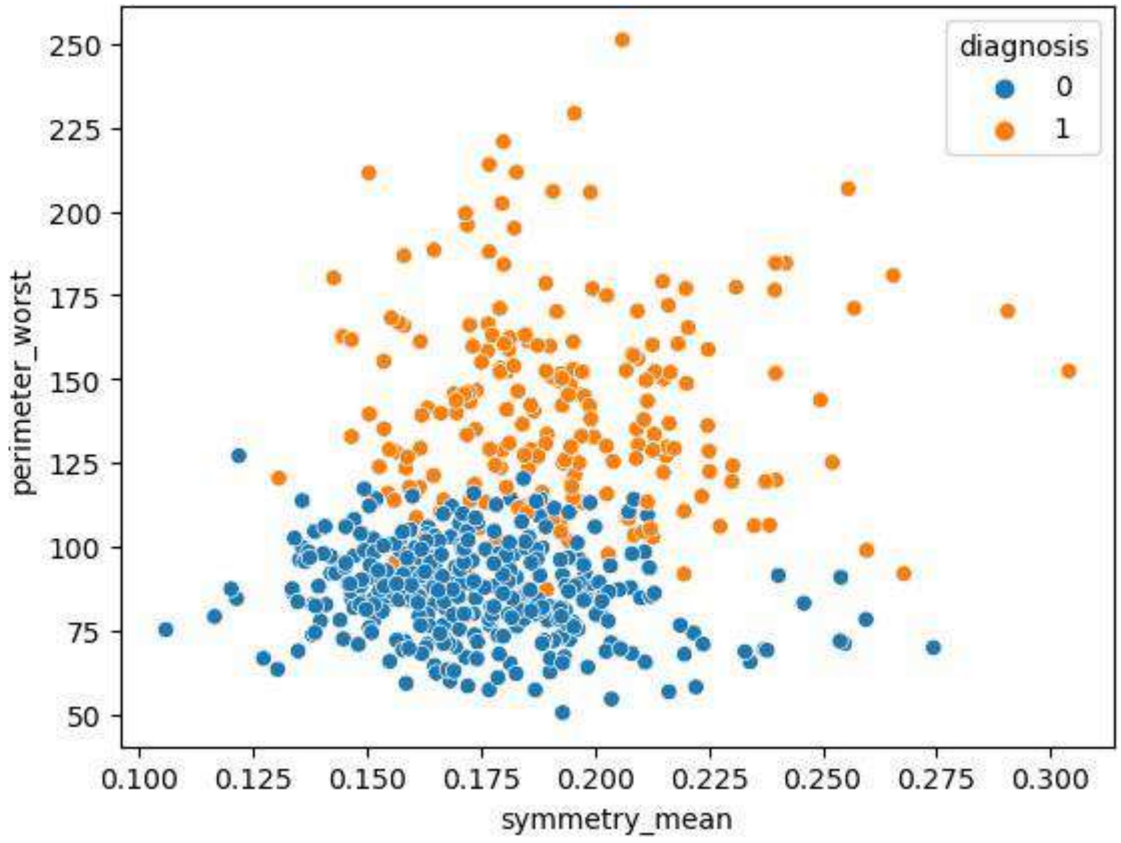
In [33]:

```
sns.scatterplot(data = breast, y = "radius_mean", x = "compactness_mean", hue="diagnosis"
```



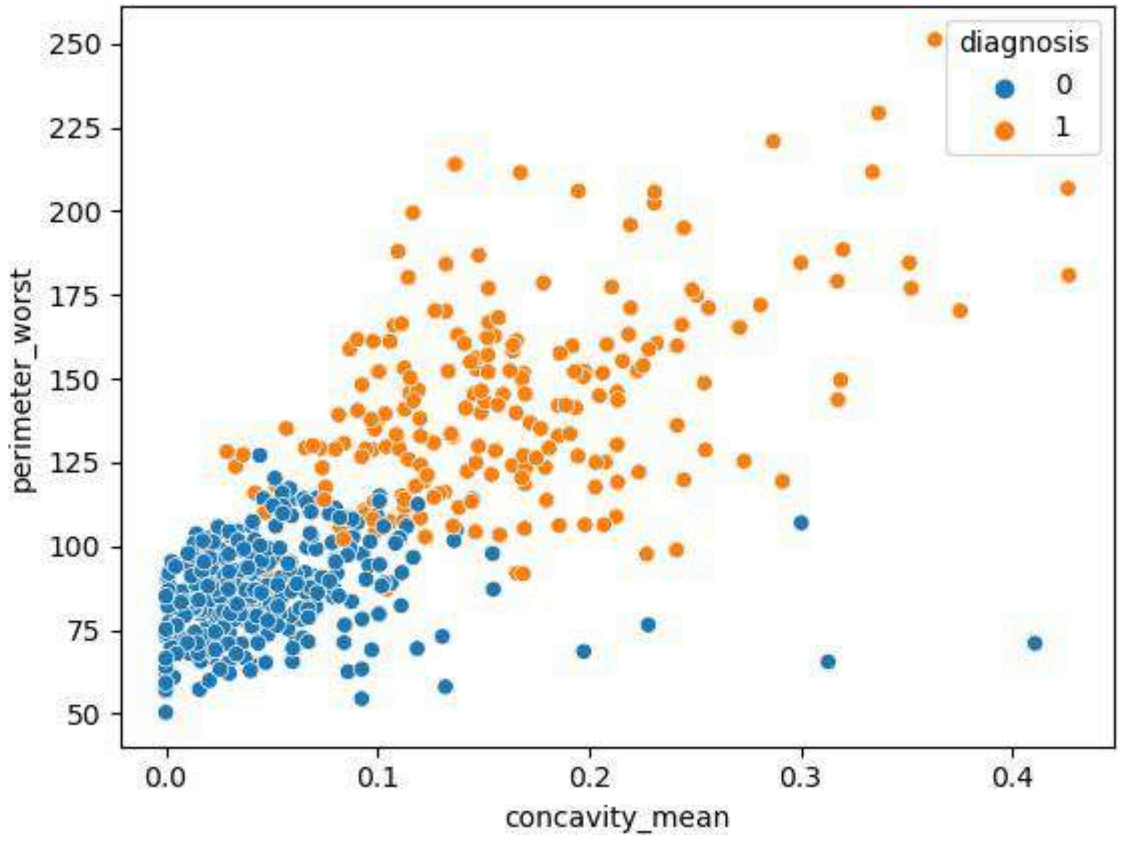
In [34]:

```
sns.scatterplot(data = breast, y = "perimeter_worst", x = "symmetry_mean", hue="diagnosis
```



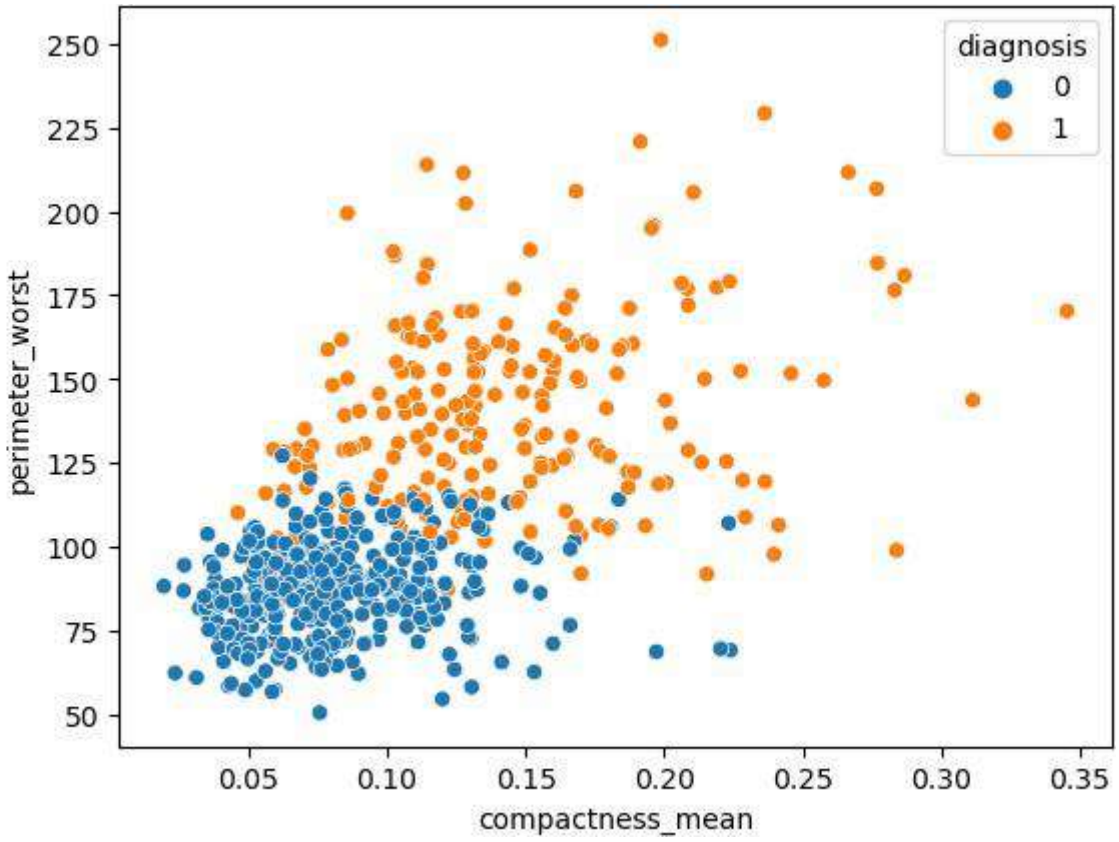
In [35]:

```
sns.scatterplot(data = breast, y = "perimeter_worst", x = "concavity_mean", hue="diagnosi
```



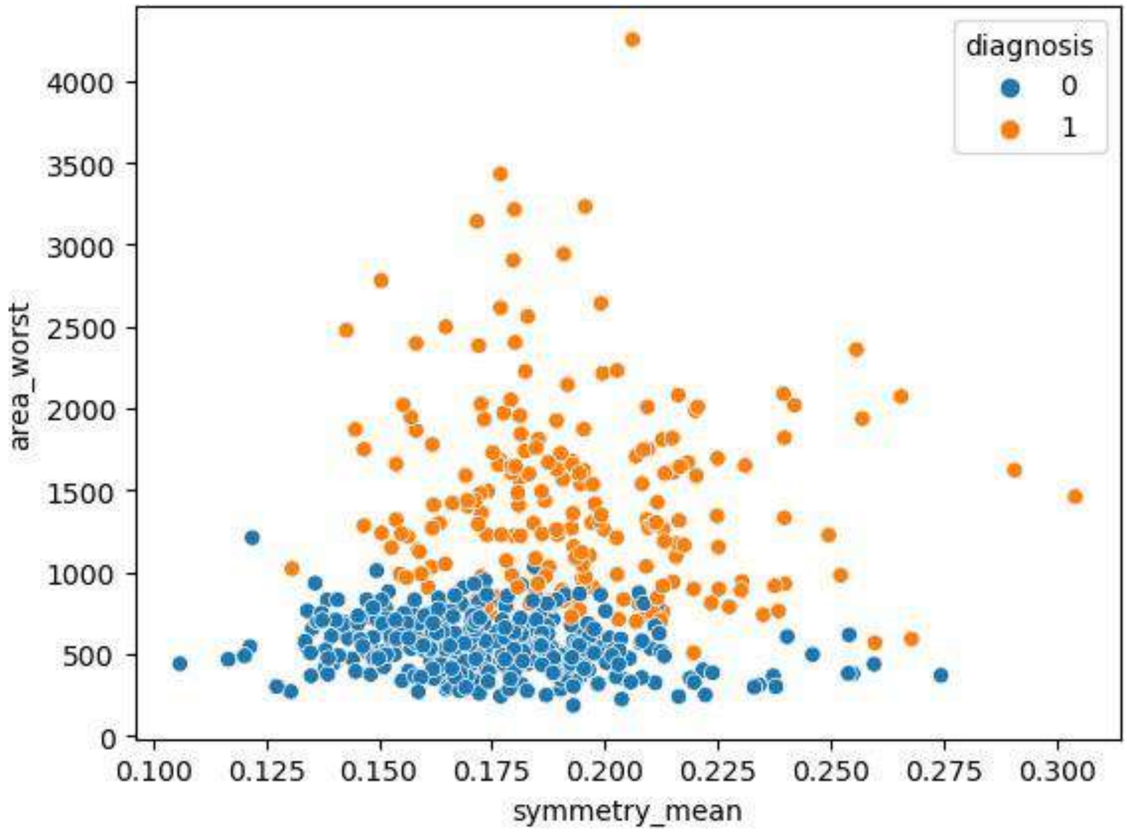
In [36]:

```
sns.scatterplot(data = breast, y = "perimeter_worst", x = "compactness_mean", hue="diagno
```



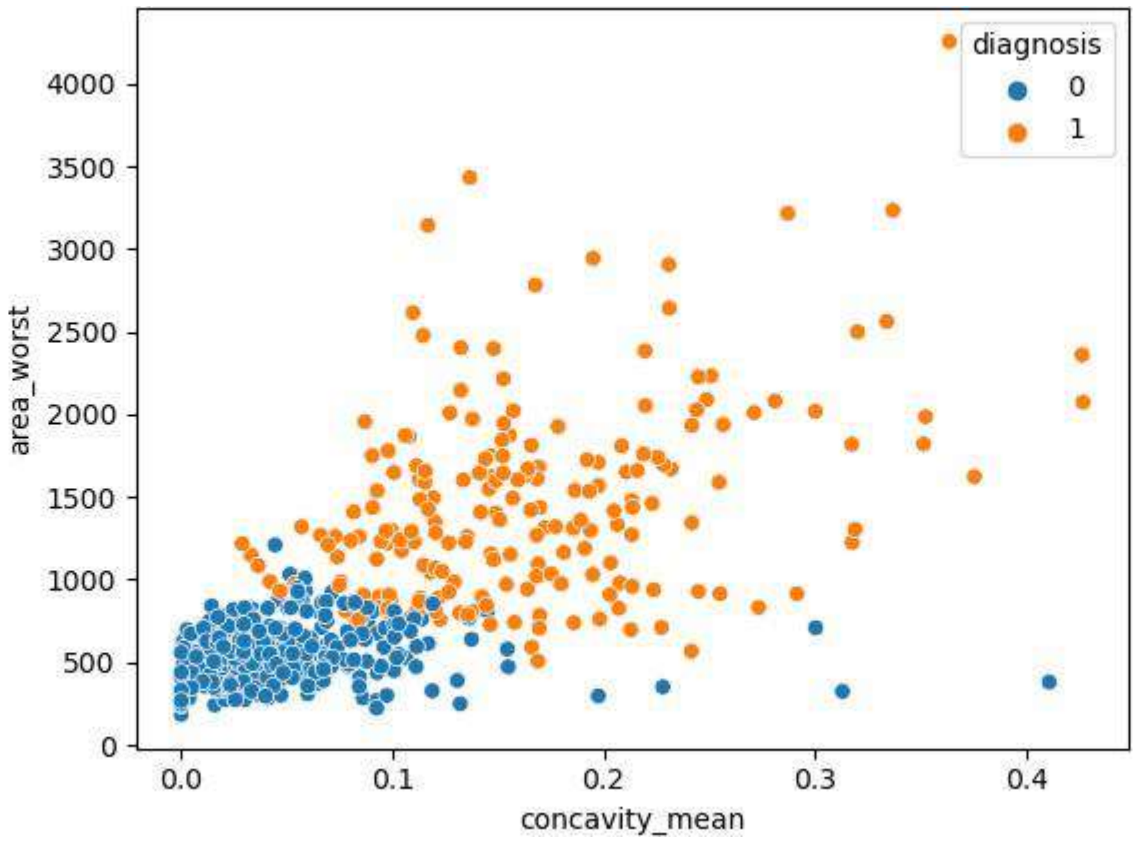
In [37]:

```
sns.scatterplot(data = breast, y = "area_worst", x = "symmetry_mean", hue="diagnosis");
```



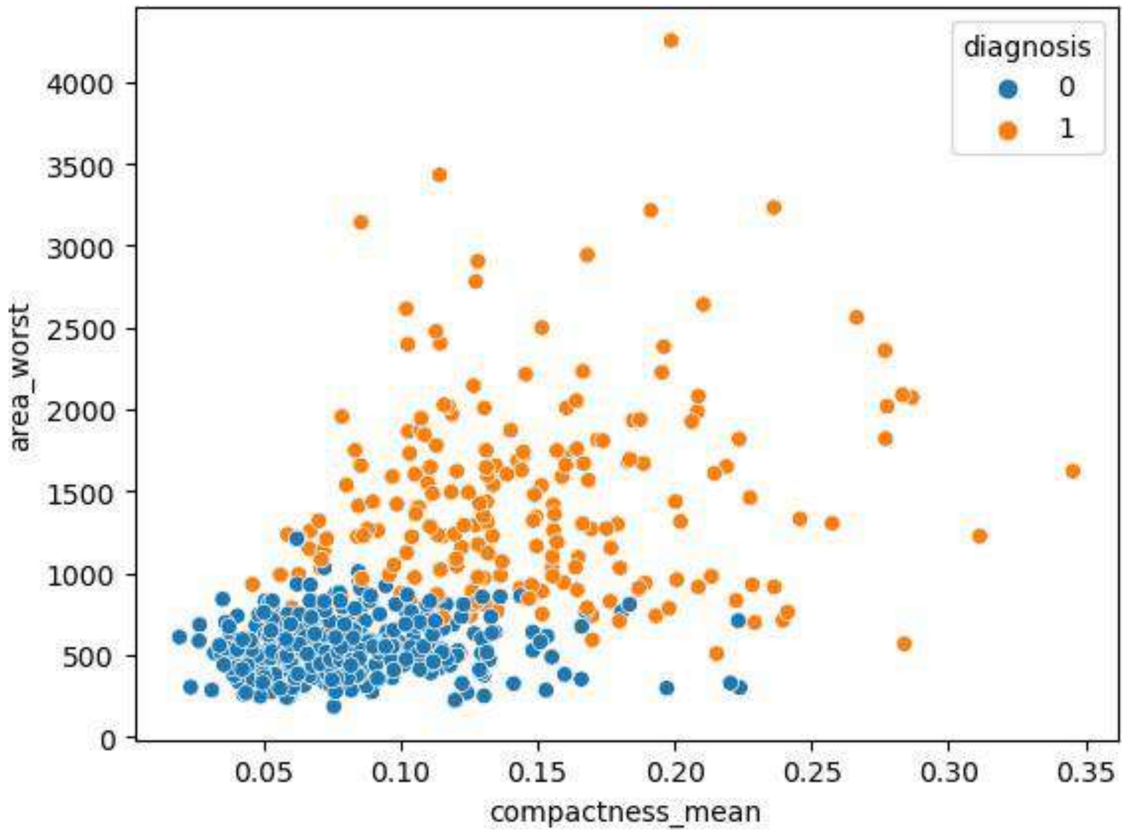
In [38]:

```
sns.scatterplot(data = breast, y = "area_worst", x = "concavity_mean", hue="diagnosis");
```



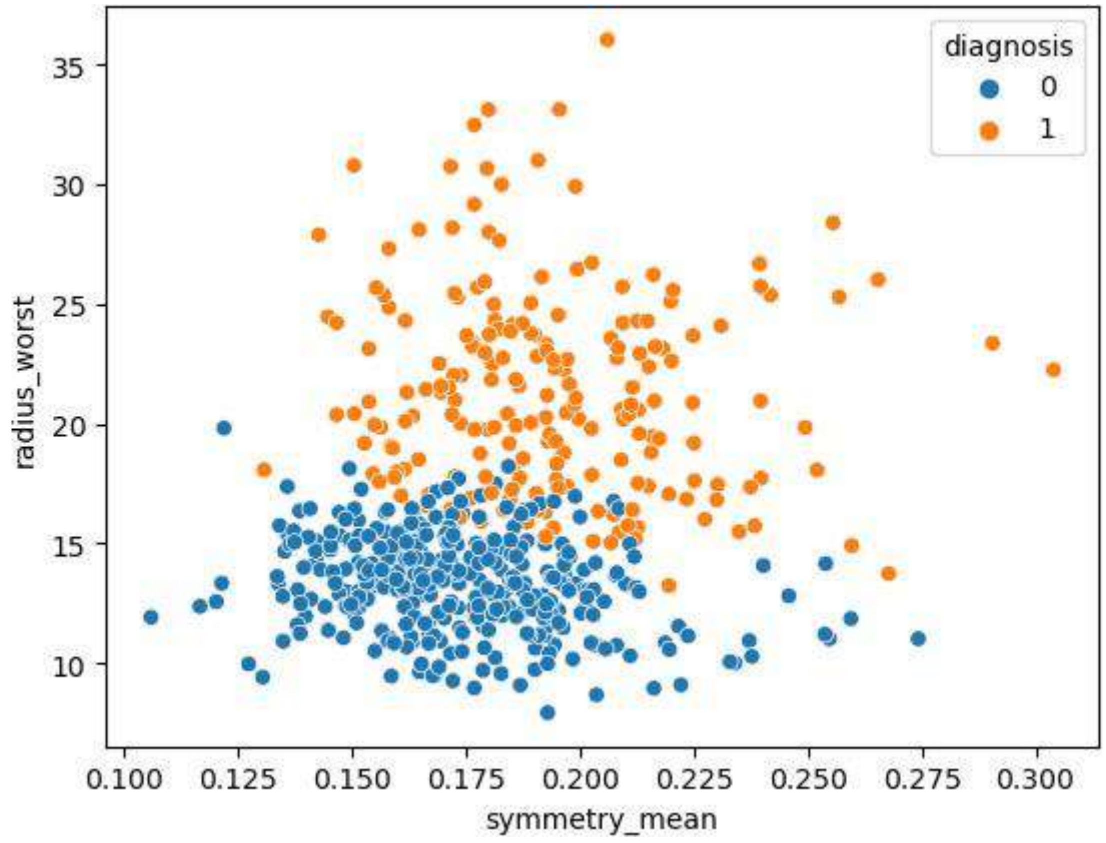
In [39]:

```
sns.scatterplot(data = breast, y = "area_worst", x = "compactness_mean", hue="diagnosis")
```



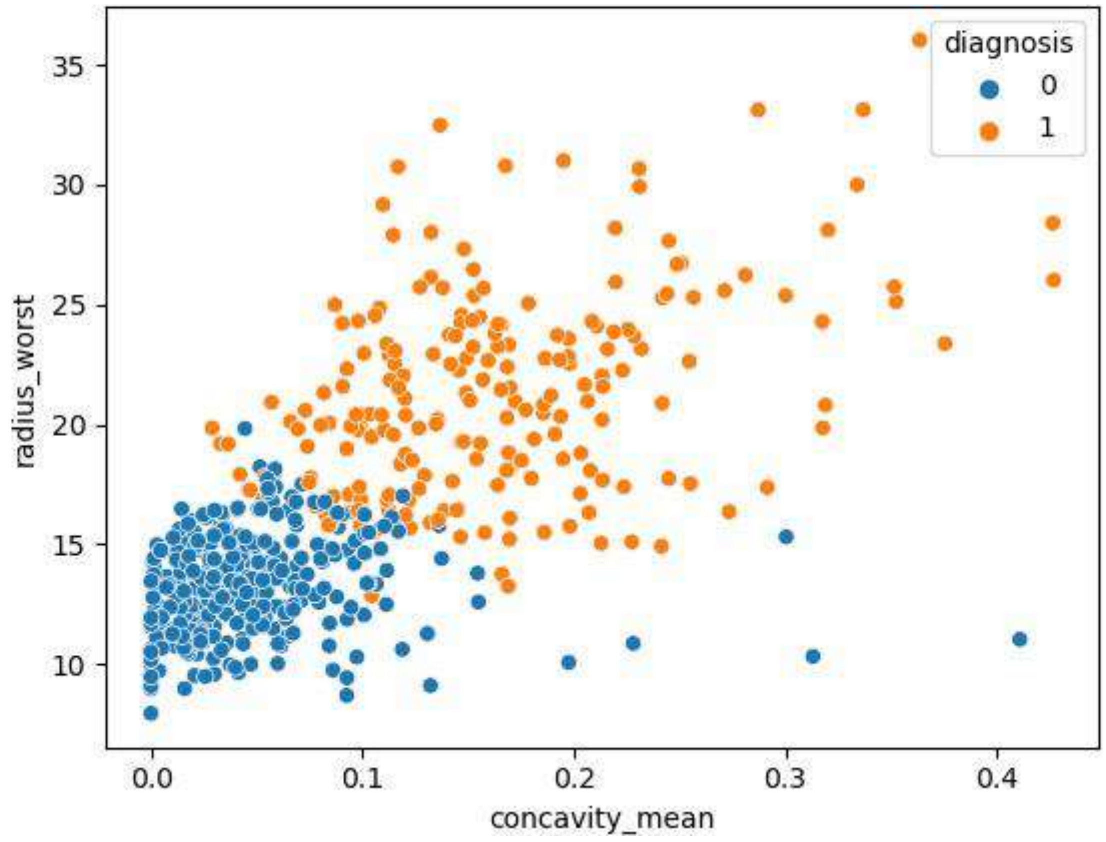
In [40]:

```
sns.scatterplot(data = breast, y = "radius_worst", x = "symmetry_mean", hue="diagnosis");
```



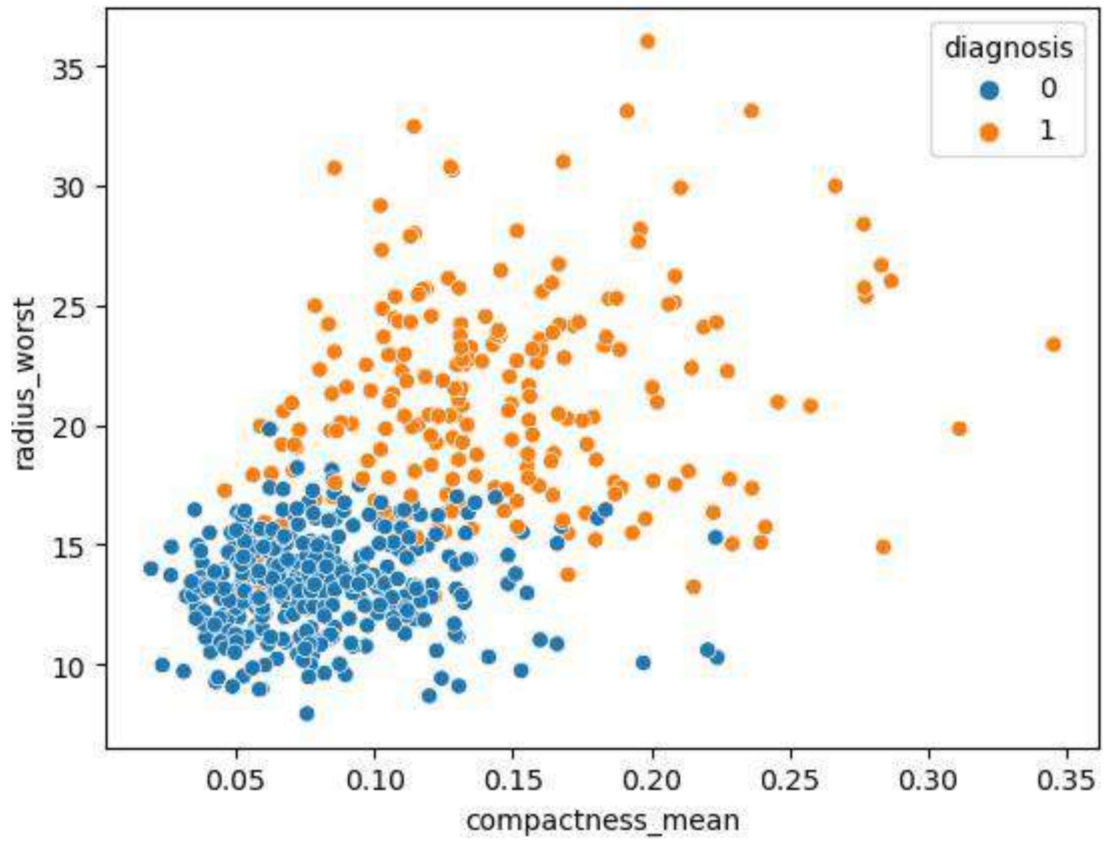
In [41]:

```
sns.scatterplot(data = breast, y = "radius_worst", x = "concavity_mean", hue="diagnosis")
```



In [42]:

```
sns.scatterplot(data = breast, y = "radius_worst", x = "compactness_mean", hue="diagnosis
```



In [ ]:

```

```

In [ ]:

```

```

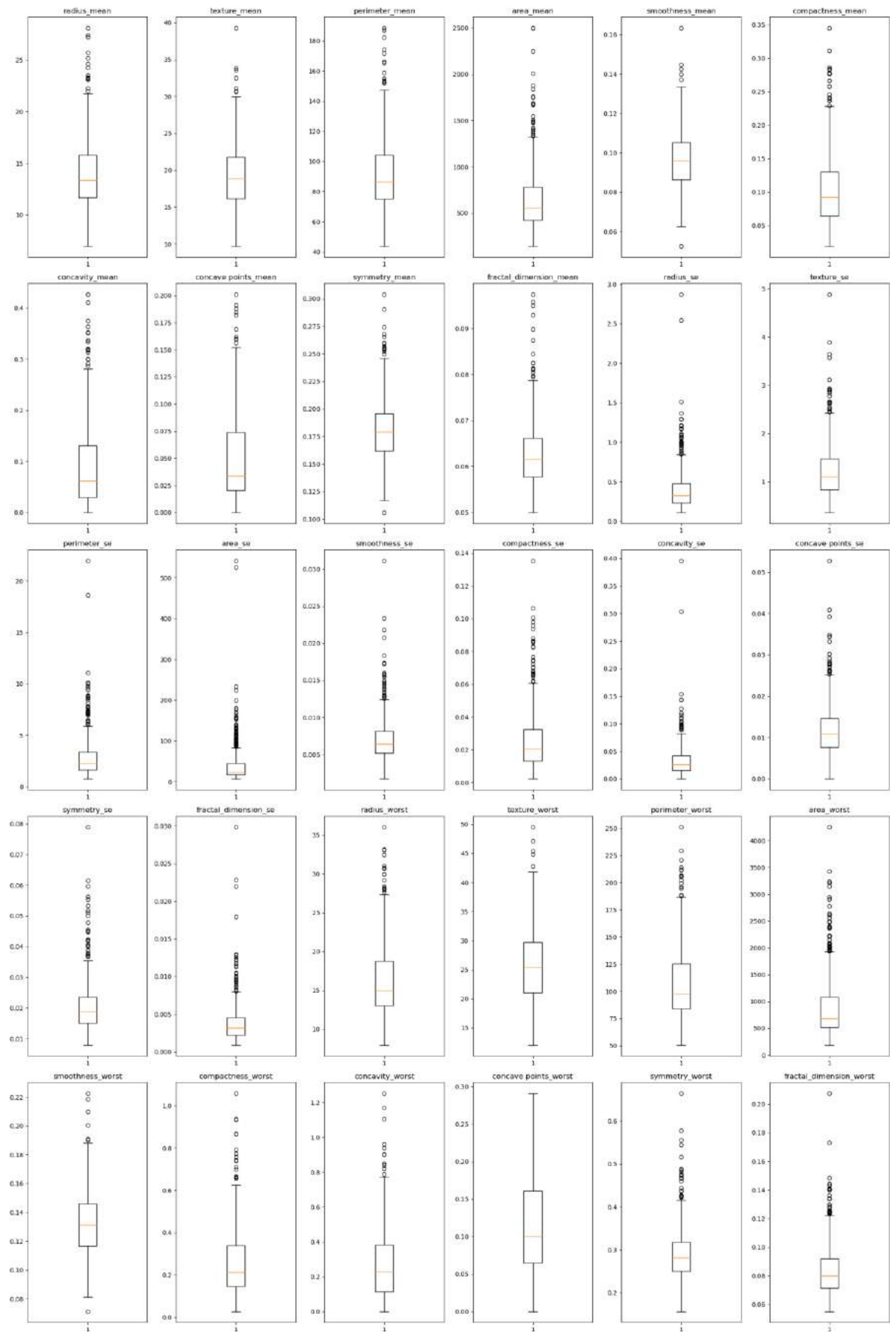
In [43]:

```
breast_cancer = breast.drop(["id", "diagnosis"], axis=1)
```

Check For Outliers

In [44]:

```
# outlier detection using boxplot

numeric_columns = breast_cancer.select_dtypes(include=np.number)
plt.figure(figsize=(20, 30))

for i, variable in enumerate(numeric_columns):
    plt.subplot(5, 6, i + 1)
    plt.boxplot(breast[variable])
    plt.tight_layout()  # to avoid the title from overlapping
    plt.title(variable)
```



Treating The Outliers

In [45]:

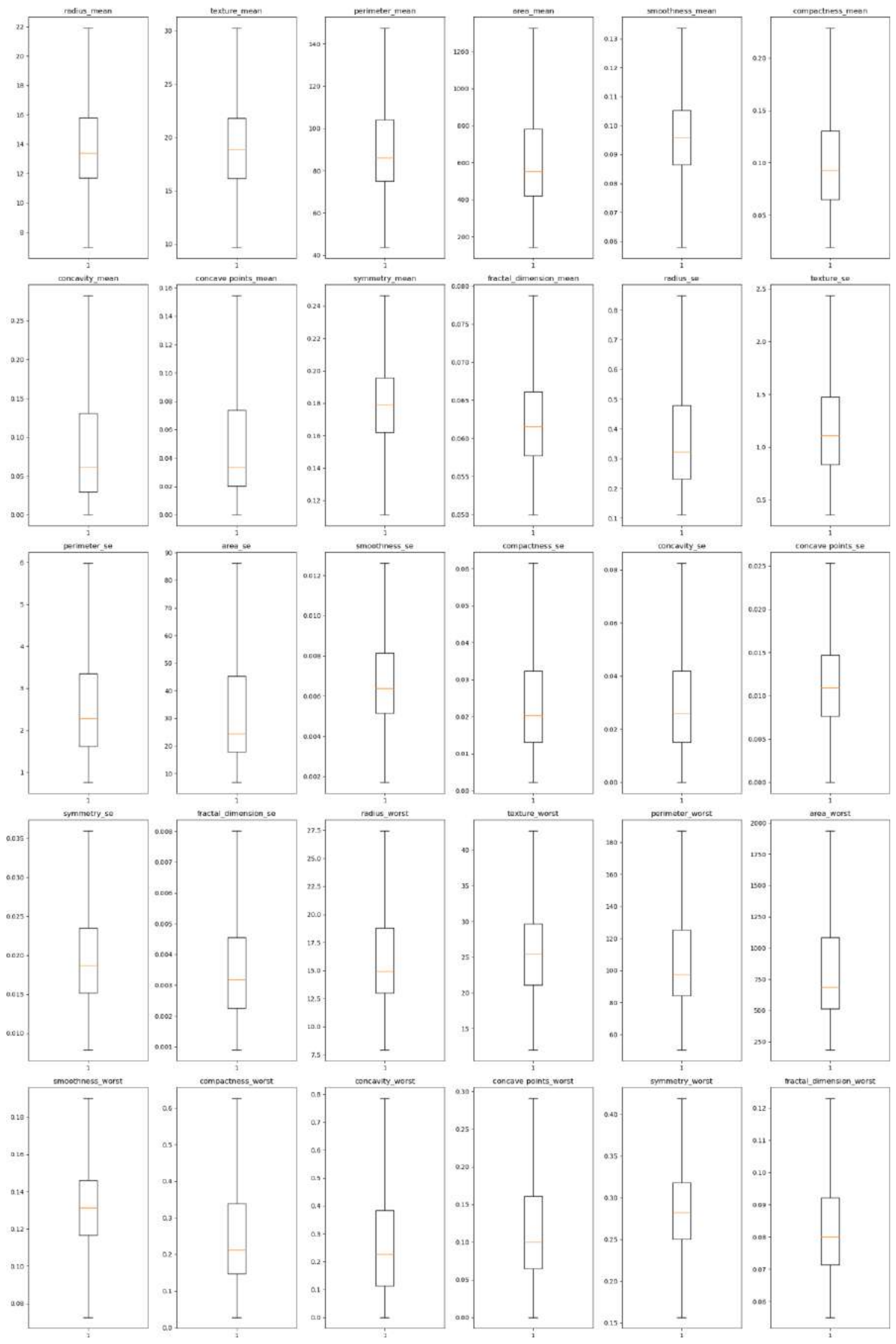
```
# Function to treat these outliers

def treat_outliers(df, col): """
    treats outliers in a varaible
    col: str, name of the numerical varaible df:
    data frame
    col: name of the column """
    Q1 = df[col].quantile(0.25)  # 25th quantile Q3
    = df[col].quantile(0.75)  # 75th quantile IQR =
    Q3 - Q1
    Lower_Whisker = Q1 - 1.5 * IQR
    Upper_Whisker = Q3 + 1.5 * IQR
    df[col] = np.clip(
        df[col], Lower_Whisker, Upper_Whisker
    )  # all the values samller than Lower_Whisker will be assigned value of Lower_whiske
    # and all the values above upper_whisker will be assigned value of upper_Whisker
    return df


def treat_outliers_all(df, col_list): """
    treat outlier in all numerical varaibles
    col_list: list of numerical varaibles
    df: data frame """
    for c in col_list:
        df = treat_outliers(df, c)

    return df
```

In [46]:

```
# treating the outliers
df_col = breast_cancer
df = treat_outliers_all(breast_cancer, df_col)
```

In [47]:

```
# let's look at the boxplots to see if the outliers have been treated or not
plt.figure(figsize=(20, 30))

for i, variable in enumerate(numeric_columns):
    plt.subplot(5, 6, i + 1)
    plt.boxplot(df[variable], whis=1.5)
    plt.tight_layout()
    plt.title(variable)
```



Type *Markdown* and LaTeX: □^{2}

Random Forest Classifier

In [48]:

```
# To confirm the values of target again

breast["diagnosis"].value_counts()
```

Out[48]:

```
0    357
1    212
Name: diagnosis, dtype: int64
```

In [49]:

```
# To select the features and the target

y = breast['diagnosis']
X = breast_cancer
```

In [50]:

```
# To split the data

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
```

In [51]:

```
# To check their dimension

x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[51]:

```
((455, 30), (114, 30), (455,), (114,))
```

In [52]:

```
# To create an instant of the model

rfcA = RandomForestClassifier(random_state=0)

# To train the model

rfcA.fit(x_train, y_train)
```

Out[52]:

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=0)
```

In [53]:

```
rfcA.get_params().keys()
```

Out[53]:

```
dict_keys(['bootstrap', 'ccp_alpha', 'class_weight', 'criterion', 'max_dep th',
'max_features', 'max_leaf_nodes', 'max_samples', 'min_impurity_decrea se',
'min_samples_leaf', 'min_samples_split', 'min_weight_fraction_leaf', 'n_estimators',
'n_jobs', 'oob_score', 'random_state', 'verbose', 'warm_st art'])
```

In [54]:

```
# function to compute different metrics to check performance of a classification model

def model_performance_classification(model, predictors, target, threshold = 0.5): """
    Function to compute different metrics to check classification model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    """

    # predicting using the independent variables
    pred_proba = model.predict_proba(predictors)[:, 1]
     # convert probability to class
    pred_class = np.round(pred_proba > threshold)
    acc = accuracy_score(target, pred_class)  # to compute accuracy
    recall = recall_score(target, pred_class)  # to compute recall
    precision = precision_score(target, pred_class)  # to compute precision
    f1 = f1_score(target, pred_class)  # to compute F1 score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {
            "Accuracy": acc,
            "Recall": recall,
            "Precision": precision,
            "F1-score": f1,
        },
        index=[0],),
    conf = confusion_matrix(target, pred_class) plt.figure(figsize=(10,
    6))
    sns.heatmap(conf, fmt="g", annot=True)
    plt.ylabel("Actual Value")
    plt.xlabel("Predicted value")
```

In [55]:

```
# To know the important values of the variables the model used

rfcA.feature_importances_
```

Out[55]:

```
array([0.03805861, 0.01264832, 0.04625365, 0.03577696, 0.00842457,
       0.00904298, 0.08822929, 0.13792714, 0.00471318, 0.00339536,
       0.02485115, 0.0044556 , 0.01200953, 0.02005183, 0.00526482,
       0.00496037, 0.00368534, 0.00368475, 0.00386233, 0.00488991,
       0.09990893, 0.01539526, 0.13698009, 0.07153559, 0.01176899,
       0.01314631, 0.04394757, 0.11660598, 0.0114667 , 0.00705889])
```

In [56]:

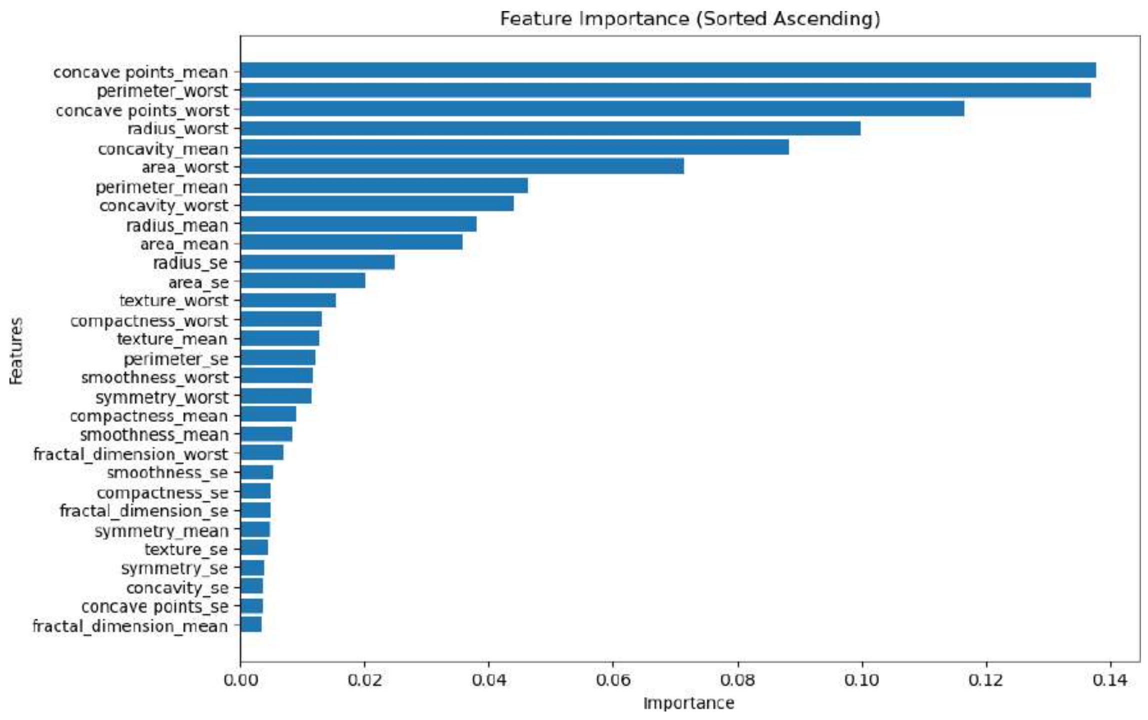
```
feature_importances = rfcA.feature_importances_
```

In [57]:

```
feature_names = [
    'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean',
    'compactness_mean', 'concavity_mean',
    'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
    'radius_se', 'texture_se', 'perimeter_se', 'area_se',
    'smoothness_se', 'compactness_se', 'concavity_se',
    'concave points_se', 'symmetry_se', 'fractal_dimension_se',
    'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst',
    'smoothness_worst', 'compactness_worst', 'concavity_worst',
    'concave points worst', 'symmetry worst', 'fractal dimension worst'
```

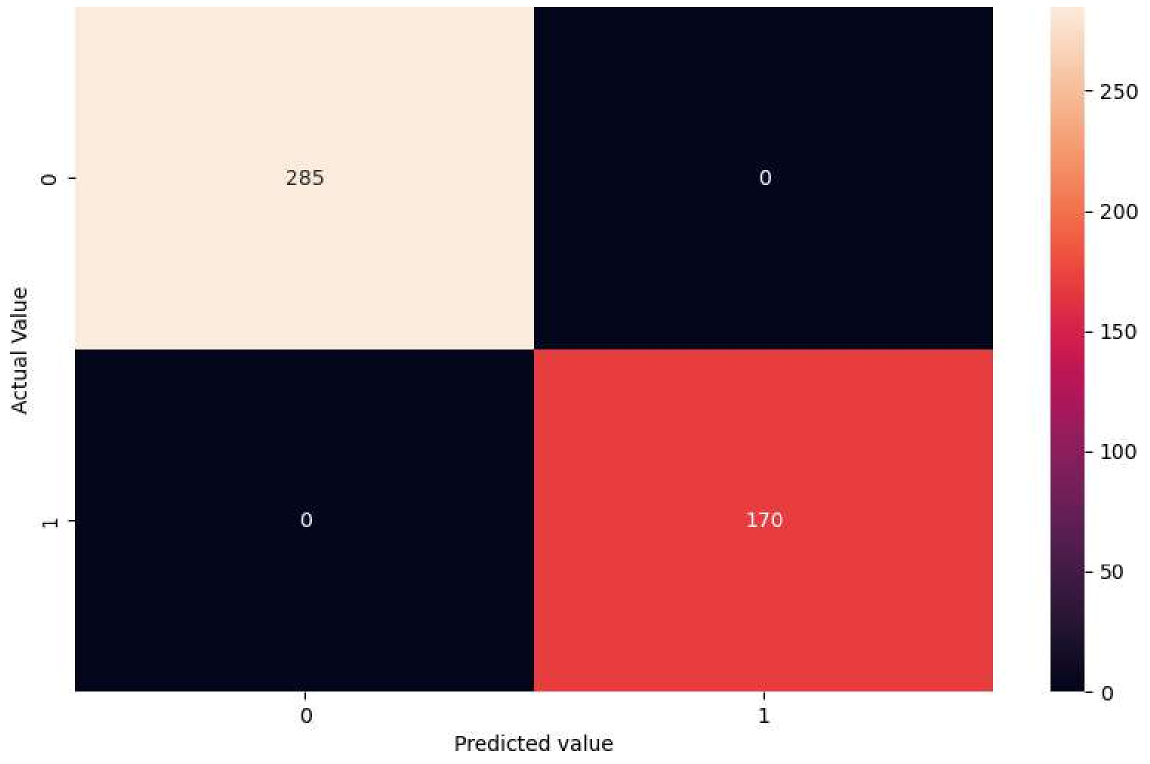
In [58]:

```
#To sorts feature importances and their corresponding names in ascending order then creat
# and visualize the sorted feature importances with their respective feature names.

sorted_indices = np.argsort(feature_importances)
sorted_feature_names = [feature_names[i] for i in sorted_indices]
sorted_feature_importances = [feature_importances[i] for i in sorted_indices]

plt.figure(figsize=(10, 7))
plt.barh(sorted_feature_names, sorted_feature_importances) plt.xlabel('Importance')
plt.ylabel('Features')
plt.title('Feature Importance (Sorted Ascending)') plt.show()
```



In [59]:

```
# To check the training performance

breast_train = model_performance_classification(rfcA, x_train, y_train) breast_train
```

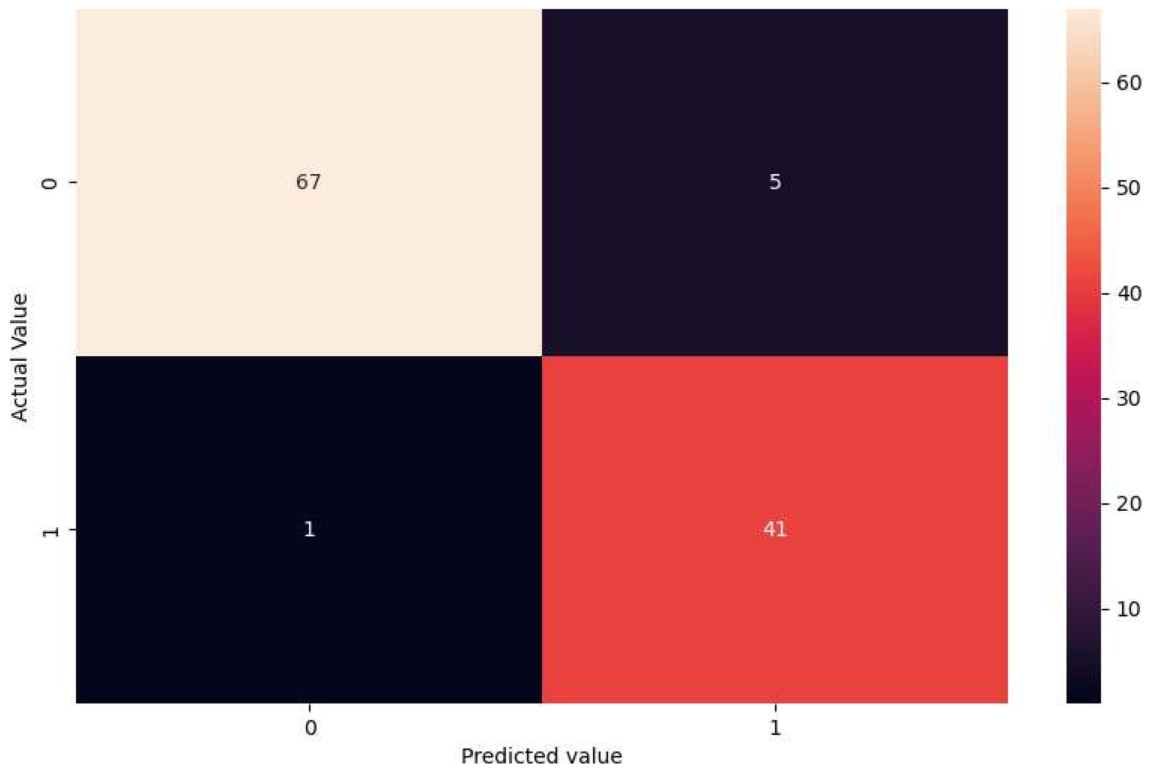


Out[59]:

```
(   Accuracy  Recall  Precision  F1-score
 0       1.0     1.0        1.0       1.0,)
```

In [60]:

```
# To check the model testing performance

breast_test = model_performance_classification(rfcA, x_test, y_test)

breast_test
```



Out[60]:

```
(   Accuracy    Recall  Precision  F1-score
 0  0.947368  0.97619    0.891304  0.931818,)
```

Type *Markdown* and LaTeX: α^2

Gradient Boosting Classifier

In [61]:

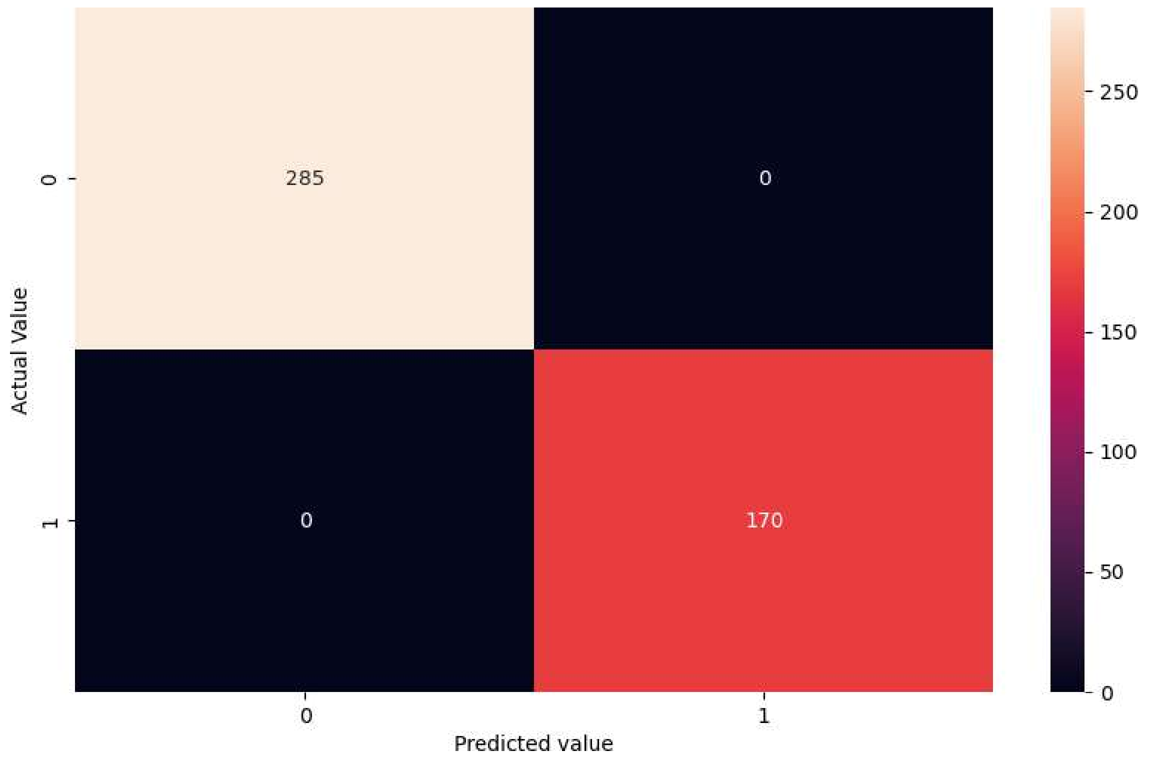
```
# To instantiate the model

gbA = GradientBoostingClassifier(random_state=0)

# To build the model

gbA.fit(x_train, y_train)
```

Out[61]:

```
▾          GradientBoostingClassifier
GradientBoostingClassifier(random_state=0)
```

In [62]:

```
breast_train_gbA = model_performance_classification(gbA, x_train, y_train)
breast_train_gbA
```

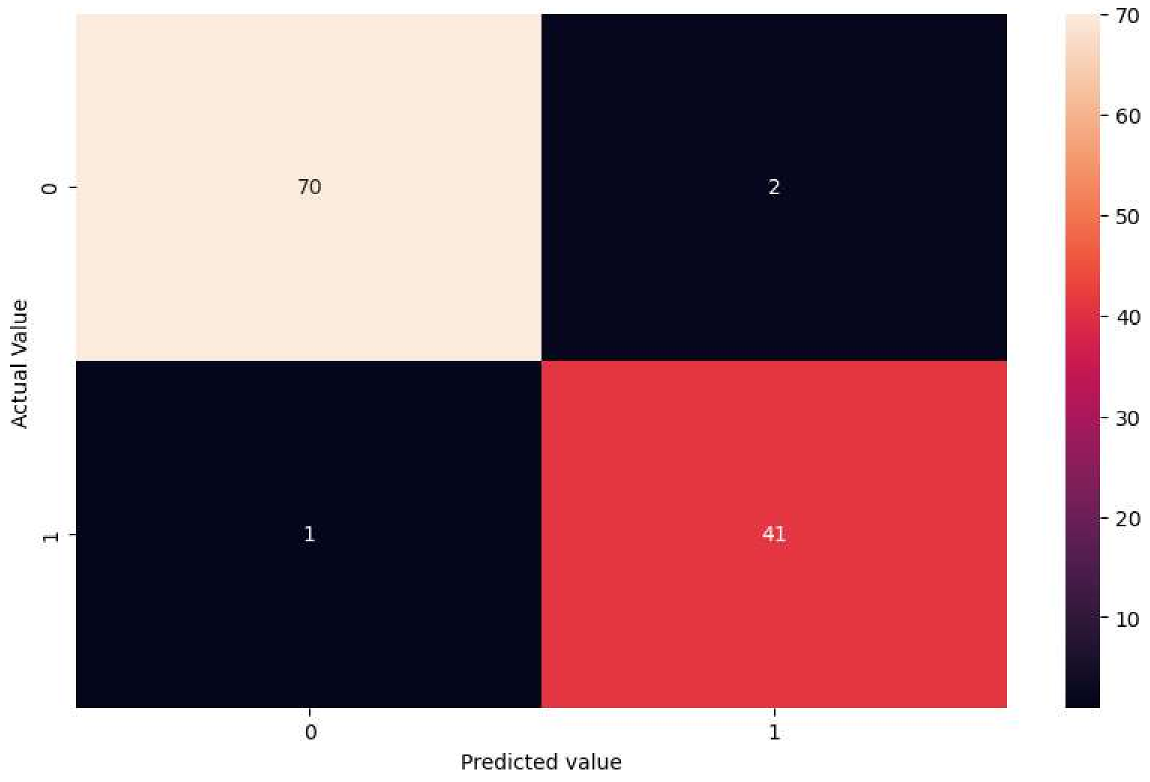


Out[62]:

```
(    Accuracy  Recall  Precision  F1-score
 0        1.0     1.0        1.0       1.0,)
```

In [63]:

```
# To check the model testing performance

breast_test_gbA = model_performance_classification(gbA, x_test, y_test) breast_test_gbA
```



Out[63]:

```
(   Accuracy    Recall  Precision  F1-score
 0  0.973684  0.97619      0.953488  0.964706,)
```

In [64]:

```
# To know the important values of the variables the model used

gbA.feature_importances_
```

Out[64]:

```
array([4.50453366e-04, 2.42201574e-02, 1.56235686e-03, 4.21032261e-03,
       5.02075900e-03, 1.30674128e-03, 4.09107831e-03, 5.48149890e-01,
       1.41698833e-04, 1.58659640e-03, 1.70521561e-03, 1.08883652e-03,
       1.66695207e-03, 1.65458024e-02, 1.67725717e-03, 5.54644937e-04,
       1.11128987e-03, 2.97616479e-04, 4.07473479e-04, 8.55980549e-04,
       5.88288814e-02, 3.18077854e-02, 7.39096875e-02, 7.86675717e-02,
       1.31266409e-02, 1.59700046e-03, 4.31689912e-02, 8.16234721e-02,
       4.55352165e-04, 1.63493590e-04])
```

In [65]:

```
# To get the predicted probabilities for the positive (malignant) class (1)
predicted_probabilities=gbA.predict_proba(x_test)[:, 1]
```

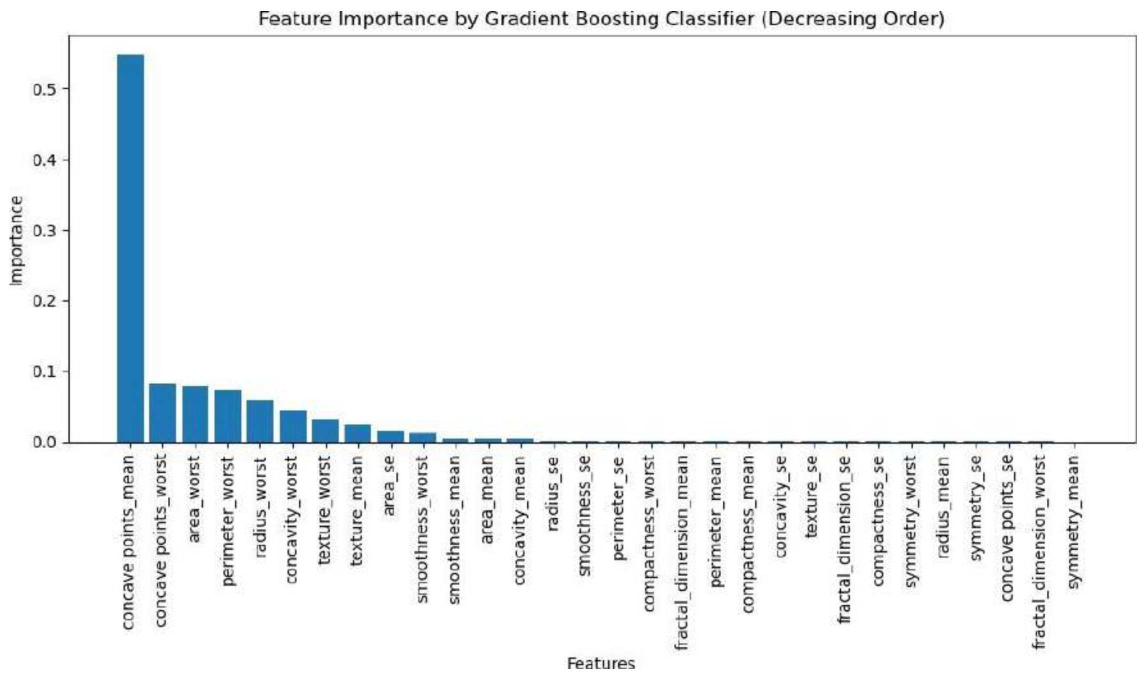
In [66]:

```
#Computing the value of AUC-ROC Score
auc_roc=roc_auc_score(y_test, predicted_probabilities) print
("AUC-ROC:", auc_roc)
```

```
AUC-ROC: 0.9900793650793651
```

In [67]:

```
# Get feature importances
feature_importances = gbA.feature_importances_

# Get indices that would sort the feature importances in descending order
sorted_indices  = np.argsort(feature_importances)[::-1]

#  Sort feature importances and feature names accordingly
sorted_importances    =    feature_importances[sorted_indices]
sorted_feature_names = X.columns[sorted_indices]

# Create a bar plot
plt.figure(figsize=(10, 6))
plt.bar(range(X.shape[1]), sorted_importances)
plt.xticks(range(X.shape[1]), sorted_feature_names, rotation=90) plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importance by Gradient Boosting Classifier (Decreasing Order)')
plt.tight_layout()
```



In [68]:

```
# Selecting the important features

breast6 =breast_cancer[['concave points_mean','area_worst','radius_worst','perimeter_wors
                   'concave points_worst']]
```

In [69]:

```
# To split the data

x_train6, x_test6, y_train6, y_test6 = train_test_split(breast6, y, test_size = 0.2, rand
```

In [70]:

```
x_train6.shape, x_test6.shape, y_train6.shape, y_test6.shape
```

Out[70]:

```
((455, 6), (114, 6), (455,), (114,))
```

In [71]:

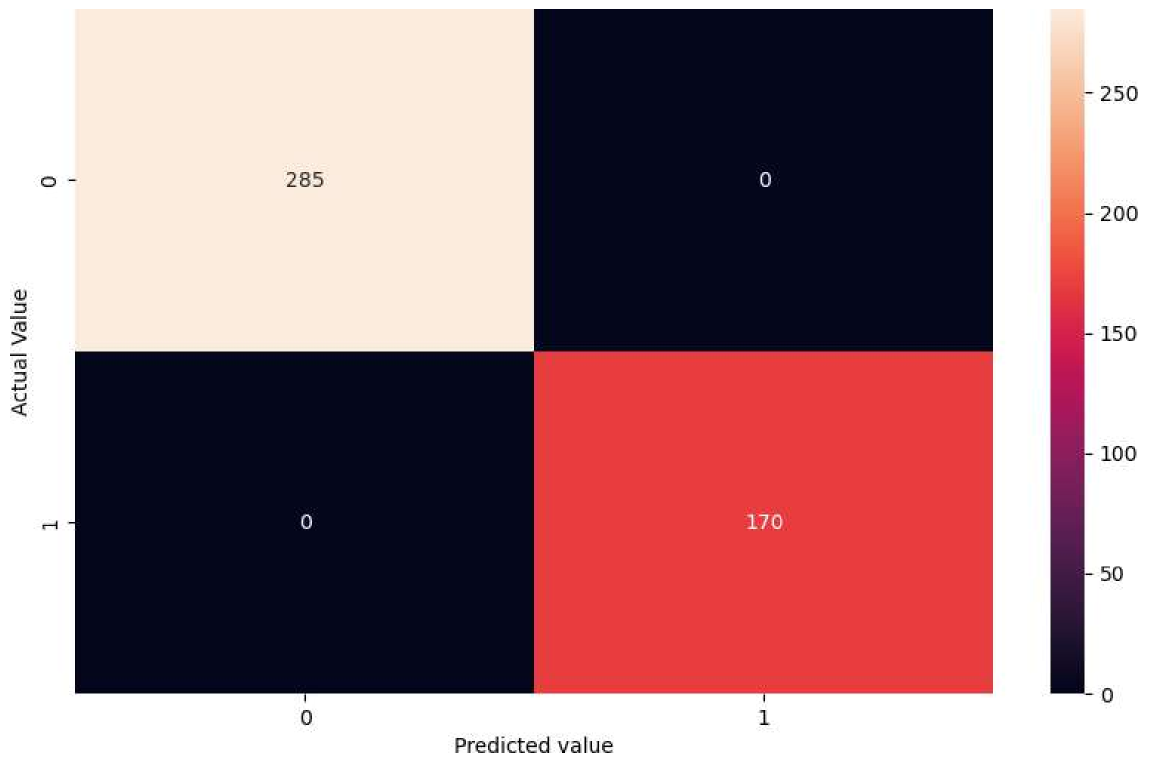
```
# To instantiate the model

gbB = GradientBoostingClassifier(random_state=0)

# To build the model

gbB.fit(x_train6, y_train6)
```

Out[71]:

```
▾          GradientBoostingClassifier
GradientBoostingClassifier(random_state=0)
```

In [72]:

```
breast_train_gbB = model_performance_classification(gbB, x_train6, y_train6) breast_train_gbB
```

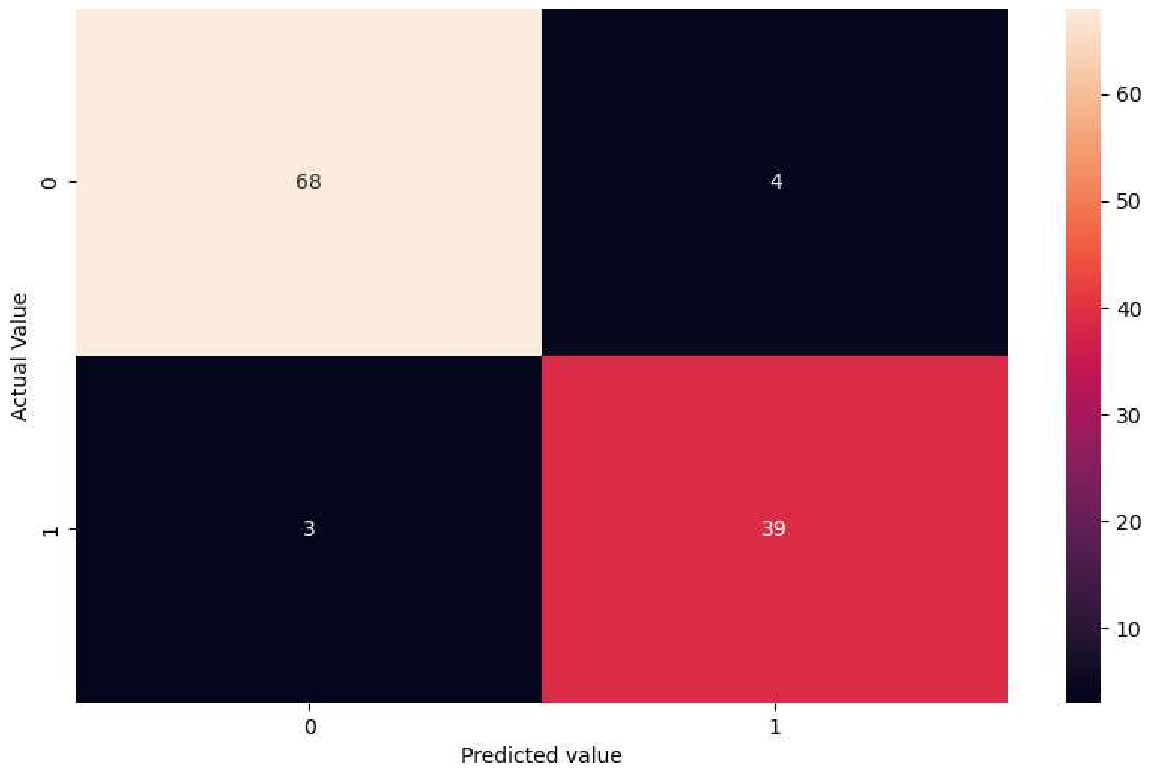


Out[72]:

```
(   Accuracy  Recall  Precision  F1-score
 0       1.0     1.0        1.0       1.0,)
```

In [73]:

```
breast_test_gbB = model_performance_classification(gbB, x_test6, y_test6)
breast_test_gbB
```



Out[73]:

```
(    Accuracy     Recall   Precision   F1-score
 0   0.938596   0.928571     0.906977   0.917647,)
```

Type *Markdown* and LaTeX: α^2

Logistic Regression

In [74]:

```
# To instantiate the Standard scaler

scaler = StandardScaler()

# To calculate and convert the variables to have a mean of 0 and standard deviation of 1

x_train_scaler = scaler.fit_transform(x_train)

x_test_scaler = scaler.transform(x_test)
```

In [75]:

```
# Create an instant of the logistics regression

logreg = LogisticRegression(random_state=0)

#To train or build the model

logreg.fit(x_train_scaler, y_train)
```

Out[75]:

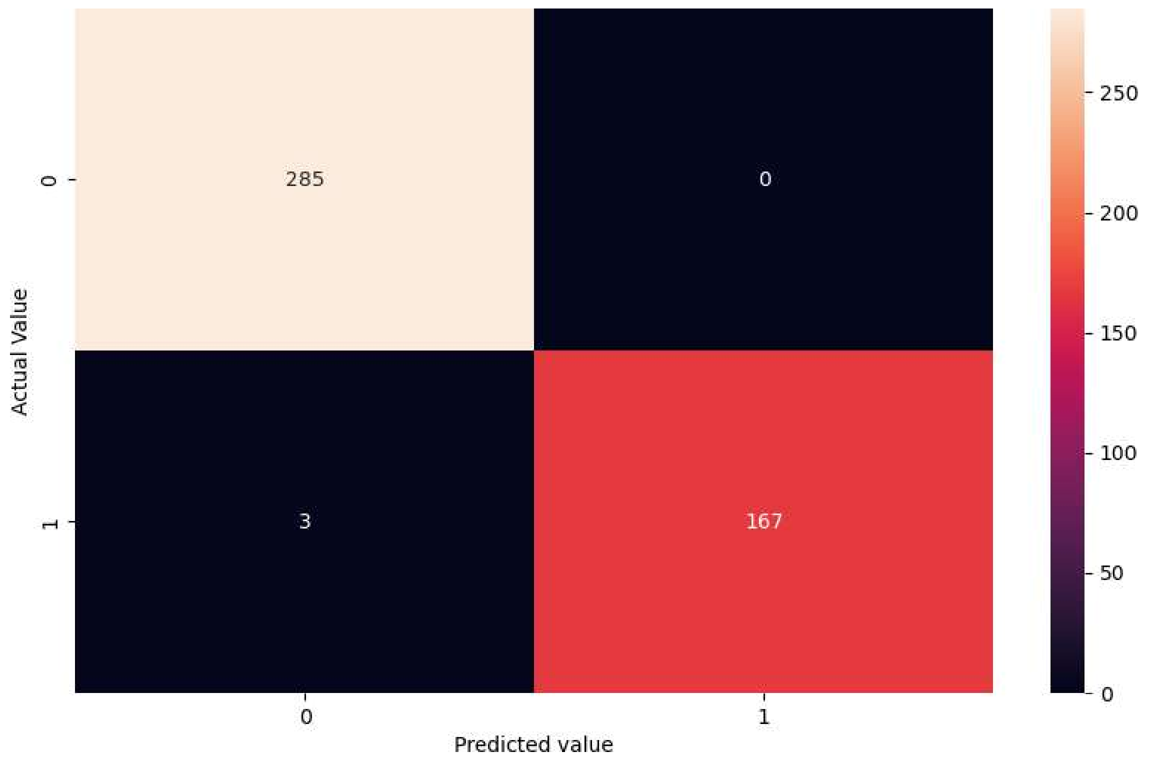
```
▾         LogisticRegression
LogisticRegression(random_state=0)
```

In [76]:

```

# function to compute different metrics to check performance of a classification model

def model_performance_classification(model, predictors, target, threshold = 0.5): """
    Function to compute different metrics to check classification model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    """

    # predicting using the independent variables
    pred_proba = model.predict_proba(predictors)[:, 1]
      # convert probability to class
    pred_class = np.round(pred_proba > threshold)
    acc = accuracy_score(target, pred_class)  # to compute accuracy
    recall = recall_score(target, pred_class)  # to compute recall
    precision = precision_score(target, pred_class)  # to compute precision
    f1 = f1_score(target, pred_class)  # to compute F1 score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {
            "Accuracy": acc,
            "Recall": recall,
            "Precision": precision,
            "F1-score": f1,
        },
        index=[0],),
    conf = confusion_matrix(target, pred_class) plt.figure(figsize=(10,
    6))
    sns.heatmap(conf, fmt="g", annot=True)
    plt.ylabel("Actual Value")
    plt.xlabel("Predicted value")
```

In [77]:

```
# Training performance

breast_train_logreg = model_performance_classification(logreg, x_train_scaler, y_train)
breast_train_logreg
```

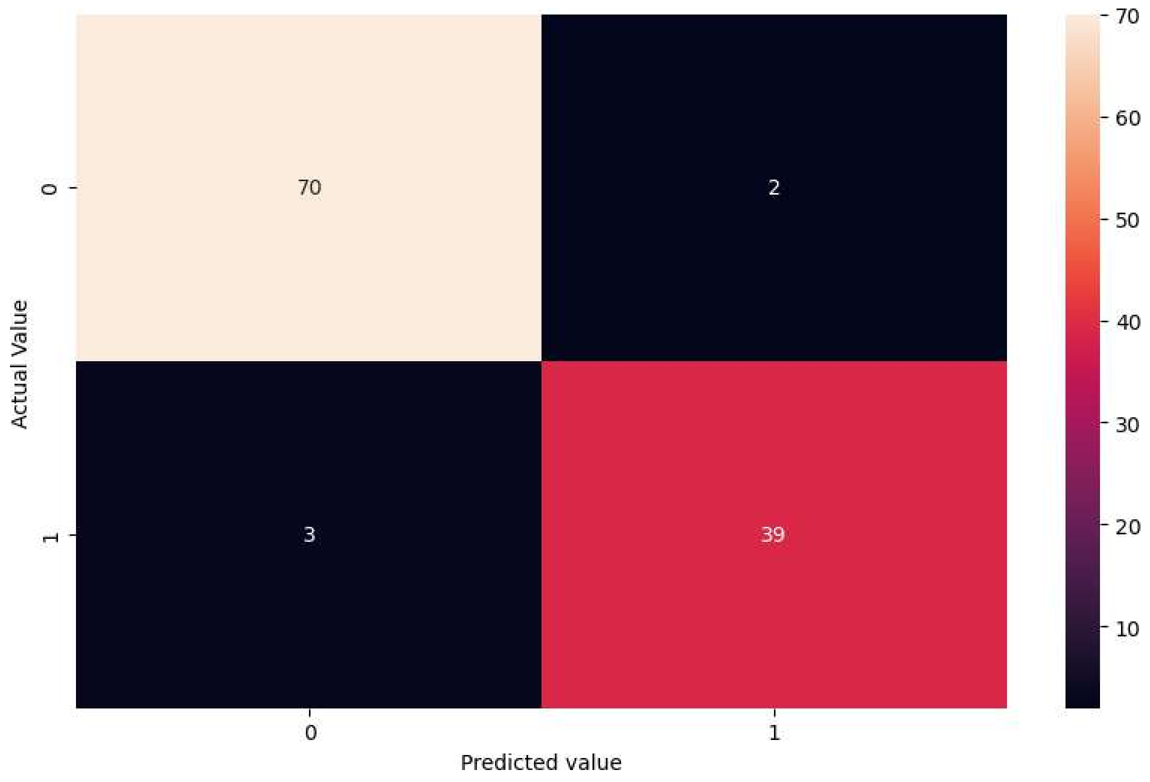


Out[77]:

```
(   Accuracy    Recall  Precision  F1-score
 0  0.993407  0.982353        1.0  0.991098,)
```

In [78]:

```
# Test Performance

breast_test_logreg = model_performance_classification(logreg, x_test_scaler, y_test)
breast_test_logreg
```



Out[78]:

```
(   Accuracy     Recall  Precision  F1-score
 0   0.95614  0.928571     0.95122  0.939759,)
```

Type *Markdown* and LaTeX: α^2

Decision Tree Classifier

In [79]:

```
# To create an instant of the decision tree classifier

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

# To train the model

clf.fit(x_train, y_train)
```

Out[79]:

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

In [80]:

```
# To create an instant of the decision tree classifier

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

# To train the model

clf.fit(x_train, y_train)
```

Out[80]:

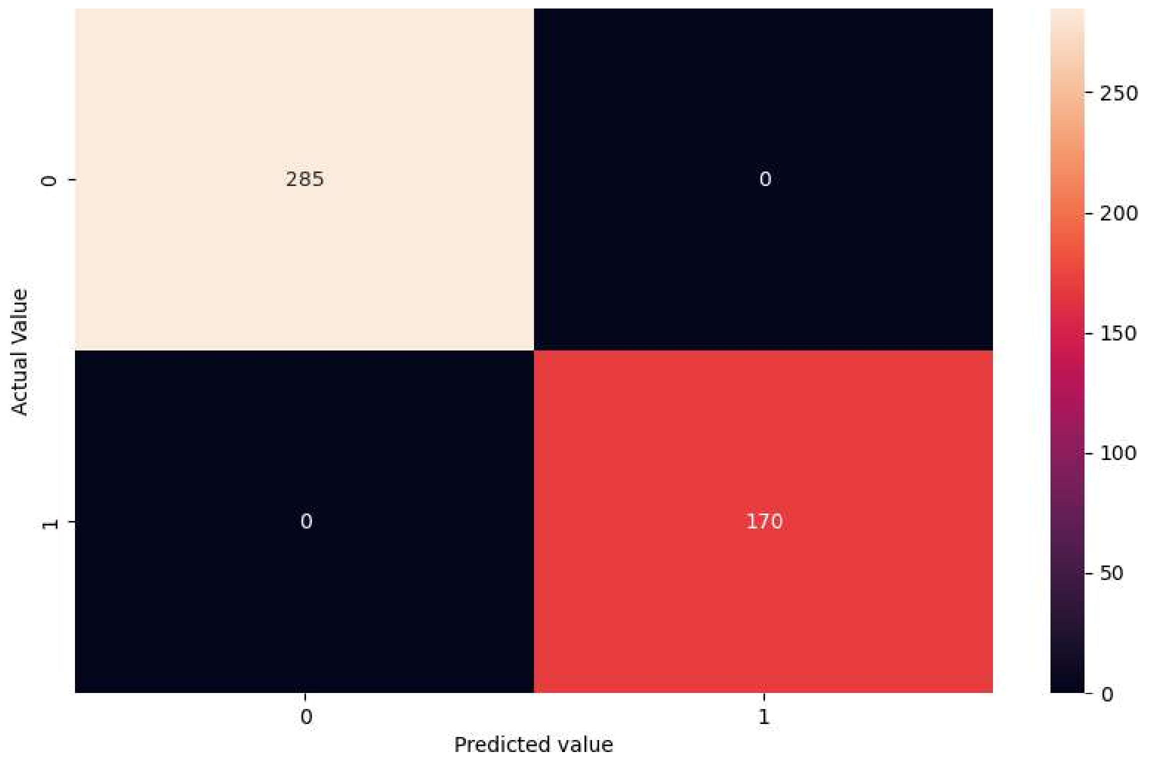
```
▾                DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In [81]:

```

# function to compute different metrics to check performance of a classification model

def model_performance_classification(model, predictors, target, threshold = 0.5): """
    Function to compute different metrics to check classification model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    """

    # predicting using the independent variables
    pred_proba = model.predict_proba(predictors)[:, 1]
     # convert probability to class
    pred_class = np.round(pred_proba > threshold)
    acc = accuracy_score(target, pred_class)  # to compute accuracy
    recall = recall_score(target, pred_class)  # to compute recall
    precision = precision_score(target, pred_class)  # to compute precision
    f1 = f1_score(target, pred_class)  # to compute F1 score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {
            "Accuracy": acc,
            "Recall": recall,
            "Precision": precision,
            "F1-score": f1,
        },
        index=[0],),
    conf = confusion_matrix(target, pred_class) plt.figure(figsize=(10,
    6))
    sns.heatmap(conf, fmt="g", annot=True)
    plt.ylabel("Actual Value")
    plt.xlabel("Predicted value")
```

In [82]:

```
# Training performance

breast_train_clf = model_performance_classification(clf, x_train, y_train)
breast_train_clf
```

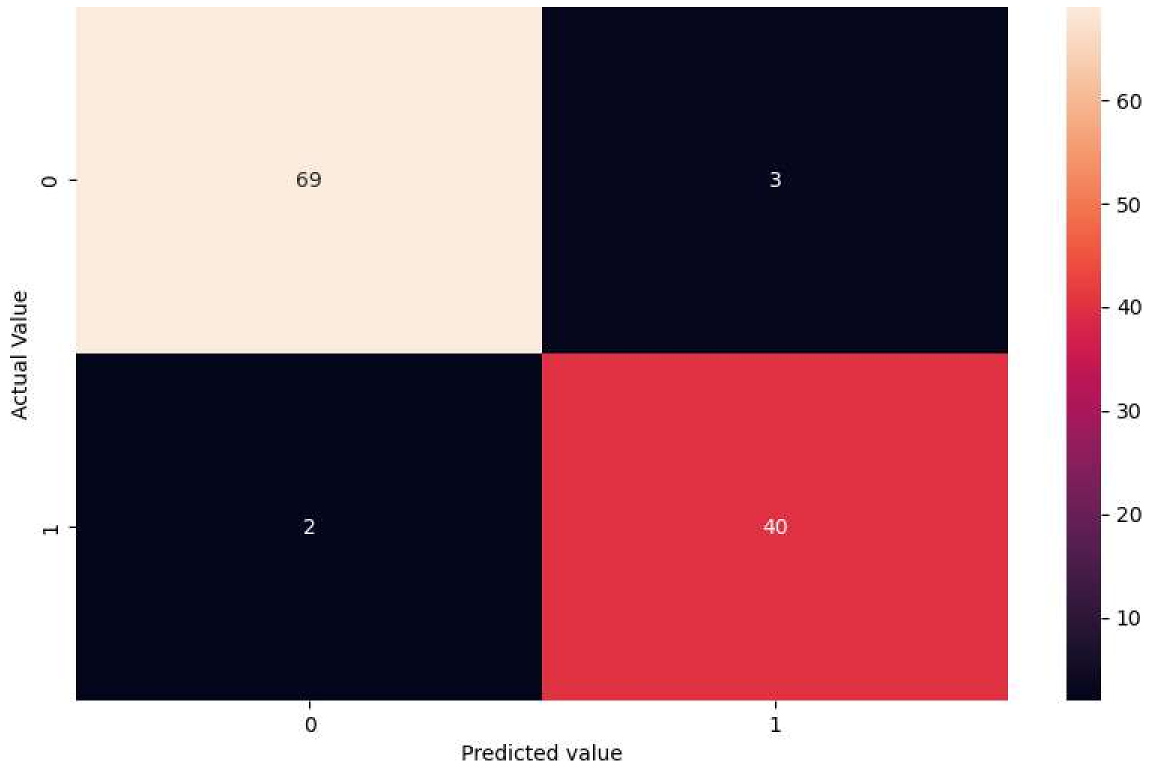


Out[82]:

```
(   Accuracy  Recall  Precision  F1-score
 0       1.0     1.0        1.0       1.0,)
```

In [83]:

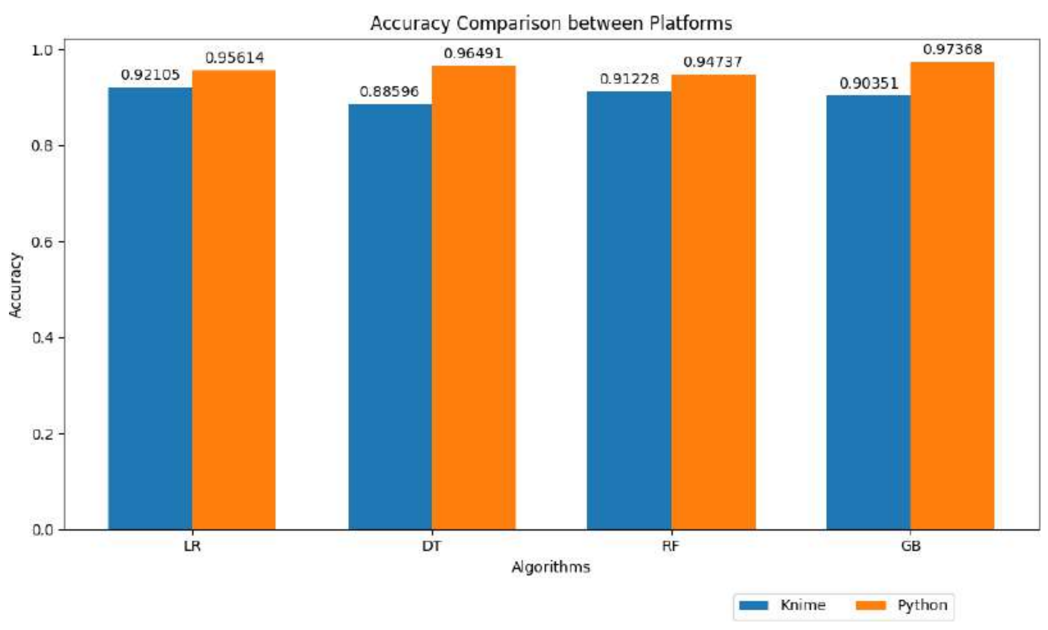
```
# Test performance

breast_test_clf = model_performance_classification(clf, x_test, y_test) breast_test_clf
```



Out[83]:

```
(   Accuracy     Recall   Precision  F1-score
 0   0.95614  0.952381     0.930233  0.941176,)
```

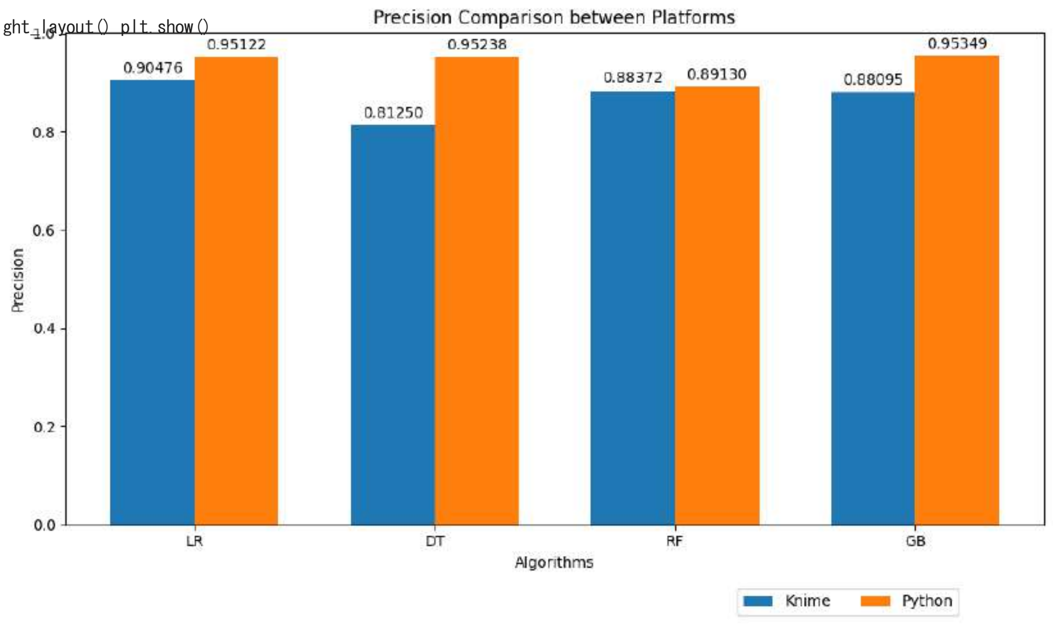
```
import matplotlib.pyplot as plt
import numpy as np

# Data
algorithms = ['LR', 'DT', 'RF', 'GB'] platforms =
['Knime', 'Python']
accuracy_data = np.array([
    [0.92105, 0.95614],
    [0.88596, 0.96491],
    [0.91228, 0.94737],
    [0.90351, 0.97368]
])

# Set up positions for bars
x = np.arange(len(algorithms))
width = 0.35

# Create the grouped bar plot
fig, ax = plt.subplots(figsize=(10, 6))
rects1 = ax.bar(x - width/2, accuracy_data[:, 0], width, label='Knime') rects2
= ax.bar(x + width/2, accuracy_data[:, 1], width, label='Python')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_xlabel('Algorithms')
ax.set_ylabel('Accuracy')
ax.set_title('Accuracy Comparison between Platforms')
ax.set_xticks(x)
ax.set_xticklabels(algorithms)
ax.legend()
plt.legend(loc='lower center', bbox_to_anchor=(0.8, -0.2), ncol=len(platforms))

# Add data labels on top of bars
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate('%.5f' % height,
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points", ha='center',
                    va='bottom')

autolabel(rects1) autolabel(rects2)

fig.tight_layout() plt.show()
```



```
import matplotlib.pyplot as plt
import numpy as np

# Data
algorithms = ['LR', 'DT', 'RF', 'GB'] platforms =
['Knime', 'Python']
recall_data = np.array([ [0.88372,
    0.92857],
    [0.90698, 0.95238],
    [0.88372, 0.97619],
    [0.86047, 0.97619]
])

# Set up positions for bars
x = np.arange(len(algorithms))
width = 0.35

# Create the grouped bar plot
fig, ax = plt.subplots(figsize=(10, 6))
rects1 = ax.bar(x - width/2, recall_data[:, 0], width, label='Knime') rects2 =
ax.bar(x + width/2, recall_data[:, 1], width, label='Python')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_xlabel('Algorithms')
ax.set_ylabel('Recall')
ax.set_title('Recall Comparison between Platforms')
ax.set_xticks(x)
ax.set_xticklabels(algorithms)
ax.legend()
plt.legend(loc='lower center', bbox_to_anchor=(0.8, -0.2), ncol=len(platforms))

# Add data labels on top of bars
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate('%.5f' % height,
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points", ha='center',
                    va='bottom')

autolabel(rects1) autolabel(rects2)

fig.tight_layout() plt.show()
```

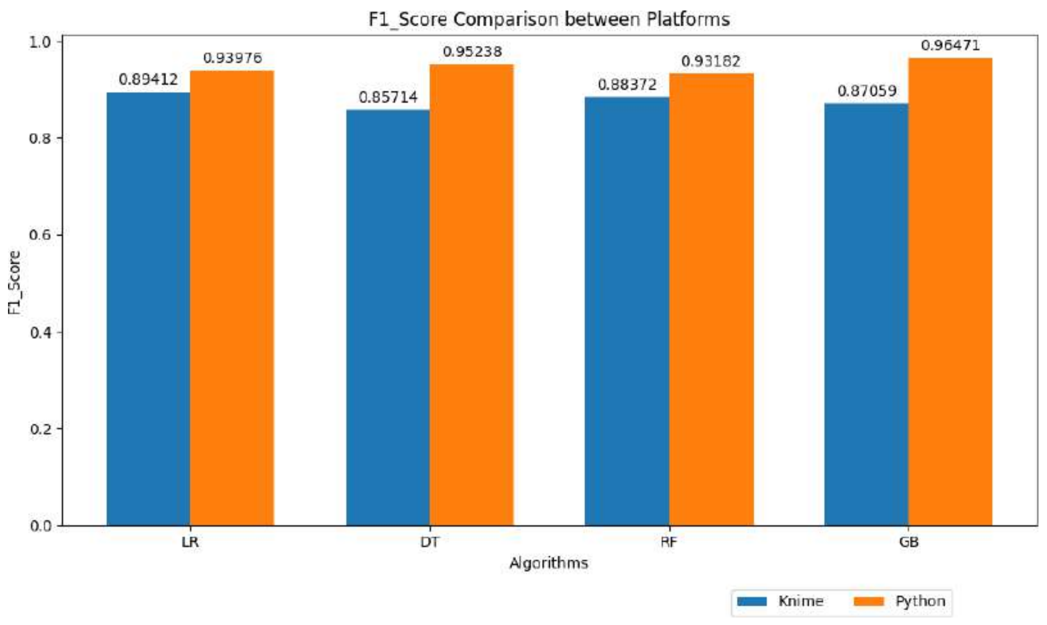
Recall Comparison between Platforms

```
# Data
algorithms = ['LR', 'DT', 'RF', 'GB'] platforms =
['Knime', 'Python']
 precision_data = np.array([
     [0.90476, 0.95122],
     [0.81250, 0.95238],
     [0.88372, 0.89130],
     [0.88095, 0.95349]
])

# Set up positions for bars
x = np.arange(len(algorithms))
width = 0.35

# Create the grouped bar plot
fig, ax = plt.subplots(figsize=(10, 6))
rects1 = ax.bar(x - width/2, precision_data[:, 0], width, label='Knime') rects2
= ax.bar(x + width/2, precision_data[:, 1], width, label='Python')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_xlabel('Algorithms')
ax.set_ylabel('Precision')
ax.set_title('Precision Comparison between Platforms')
ax.set_xticks(x)
ax.set_xticklabels(algorithms)
ax.legend()
plt.legend(loc='lower center', bbox_to_anchor=(0.8, -0.2), ncol=len(platforms))

# Add data labels on top of bars
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate('%.5f' % height,
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points", ha='center',
                    va='bottom')

autolabel(rects1) autolabel(rects2)

fig.tight_layout() plt.show()
```



```
# Data
algorithms = ['LR', 'DT', 'RF', 'GB']
```

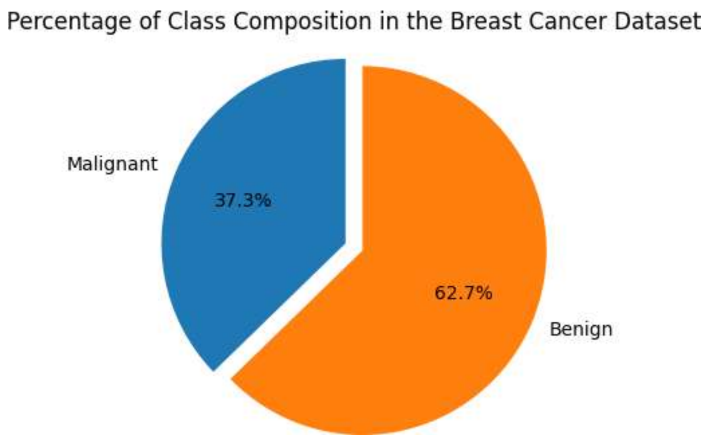
```
platforms = ['Knime', 'Python']
F1_Score_data = np.array([
    [0.89412, 0.93976],
    [0.85714, 0.95238],
    [0.88372, 0.93182],
    [0.87059, 0.96471]
])

# Set up positions for bars
x = np.arange(len(algorithms))
width = 0.35

# Create the grouped bar plot
fig, ax = plt.subplots(figsize=(10, 6))
rects1 = ax.bar(x - width/2, F1_Score_data[:, 0], width, label='Knime') rects2
= ax.bar(x + width/2, F1_Score_data[:, 1], width, label='Python')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_xlabel('Algorithms')
ax.set_ylabel('F1_Score')
ax.set_title('F1_Score Comparison between Platforms')
ax.set_xticks(x)
ax.set_xticklabels(algorithms)
ax.legend()
plt.legend(loc='lower center', bbox_to_anchor=(0.8, -0.2), ncol=len(platforms))

# Add data labels on top of bars
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate('%.5f' % height,
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points", ha='center',
                    va='bottom')

autolabel(rects1) autolabel(rects2)

fig.tight_layout() plt.show()
```



```
import matplotlib.pyplot as plt #

Class label composition
```

```
    labels = ['Malignant', 'Benign']
    sizes = [212, 357]  # Replace with the actual counts
    colors = ['#ff9999', '#66b3ff']  # Colors for the slices
```

```
explode = (0.1, 0)  # Explode the 1st slice (Malignant)

# Create a pie chart
plt.figure(figsize=(4, 4))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90, explode=explode)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.title('Percentage of Class Composition in the Breast Cancer Dataset') #

Save the figure with higher DPI for better quality
plt.savefig('pie_chart.png', dpi=300)  # Adjust the DPI value as needed
plt.show()
```

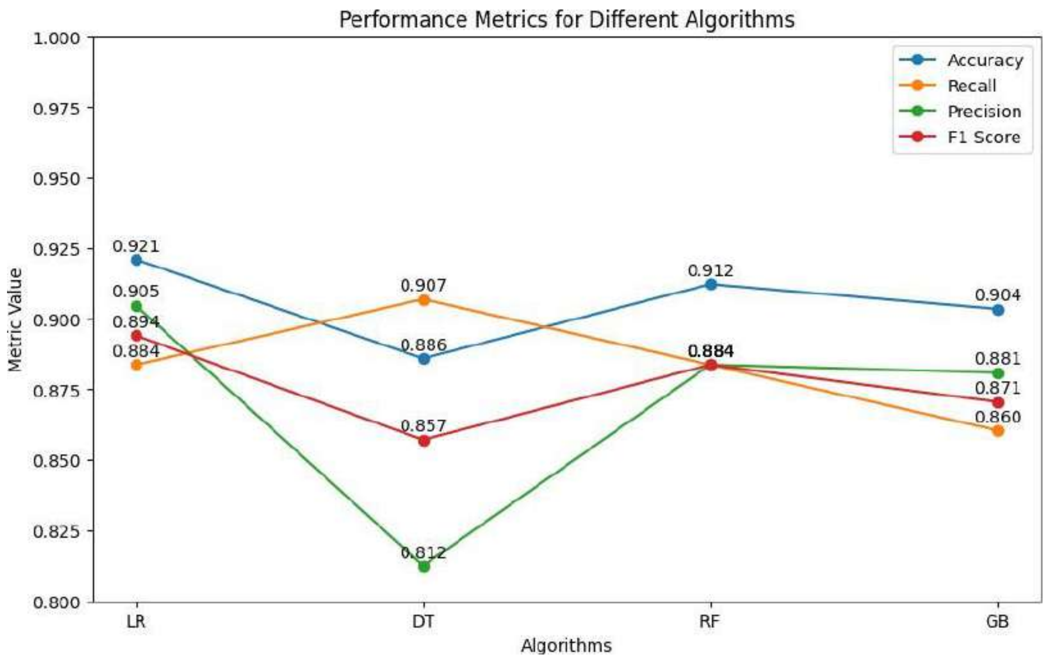


```
import numpy as np
from scipy import stats

# Create arrays with the accuracy values for each algorithm on Knime and Python
knime_accuracy = np.array([0.945, 0.927, 0.939, 0.935])
python_accuracy = np.array([0.956, 0.965, 0.947, 0.974])

# Perform paired t-test
t_statistic, p_value = stats.ttest_rel(knime_accuracy, python_accuracy)

# Print results
print("Paired t-test results:")
print("T-statistic:", t_statistic)
print("P-value:", p_value)

# Check if the difference is statistically significant alpha =
0.05
if p_value < alpha:
    print("The observed differences are statistically significant.") else:
    print("The observed differences are not statistically significant.")
```

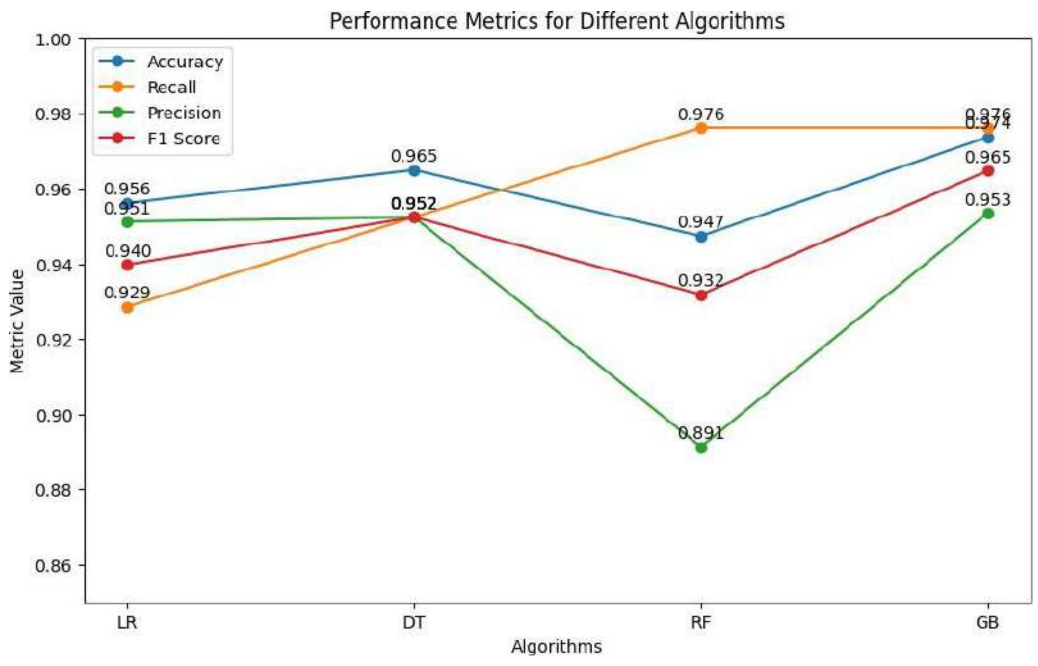
```
Paired t-test results:
T-statistic: -2.85835840404199
P-value: 0.06466393746357252
The observed differences are not statistically significant.
```

```
import matplotlib.pyplot as plt #

Data
algorithms = ['LR', 'DT', 'RF', 'GB']
metrics = ['Accuracy', 'Recall', 'Precision', 'F1 Score']
values = [
    [0.92105, 0.88372, 0.90476, 0.89412],
    [0.88596, 0.90698, 0.81250, 0.85714],
    [0.91228, 0.88372, 0.88372, 0.88372],
    [0.90351, 0.86047, 0.88095, 0.87059]
]

# Plotting
plt.figure(figsize=(10, 6))
```

```
for i, metric in enumerate(metrics):
    plt.plot(algorithms, [row[i] for row in values], marker='o', label=metric)

# Adding annotations
for i, metric in enumerate(metrics):
    for j, alg in enumerate(algorithms):
        plt.annotate(f'{values[j][i]:.3f}', xy=(alg, values[j][i]), textcoords='offset points', xytext=(0,5), ha='center')

plt.title('Performance Metrics for Different Algorithms')
plt.xlabel('Algorithms')
plt.ylabel('Metric Value')
plt.ylim(0.8, 1)  # Set y-axis limit between 0.7 and 1 for better scaling
plt.legend()
plt.show()
```



```
    import matplotlib.pyplot as plt #

    Data
    algorithms = ['LR', 'DT', 'RF', 'GB']
    metrics = ['Accuracy', 'Recall', 'Precision', 'F1 Score']
    values = [
        [0.95614, 0.92857, 0.95122, 0.93976],
        [0.96491, 0.95238, 0.95238, 0.95238],
        [0.94737, 0.97619, 0.89130, 0.93182],
        [0.97368, 0.97619, 0.95349, 0.96471]
    ]

    # Plotting
    plt.figure(figsize=(10, 6))

    for i, metric in enumerate(metrics):
        plt.plot(algorithms, [row[i] for row in values], marker='o', label=metric)

    # Adding annotations
    for i, metric in enumerate(metrics):
        for j, alg in enumerate(algorithms):
            plt.annotate(f'{values[j][i]:.3f}', xy=(alg, values[j][i]), textcoords='offset points', xytext=(0,5), ha='center')

    plt.title('Performance Metrics for Different Algorithms')
    plt.xlabel('Algorithms')
    plt.ylabel('Metric Value')
    plt.ylim(0.85, 1)  # Set y-axis limit between 0.7 and 1 for better scaling plt.legend()
    plt.show()
```


Performance Metrics for Different Algorithms
Accuracy
Recall
Precision
F1 Score
0.956
0.951
0.940
0.929
0.965
0.952
0.976
0.947
0.932
0.891
0.976
0.974
0.965
0.953
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
Metric Value
LR
DT
RF
GB
Algorithms