*Leen Shareef Saleh*

*HTU | Amman - Jordan*

Implementing and developing multiple models For Predicting Liver Cirrhosis Stage.

Machine Learning

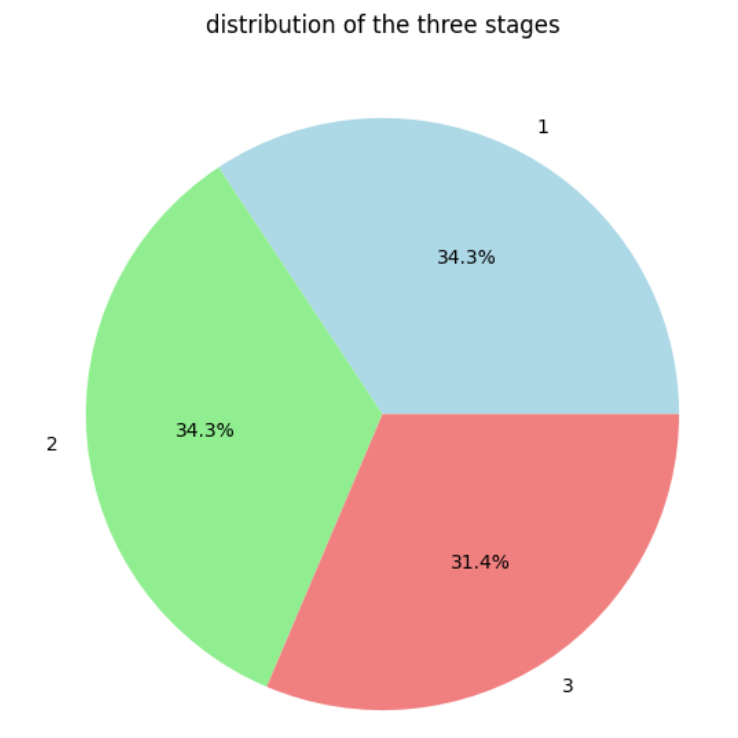
## Provide a well-defined problem statement with a clear set of requirements

*Define a clear problem to solve using machine learning. Discuss all the details of the problem, such as the number of classes in a classification problem and each category (classification)), and any other important detail.*

**Problem Statement: Predicting Liver cirrhosis Disease Stage**

**Objective**

The goal is to develop a robust, good machine learning model that can accurately predict the stage of liver cirrhosis in patients. The model will get the pattern of a various health measurements collected from patients to make these predictions. This project aims to help healthcare professionals in assessing the advancement of the disease. This problem is a multiclass classification problem this means that the model will predict one of the three outcomes (stage 1 – early stage, stage 2 – intermediate stage, stage 3 – advanced stage).



**Background**

Cirrhosis is a severe liver condition resulting from prolonged damage, leading to extensive scarring and compromised liver function. Common causes include chronic hepatitis, prolonged alcohol consumption, and other liver diseases.[[1]](#footnote-1) Early detection and accurate staging of cirrhosis are crucial for effective disease management and treatment planning.

When the liver is damaged, whether due to excessive alcohol consumption or other causes such as infection, it attempts to repair itself. This repair process results in the formation of scar tissue. As cirrhosis progresses, the accumulation of scar tissue increases, making it difficult for the liver to perform its functions. Advanced cirrhosis is a life-threatening condition.[[2]](#footnote-2)

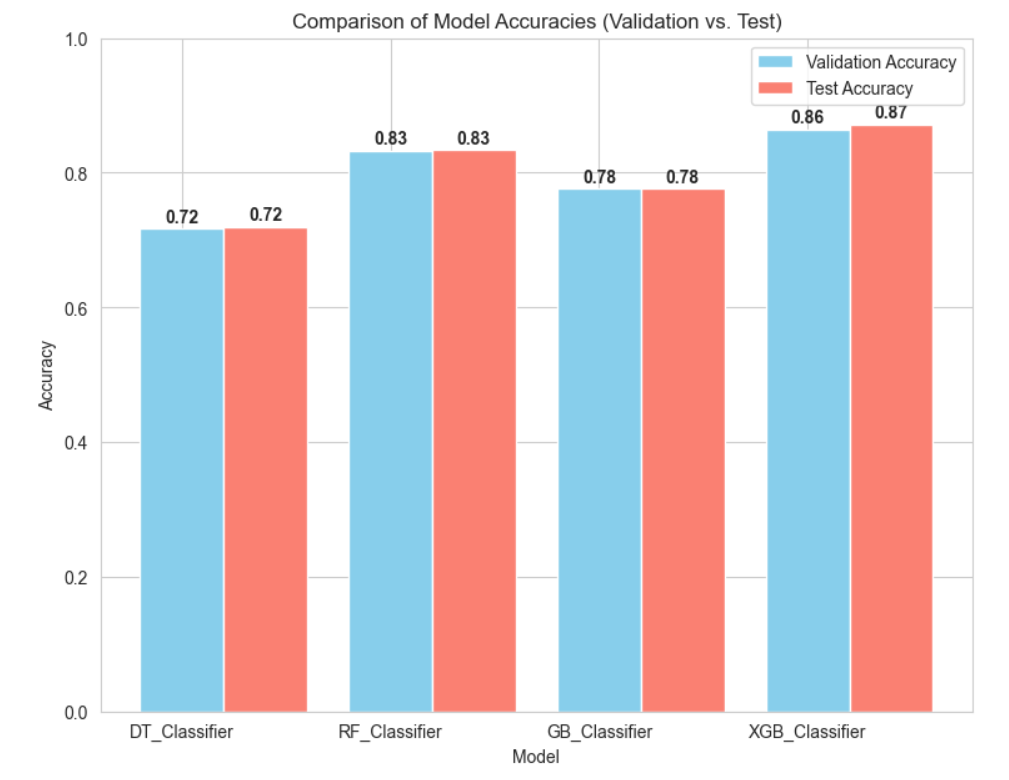
The damage to the liver caused by cirrhosis is usually irreversible. However, if cirrhosis is detected early and the underlying cause is addressed, further damage can be prevented. In rare instances, the condition may even be reversed.

## Find the best ML algorithms used to train models for such a problem.

Decision Tree, Random Forest, Gradient Boosting, and XGBOOST where used.

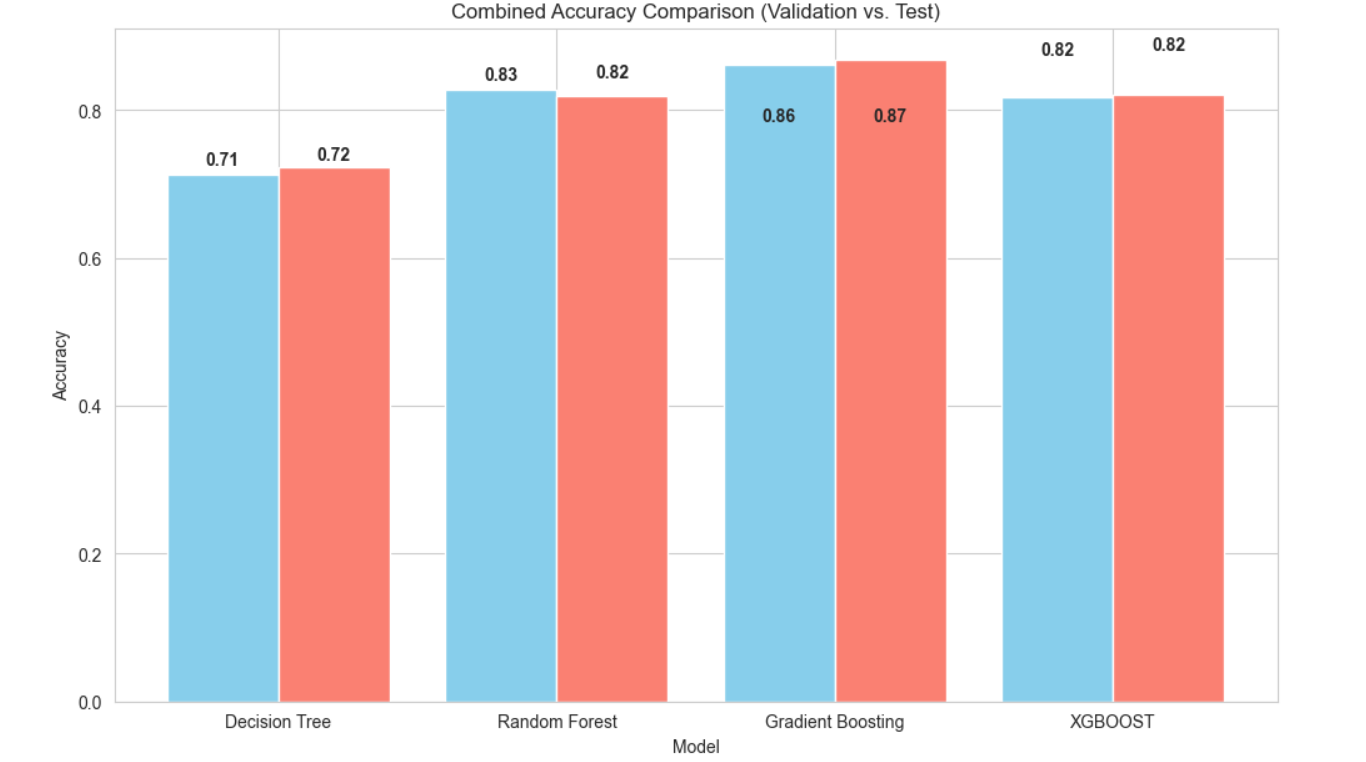
### Baseline Models (Best)

Best model was XGB\_classifier with an accuracy of 0.87



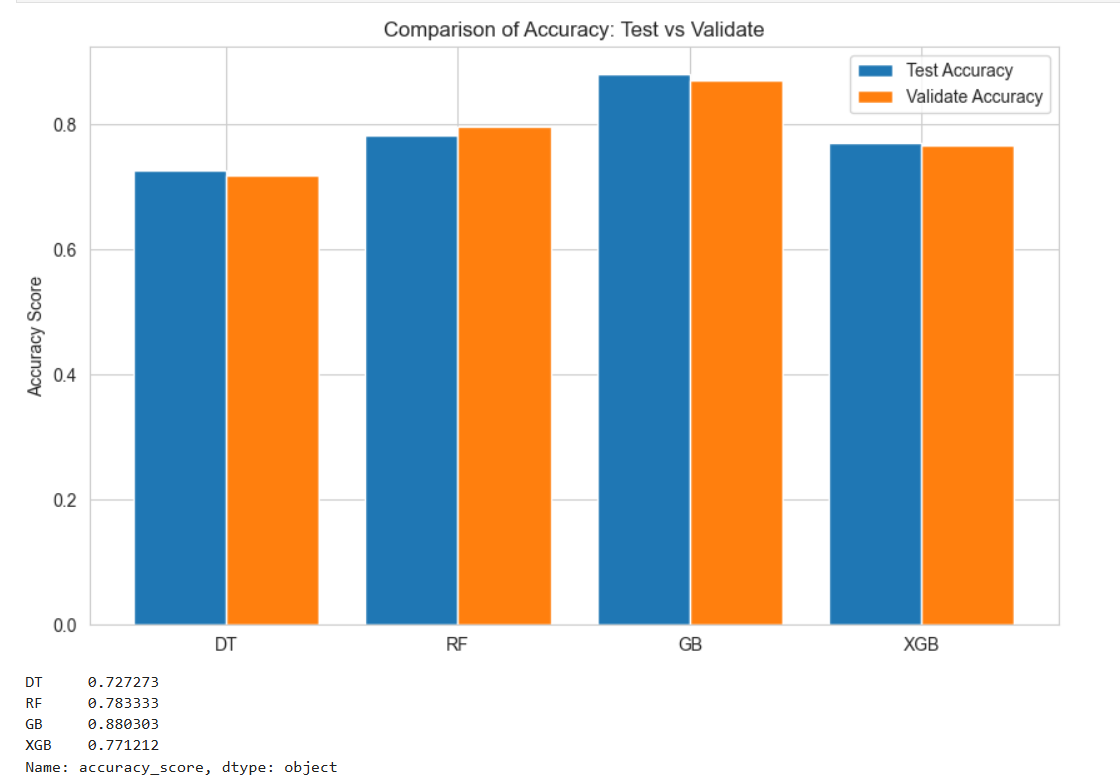
### Models with Tuned Parameters (Best)

Best model was Gradient Boosting with accuracy of 0.87



## Models with Grid Search

Best model was Gradient Boosting with accuracy of 0.88



Compare the ML algorithms found and make a summary of their advantages/disadvantages. (Use this to justify your choice of the algorithms you are going to use. Use a table for comparison. )

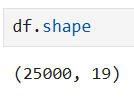
|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Advantage** | **Disadvantages** | **Justify your choice** |
| **Decision Trees [[3]](#footnote-3)** | Easy to explain and understand  Trees can be displayed graphically and are easily interpreted even by a non-expert.  trees are more closely in mirroring human decision-making | trees more closely mirror human decision-making.  They can overfit, create complex trees that do not generalize well.  Unstable: small changes in data can result in different trees. | Decision Trees are helpful when you want to explain the logic behind predictions, especially for non-technical audiences. They are quick to build and easy to visualize, which makes them good for small datasets or exploratory work. However, they might not perform well on larger datasets or complex problems, so they work best when simplicity and explainability are more important than accuracy. |
| **Random Forest[[4]](#footnote-4)** | Reduces overfitting  Diversity: Each tree in a random forest uses different features, making each tree unique  Works well with large datasets  Immune to high dimensions: By not using all features, the feature space is smaller.  Parallelization: Trees are built independently, allowing full CPU use.  Train-Test split: Random forests automatically have 30% of data not seen by each tree, so no need for a separate test set.  Stability: Results are stable because they are based on majority voting or averaging. | Computational time is so high  Complex and less interpretable compared to a single decision tree | Random Forests are great when you need a reliable and accurate model that performs well on complex datasets. They work particularly well when your dataset has many features and you want to reduce overfitting. They are also stable because they combine results from many trees. However, they are harder to explain than Decision Trees and need more time and resources to train, so they are better suited for larger or critical projects. |
| **Gradient Boosting[[5]](#footnote-5)** | main advantages of Gradient Boosting is its predictive power. | Sensitivity to outliers  Risk of overfitting  computationally expensive and slow to train[[6]](#footnote-6) | Gradient Boosting is ideal when accuracy is your top priority, especially for difficult problems where small improvements can make a big difference. It is also useful for datasets with imbalanced classes because it can focus on the harder-to-classify examples. However, it requires significant computational power and time to train, so it’s better for projects where you have sufficient resources and don't need instant results. |
| **XGBOOST[[7]](#footnote-7)** | XGBOOST often produces excellent results  Designed for efficient training, XGBoost works well with large datasets | Can overfit, especially with small datasets  Many settings need to be tuned for best performance, which can be time-consuming and require expertise. | XGBoost is a powerful algorithm that combines speed and accuracy, making it perfect for large datasets and real-world applications where efficiency matters. It is especially good for competitions or business cases where the smallest accuracy improvements are important. However, it requires careful tuning, which means you need time, knowledge, and tools to find the best settings. It works best when you have a large dataset and enough time to optimize the model for the best performance. |

**PART-4: Model Development**

Identify and obtain a relevant dataset for training models to address the problem at hand. (Provide a detailed description of the dataset, including its source, method of collection, key attributes, size, and the domain it represents.)

**Dataset Description**

The dataset is derived from a Mayo Clinic study on primary biliary cirrhosis (PBC) conducted from 1974 to 1984. It contains 25,000 entries with 19 columns, capturing various attributes related to patients' health.

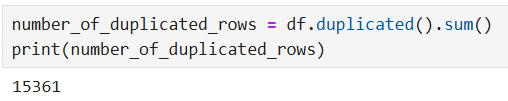


With no null values and a categorical and numerical columns

A screenshot of a computer

Description automatically generated

But with one major problem that there are 15361 rows duplicated



**Numerical Attributes**

*N\_Days:* Number of days of follow-up. (length of time patients were observed n the study until it end by death, trans. or end of study)

*Age:* Age in years. [mean of age by days --> 18495 / 365 = 50 years] / [min of age 9598 approximately = 26.3 years and max = 78.5 years]

*Bilirubin:* Serum bilirubin level (mg/dl).

*Cholesterol:* Serum cholesterol level (mg/dl).

*Albumin:* Serum albumin level (g/dl).

*Copper:* Urine copper level (ug/day).

*Alk\_Phos*: Alkaline phosphatase level (U/liter).

*SGOT:* Aspartate aminotransferase level (U/ml).

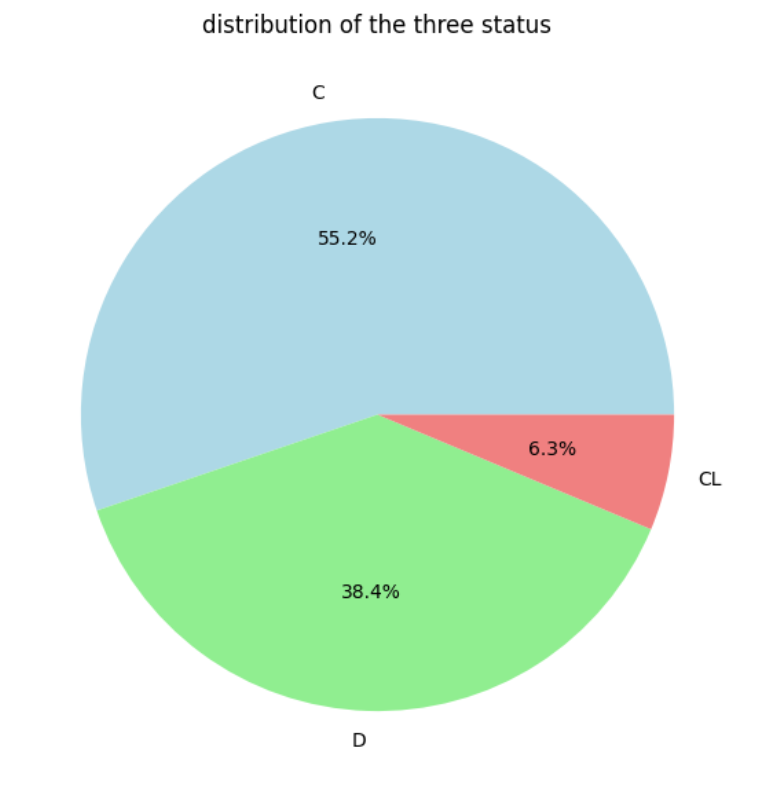
*Tryglicerides:* Serum triglycerides level (mg/dl).

*Platelets:* Platelet count (per cubic ml/1000).

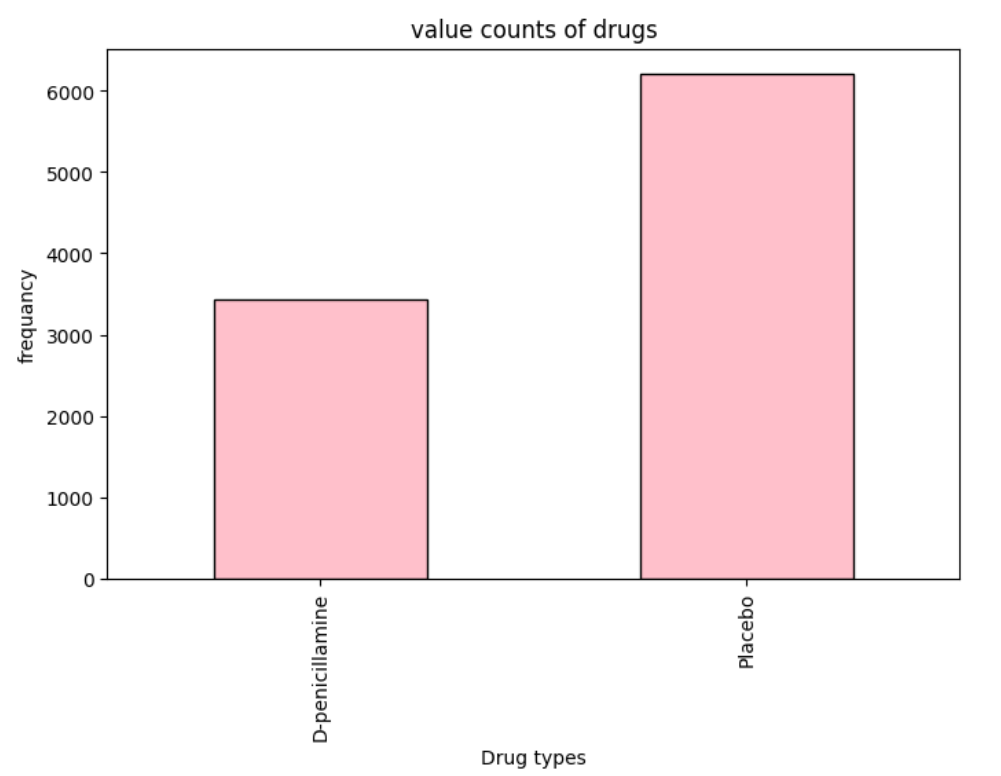
*Prothrombin:* Prothrombin time (seconds).

**Categorical Attributes**

*Status:* Patient's status, C 🡪 censored / CL 🡪 censored due to liver transplantation / D 🡪 death



*Drug*: Type of drug treatment. (D-penicillamine or placebo)



*Sex*: Male or Female.

*Ascites*: Presence of ascites (Yes or No).

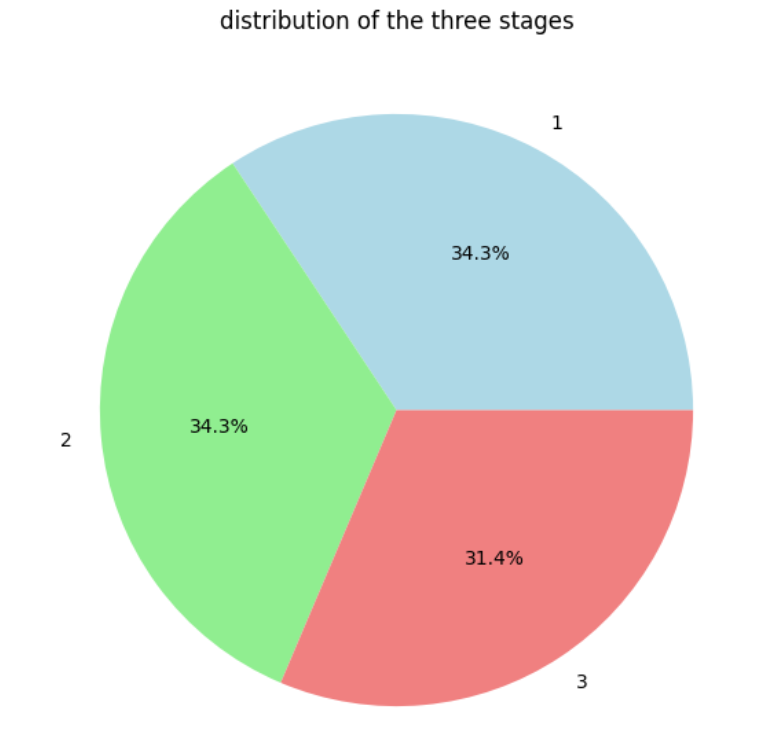
*Hepatomegaly*: Presence of hepatomegaly (Yes or No).

*Spiders*: Presence of spider nevi (Yes or No).

*Edema*: Presence of edema presence of edema N (no edema and no diuretic therapy for edema), S (edema present without diuretics, or edema resolved by diuretics), or Y (edema despite diuretic therapy)

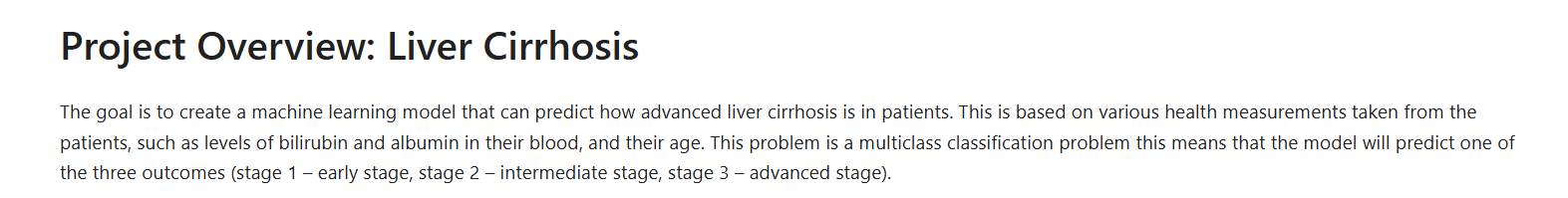
**Target Attribute**

Stage: Stage of the disease (1🡪 early stage, 2🡪 intermediate stage, 3🡪 advanced stage).



## Prepare the dataset by performing the necessary preprocessing and splitting it into training, validation, and test subsets.

Starting the code with a project overview description



Reading the dataset to start the EDA

A white rectangular sign with red and blue text

Description automatically generated

The head of the data

A screenshot of a computer

Description automatically generated

Columns in the dataset

A screenshot of a computer

Description automatically generated

Getting to know the three stages in the target column

A close-up of a computer code

Description automatically generated

The dataset has no null-values

A list of medical records

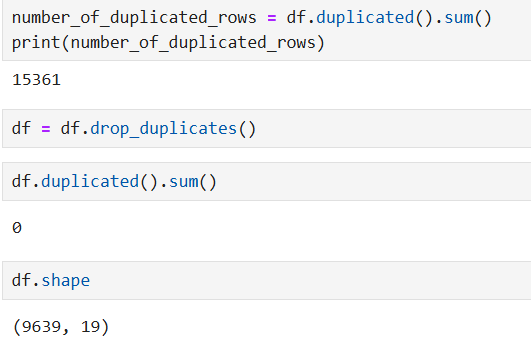
Description automatically generated with medium confidence

Getting the statistical measures

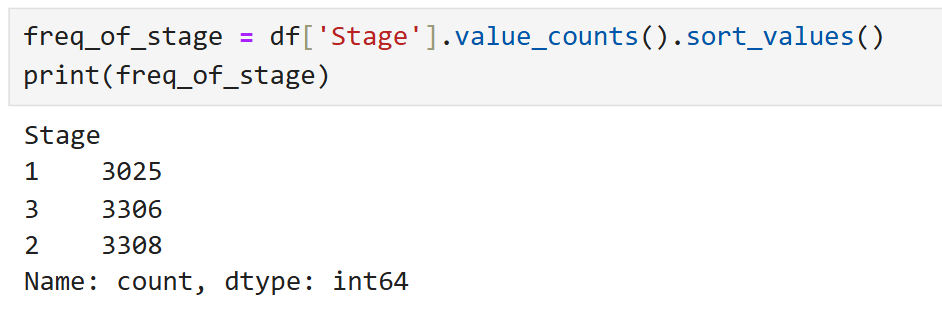
A screenshot of a computer screen

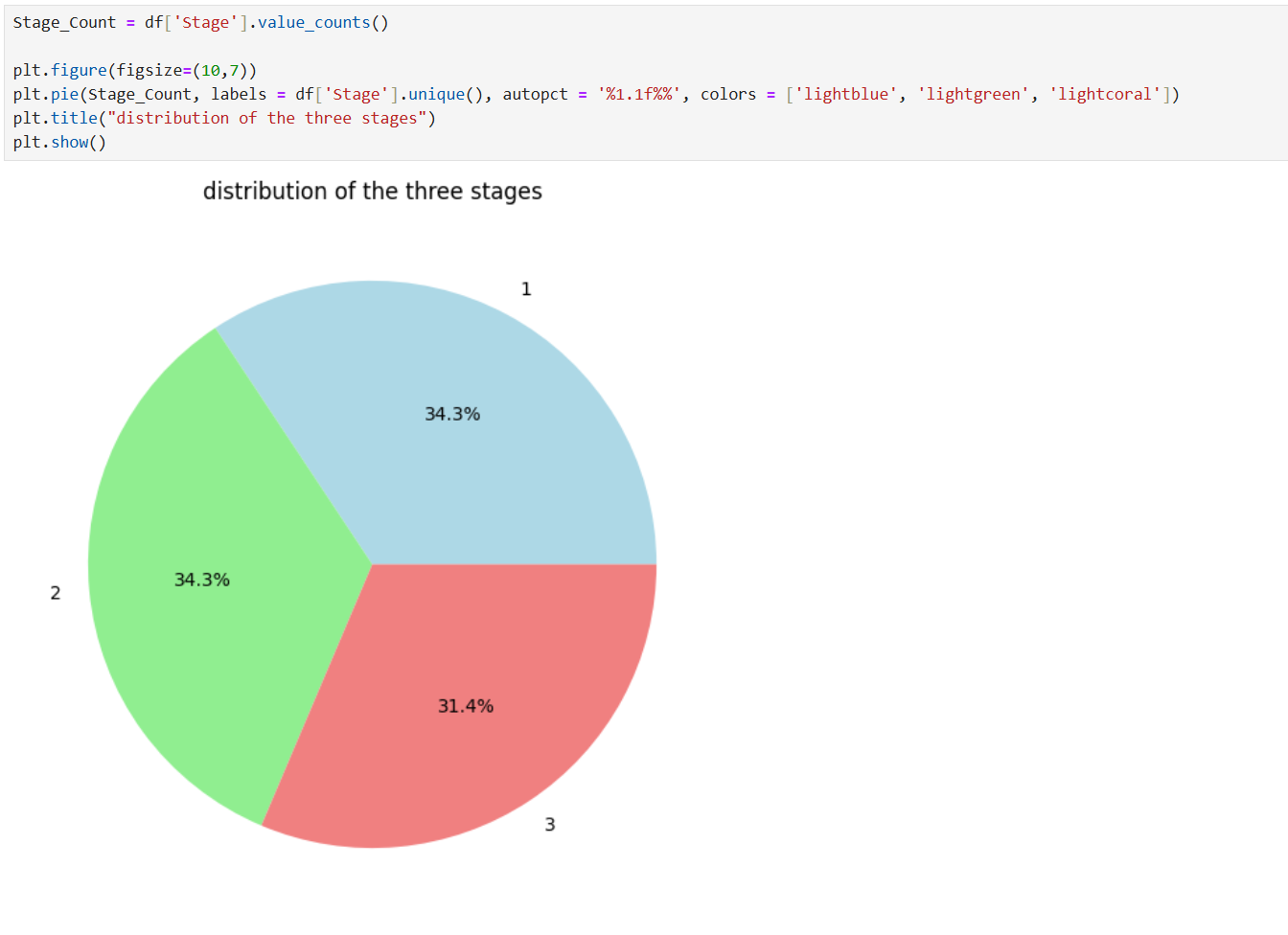
Description automatically generated

Checking duplicated rows and dropping them so the dataset has the shape of (9639, 19)

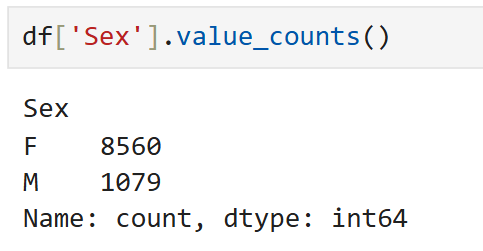


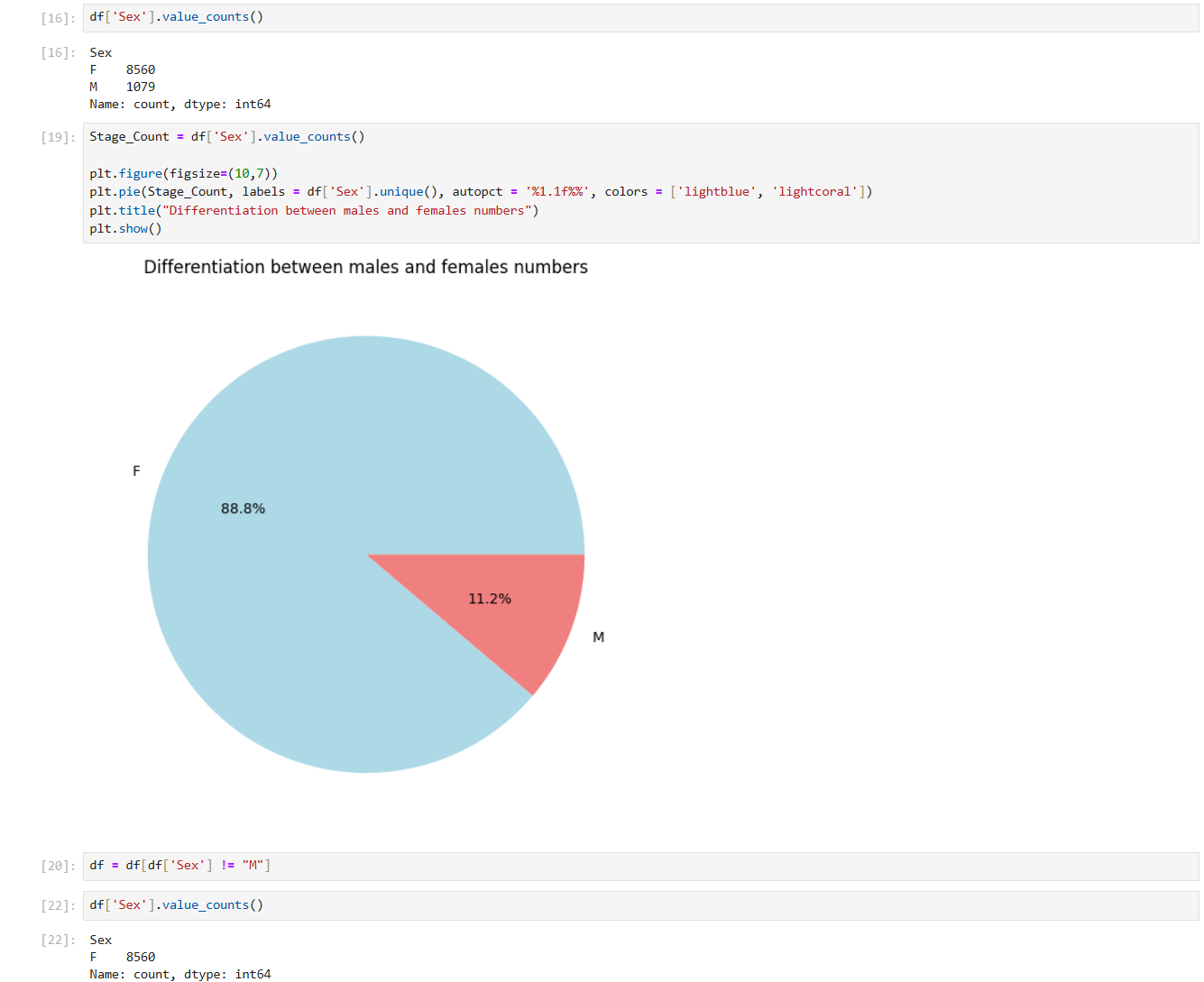
The frequency of the three stages seems to be somehow close to each other, however stage one is less than the other 2 by 300 (approximately)





Checking the counts of the two genders (this indicates that the data is biased, however I have contacted a friend of mine that she is a doctor and she told me that gender has to do nothing with the liver cirrhosis disease)

  
dropping all rows with M in Sex column



Getting the relative frequency of the status

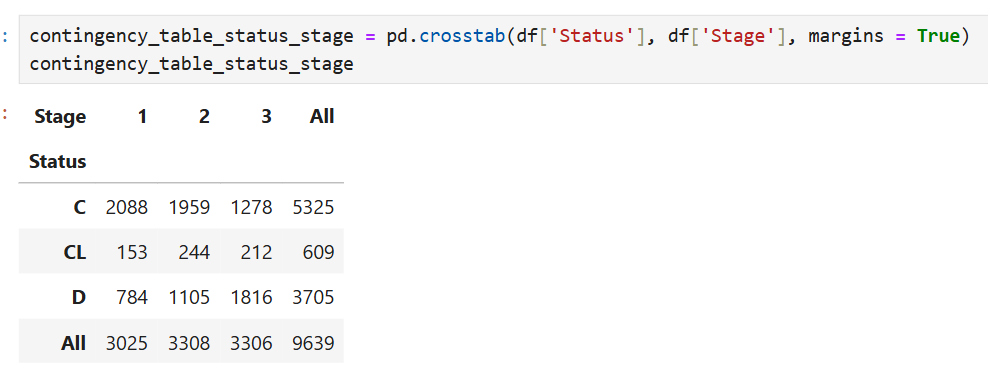
A screenshot of a computer code

Description automatically generated

A pie chart with numbers and a diagram

Description automatically generated

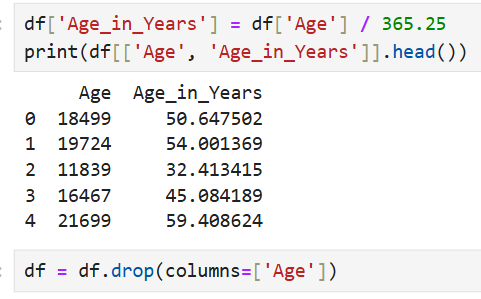
contingency table provides a breakdown of the Status categories (C, CL, D) across different stages (1, 2, 3), status C has the highest frequency 5325, status CL is the least 609. Stage **1** has the most observations (3025), closely followed by Stage **2** (3308) and Stage **3** (3306)



A chart with different colored squares

AI-generated content may be incorrect.

Age column was provided by days, but to make it more practical it was converted to years, and the Age column by days was dropped to not have noises + understating the distribution of the ages.

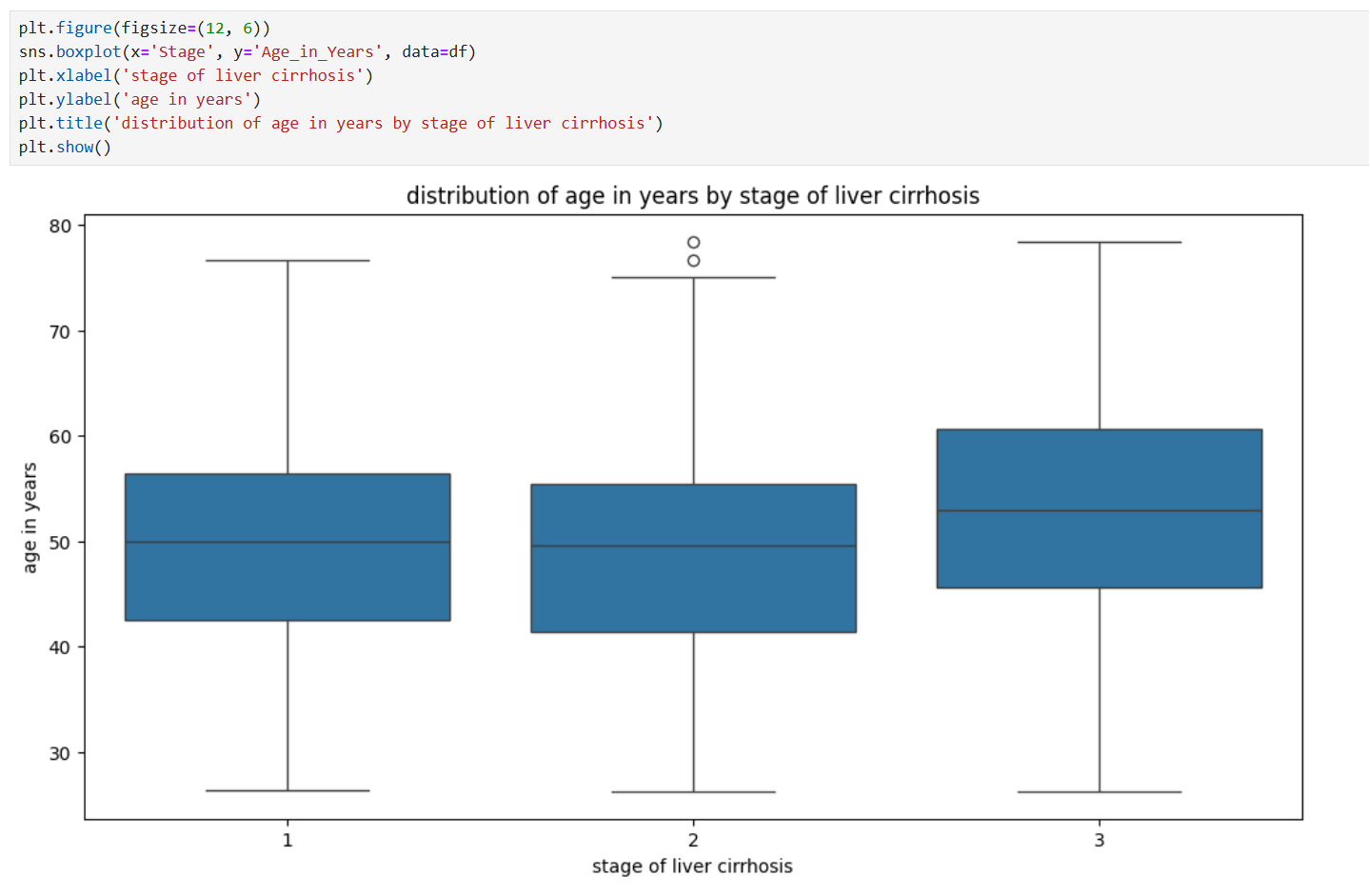


A screenshot of a graph

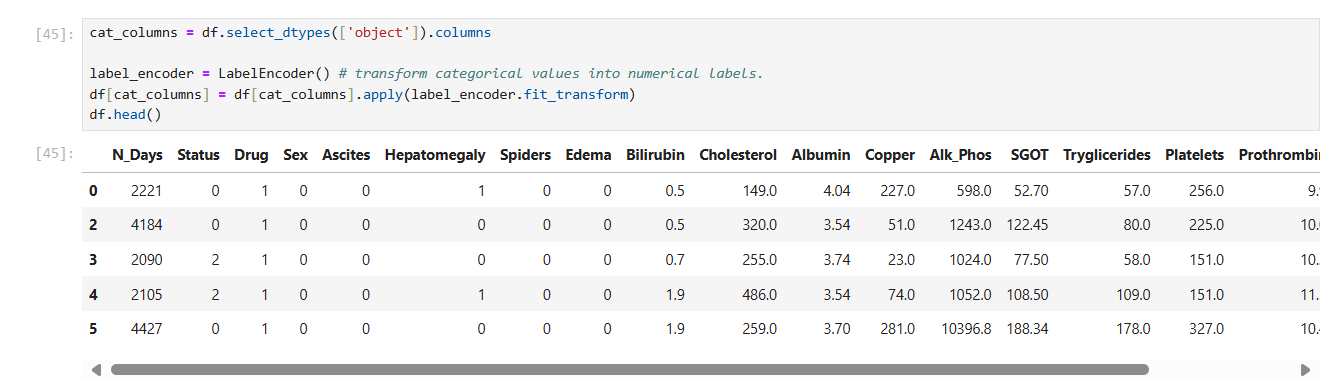
Description automatically generated

Stage 1, the median appears to be approximately 50 years, same as stage 2. And for Stage 3, the median is slightly higher, close to 55 years. (older patients are more common in later stages of liver cirrhosis)

A few outliers are visible in Stage 2, representing older patients whose ages fall above the upper limit.



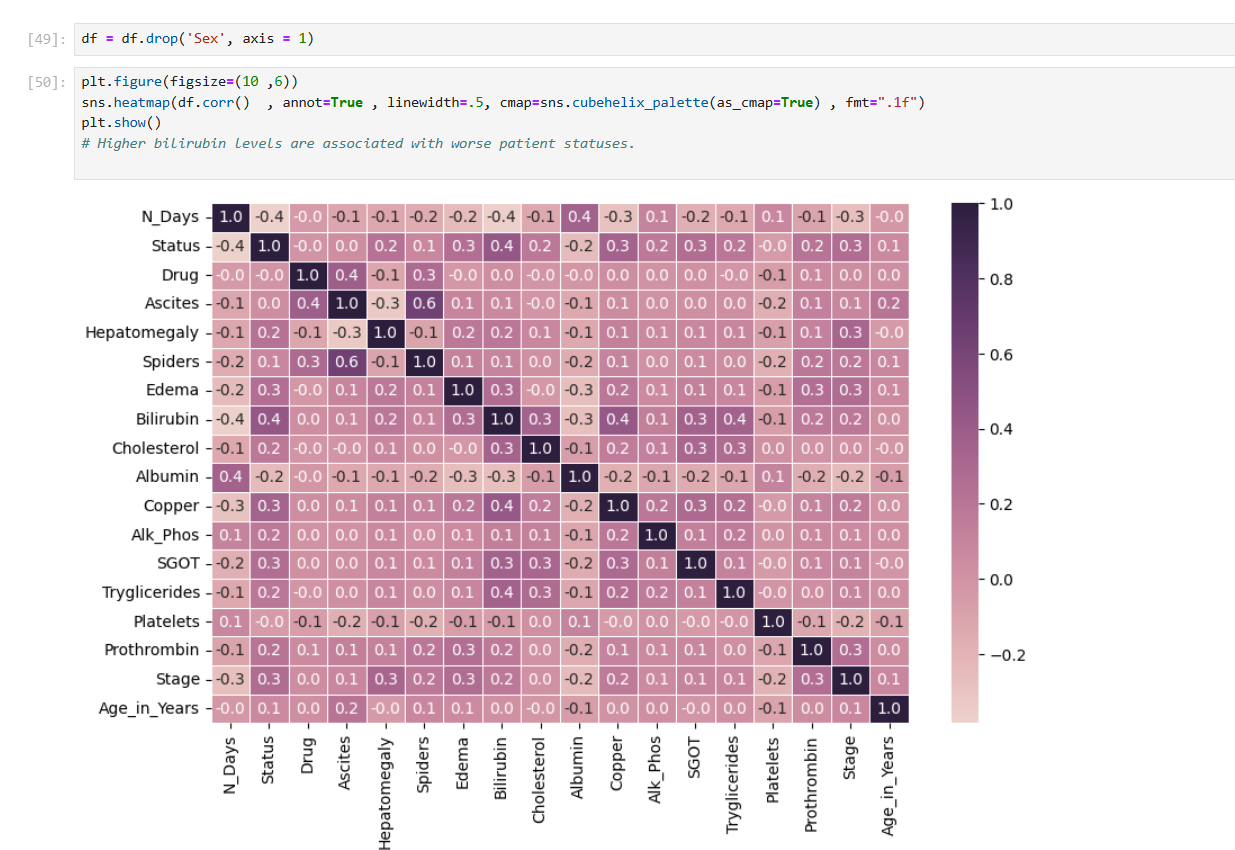
Label encoder



Correlation heatmap (because we have dropped all the rows for male to avoid bias, now the Sex column has no meaning so we will be dropping it to avoid noise)

A screenshot of a computer

Description automatically generated



Preparing the dataset for modeling

A white rectangular sign with red and blue text

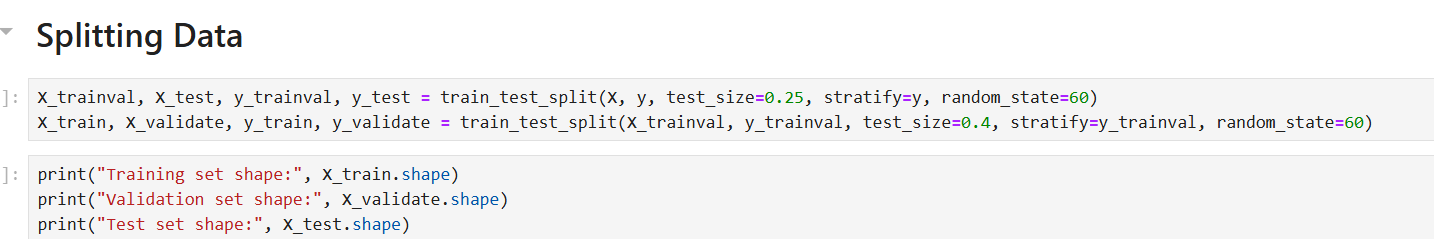
Description automatically generated

Data normalization

A screenshot of a computer

Description automatically generated

Splitting the dataset into train, validate and test



## Train the models designed for making inferences and solving the problem under consideration. Show the steps for training and use screenshots from your code.

### Baseline model

Decision tree



Random forest

A screenshot of a computer code

Description automatically generated

Gradient boosting

A screenshot of a computer program

Description automatically generated

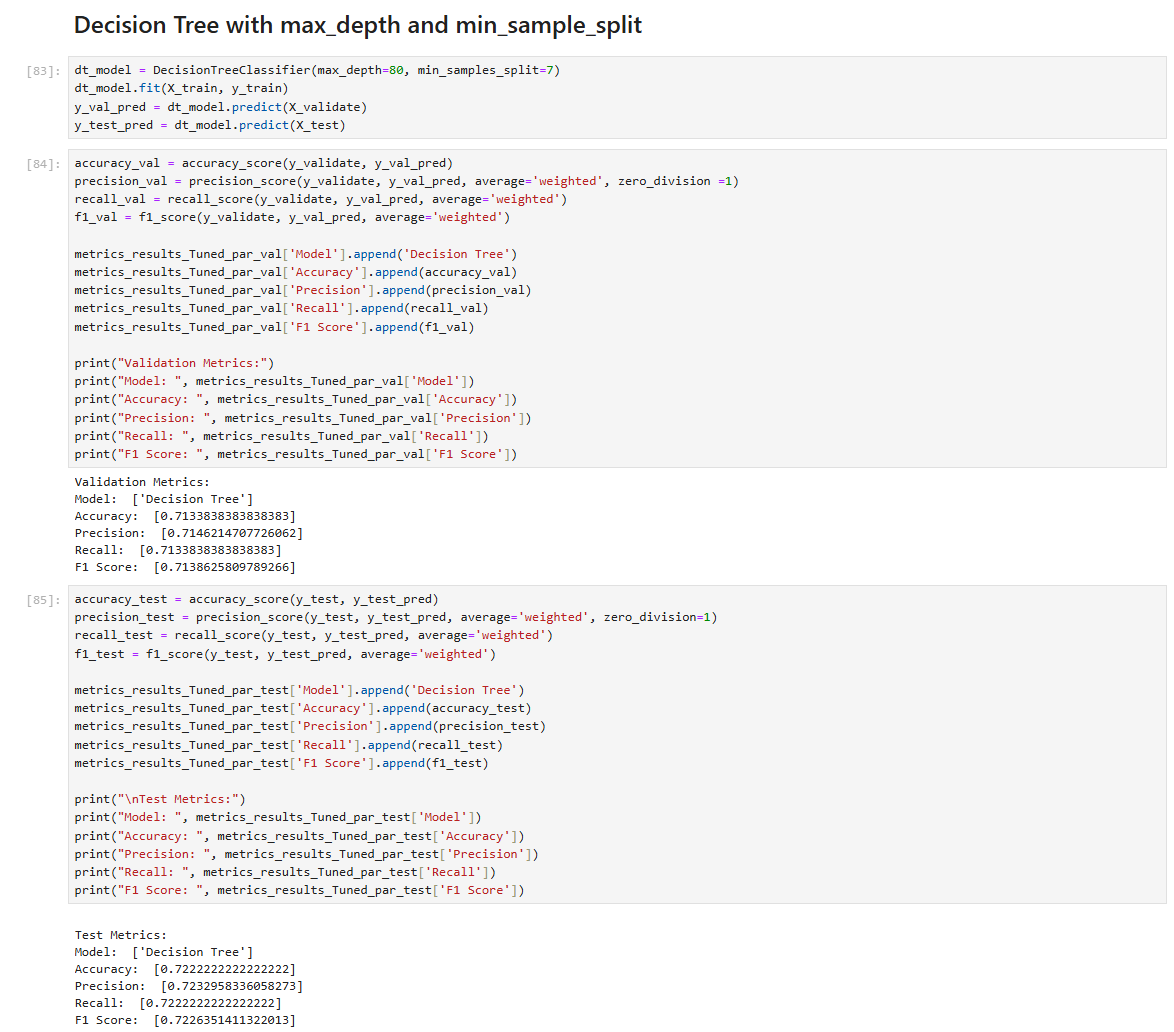
XGBOOST

A screenshot of a computer program

Description automatically generated

### Models with parameters

Decision tree with max depth and min sample split



Random forest

A screenshot of a computer program

Description automatically generated

Gradient boosting

A screenshot of a computer program

Description automatically generated

XGBOOST

A screenshot of a computer program

Description automatically generated

### Models with grid search

Decision tree

A screenshot of a computer program

Description automatically generated

Random forest

A screenshot of a computer program

Description automatically generated

Gradient boosting

A screenshot of a computer program

Description automatically generated

XGBOOST

A screenshot of a computer program

Description automatically generated

## Provide a comprehensive explanation of the training process, including the hyperparameters considered and their values, the algorithms selected, and any other relevant details. Use a table to summarize your selections.

Table: Description of the hyperparameters considered and best value for each hyperparameter or hyperparameter combination.

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Hyper-parameter** | **Description** | **Value** |
| **Decision tree** | **Max\_depth**  **Min\_sample\_split** | The maximum depth of the tree. Controls overfitting.  The minimum number of samples required to split an internal node. | Values used:    Best values:  A black and white text  AI-generated content may be incorrect.  Best parameters found: {'max\_depth': 20, 'min\_samples\_split': 4} |
| **Random Forest** | **n\_estimator**  **max\_depth**  **min\_samples\_split** | The number of trees in the forest. More trees can improve performance.  The maximum depth of the individual trees. Controls complexity.  The minimum number of samples required to split an internal node. | Values used:    Best values:    Best parameters found: {'max\_depth': 10, 'min\_samples\_split': 3, 'n\_estimators': 26} |
| **Gradient Boosting** | **n\_estimators**  **learning\_rate**  **max\_depth** | The number of boosting stages to be run.  How much each tree contributes to the final prediction.  The maximum depth of the individual trees. Controls overfitting. | Values used:    Best values:    Best parameters found: {'learning\_rate': 0.2, 'max\_depth': 15, 'min\_samples\_split': 18, 'n\_estimators': 39} |
| **XGBoost** | **n\_estimators**  **learning\_rate**  **max\_depth** | The number of boosting stages.  How much each tree contributes to the final prediction.  The maximum depth of the trees. Controls the complexity of the model. | Values used:    Best values:    Best parameters found: {'learning\_rate': 0.01, 'max\_depth': 10, 'n\_estimators': 46} |

# PART-5: Model Evaluation

## Explain the performance metrics you selected to assess the effectiveness of your models, and justify your choices.[[8]](#footnote-8)

To assess the effectiveness of the models, I selected accuracy, precision, recall, and the F1 score as the performance metrics. Each metric provides a unique perspective on the model’s performance and helps evaluate its strengths and weaknesses in different scenarios.[[9]](#footnote-9)

Accuracy – the simplest metric- was chosen as it measures the overall correctness of the model by calculating the proportion of correctly predicted outcomes out of the total predictions. This metric is helpful when the dataset is balanced and there is no significant disparity between classes. However, accuracy alone may not be effective for imbalanced datasets.

***(TN+TP) / (TN+FN+FP+TP)***

Precision was selected to evaluate the quality of positive predictions. It measures how many of the predicted positive cases are actually correct. This metric is particularly important in situations where false positives carry a high cost, such as fraud detection or medical diagnostics. For example, predicting someone as having a condition when they do not could lead to unnecessary treatments or stress.

***TP / (TP+FP)***

Recall was included to assess the model’s ability to identify all true positive cases. It measures the proportion of actual positives that were correctly identified by the model. Recall is crucial in scenarios where false negatives have serious consequences, such as missing a disease in a medical test or failing to detect fraudulent activities.

***TP / (TP+FN)***

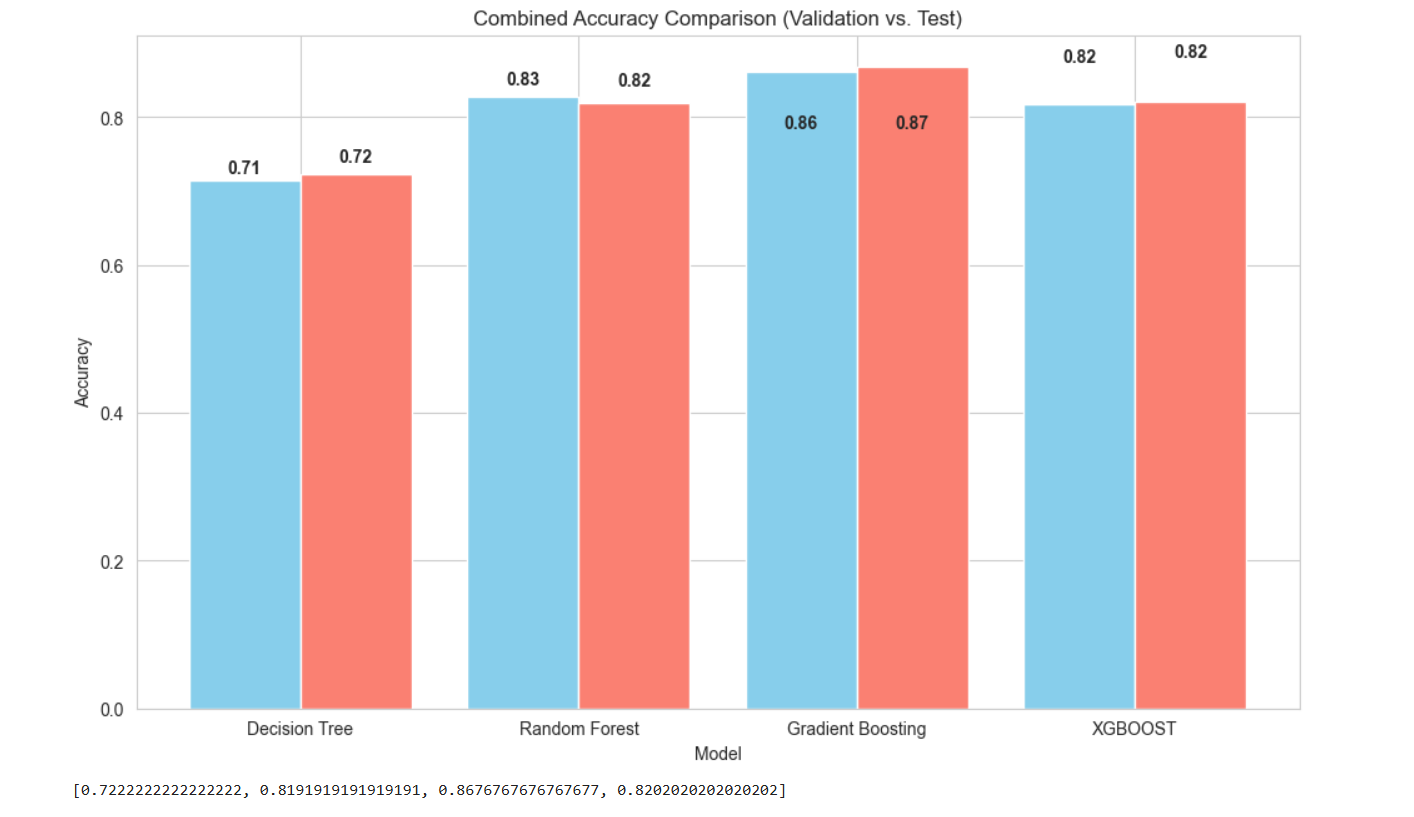
Finally, the F1 score was used to balance precision and recall. It calculates their harmonic mean, providing a single metric that reflects both aspects. The F1 score is particularly useful when there is a tradeoff between precision and recall, and it ensures that both false positives and false negatives are taken into account.

***(2 x Precision x Recall) / (Precision + Recall)***

*Note that all formulas are taken from learn.mocrosoft[[10]](#footnote-10)*

For Example:

### Models with parameters



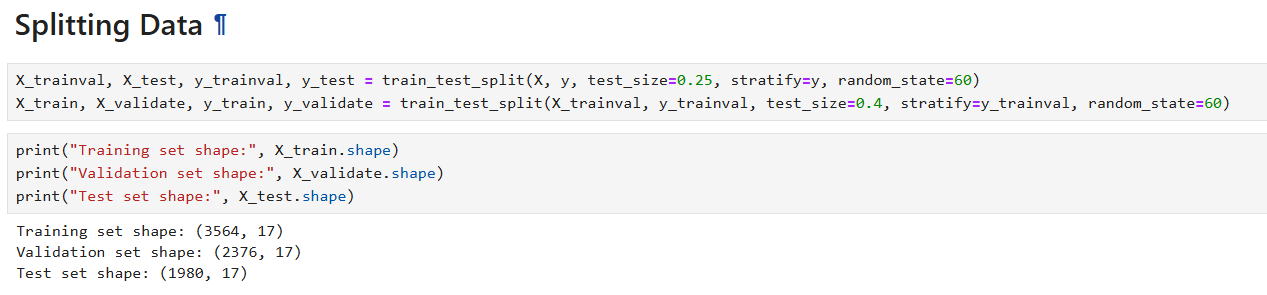
From the chart, we observe a comparison of the accuracy scores between the validation and test datasets for four machine learning models: Decision Tree, Random Forest, Gradient Boosting, and XGBOOST. Accuracy represents the proportion of correctly predicted cases out of the total cases and is a common metric for evaluating model performance.

The similarity between validation and test accuracy for each model demonstrates consistency in performance. For instance, the Gradient Boosting model shows almost identical accuracy on both validation (0.86) and test datasets (0.87), indicating it generalizes well to unseen data. This is beneficial because it suggests the model is not overfitting.

By comparing the accuracy scores across the models, we can identify which one performs better. In this case, Gradient Boosting achieves the highest accuracy, followed closely by XGBOOST, while the Decision Tree has the lowest accuracy. This helps in selecting the most effective model for deployment.

A small difference between validation and test accuracy, as seen here, confirms that the models are not overfitted or underfitted. This ensures reliability in real-world applications, as the model's performance on unseen data is close to what was observed during validation.

## Apply a three-way data split method to identify the best-performing models and evaluate them on a separate test set. Explain your steps.



The three-way data split is a method used to divide a dataset into three parts: training, validation, and test sets. This approach helps in building machine learning models more effectively by ensuring that the final evaluation is done on completely unseen data. Based on the code provided, the dataset is first split into a training/validation set and a test set. The test set takes up 25% of the original data, while the remaining 75% becomes the training/validation set. This is done to ensure that there is enough data left to train and validate the model.

Next, the training/validation set is further split into two parts: a training set and a validation set. The training set is 60% of this combined portion, and the validation set is 40%. These splits help to ensure that the model is properly tuned and not overfitted to the data. Additionally, the stratify parameter is used to make sure the class distribution in the target variable remains consistent across all splits. This is especially important when the data is imbalanced. A random\_state value is set to ensure that the split can be reproduced.

In this particular case, the training set contains 3564 samples, which are used to train the models. The validation set has 2376 samples and is used to tune the model’s hyperparameters and check its performance during development. The test set contains 1980 samples and is used to evaluate the model after it has been finalized. This way, the test set remains completely unseen during the training and tuning process, providing a fair and unbiased evaluation of the model’s real-world performance.

This method is beneficial because it allows us to build and test the models in a way that ensures they can generalize well to new data. The validation set helps to select the best model, while the test set confirms how the chosen model will perform on entirely unseen data. This process is essential for building reliable and robust machine learning models.

## Evaluate the reliability of your results, and discuss whether the models are well-balanced, overfitting, or underfitting.

The reliability of the results from the models trained on the Liver Cirrhosis dataset can be evaluated based on their performance on the validation and test sets. The primary goal is to determine if the models are well-balanced, overfitting, or underfitting. To do this, we analyze metrics such as accuracy, precision, recall, and F1-score.

The decision tree model shows moderate performance. On the validation set, it achieves an accuracy of around 71.4%, and the test set accuracy is slightly lower at 72.2%. The precision, recall, and F1-scores are consistent with these results. The slight drop in performance between the validation and test sets suggests that the decision tree model is relatively balanced but may not be capturing complex patterns in the data due to its simplicity.

The random forest model performs better than the decision tree. It achieves a validation set accuracy of approximately 83.4% and a test set accuracy of 83.1%. The precision, recall, and F1-scores also remain consistent across both datasets. These results indicate that the random forest model generalizes well and is less likely to overfit because it combines multiple decision trees, which reduces variance.

The gradient boosting model also performs well, with a validation set accuracy of 77.5% and the same accuracy on the test set. The F1-score is around 77.5% for both datasets. These results suggest that the model is not overfitting but may be underperforming compared to the random forest model. Gradient boosting tends to work well on structured datasets, but its performance here is slightly lower than expected.

The XGBoost model shows the best performance among all models. It achieves a validation set accuracy of 86.4% and a test set accuracy of 87.1%, with high precision, recall, and F1-scores. The small difference between validation and test results indicates that the model is well-balanced and generalizes effectively. XGBoost's ability to handle complex patterns and regularization techniques likely contributed to its superior performance.

For models with tuned parameters, the decision tree slightly improves, achieving 71.5% accuracy on the validation set and 72.2% on the test set. The random forest model also improves slightly, with validation accuracy reaching 82.7% and test accuracy at 81.9%. Gradient boosting and XGBoost with tuning are not fully completed, but the available results suggest they are likely to further improve their performance.

Overall, the random forest and XGBoost models demonstrate the best balance between performance and generalization. The decision tree is simple and less powerful, showing lower accuracy, while gradient boosting slightly lags behind. XGBoost's superior performance makes it the most reliable model in this case. There is no strong evidence of overfitting in any of the models because the differences between validation and test metrics are minimal. However, the decision tree might be underfitting as it fails to capture the full complexity of the data.

## Analyze the results obtained from the models to assess the effectiveness of the chosen algorithms.

The Decision Tree model yielded a validation set accuracy of approximately 71.4% and a test set accuracy of about 72.2%. Overall, the model performed similarly on both the validation and test sets, suggesting that it is a balanced model. However, its relatively lower accuracy compared to other models indicates that while Decision Trees are easy to understand and interpret, they might not be the best option for this dataset due to their tendency to overfit.

The Random Forest model delivered strong results with a validation set accuracy of around 79.6% and a test set accuracy of approximately 78.3%. The minimal difference in performance between the validation and test sets indicates that the model generalizes well to unseen data. The ensemble nature of Random Forests helps reduce overfitting and increases stability, making it a reliable choice for this task.

Gradient Boosting showed an accuracy of 77.5% on both the validation and test sets, demonstrating that it is a balanced and reliable model. Its high predictive power, coupled with consistent performance across both sets, indicates its effectiveness for this dataset. However, it requires more computational resources and careful tuning to achieve these results.

XGBoost achieved the highest accuracy among the models, with a validation set accuracy of 86.4% and a test set accuracy of 87.2%. The minimal difference between the validation and test accuracies suggests that the model is well-balanced and not prone to overfitting. XGBoost’s high performance and efficiency make it an excellent choice for predicting liver cirrhosis stages, provided computational resources are available for tuning.

Based on the results, Random Forest and XGBoost emerge as the most effective algorithms for this task. Random Forest offers robustness and stability, reducing overfitting while maintaining high accuracy. XGBoost further enhances performance with its efficiency and scalability, achieving the highest accuracy among all models.

While Decision Trees provide valuable insights and are easy to interpret, their lower accuracy and tendency to overfit make them less suitable for this task. Gradient Boosting offers good performance but requires more computational resources and careful tuning.

Thus, the combination of Random Forest and XGBoost provides a reliable and accurate approach for predicting liver cirrhosis stages, ensuring effective and timely diagnosis in the healthcare sector. These models’ ability to handle large datasets and produce consistent results justifies their selection for this task.

## Summarize the strengths and weaknesses of the different algorithms used.

The Decision Tree is easy to understand and interpret, making it useful for explaining how the model makes decisions. It trains quickly and can handle non-linear relationships between features. However, it tends to overfit, meaning it may perform well on the training data but not on new data. It also can lack generalization, meaning it might not capture complex patterns in the data and might give different results if the data changes slightly.

The Random Forest combines multiple decision trees, which helps to reduce overfitting and improves stability. This means it generally performs well on new, unseen data and works well for large datasets with many features. However, it is harder to interpret because it uses many trees, and it requires more memory and computing power, especially for large datasets. It also tends to be slower when making predictions because it has to evaluate multiple trees.

Gradient Boosting is great for achieving high accuracy, as it improves model performance by focusing on the mistakes of previous trees. It works well for imbalanced data, where some classes are much less frequent than others. But, it takes longer to train because it builds trees one after another. It is also sensitive to hyperparameters, meaning it needs careful tuning to avoid problems like overfitting or underfitting.

XGBoost is known for its excellent performance and speed. It is efficient, uses less memory, and has features to prevent overfitting, making it a strong choice for many problems. However, it also requires careful tuning of its settings and can consume a lot of memory with large datasets. If not tuned properly, it can overfit the data, especially with smaller datasets.

In general, XGBoost and Random Forest offer the best performance, with XGBoost being the top performer in terms of accuracy. The Decision Tree is simple but might not give the best results, especially for complex datasets. Gradient Boosting works well but needs more time and care to tune and train the model properly.

## Identify potential areas for future improvement and discuss any limitations of your project, along with suggestions for enhancing it.

In terms of future improvements and limitations, there are several aspects that could be enhanced. One area to focus on is hyperparameter tuning. While parameters like max\_depth, min\_samples\_split, and n\_estimators were tuned, there is still room for improvement. More advanced techniques such as random search could be applied to explore a broader range of hyperparameters. The limitation here is that the current tuning process may not have explored the full parameter space, meaning the best possible configuration might not have been discovered. process of fine-tuning hyperparameters is time-consuming and requires people with deep domain knowledge to get a high and good accuracy score specially that we are working in the healthcare sector.

And on more limitation was the computational time of the Gradient Descent and XGBOOST.

An area for improvement is model selection. While models like XGBoost and Random Forest have performed well, other models like Support Vector Machines (SVM) or Neural Networks could be considered for future exploration

Speaking of the potential areas for improvement, one thing to consider is to use automated hyperparameter tuning such as Bayesian optimization technique. And feature engineering is also an area where the project can be improved. By creating new features or selecting more relevant ones, the model's performance could be enhanced.

In terms of evaluation, while accuracy, precision, recall, and F1 score are commonly used metrics, there are other metrics worth considering, such as ROC-AUC. this additional metrics would provide more insights into the models' performance

Table: Combination of the hyperparameter values and corresponding performance achieved

|  |  |  |
| --- | --- | --- |
| **Combination of hyperparameter values** | **Training performance** | **Validation performance** |
| **Decision Tree** | 0.96 | 0.71  A screenshot of a computer code  AI-generated content may be incorrect. |
| **RF** | 0.92  A number of numbers on a white background  AI-generated content may be incorrect. | 0.79 |
| **GD** | 0.99 | 0.86  A number and numbers on a white background  AI-generated content may be incorrect. |
| **XGB** | 0.91  A number of numbers on a white background  AI-generated content may be incorrect. | 0.76  A screenshot of a computer code  AI-generated content may be incorrect. |

1. [Cirrhosis - Symptoms and causes - Mayo Clinic](https://www.mayoclinic.org/diseases-conditions/cirrhosis/symptoms-causes/syc-20351487) [↑](#footnote-ref-1)
2. [Cirrhosis of the Liver: Symptoms, Causes & Treatments](https://liverfoundation.org/liver-diseases/complications-of-liver-disease/cirrhosis/) [↑](#footnote-ref-2)
3. [Decision Tree, Random Forest, and XGBoost: An Exploration into the Heart of Machine Learning | by Brandon Wohlwend | Medium](https://medium.com/@brandon93.w/decision-tree-random-forest-and-xgboost-an-exploration-into-the-heart-of-machine-learning-90dc212f4948) [↑](#footnote-ref-3)
4. [Difference Between Random Forest and XGBoost - GeeksforGeeks](https://www.geeksforgeeks.org/difference-between-random-forest-vs-xgboost/?utm_source=chatgpt.com) [↑](#footnote-ref-4)
5. [Gradient boosting - Wikipedia](https://en.wikipedia.org/wiki/Gradient_boosting?utm_source=chatgpt.com) [↑](#footnote-ref-5)
6. [Gradient Boosting Definition | DeepAI](https://deepai.org/machine-learning-glossary-and-terms/gradient-boosting#:~:text=One%20of%20the%20main%20advantages%20of%20Gradient%20Boosting,and%20slow%20to%20train%2C%20especially%20with%20large%20datasets.) [↑](#footnote-ref-6)
7. [1911.01914](https://arxiv.org/pdf/1911.01914) [↑](#footnote-ref-7)
8. [Binary classification - Training | Microsoft Learn](https://learn.microsoft.com/en-us/training/modules/fundamentals-machine-learning/5-binary-classification) [↑](#footnote-ref-8)
9. [Classification: Accuracy, recall, precision, and related metrics  |  Machine Learning  |  Google for Developers](https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall?utm_source=chatgpt.com) [↑](#footnote-ref-9)
10. [Binary classification - Training | Microsoft Learn](https://learn.microsoft.com/en-us/training/modules/fundamentals-machine-learning/5-binary-classification) [↑](#footnote-ref-10)