# Models Comparison for Liver Cirrhosis Survival Prediction

Cleto Pellegrino

July 5, 2024

## Contents

# 1 Introduction

The liver is the **second largest organ** in the human body and **one of the most important for human health**. Cirrhosis is a progressive condition that puts both a person's liver and life at risk. [1]

Different models can be compared to determine **which one best predicts the likelihood of a patient to survive against the disease**.

# 2 Dataset

This study utilizes the Cirrhosis Prediction Dataset available on **Kaggle**[2], comprising data collected from the **Mayo Clinic trial in primary biliary cirrhosis** (*PBC*) of the liver conducted between 1974 and 1984.

The dataset includes various information such as Cholesterol, Triglycerides, and, crucially, the patient's **status**. In total, there are 418 observations with 20 attributes.

# 3 Data Visualization

## 3.1 Preliminary Adjustments

Before commencing with a complete dataset visualization, several **adjustments** were made for some features:

- All features of type "object" were converted to "**category**" (in order to plot barplots and use the describe function provided in pandas with specific characteristics for this type of features).
- The "ID" column was **dropped** as it is useless for the analysis, solely serving as an **identifier**.
- The "Stage" feature, originally represented as *float64*, was converted to **categorical**, since it can be considered the **name** of the current liver condition.
- Age, initially reported in **days**, was converted to years using the formula: $\text{round}(\frac{x}{365}, 0)$.

  This was just made for visualization reasons.
- Status **C** was converted to "High chance", Status **CL** was converted to "Transplant needed" and Status **D** was converted to "Low chance" (again, still just for visualization reasons).

## 3.2 Plot Analysis

By looking at the **heatmap** of the numerical features (Figure 1),

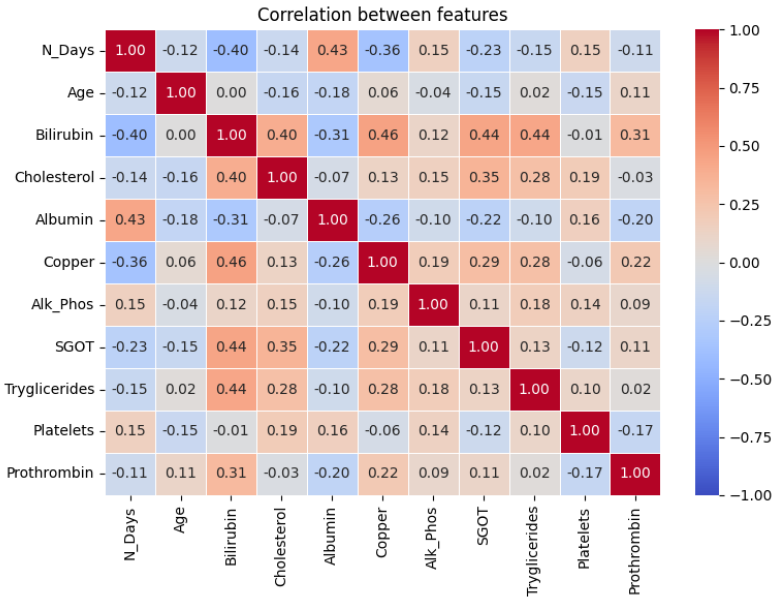



Figure 1: Heatmap of all numerical features.

one can see that there is **no strong positive/negative correlation** between them. For this reason, we cannot drop features based on their correlation, indeed no couples contain the **same type of information**.

Figure 2 demonstrates the **imbalanced distribution** of the dataset labels.

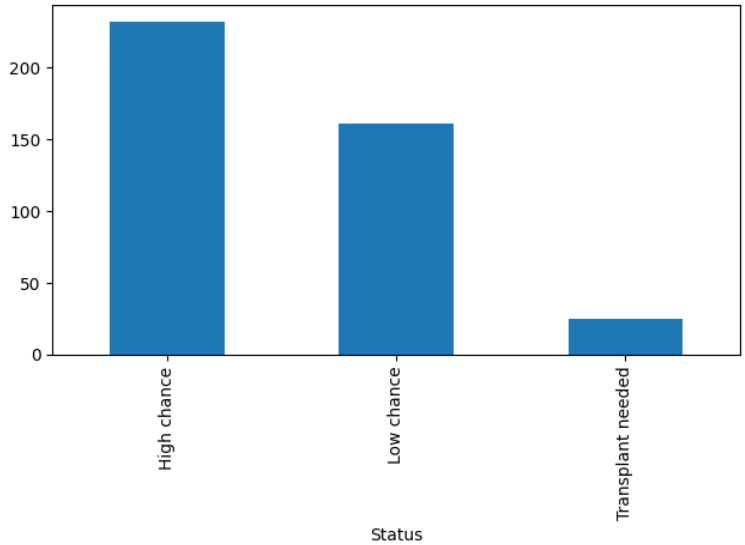



Figure 2: Distribution of patient statuses.

This is an important information not only for the **correct implementation of the pipelines** (imputing missing values, re balancing) but also to decide **how we will evaluate their performances** to select the best at predicting the correct chance of survival of the patient.

# 4 Preprocessing

The Train set showed lots of ***NaN* values**, but due to the low number of observation found, removing all of them was not a possible option in order to **maintain a substantial number of information**.

In order to replace *NaN* values in numerical features with plausible values, a **CustomKNNImputer** with **different values of *k_neighbours*** was used. The imputing was done also by dividing patients **by label** in order to create new values only considering **neighbours of the same class** (this can lead to more separate classes and better classification. Indeed by doing this we assure that, when we have an imbalanced problem, imputed values are **drawn from instances within the same class**, reducing the risk of **bias** introduced by imputing based on the **majority** class).

For categorical features, in order to replace *NaN* values with plausible values, a **CustomSimpleImputer** with "mode" (reported in python as *most_frequent*) was used. Also in this case, patients where divided by label.

Before deciding for the best **scaler**, it's important to analyze the **boxplots** (and so the distribution) of our numerical features one by one:

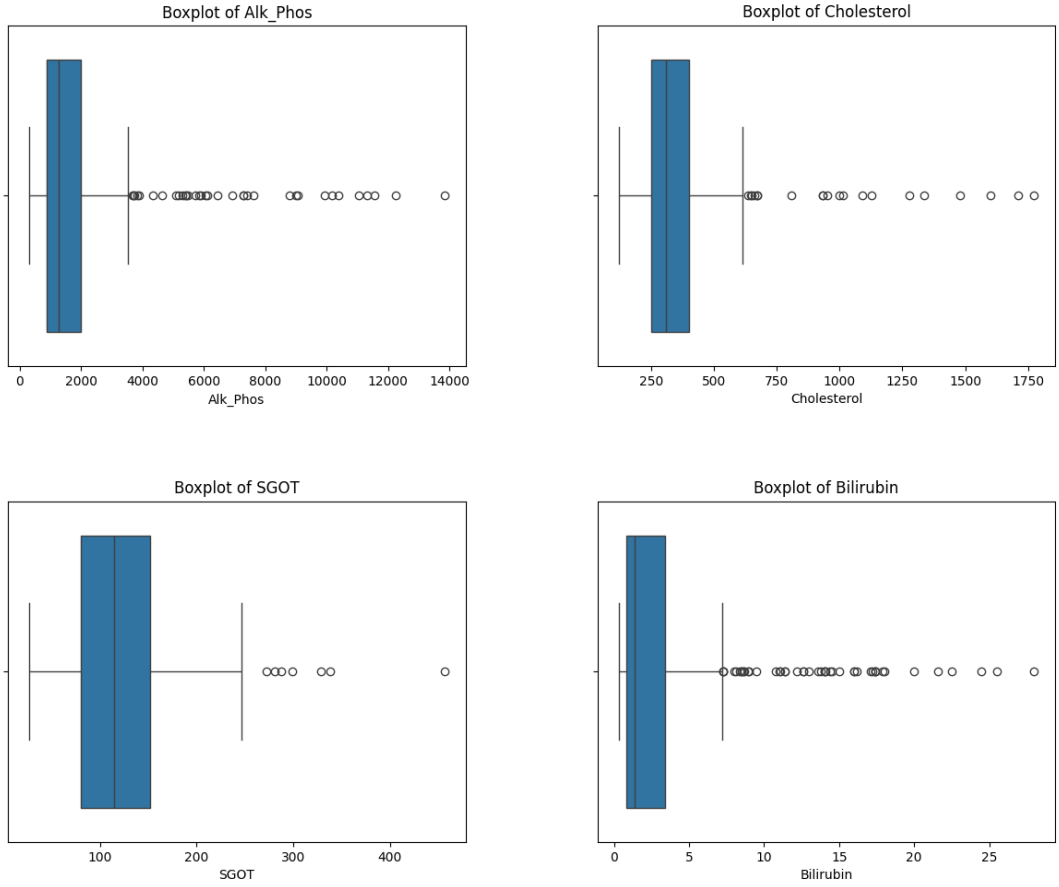



Figure 3: Boxplots of some numerical features as examples.

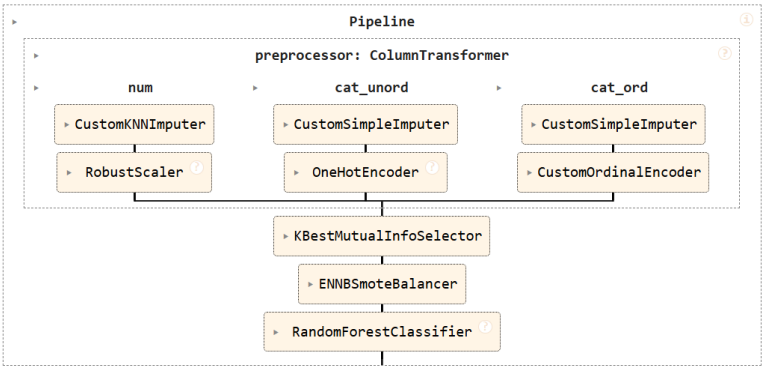
Given the relevant presence of outliers, the best choice would be to apply a **RobustScaler**.

Furthermore, in order to convert categorical features to numerical ones, **OneHotEncoding** and **CustomOrdinalEncoding** were used, respectively "Drug" was converted in "Drug_D-penicillamine" and "Drug_Placebo", while "Sex" became "Sex_M" and "Sex_F".

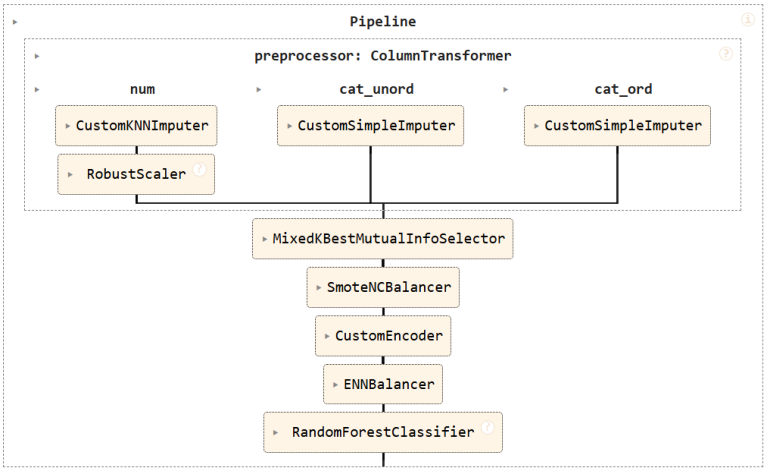
For the remaining categorical features, the CustomOrdinalEncoder implements and OrdinalEncoder for each feature, with sorting strategy that represent with high values **serious conditions** (the higher the values, the worse the condition). This steps were possible by exploiting ad-hoc pipelines (**ColumnTrasformers**) provided by the **imblearn** library.

Furthermore, different pipelines were tested:

- both Unordered and Ordered categorical features encoded before re-sampling:




- both Unordered and Ordered categorical features encoded after over sampling.




For the feature selection **mutual information** was used, applied to a **SelectKBest** strategy with various values of K.

The **KBestMtutalInfoSelector** was created in order to select a fixed random_state in order to **replicate the results** (since in sklearn it's not possible to pass the value while using the mutual information as a score function).

In the **MixedKBestMutualInfoSelector** class of the pipeline, there is a combination of mutual_info_classif for numerical features, while for categorical features, that cannot be analyzed with the same function (not implemented), there is an **handwritten implementation of the mutual information**.

Another important thing that is done inside the class is to leave **at least one feature for both numerical and categorical**, since SMOTENC needs at least one categorical feature (instead if there were only numerical, the other pipeline is preferred).

PCA was not tested since it's not **recommended when dealing with encoded features**.

To balance instances, a double balancing technique was used. The undersampling technique was the **EditedNearestNeighbour**, with sampling strategy to decrease the number of instances in the "High chance" class, along with different values of *n_neighbors*.

This method was used since it can not only decrease the number of instances, but it's also capable to **separate** classes in the process, by deleting instances that may be close while being in different classes. This was done with the goal to delete instances of the High chance class that may make more difficult to classify the others.

The only problem with that approach is that there is no way to control **how much instances are deleted** (due to how it's implemented), differently from the other approaches.

The oversampling approach was done in order to increase the number of instances in the Transplant needed class. Given the SMOTENC (SMOTE for numerical and categorical) approach for a pipeline, the other one has a **BorderlineSMOTE** approach, since it can increase the chance to classify better instances which are considered hard to classify, by increasing the number of instances at the "**border**". All the methods were also used by comparing different values of *n_neighbors*.

This distinction was done to try and test what happens when applying a classic SMOTE approach against a specific type of SMOTE for mixed data type.

Indeed SMOTENC creates instances with a **coherent** value based on the feature, while BorderlineSMOTE, like SMOTE, creates values by **interpolating** with existing values (for instance, if Sex is represented by 0 and 1, SMOTENC may create 1 or 0, while Borderline SMOTE a value also in between, which is **meaningless** in this case)

Overall, the goal of the whole sampling approach was to **decrease the number of instances of the majority class down to the closest class in terms of instances, then increasing the lowest up to the closest**. e.g.) Suppose:

- Starting labels: Low chance = 100, High chance = 200 and Transplant needed = 20;
- Final result: 100 for all the classes.

This decision was driven by the fact that only applying oversampling could have caused **over fitting**, since creating to much synthetic values may lead the model to learn the **noise in the data rather than the underlying patterns**.

# 5 Models Definition and Evaluation

In order to conduct the experiments, four different models where used:

- RandomForest;
- KNNeighbour;
- ADABOOST with naive Bayes classifier;
- Logistic regression.

All the model where optimized using the **Randomized Cross Validation** method.

Also it was important to decide the scorer in order to classify the best classifier, and the decision was to choose the classifier that improved the *f1_macro_average* score. This was done to **penalize** classifiers that did not achieve good performances in classifying the Transplant needed class, which is actually the hardest given the very low number of instances.

The hardest decision was made when choosing **how many folds** where needed in order to obtain a significant amount of information to perform statistical analysis (that could be considered **valid**) on the results. Indeed most of the tests may not be **efficient** when using few values, but on the contrary, a confusion matrix with not so many instances may be **useless to compare performances**.

The chosen method was the **Kruskall-Vallis** test (combined with the **Wilcoxon** test if statistically relevant differences were found between groups), which is a **non-parametric** test, that can be used to compare three or more independent distributions, and has as an assumption, as mentioned in the sklearn page, to have **5 or more values**, together with independence and equal variances (in order to avoid staying "at the border" of the assumption, **10 folds** were selected).

### 5.1 Validation of the Assumptions

Both the test mentioned before were used since the Q-Q plot Figure 4 of the scores showed that the distributions are **not normally distributed**, so non-parametric tests were selected.

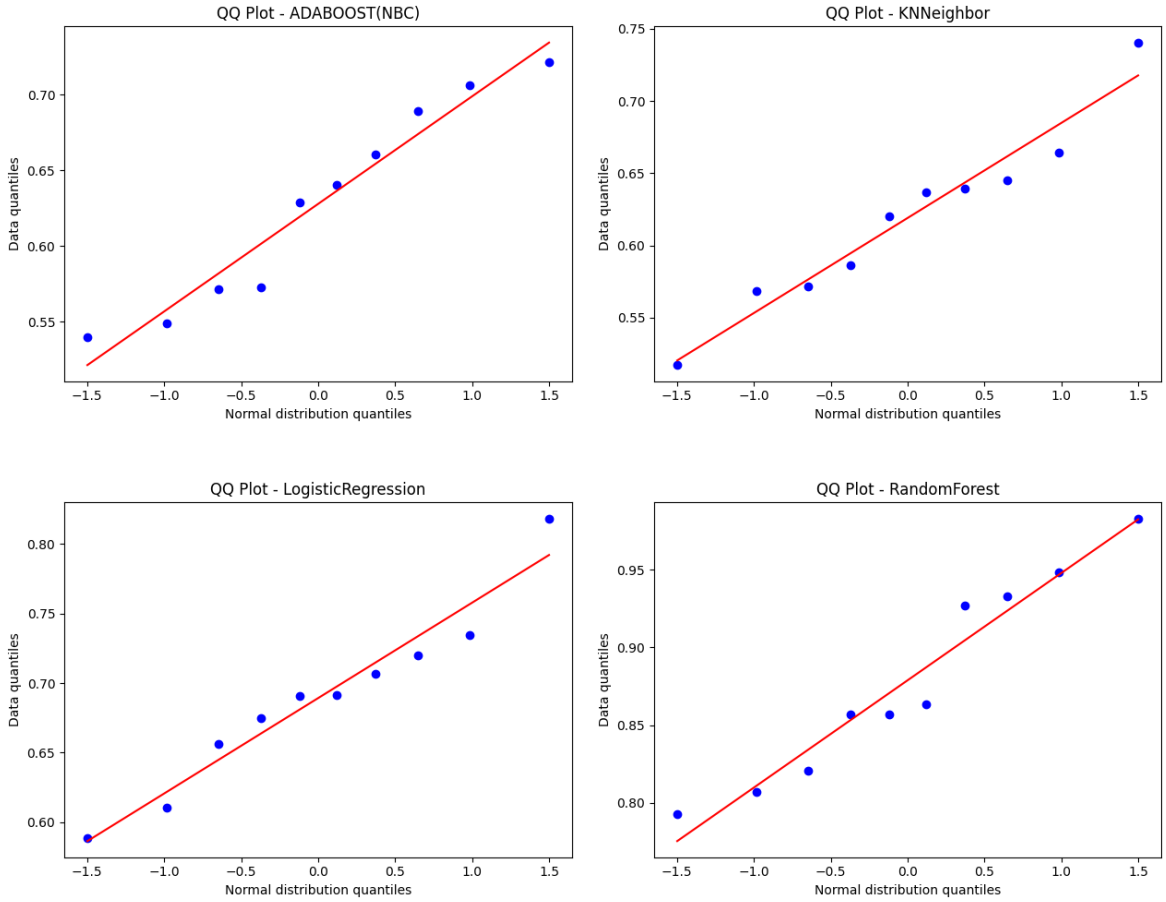



Figure 4: Q-Q plots for non SMOTENC pipeline as examples.

After seeing this, the equal variances can be proved by the **Levene's test**:

```
Non SMOTENC pipeline)
Levene's statistic: 0.1621720253985203
p value: 0.9211077886144717
There is no evidence to reject the null hypothesis of equal variances.

SMOTENC pipeline)
Levene's statistic: 0.0809357948703925
p value: 0.9699625551028712
There is no evidence to reject the null hypothesis of equal variances.
```

## 6 Final Results

The result of the pipelines showed that overall the RandomForest approach **performed better in every situation**.

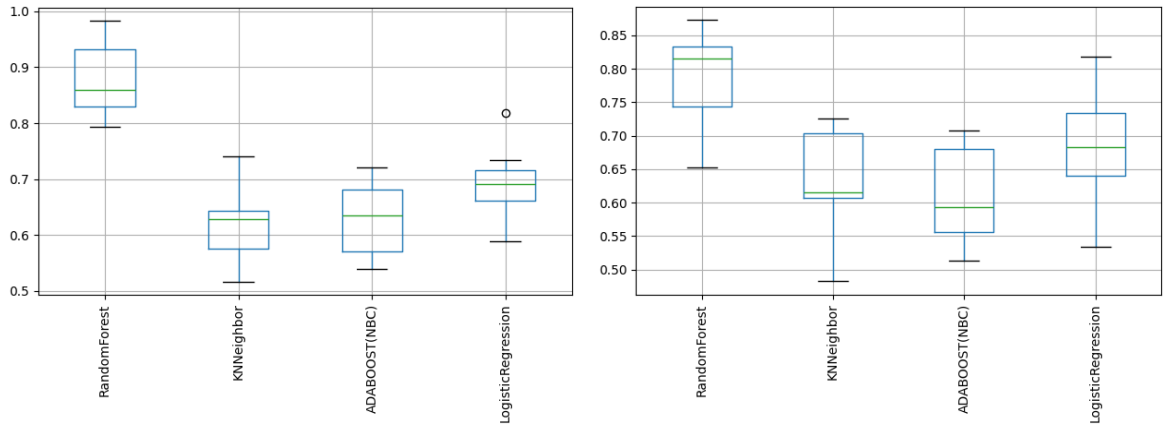
The following are the best parameters for both the pipeline:

- Non SMOTENC pipeline:

```
{
    'sampler__k_neighbors_cleaning': 3,
    'sampler__k_neighbors_borders': 7,
    'preprocessor__num__imputer__n_neighbors': 11,
    'feature_selector__k': 14,
    'classifier__max_depth': 7,
    'classifier__criterion': 'entropy'
}
```

- SMOTENC pipeline:

```
{
    'sampler2__k_neighbors_cleaning': 3,
    'sampler1__k_neighbors': 8,
    'preprocessor__num__imputer__n_neighbors': 15,
    'feature_selector__k_num': 11,
    'feature_selector__k_cat': 2,
    'classifier__max_depth': 5,
    'classifier__criterion': 'gini'
}
```

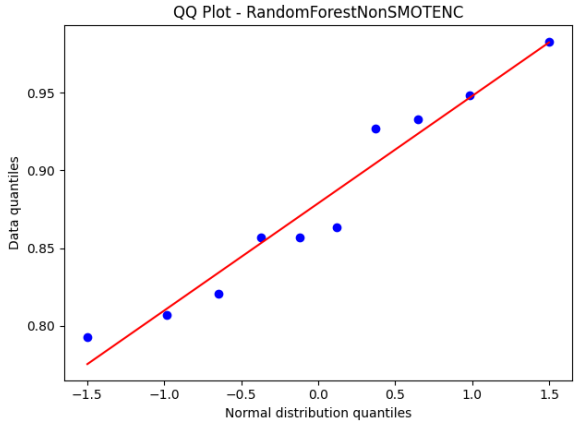



(a) Result with non SMOTENC pipeline.

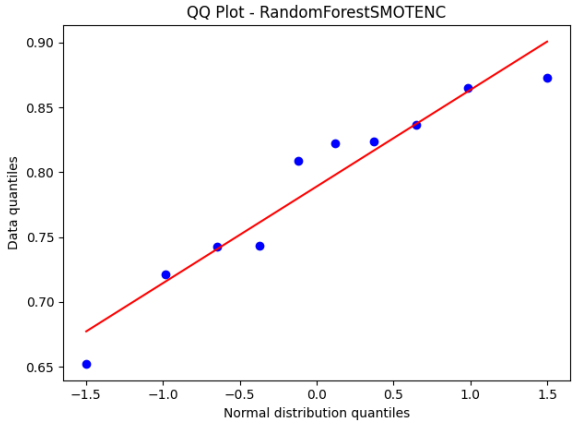
(b) Result with SMOTENC pipeline.

As a final step we can compare just the RandomForest pipelines and see if there are further considerations that can be done.

The distribution are still not normally distributed so we preocede with the Wilcoxon test:




(a) Q-Q Plot with non SMOTENC pipeline.



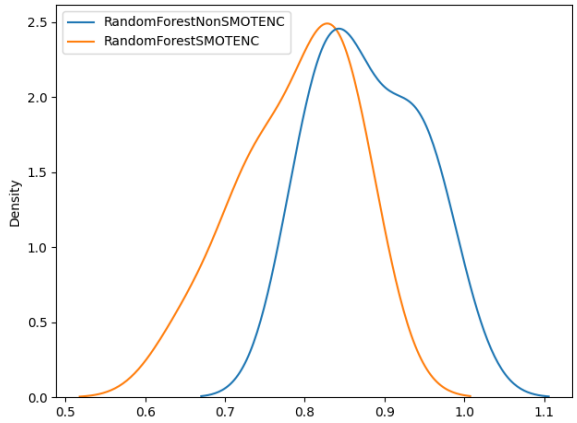

(b) Q-Q Plot with SMOTENC pipeline.

The following proves that the homoscedasticity is still valid:

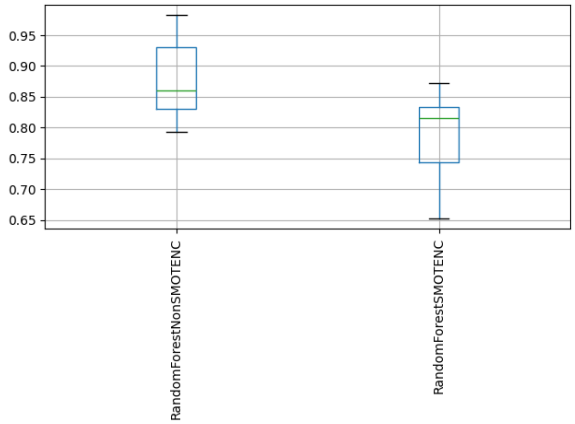
```
Levene's statistic: 0.026357926615570314
p value: 0.8728379042964531
There is no evidence to reject the null hypothesis of equal variances.
```

The test showed a statistically relevant difference between the pipelines. We can conclude that **the approach where we didn't use SMOTENC actually performed better**.

```
Wilcoxon statistic: 1.0
p value: 0.00390625
There is a statistically relevant difference.
```




(a) KDE Plots.




(b) Boxplots.

# 7 Appendix

## 7.1 Evaluation with non SMOTENC pipeline

```
Kruskal-Wallis statistic: 24.914634146341484
p value: 1.608825352331587e-05
There is a statistically relevant difference between at least two groups.
Comparing RandomForest and KNNeighbor
Wilcoxon statistic: 0.0
p value: 0.001953125
There is a statistically relevant difference between RandomForest and KNNeighbor
----
Comparing RandomForest and ADABOOST(NBC)
Wilcoxon statistic: 0.0
p value: 0.001953125
There is a statistically relevant difference between RandomForest and ADABOOST(NBC)
----
Comparing RandomForest and LogisticRegression
Wilcoxon statistic: 0.0
p value: 0.001953125
There is a statistically relevant difference between RandomForest and LogisticRegression
----
Comparing KNNeighbor and ADABOOST(NBC)
Wilcoxon statistic: 24.0
p value: 0.76953125
There is no statistically relevant difference between KNNeighbor and ADABOOST(NBC)
----
Comparing KNNeighbor and LogisticRegression
Wilcoxon statistic: 0.0
p value: 0.001953125
There is a statistically relevant difference between KNNeighbor and LogisticRegression
----
Comparing ADABOOST(NBC) and LogisticRegression
Wilcoxon statistic: 6.0
p value: 0.02734375
There is a statistically relevant difference between ADABOOST(NBC) and LogisticRegression
----
```

### 7.2 Evaluation with SMOTENC pipeline

```
Kruskal-Wallis statistic: 18.045365853658552
p value: 0.0004304744375195307
There is a statistically relevant difference between at least two groups.
Comparing RandomForest and KNNeighbor
Wilcoxon statistic: 0.0
p value: 0.001953125
There is a statistically relevant difference between RandomForest and KNNeighbor
----
Comparing RandomForest and ADABOOST(NBC)
Wilcoxon statistic: 0.0
p value: 0.001953125
There is a statistically relevant difference between RandomForest and ADABOOST(NBC)
----
Comparing RandomForest and LogisticRegression
Wilcoxon statistic: 1.0
p value: 0.00390625
There is a statistically relevant difference between RandomForest and LogisticRegression
----
Comparing KNNeighbor and ADABOOST(NBC)
Wilcoxon statistic: 21.0
p value: 0.556640625
There is no statistically relevant difference between KNNeighbor and ADABOOST(NBC)
----
Comparing KNNeighbor and LogisticRegression
Wilcoxon statistic: 9.0
p value: 0.064453125
There is no statistically relevant difference between KNNeighbor and LogisticRegression
----
Comparing ADABOOST(NBC) and LogisticRegression
Wilcoxon statistic: 7.0
p value: 0.037109375
There is a statistically relevant difference between ADABOOST(NBC) and LogisticRegression
----
```

# 8 References


1. Medical News Today, "Cirrhosis of the liver: Life expectancy,"
   https://www.medicalnewstoday.com/articles/cirrhosis-of-the-liver-life-expectancy

2. Cirrhosis Prediction Dataset
   https://www.kaggle.com/datasets/fedesoriano/cirrhosis-prediction-dataset