

Assignment 6

Computer Science Department, University of Crete
MACHINE LEARNING - CS 577, Fall 2022

Student: Mourouzidou Eleni, Msc Bioinformatics

PART 1 - Data Exploration

Begin by identifying which features are continuous or categorical, reporting how many unique values they have, and calculating basic statistics like mean and variance. Look for features with very high or low variance/mean, imbalanced feature distributions or highly dependent features with the target variable Y. This will give you insight on selecting the correct configuration setup during model development. Visualizing the distribution of features, outliers or feature relationships is also helpful in identifying data quality issues or other issues that could affect model performance.

Since we have no other information about the features data types and due to the fact that all features consist only of integer values, we need to consider some other characteristics of the vectors (columns) to determine whether a feature is categorical or continuous. A common naive approach could be that since the sample size in the given dataset *File Dataset6.A_XY.csv* is relatively large (2199 rows) but there are not many unique values for most of the features (max number of unique values in the dataset = 37 for the 14th feature). Thus, we can assume that the overall ratios $\text{unique_values} / \text{total sample size}$ is low for each feature. Under this assumption it is reasonable to consider features with a small ratio as categorical, since it is not that likely to meet certain values again and again in a scenario of continuous variables. On the other hand, we could confidently assume that variables that strictly appear to take 2 unique values out of such a large sample size are binary categorical. To effectively classify the features of our dataset as continuous or categorical it is also important to observe their distribution. Thus, we created the following histograms (*Figure 1*), one for every feature, indicating the distribution of the values, the amount of unique values and the variance of each variable:

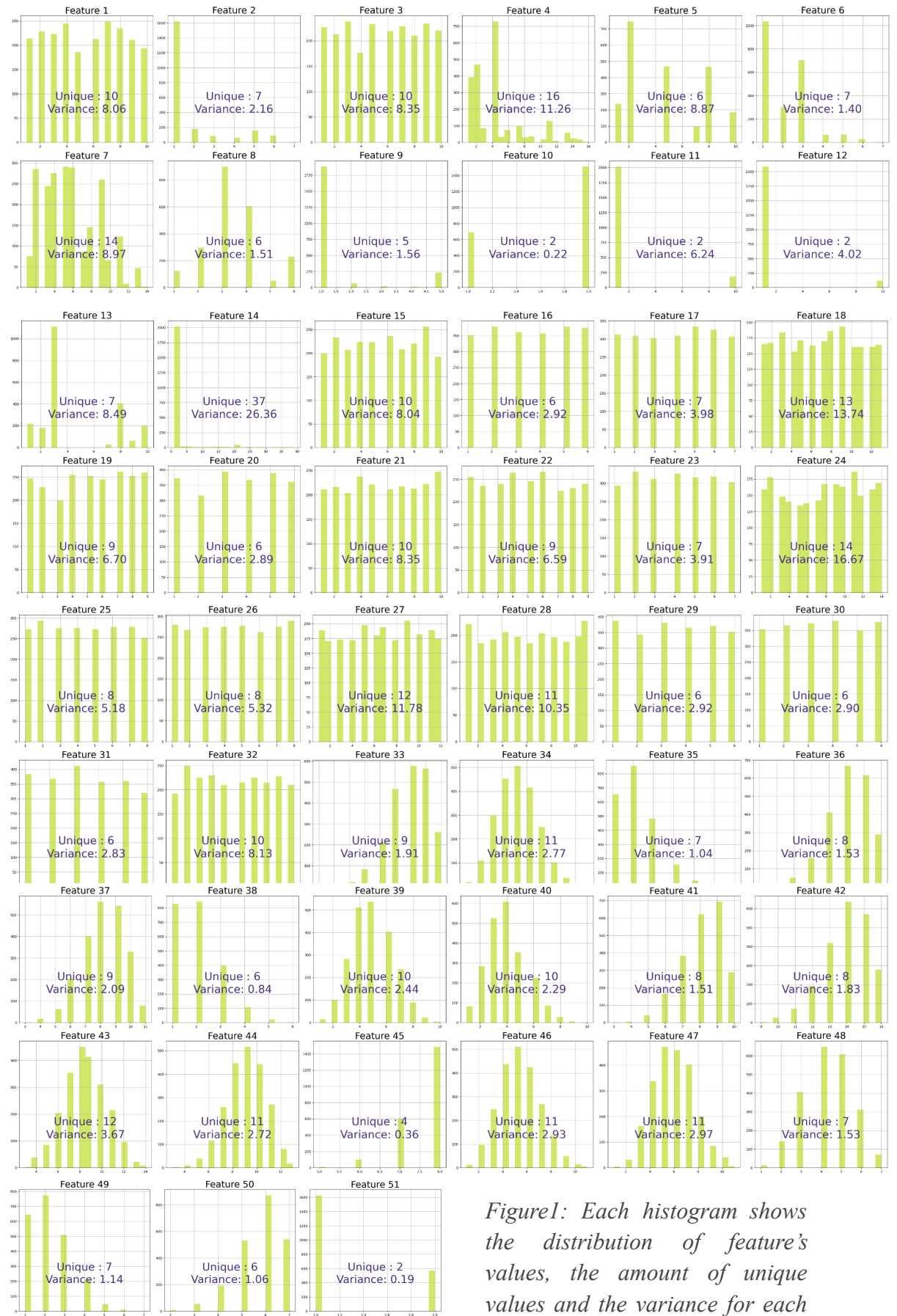


Figure1: Each histogram shows the distribution of feature's values, the amount of unique values and the variance for each feature

According to the distribution of the values, the unique values number and the variance of each variable, we will classify the features to continuous and categorical based on the following criteria:

- Features with exactly 2 unique values will be classified as categorical (specifically binary categorical)
- Smooth and bell shaped distributions will be considered as continuous variables, especially those with higher variance.
- Distributions with distinct clusters will be considered as categorical, especially the ones with low variance.
- Features with a limited number of unique values (empirically chosen this time) are also more likely to be classified as categorical than those with a higher one (that are more likely to be classified as continuous)

Based on the criteria mentioned above we decided to classify the data types as follows:

- Define empirically which features show a normal shaped distribution (*Figure 1*)
- Create a list to match boolean values (0 : No bell shaped distribution observed, 1 : Yes - bell shaped distribution observed) to the corresponding feature indices
- Iterate over all the unique thresholds (set of possible unique values met over all features) and each time set features with unique values $<$ threshold \rightarrow classified as categorical, thus features with unique values \geq threshold \rightarrow classified as continuous.
- Finally for each threshold we will exhaustively compute a percentage of agreement between the two criteria and we will pick the unique values threshold that corresponds to the highest agreement percentage. (Result: optimal threshold = 9).
- The scatterplots below (*Figure 2*), shows how the features are classified to “categorical” = 0 or “continuous” = 1 according to their distribution shape and are colored based on the threshold split classification.

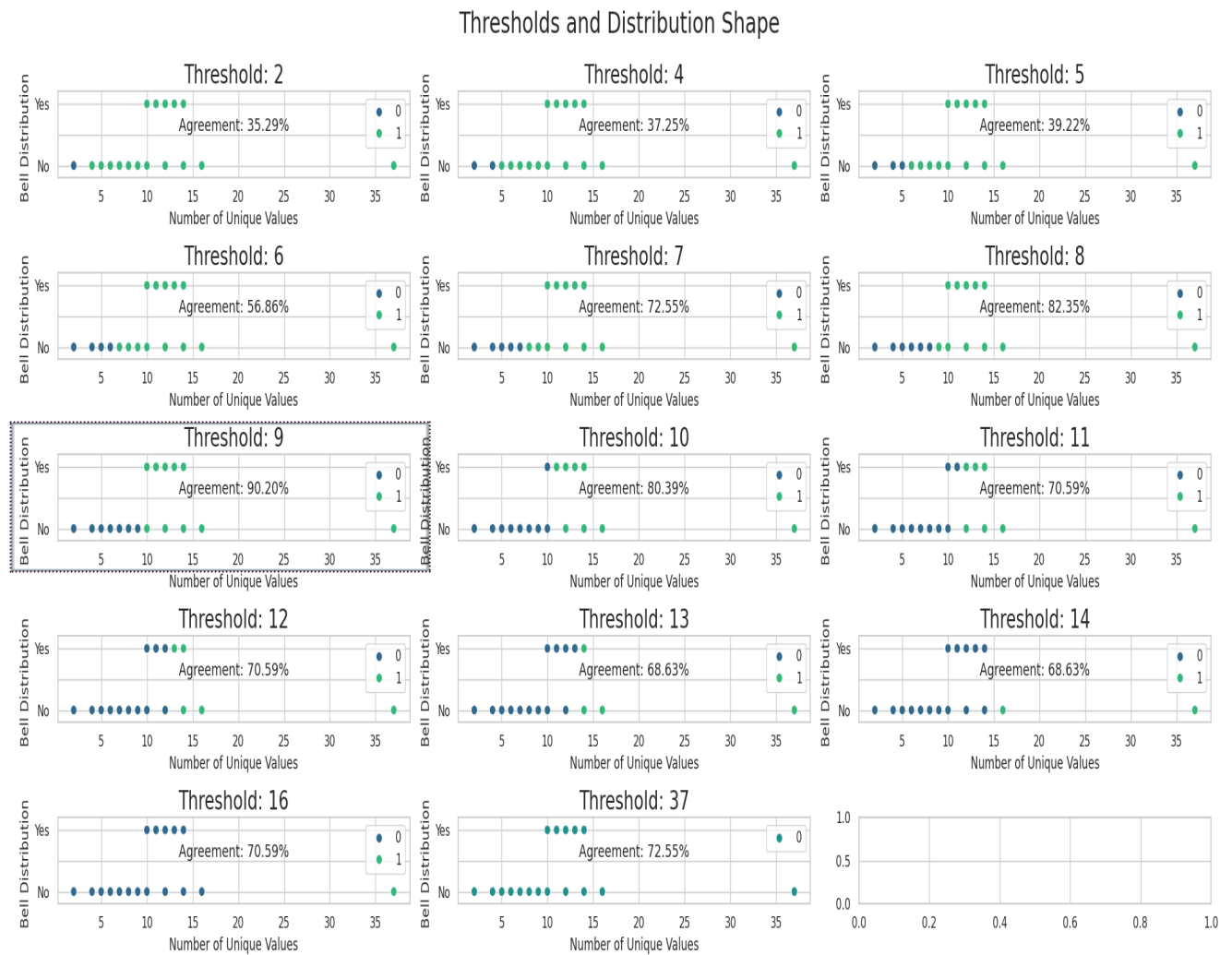


Figure 2: This scatterplot shows the relationship between bell-shaped distribution and unique values among features across varying thresholds, targeting to select the optimal threshold that classifies the features to “continuous” (green) or “categorical” (blue).

Based on the approach explained above, we will handle features with unique values number $<$ threshold = 9 as categorical variables and features with unique values number \geq 9 as continuous variables.

Here, we can see the indices (in the original dataset that correspond to the categorical and continuous features)

```
categorical features [1, 4, 5, 7, 8, 9, 10, 11, 12, 15, 16, 19, 22, 24, 25, 28, 29, 30, 34, 35, 37, 40, 41, 44, 47, 48, 49, 50]
continuous features [0, 2, 3, 6, 13, 14, 17, 18, 20, 21, 23, 26, 27, 31, 32, 33, 36, 38, 39, 42, 43, 45, 46]
```

Dependence between features and Y

Now we would like to check whether there is some interesting relationship (dependence) between any features and the target variable Y. To do so, we will have to approach continuous and categorical features differently.

Continuous:

We will use the *corrwith* method to compute the pairwise correlation between the columns of our dataframe(features) and the last column (Y target variable).

We will consider that feature pairs with correlation coefficient:

- [0.1, 0.3]: have weak positive correlation / [-0.1, -0.3] : weak negative
- [0.3, 0.5] : have moderate positive correlation / [-0.3, -0.5]: moderate negative
- [0.5, 1]: have strong positive correlation / [-0.5, 1]: strong negative correlation

We can observe that there are no continuous features with a strong correlation (< -0.3 or > 0.3) with the Y target variable. There is just one feature, *feature 4* with moderate positive correlation with the target variable (correlation coefficient = 0.29908) and just one feature, *feature 7* with moderate negative correlation with Y (correlation coefficient = -0.21082). The remaining features have an extremely weak correlation with the target variable (< 0.1 or > -0.1). (Fig3)

1	0.075060	28	0.007235
3	0.018320	32	-0.020772
4	0.299083	33	0.047005
7	-0.210819	34	0.007230
14	0.007137	37	0.013520
15	0.015646	39	-0.005455
18	0.021084	40	-0.000479
19	0.025415	43	0.021017
21	-0.016029	44	0.002991
22	0.002056	46	-0.033877
24	-0.002532	47	0.027113
27	-0.005100		

Figure 3: Correlation between continuous features and the target variable Y

Categorical:

To check whether some categorical features are highly dependent of target variable Y, we will perform chi squared test and set a threshold for the p-value : $\alpha = 0.05$

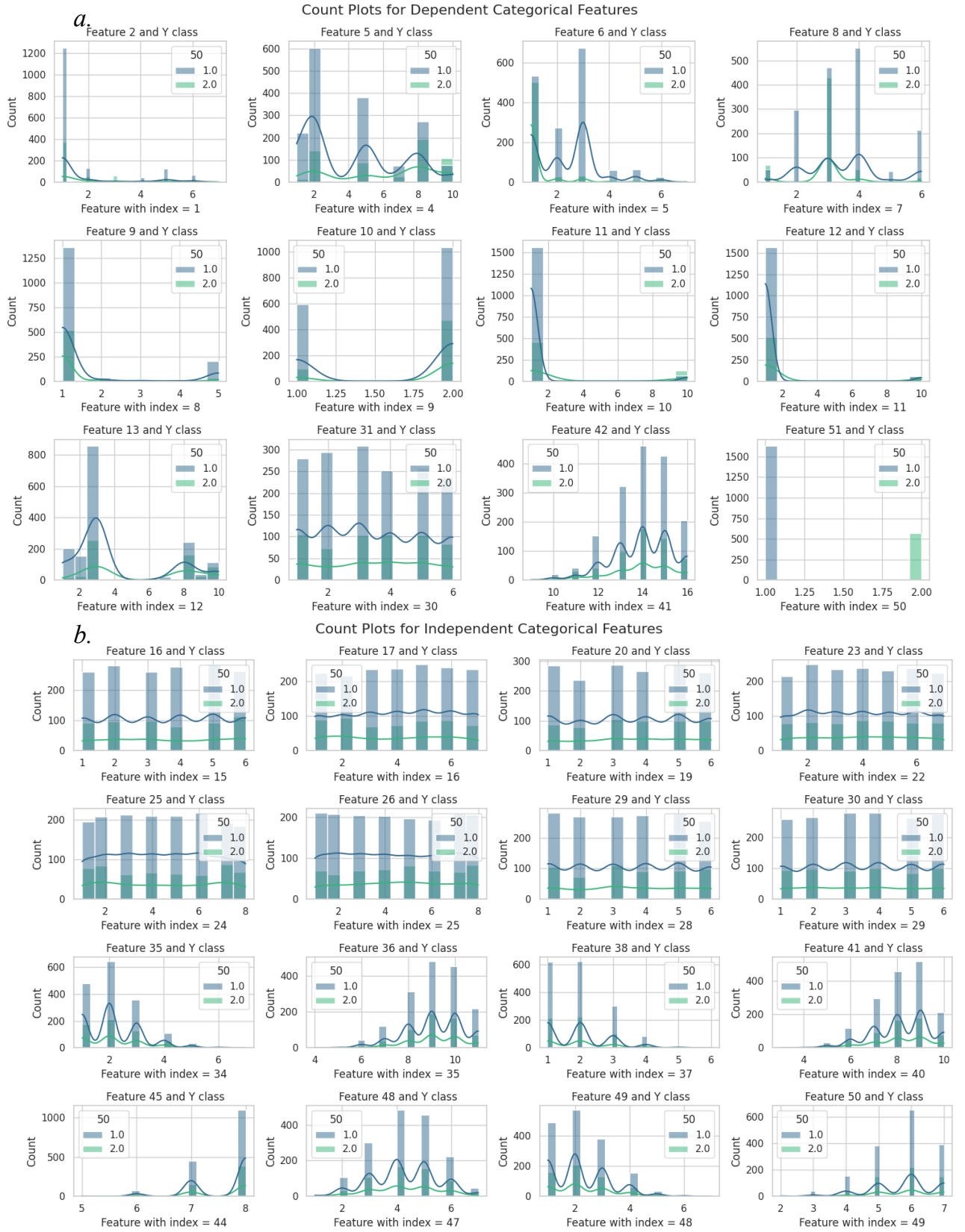


Figure 4: Representation of dependent (4a) and independent (4b) features, ($p\text{-value} < 0.05$) of Y class. Different colors correspond to samples of two different classes of Y . KDE lines underline the differences among the two distributions

Features that after performing chi square test with the target variable Y, resulted in a p-value lower than 0.05, were selected and considered the ones with the high dependence with class Y.

In the figures 4a and 4b we can see the distribution of each feature by representing with different colors the samples of the 2 different classes of variable Y. Figure 4a consists of the features that were considered dependent, while figure 4b consists of features considered independent. This visualization aims in the interpretation of features relationship with variable Y, as it is clear that dependent features (*figure 4*) exhibit a different distribution among the two classes (Y=1, Y=2), while independent ones (*figure 4a*) exhibit a more similar (almost identical) distribution between the two classes. The added smooth lines - kernel density estimation - (kde = True) in the first plot (dependent features) show that the feature patterns vary between the two classes (Y=1, Y=2), suggesting a connection with the Y variable. In the second plot (independent features), the lines are very similar, almost like parallel lines. This strong similarity indicates that these features don't really differ much between the two classes. So, the smooth lines help us see the differences in feature patterns, showing that some features are linked to Y (dependent), while others don't seem to have a strong connection (independent).

Missing values:

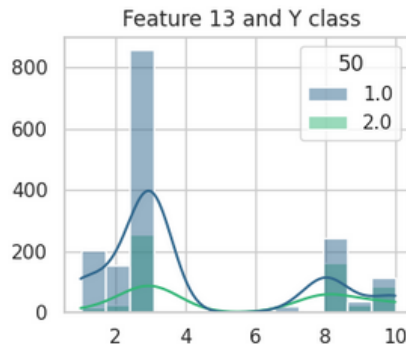
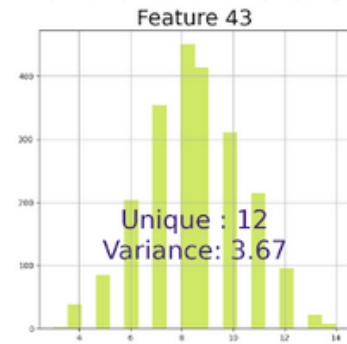
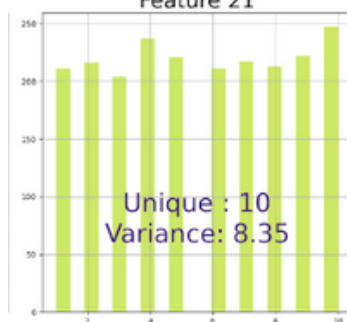
Missing values do not seem to appear in the given dataset "Dataset6.A_XY.csv"

```
print(missing_values = df.isna().sum())
```

Interesting features selection

Based on the previous analysis, and after considering criteria such as the dependence relationship between some features and the Y target class, the variances, the number of unique values and the nature of the data (continuous - categorical) of some variables we selected the following features:

Feature	Histogram	Statistics												
5	<p>Feature 5 and Y class</p>	<p>Data type: Categorical Unique values: 6 Class Distribution (%):</p> <table><tr><td>2.0</td><td>33.879036</td></tr><tr><td>5.0</td><td>21.327876</td></tr><tr><td>8.0</td><td>21.191451</td></tr><tr><td>1.0</td><td>10.777626</td></tr><tr><td>10.0</td><td>8.412915</td></tr><tr><td>7.0</td><td>4.411096</td></tr></table> <p>Comments: Distribution of classes shows variability, interesting in capturing class-dependent patterns, dependence from Y target variable (chi - square test : p-value <0.05). Significant differences among kdes of different classes.</p>	2.0	33.879036	5.0	21.327876	8.0	21.191451	1.0	10.777626	10.0	8.412915	7.0	4.411096
2.0	33.879036													
5.0	21.327876													
8.0	21.191451													
1.0	10.777626													
10.0	8.412915													
7.0	4.411096													
8	<p>Feature 8 and Y class</p>	<p>Data type: Categorical Unique values: 6 Class Distribution (%) :</p> <table><tr><td>3.0</td><td>40.836744</td></tr><tr><td>4.0</td><td>27.421555</td></tr><tr><td>2.0</td><td>13.506139</td></tr><tr><td>6.0</td><td>10.459300</td></tr><tr><td>1.0</td><td>5.547976</td></tr><tr><td>5.0</td><td>2.228286</td></tr></table> <p>Comments: Selected for the same reason as feature 5, class 1 distribution is very different from class 2 (e.g. feature's class 4 is the most frequent one Y=1 but at the same time the less frequent for Y=2)</p>	3.0	40.836744	4.0	27.421555	2.0	13.506139	6.0	10.459300	1.0	5.547976	5.0	2.228286
3.0	40.836744													
4.0	27.421555													
2.0	13.506139													
6.0	10.459300													
1.0	5.547976													
5.0	2.228286													

13	 <p>Feature 13 and Y class</p>	<p>Data type: Categorical</p> <p>Unique values: 7</p> <p>Class distribution (%):</p> <table><tr><td>3.0</td><td>50.477490</td></tr><tr><td>8.0</td><td>18.371987</td></tr><tr><td>1.0</td><td>9.959072</td></tr><tr><td>10.0</td><td>8.958618</td></tr><tr><td>2.0</td><td>8.140064</td></tr><tr><td>9.0</td><td>2.864939</td></tr><tr><td>7.0</td><td>1.227831</td></tr></table> <p>Comments :</p> <p>High variability among classes. Obvious differences between kdes of Y=1 and Y=0. Dependence from target variable Y might result in capturing distinct class patterns.</p>	3.0	50.477490	8.0	18.371987	1.0	9.959072	10.0	8.958618	2.0	8.140064	9.0	2.864939	7.0	1.227831
3.0	50.477490															
8.0	18.371987															
1.0	9.959072															
10.0	8.958618															
2.0	8.140064															
9.0	2.864939															
7.0	1.227831															
43	 <p>Feature 43</p> <p>Unique : 12 Variance: 3.67</p>	<p>Data type: Continuous</p> <p>Unique values: 12</p> <p>Mean: 8.47</p> <p>Standard deviation: 1.92</p> <p>Variance : 3.67</p> <p>Comments:</p> <p>Exhibits a normal distribution pattern based on mean, standard deviation, and variance. Moderate number of unique values. Might contribute to a good predicting accuracy (for models that consider normal distributions)</p>														
21	 <p>Feature 21</p> <p>Unique : 10 Variance: 8.35</p>	<p>Data type: Continuous</p> <p>Unique values: 10</p> <p>Mean: 5.58</p> <p>Standard deviation: 2.89</p> <p>Variance : 8.35</p> <p>Comments:</p> <p>Uniform shaped distribution - less sensitive to extreme values/outliers, making,more robust, might provide a different approach to the application of the model</p>														

In summary, we selected three categorical features that are highly dependent on the target variable Y to capture distinct class patterns. Adding a continuous feature with a normal distribution provides smooth information, while another with a uniform distribution adds diversity and robustness. This feature mix aims to boost the model for effective pattern recognition and more accurate predictions.

PART 2 - Model Selection

In this exercise you will implement a learning method from start to finish. You will train multiple classifiers with different hyper-parameter values (aka configurations), select the best model and report its performance. ¹

Classifiers	Param 1	Param 2
NBC	var_smoothing (alpha)	-
LR	L1/L2/No Penalty	C (Regularization Parameter)
SVM	L1/L2 Penalty	C (Regularization Parameter)
KNN	K	Uniform / Weighted Average
DT	max_depth	min_samples_split
RF	Number of Trees	criterion

Table 1: Available classifiers and hyper-parameters.

Before you begin, from Table 1 select 2-3 classifiers and use 2-5 different values for each hyper-parameter to construct your candidate configurations set using all the different hyper-parameter value combinations for each classifier.

Preprocessing

For this model selection analysis we will take into account the data types that were specified in the context of the previous question (data exploration) and will define a list for categorical indices and a list for continuous indices in the original dataframe.

The next data preprocessing step was to convert categorical data to one hot encoded data as we will further implement certain classifiers (such as logistic regression, knn) that handle numerical inputs. This technique transforms each categorical value into a new binary column - a format that can be efficiently processed by algorithms that require numerical input (such as in this case). We also converted the categorical variable Y to a binary format of 0, 1 (from 1, 2).

Define model configuration - data structure

In order to proceed in the model selection part of our analysis, it is crucial to choose diverse configurations of several classifiers. Herein we decided to proceed with the following machine learning algorithms :

- Logistic Regression
- KNN
- Random forest Classifier

We choose logistic regression since it seems that here we have to handle a binary classification task, knn to uncover more local patterns and random forest to try capturing non linear relationships and avoid overfitting.

```
configurations = [  
    {'classifier': LogisticRegression, 'params': {'C': [0.01,  
0.1, 1, 10, 100], 'penalty': ['l2', None], 'max_iter': [1000]}},  
  
    {'classifier': KNeighborsClassifier, 'params':  
{'n_neighbors': [3, 5, 7], 'weights': ['uniform', 'distance']}},  
  
    {'classifier': RandomForestClassifier, 'params':  
{'n_estimators': [100, 200], 'max_depth': [5, 10, None]}}  
]
```

The chosen configurations for Logistic Regression, K-Nearest Neighbors, and Random Forest classifiers build a diverse range of hyperparameters, aiming to explore various aspects of each model's learning capability.

Logistic Regression:

- I. C: [0.01, 0.1, 1, 10, 100], this set of hyperparameters is used to cover a range of possible values for constant C, to control inversely (higher values of C → less regularization) the regularization strength of the model. We would expect lower values such as 0.01 to significantly constrain the model complexity, and avoid overfitting (also potentially leading to underfitting, where the model is generalizing a lot). On the other hand, higher values such as 100 make the model less strict and correlated to the training data, which might perform good in recognizing complex patterns in the dataset but at the same time overfit and result to a low performance on new unknown data.
- II. penalty : ['l2', None], 'l2' adds a penalty proportional to the square of the magnitude of coefficients (L2 regularization), which can prevent overfitting by discouraging large coefficients. None means that no regularization is

applied, which might be useful for testing how the model performs without this constraint.

- III. `max_iter`: we increased this parameter to 1000 to address the *ConvergenceWarning* indicating that the default iterations were insufficient for the solver to converge. This change allows the algorithm more time to effectively learn and find optimal solutions.

K-Nearest Neighbors:

- I. `n_neighbors`: [10, 47, 60], these values indicate how many nearest neighbors are considered in making predictions. Following a common heuristic approach we first decided to pick as k the sqrt of the sample size (~ 47) as a theoretical ideal value, and will also try different configurations for lower and higher k , to compare the performance. We expect for lower k values to be more sensitive to noise unlike higher values of k where the model is supposed to generalize more, while a more moderate value of k should balance these pros and cons.
- II. `weights`: ['uniform', 'distance'], 'Uniform' treats all neighbors equally, while 'distance' gives more weight to closer ones. This comparison will help us assess the importance of proximity in determining the model's accuracy.

Random Forest:

- I. `n_estimators`: [100, 200, 500], higher value for `n_estimator` hyperparameter will set a higher number of trees in the forest, which would typically lead to a better performance (with a computational power cost).
- II. `max_depth`: [5, 10, None], setting `max_depth` to lower values such as 5 or 10 will restrict the growth of the trees which will probably lead the model to fail recognizing complex patterns and relationships in the dataset (underfitting). Thus the model might perform relatively good to unseen data but at the same time will be too simplistic to capture complexities. On the other hand, high or None `max_depth`, might lead to more complex trees where the model will be able to learn detailed patterns but with the risk of overfitting.

create_folds:

The `create_folds` function sets up cross-validation by dividing the data into k parts, ensuring each part has a similar mix of target classes. This method tests the model on different data sections, providing a fair and consistent evaluation. The function we implemented, first sorts and groups all samples (rows) by the last column (Y target variable), calculates the occurrences for each class and stores the indices.

Finally shuffles the indices and evenly splits them into k groups - folds and shuffle each fold to achieve a mixture of each class in each fold. The function returns a list of k lists (one for each fold)

run_model:

This function trains the models specified by *configurations* list, by iterating over the specified hyperparameters, on the training dataset. The classifier is trained using the fit method on the training data. `train_data[0]` and `train_data[1]` represent the features (X_train) and the target variable (Y_train) of the training dataset, respectively. The `predict_proba` method is used, to give the probability estimates for each class. The `[:, 1]` part extracts the probabilities of the positive class. We need to use *predict_proba* method because the ROC-AUC score requires probability scores rather than binary predictions. The function finally calculates (based on the probability estimates of each class) the and returns the roc-auc score.

performance_estimation:

This function evaluates the performance of different configurations (combinations of classifiers and their hyperparameters) using cross-validation. It calculates the average performance for each fold and each configuration, calculates the roc-auc for each fold and the mean score is appended to the performances list. It finally returns a list of average ROC AUC scores for each configuration.

best_model:

Identifies the most effective model configuration and returns it based on the highest configuration score.

Run analysis:

After having set the number of desired fold to create (`create_fold` function), we need to define an “*expanded_configurations*” where each element in it will have a specific model and its corresponding hyperparameter settings. We then use this list in the “*performance_estimation*” function to systematically train and evaluate each model configuration across the different folds. This process involves training each model on the training set of each fold and assessing its performance on the test set using the ROC AUC score. By doing this for all configurations across all folds, we can identify which configuration yields the best average performance, effectively selecting the best model and hyperparameters for our data. Finally we trained the model based on the best configuration by fitting it to the entire original dataset.

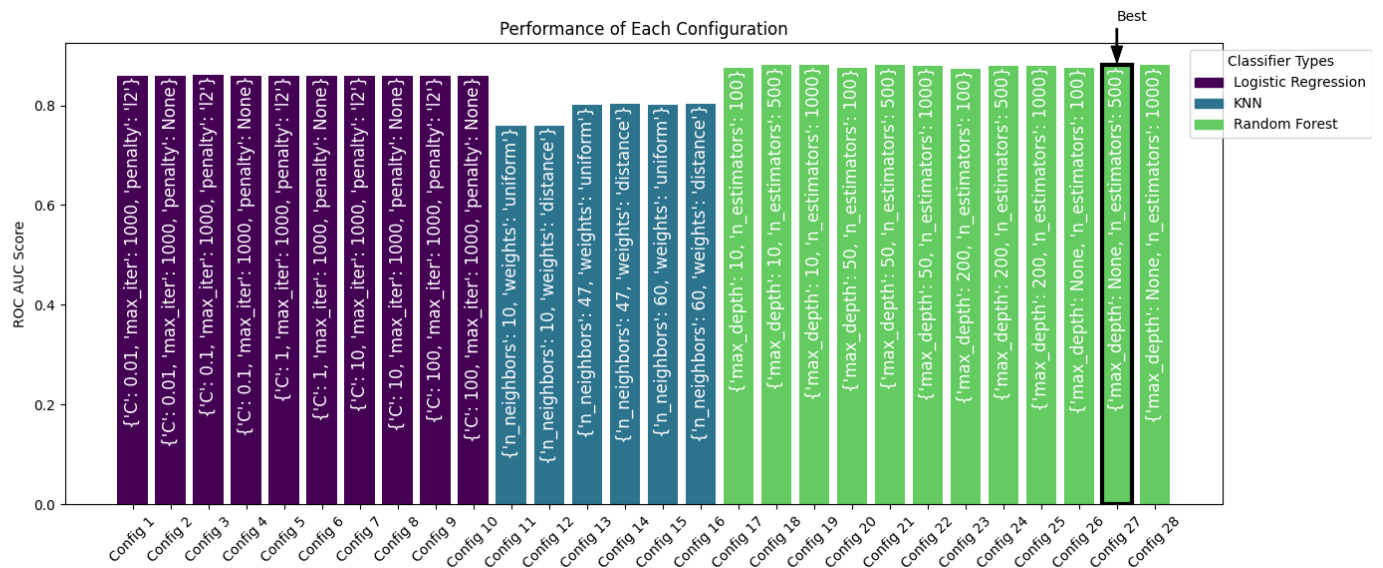
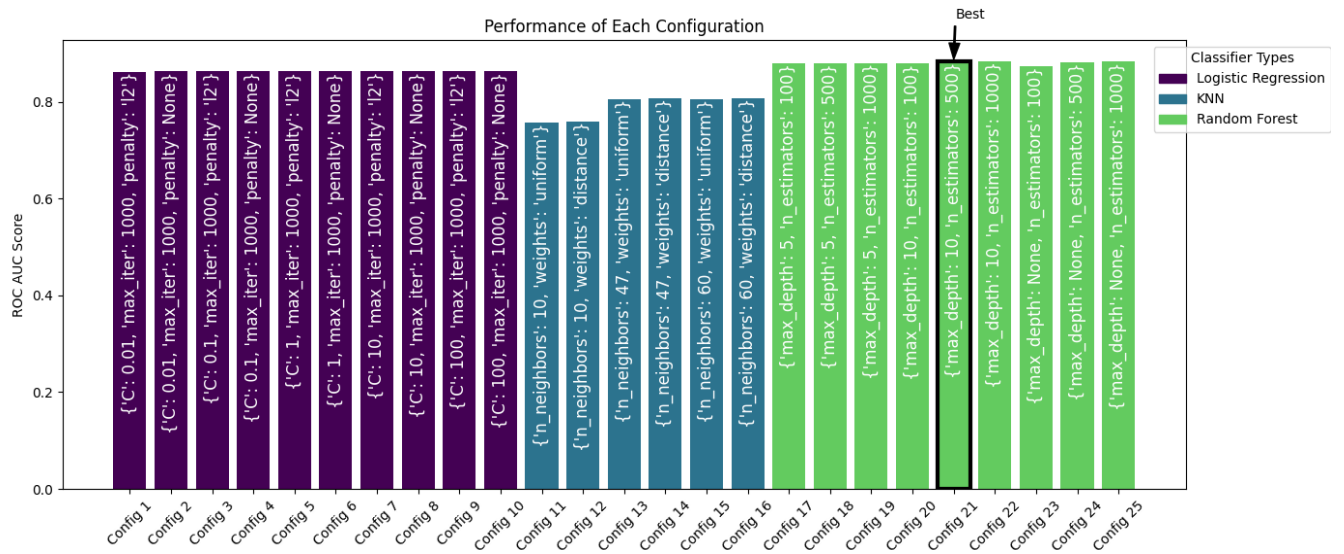
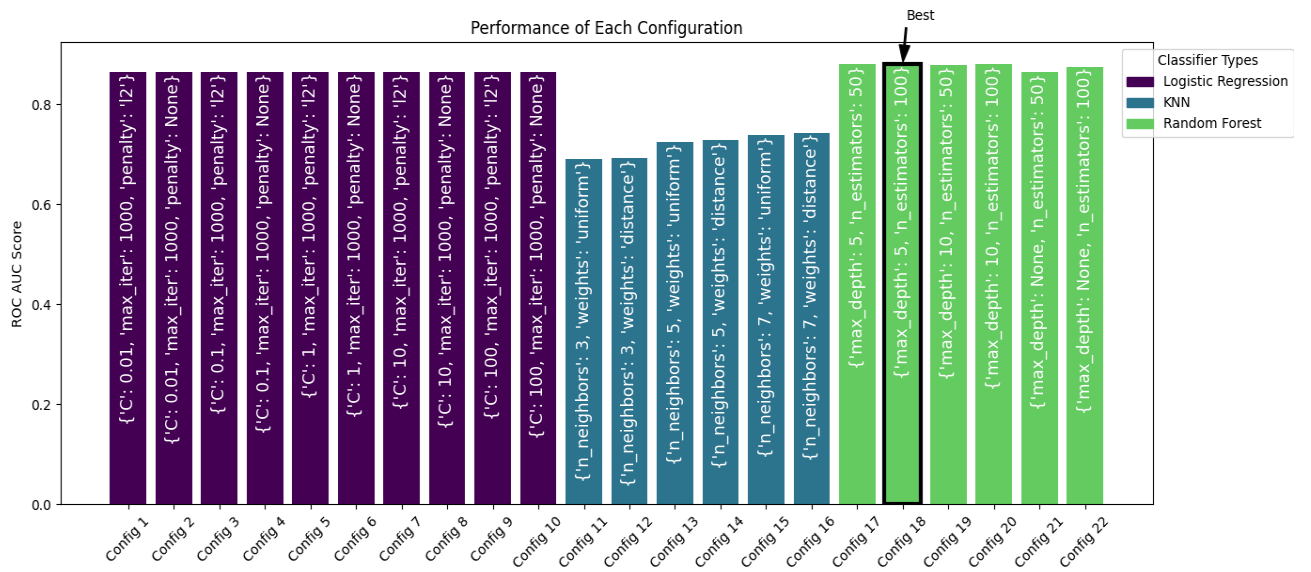


Figure 5. Barplot shows the performance of each configuration according to the ROC-AUC score. Bars with purple color correspond to Logistic Regression classifier; bars with blue color correspond to KNN classifier and bars with green color correspond to Random Forest classifier. In the body of each bar are shown the hyperparameter combinations. The best model configuration is highlighted with a black outline

PART 3

Computing the out-of-sample performance

Building on the previous exercise, let's now suppose you had previously held-out a test set (in the file: Dataset6.B_XY.csv). Use this set to assess the out-of-sample performance of your model.

Fit best model to new dataset

Since the test dataset has the same format as the previous one (dataset in file "Dataset6.A_XY.csv"), we will again prepare the features data and the target label data in to variables `X_test`, `Y_test`.

Then , we will use the final model to predict the class probabilities of the test set (ignoring the Y class - last column)

```
y_pred_proba = final_model.predict_proba(X_test_transformed)[:, 1]
```

`y_test_proba` and `y_test` will be used to compute the roc auc score and the fpr tpr ratios.

Produce the ROC curve for the selected best model. Compare against the trivial (naive) classifier.

We chose as a naive classifier a model that will always predict the majority class. Thus we found the majority class of the training dataset, and then created the `y_pred_naive` variable that will store the prediction class of all instances. Finally the true positive and false positive ratios will be computed and auc value will be stored in the variable `roc_auc_naive`. We also "calculated" the roc auc score of a random classifier (0.5) for a better representation of our model's performance comparing to the diagonal (*Figure 6*). The graph shows our model works much better than guessing or just picking the most common class every time. With an area of 0.87 under its curve, it outperforms the naive classifier. In contrast, the naive classifier is represented as a point, which typically corresponds to the performance of always predicting the most frequent class (lack of any predictive capability). This visualization underscores the efficacy of the best model and its ability to accurately classify instances compared to baseline approaches that do not consider input features.

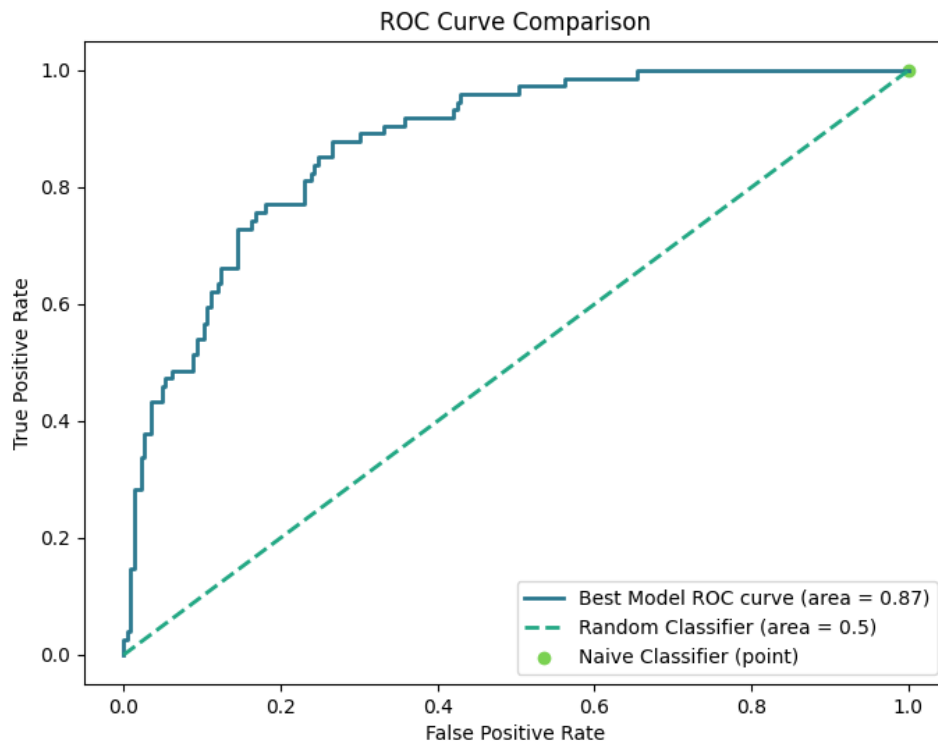


Figure 6: The best model's ROC curve (solid line) with an AUC of 0.87 shows good predictive accuracy, outperforming the random (dashed line, AUC = 0.5) and naive (dot) classifiers, which have no predictive value.

- Does the out-of-sample performance differ from the performance obtained by the cross validation? If yes, why?

The model achieved a slightly higher score in cross-validation (ROC AUC = 0.8837) than in testing on new data (ROC AUC = 0.876). This difference is something that we would expect, since models often perform a bit better on familiar training data compared to new, unseen data. The decrease in test performance might be due to small differences in the data patterns between the training and test data or the overfitting behavior of the model. However, the results are still quite close, showing that our model generally predicts well on new data.

```
cross validation performance = 0.8836909780463355
out of sample performance = 0.8760762975364746
cross validation - out of sample = 0.8760762975364746
```


PART 4

Calculating the 95% Confidence Intervals

When we obtain the predictive performance for a model, we are also interested in how accurate this performance actually is. To do that, we use a technique called **Confidence Intervals**. In this step, we will calculate the confidence intervals of the best model by bootstrapping the hold-out set. To do so, store the performance obtained at each boot-strapped sampling of the hold-out set, and then create a histogram of the results: you can now define the 0.025 and 0.975 quantiles to obtain the 95% confidence intervals.

In our analysis, we conducted bootstrapping on the hold-out test set to ascertain the 95% Confidence Intervals for the model's performance. For a 95% confidence interval, we are looking at the percentiles that cut off the lowest 2.5% and the highest 2.5% of your data. These are the 2.5th percentile and the 97.5th percentile, respectively. This approach enabled us to evaluate the model's robustness and the stability of its roc auc score across varied samples. Bootstrapping repeatedly resamples the existing dataset with replacement. This creates multiple new datasets, each resembling the original but with slight variations due to the random sampling process. The already trained model is then used to make predictions on these bootstrapped datasets. Since the model is already trained, there's no need for additional training with each new sample. By testing the model on various resampled datasets, we get a better idea of the model's ability to generalize in new, unseen data. The dashed lines representing the lower and upper bounds of the 95% Confidence Interval suggest that we can be 95% confident that the true roc auc score of the model lies between these two bounds.

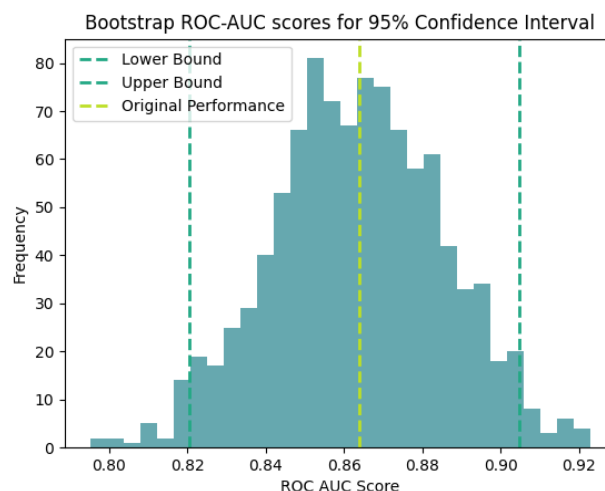


Figure 7. Histogram of ROC AUC scores from 1000 bootstrapped test samples with 95% Confidence Intervals, demonstrating the model's stable performance (Lower bound = 2.5 quartile, Upper bound = 97.5 quartile).

