Machine learning used to solve:

Telecon Customer Churn Prediction

1)Description

The task of this project is to build predictive models that can accurately predict whether a Telecon customer will churn or not based on historical customer data.

Customer churn is a critical issue for telecommunication companies. Losing customers to churn can lead to significant revenue loss and impact the company's growth. Therefore, it is essential to predict potential churners accurately and take proactive measures to retain them. In this project I can work with historical customer data, and my goal is to develop machine learning models that can predict customer churn effectively.

The dataset consists of various customer attributes, such as customer demographics, subscription details, service usage, and billing information. The target variable is the "Churn" column, which indicates whether the customer churned (1) or not (0).

Evaluation

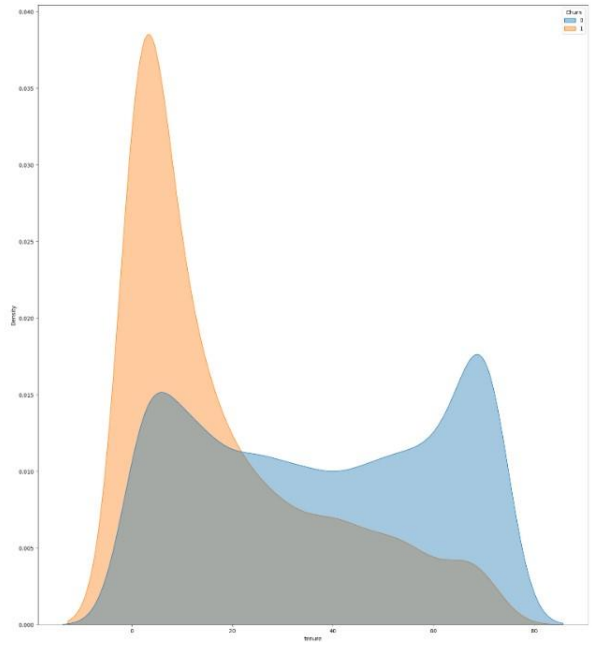
Model is been evaluated based on the Area Under Curve (AUC). The AUC is a commonly used metric to evaluate the performance of binary classification models, and it measures the trade-off between the true positive rate (sensitivity) and the false positive rate (1-specificity).

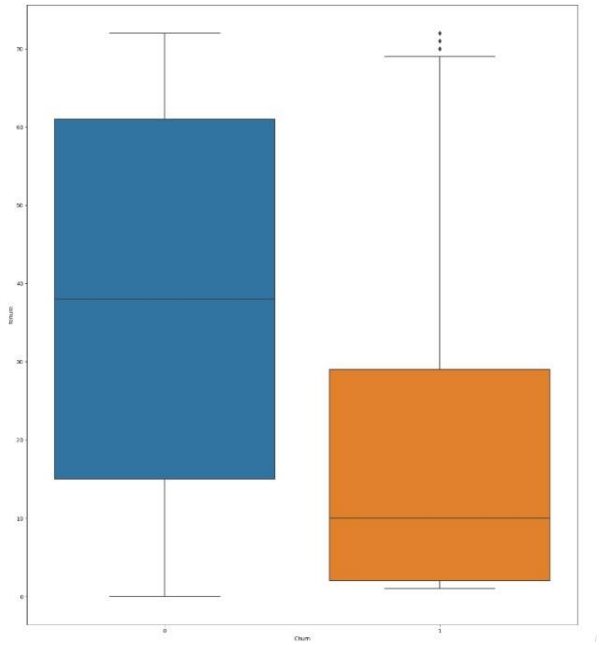
The AUC was the metric for which the models were trained, hence the metric to optimize. For each model, I present the results in terms of both AUC and accuracy

The main objective was to beat the benchmark of an online competition, held on kaggle, in which the threshold was set at 0.73 AUC on test data.

2)Results & Analysis

First of all, in the preprocessing phase, I tried to understand which features could be most separable with the output column (churn), obtaining that the most linearly separable property with the output was tenure, as shown below and in the script.





Training phase:

I tried various types of models.

The first was the Random Forest Classifier.

Initially the parameters used were:

```
# Random Forest classifier
classifier_random_forest = RandomForestClassifier(
    n_estimators=80,
    random_state=17,
    criterion='gini',
    max_features='log2',
    max_depth=5,
)
```

With these results:

```
Training Accuracy: 0.80877504438245
Test Accuracy: 0.7983441750443524
Training AUC: 0.6997434110700077
Test AUC: 0.6979345022648952
```

After a long search for parameter remodeling, both manual and automated, I found that the best performance in terms o f ROC-AUC for the test data was obtained from the following model:

[25]:

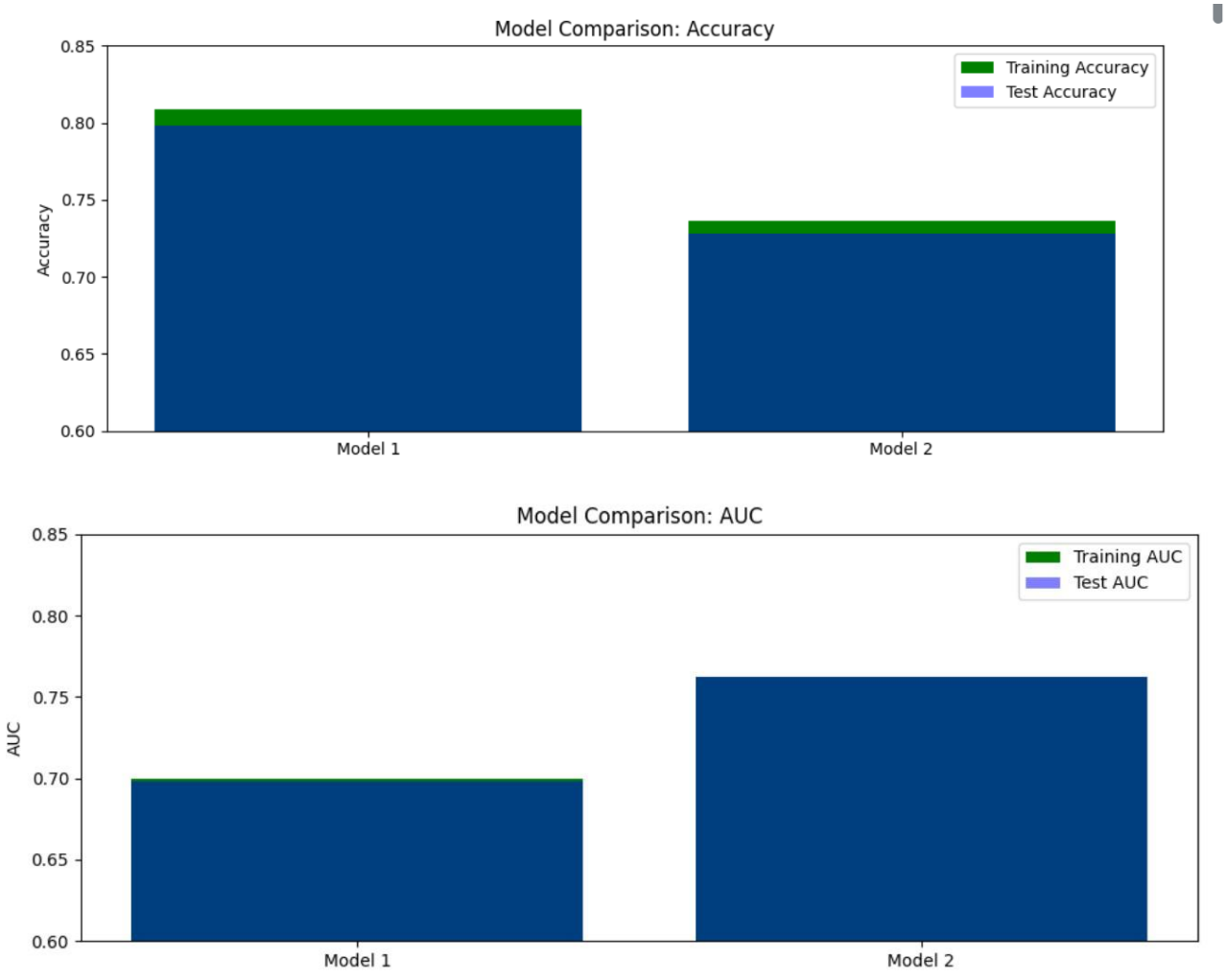
```
# Random Forest classifier
classifier_random_forest = RandomForestClassifier(
    n_estimators=106,
    random_state=17,
    criterion='gini',
    max_features='log2',
    max_depth=12,
    #new params
    class_weight='balanced',
    oob_score=True,
    min_impurity_decrease=0.012
)
```

That leds to the following result:

```
Training Accuracy: 0.7364950545270099
Test Accuracy: 0.7279716144293318
```

```
Training AUC: 0.7625228163127259
Test AUC: 0.7626203789996894
```

Graph that shows the difference between the 2 setups:



It is clearly visible that the first setup has better performance in terms of accuracy, while the second maximizes the ROC curve.

Both of the setups shows little gaps between performace on test and training datasets, highlighting signs of possible underfitting.

The second classifier used was the Decision Tree Classifier, with the following parameters:

```
max_depth=7, min_samples_split=3, min_samples_leaf=3, max_features='log2'
```

And it led to these results:

```
Training Accuracy: 0.8113111843773776
Test Accuracy: 0.7717327025428741
Training AUC: 0.7405395375279816
Test AUC: 0.7110732424485433
```

I then moved on to the K-Nearest Neighbors classifier and maximum result I had with this classifier was:

```
Training Accuracy: 0.8021810803956378
Test Accuracy: 0.7882909520993495
Training AUC: 0.73431547113794
Test AUC: 0.7324524805479096
```

Then, I Tried with the Perceptron Classifier, but from the beginning the results left no room for optimizations via parameter tuning.

These are the strating results:

```
Training Accuracy: 0.7476540705046919
Test Accuracy: 0.7321111768184506
Training AUC: 0.6535340445048053
Test AUC: 0.6385234472644256
```

With an initial AUC-ROC score of less than 0.64, there was no point in trying to opti mize this model.

Then I tried the Logistic Regression Classifier, with different versions of parameters, and this is the result I have obtained :

```
Training Accuracy: 0.8105503423788993
Test Accuracy: 0.7953873447664104
Training AUC: 0.7269984589530092
Test AUC: 0.7198600966644271
```

In the end, as the last model I tried gradient boosting classifi er.
This gave me the best results in terms of AUC on test data t ogether with the random forest classifier.
The initial results of the model were:

```
Training Accuracy: 0.8110726203348112
Test Accuracy: 0.77123211272283
Training AUC: 0.71112821998567
Test AUC: 0.7182213338294
```

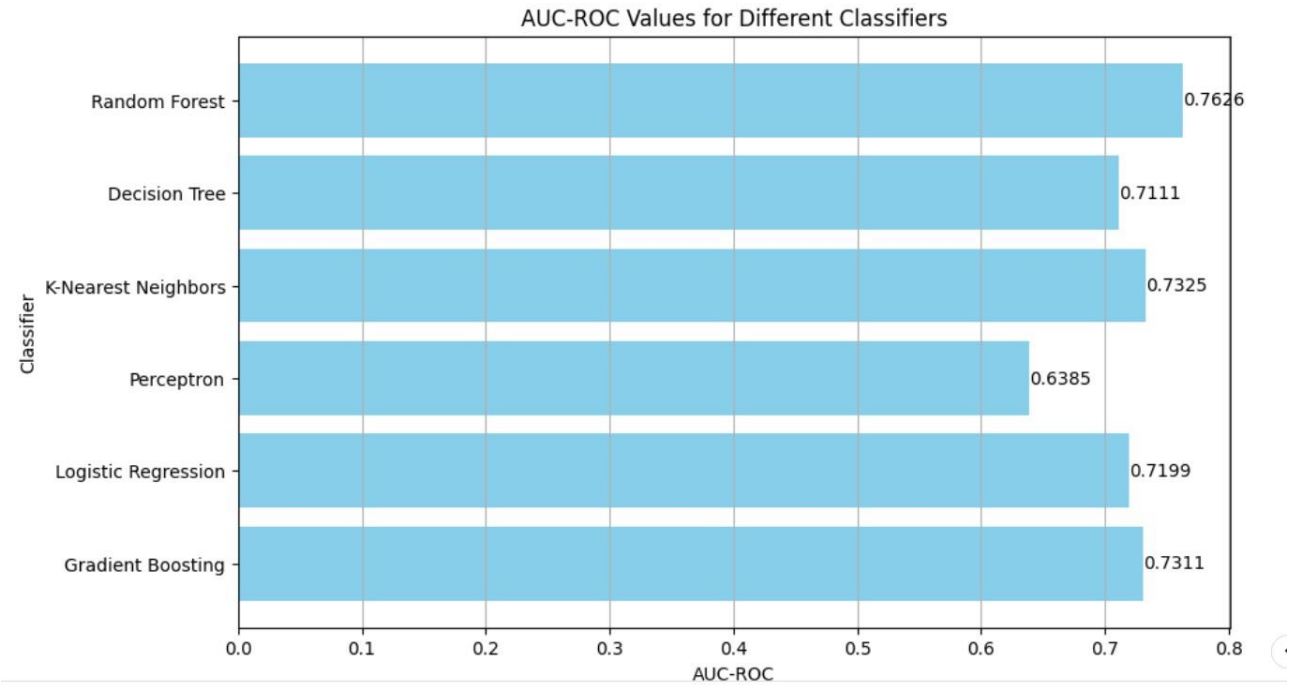
After manually tuning the parameters, I got the best perform ance
with the following setup:

```
    n_estimators=77,
    learning_rate=0.1,
    max_depth=3,
    min_samples_split=2,
    min_samples_leaf=1,
    subsample=1,
    max_features=None,
    loss='log_loss',
    criterion='friedman_mse',
    warm_start=False,
    validation_fraction=0.95,
    n_iter_no_change=None
```

That gaves me this outcome:

```
Training Accuracy: 0.8300786203398428
Test Accuracy: 0.8013010053222945
Training AUC: 0.7506078318350567
Test AUC: 0.7311213579256884
```

Below is the graph comparing all the models on the AUC cur ve for
the test dataset:



After obtaining these results, I chose to proceed with a grid search on two specific models: The RandomForestClassifier and the GradientBoostDescent.

The random forest because it had the most important result in terms of AUC, while the Gradient Boost Classifier because it was the most sensitive to parameter changes.

The results obtained with the grid search applied to the random forest classifier are the following:

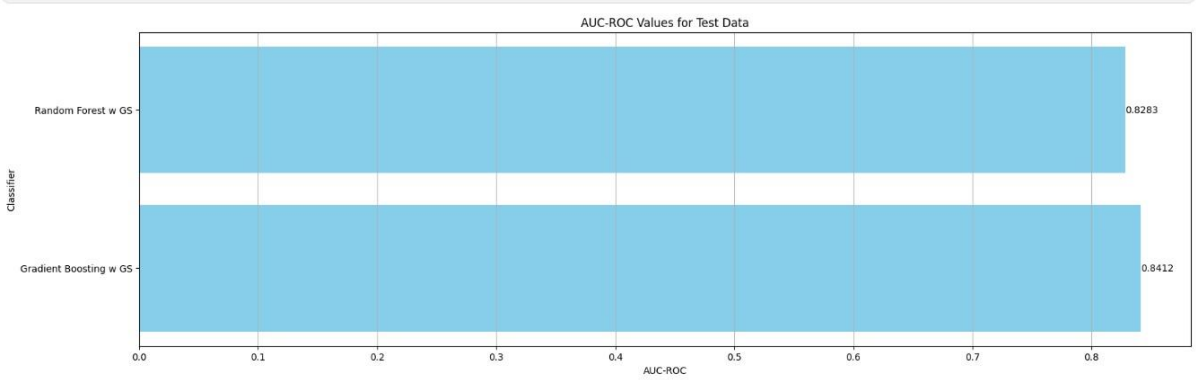
```
Best Parameters:
{'criterion': 'entropy',
 'max_depth': 12,
 'min_impurity_decrease': 0.012,
 'n_estimators': 91}

Test Accuracy: 0.7936132465996452
Test AUC-ROC: 0.8282716357094866
```

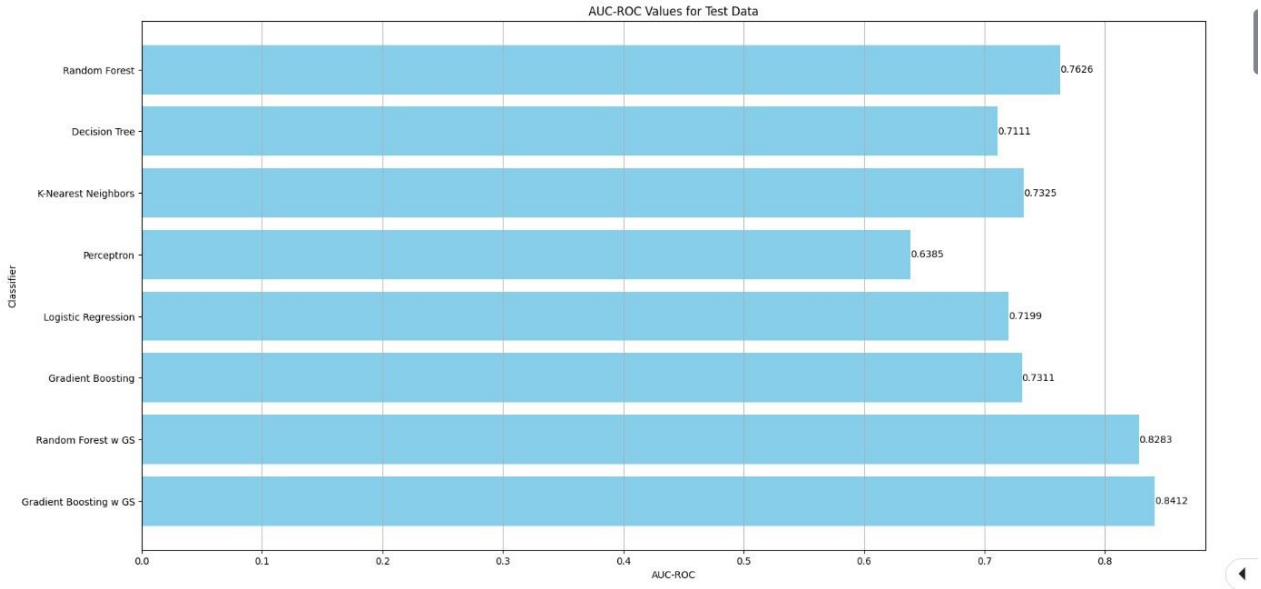
And the results obtained with the grid search applied to the gradient boosting classifier are the following:

```
Best Parameters:{'learning_rate': 0.1,'max_depth': 3,'n_estimators': 76}
Test Accuracy: 0.8018923713778829
Test AUC-ROC: 0.8412415021276288
```

Graph showing the two classifiers obtained with grid search:



And to compare them again with all the models tested:



This shows a great upgrade, and the models obtained with grid search far exceed the threshold set in this project.

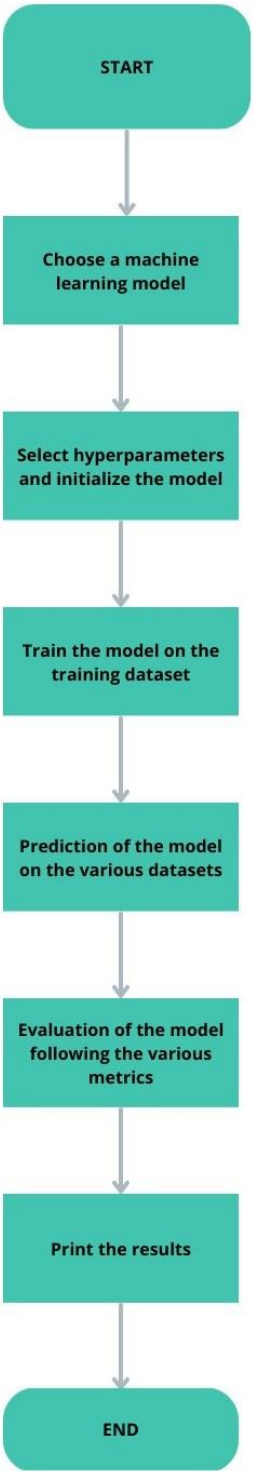
3)Flow Chart

First of all, for the creation of the flowchart, a general scheme can be created which brings together all the possible creations and implementations of the models tested in this project.

The models all follow a sequential construction and make use of very renowned libraries in the ML field.

However, it is possible to reconstruct the sequentiality of both the normal classifier and the grid search steps applied on the classifier.

The following represents the construction and execution of a general classifier:



Choose the classifier, for instance here is the Perceptron. There is also the **selection of the initial hyperparameters** (in this case max_iter and radom_state.

```
#Perceptron Classifier
classifier_perceptron = Perceptron(max_iter=1000, random_state=42)
classifier_perceptron.fit(X_train_scaled, Y_train)
```

Training of the model on the different datasets:

```
Y_pred_train = classifier_perceptron.predict(X_train_scaled)
Y_pred_test = classifier_perceptron.predict(X_test_scaled)
Pred_Perceptron = classifier_perceptron.predict(X_final_test_scaled)
```

Evaluation of the model, following the metrics chosen (with the **printing phase** included):

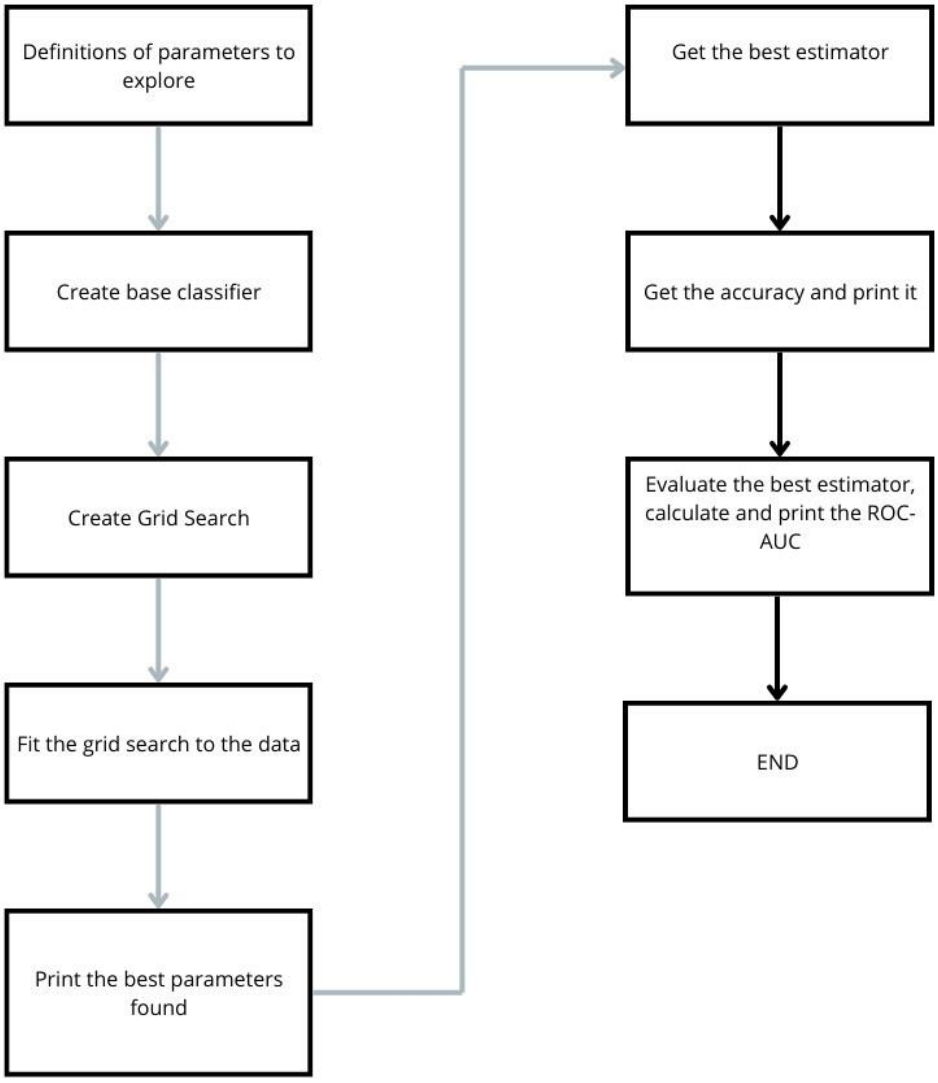
```
train_accuracy = accuracy_score(Y_train, Y_pred_train)
print("Training Accuracy:", train_accuracy)

test_accuracy = accuracy_score(Y_test, Y_pred_test)
print("Test Accuracy:", test_accuracy)

# Calculate AUC for the training set
train_auc = roc_auc_score(Y_train, Y_pred_train)
print("Training AUC:", train_auc)

# Calculate AUC for the test set
test_auc = roc_auc_score(Y_test, Y_pred_test)
print("Test AUC:", test_auc)
```

By applying grid search the scheme can be slightly modified as follows:



Definitions of parameters to explore:

```
gb_hyperParams = {
    'n_estimators': range(74, 82),
    'learning_rate': [ 0.09, 0.1, 0.11, 0.12],
    'max_depth': range(3, 6)
}
```

Create Gradient Boosting Classifier, the base classifier

```
gb_classifier = GradientBoostingClassifier(
```

```
    loss='log_loss',           # 'deviance' is used for binary classification
    criterion='friedman_mse',  # Quality of split function
    subsample=1,               # Use all samples for fitting individual trees
    max_features=None,         # Number of features to consider when looking for
the best split
    min_samples_split=2,       # Minimum number of samples required to split an
internal node
    min_samples_leaf=1,        # Minimum number of samples required to be at a leaf
node
    warm_start=False,          # Whether to reuse solution of the previous call to
fit and add more estimators
    validation_fraction=0.95,  # Fraction of training data to set aside as
validation set for early stopping
    n_iter_no_change=None     # Number of iterations with no improvement before
stopping fitting
)
```

Create Grid Search istance

```
grid_search = GridSearchCV(estimator=gb_classifier, param_grid=gb_hyperParams,
cv=5, scoring='roc_auc')
```

Fit the grid search to the data

```
grid_search.fit(X_train_scaled, Y_train)
```

Print the best parameters found

```
print("Best Parameters:", grid_search.best_params_)
```

Get the best estimator

```
best_classifier = grid_search.best_estimator_
```

Get the accuracy and print it

```
test_accuracy = best_classifier.score(X_test_scaled, Y_test)
print("Test Accuracy:", test_accuracy)
```

Evaluate the best estimator, calculate and print the ROC-AUC

```
Y_pred_proba = best_classifier.predict_proba(X_test_scaled)
test_auc_roc = roc_auc_score(Y_test, Y_pred_proba[:, 1])
print("Test AUC-ROC:", test_auc_roc)
```

4)Code

Below I leave the link to the driver containing the notebook file, in which the data preprocessing part is also visible.

https://drive.google.com/file/d/18gFXnVbWtg30Jdd_dfqKV8TubD6RGIjh/view?usp=drive_link