Laura Calzoni 0001058438 Francesca Paradiso 0001037825

Optimization and Machine Learning M

Machine Learning Project

Prediction over the "Titanic" dataset

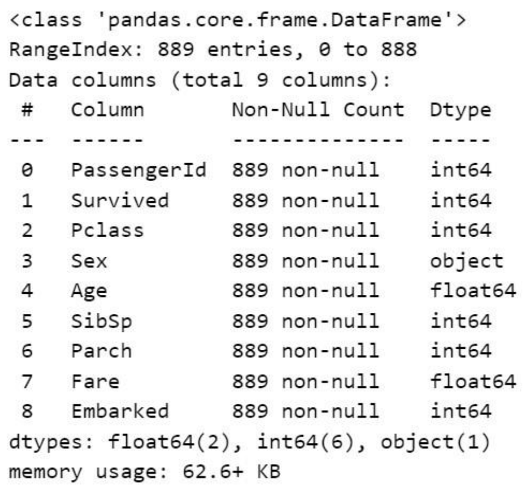
The aim of this project is to develop a machine learning model to predict the survival status of an individual after the Titanic shipwreck, based on relevant features such as sex, age and class, using the 'Titanic' dataset. It consists in a binary classification problem, in which data have to be split into two classes: survived and not survived. For the purpose, we chose to use Scikit-learn, an open-source machine learning library for Python language, that features various classification, regression and clustering algorithms.

1. Analysis of the Dataset

We were provided with 889 labelled data with 8 features plus the target variable ('Survived'):

- PassengerId: Unique identifier for each passenger
- Survived (label): Survival status of the passenger (0 = Not Survived, 1 = Survived)
- Pclass: Passenger class (1 = First class, 2 = Second class, 3 = Third class)
- Sex: Gender of the passenger
- Age: Age of the passenger
- SibSp: Number of siblings/spouses aboard the Titanic
- Parch: Number of parents/children aboard the Titanic
- Fare: Fare paid by the passenger
- Embarked: Port of embarkation

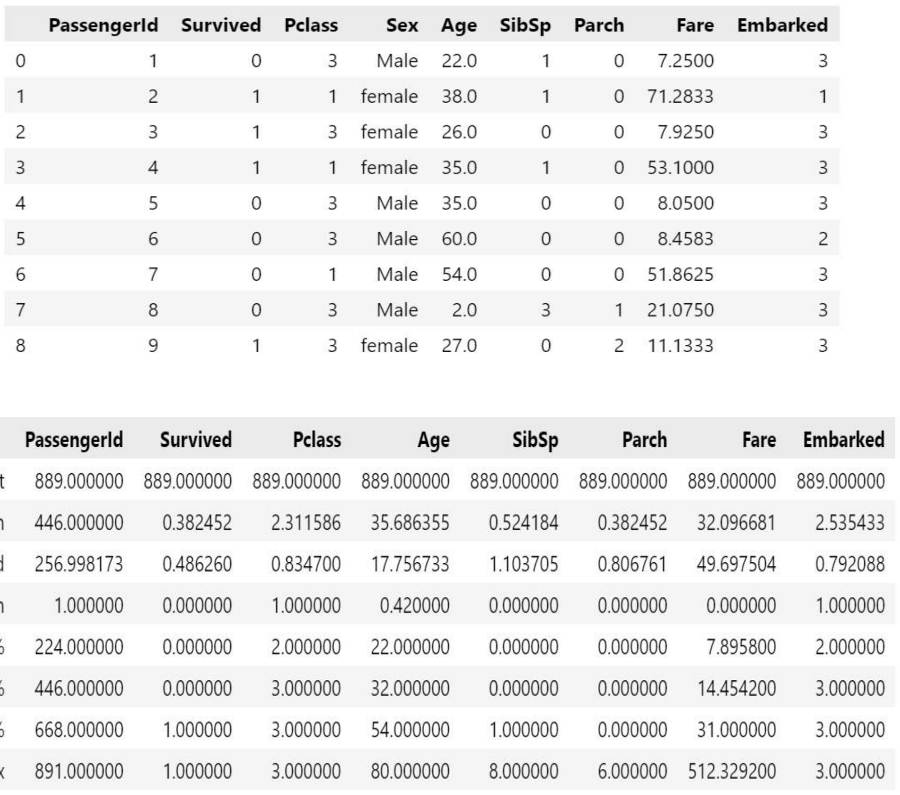
By plotting the dataset characteristics (Figure 1), it can be noticed that the data features are not homogenous and not all numerical, in particular two are floats and one is an object, while the remaining ones are integers. Thus, it will be of our concern to properly transform them later on, so that they can be processed by a learning algorithm.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 889 entries, 0 to 888
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  889 non-null    int64  
 1   Survived     889 non-null    int64  
 2   Pclass       889 non-null    int64  
 3   Sex          889 non-null    object 
 4   Age          889 non-null    float64
 5   SibSp        889 non-null    int64  
 6   Parch        889 non-null    int64  
 7   Fare         889 non-null    float64
 8   Embarked     889 non-null    int64  
dtypes: float64(2), int64(6), object(1)
memory usage: 62.6+ KB
```

Figure 1 - Data characteristics

Looking more into details, we can see that just 38% out of the dataset survived the Titanic. The passengers ages range from 0.4 to 80 years and most of them belonged to the third class. The statistics are reported in Figure 2, where at the top it is displayed the description of the first nine data in the set, as an example.



	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	Male	22.0	1	0	7.2500	3
1	2	1	1	female	38.0	1	0	71.2833	1
2	3	1	3	female	26.0	0	0	7.9250	3
3	4	1	1	female	35.0	1	0	53.1000	3
4	5	0	3	Male	35.0	0	0	8.0500	3
5	6	0	3	Male	60.0	0	0	8.4583	2
6	7	0	1	Male	54.0	0	0	51.8625	3
7	8	0	3	Male	2.0	3	1	21.0750	3
8	9	1	3	female	27.0	0	2	11.1333	3

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Embarked
count	889.000000	889.000000	889.000000	889.000000	889.000000	889.000000	889.000000	889.000000
mean	446.000000	0.382452	2.311586	35.686355	0.524184	0.382452	32.096681	2.535433
std	256.998173	0.486260	0.834700	17.756733	1.103705	0.806761	49.697504	0.792088
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000	1.000000
25%	224.000000	0.000000	2.000000	22.000000	0.000000	0.000000	7.895800	2.000000
50%	446.000000	0.000000	3.000000	32.000000	0.000000	0.000000	14.454200	3.000000
75%	668.000000	1.000000	3.000000	54.000000	1.000000	0.000000	31.000000	3.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200	3.000000

Figure 2 - Dataset description

According to us, the most relevant features that can potentially influence a passenger's survival are 'Age', 'Sex' and 'Pclass', as we believed that, in general, women, younger or richer people may have had more chances to survive. Below, it's visualized the distribution of such variables.

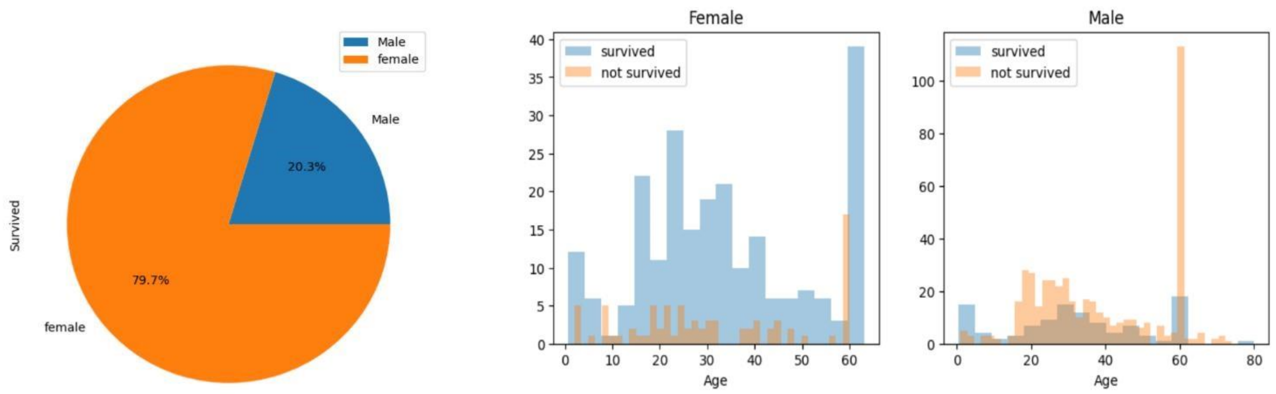


Figure 3 - Survivors according to their gender and age

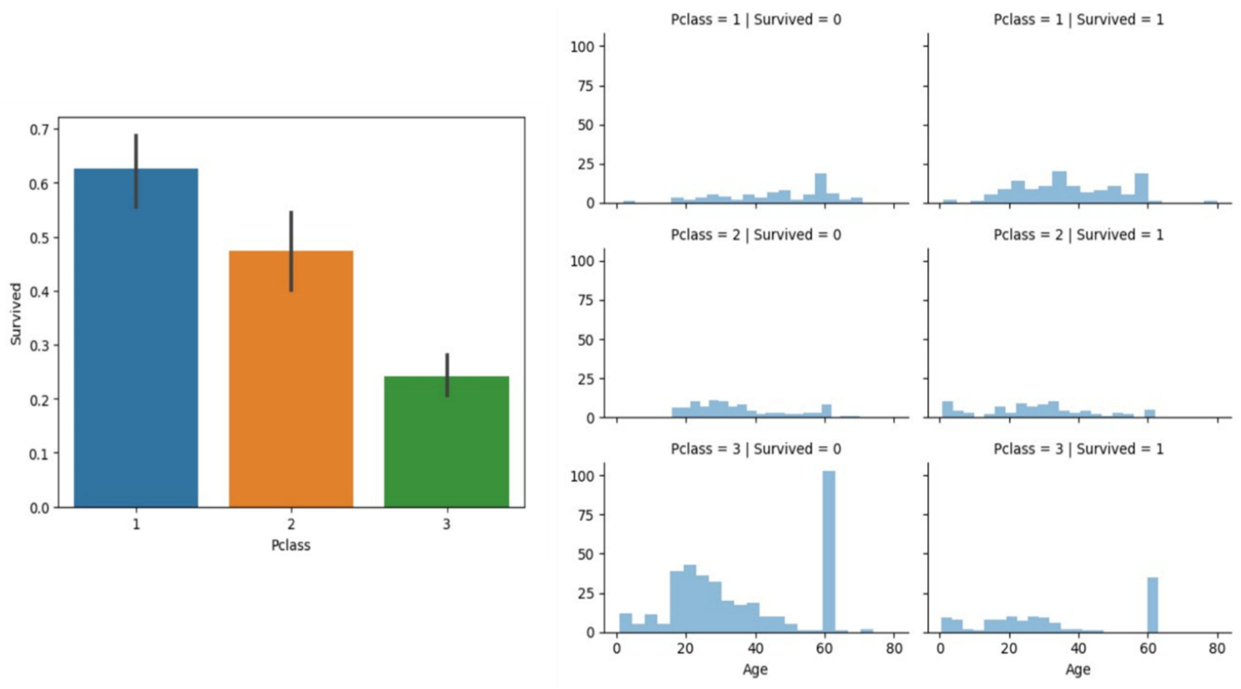
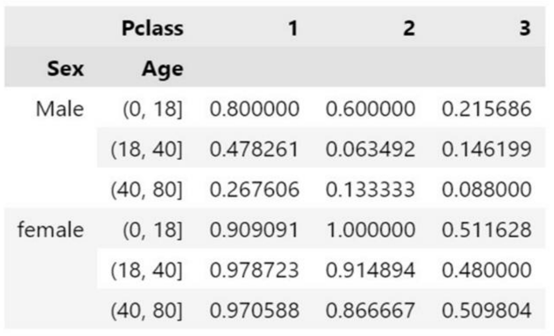


Figure 4 - Survivors according to their class and age



	Pclass	1	2	3
Sex	**Age**			
Male	(0, 18]	0.800000	0.600000	0.215686
	(18, 40]	0.478261	0.063492	0.146199
	(40, 80]	0.267606	0.133333	0.088000
female	(0, 18]	0.909091	1.000000	0.511628
	(18, 40]	0.978723	0.914894	0.480000
	(40, 80]	0.970588	0.866667	0.509804

Figure 5 - Survival rate by sex, age, and class

As expected, it can be appreciated that the highest number of deaths were within the middle-aged men of the last class.

Furthermore, we analysed the distribution of the ‘Fare’ feature and we observed how it is more likely that people who paid the ticket less (thus realistically poorer) would die.

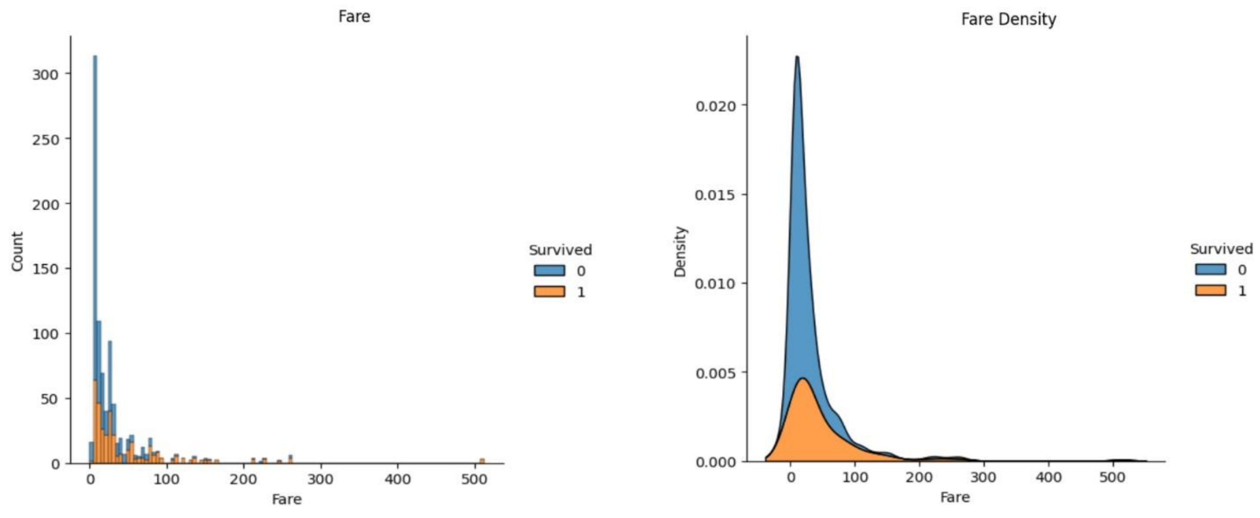
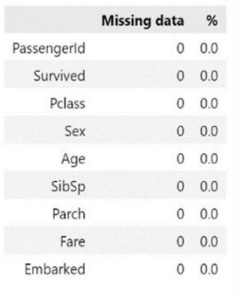


Figure 6 - 'Fare' distribution

2. Data Preprocessing and Feature Engineering

Before choosing an appropriate learning algorithm, it's necessary to preprocess and properly manipulate the data samples.
Firstly, we verified that no missing values are present in our dataset, as is shown in Figure 7. Then, we detected the presence of outliers by plotting the data through box plots, and we proceeded removing them by checking the z-score for the categories affected. The z-score is a statistical measure that informs on how far a point from the rest of the dataset is, in terms on how many standard deviations away a given observation is from the mean. Setting a threshold equal to 3, 15 outliers have been identified for 'Parch', 30 for 'SibSp' and 20 in 'Fare', for a total amount of 64 outliers in the whole dataset (one outlier is shared by more than one category).



	Missing data	%
PassengerId	0	0.0
Survived	0	0.0
Pclass	0	0.0
Sex	0	0.0
Age	0	0.0
SibSp	0	0.0
Parch	0	0.0
Fare	0	0.0
Embarked	0	0.0

Figure 7 - Missing data

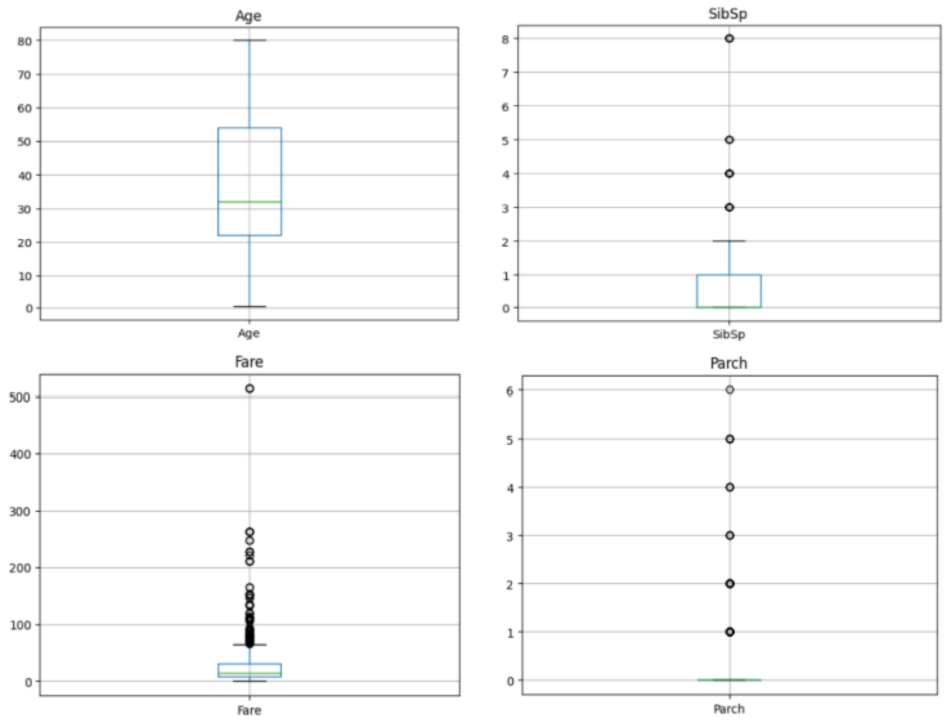
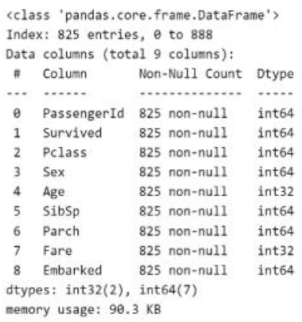


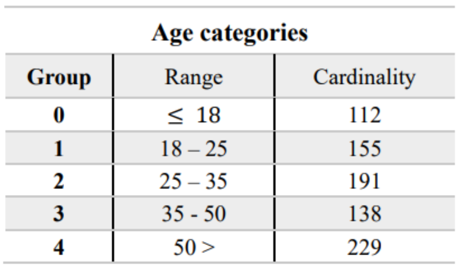
Figure 8 - Outliers

Next, we performed data preprocessing in order to rewrite them in a more suitable way for being elaborated by the learning algorithm. We proceeded by converting 'Fare' and 'Age' from float to int64, using the "astype()" function pandas provides. Secondly, we convert the feature 'Sex' (of type object) into numerical, by assigning the label 0 to the males and 1 to the females (Figure 9). Furthermore, since both the features 'Fare' and 'Age' have widely different ranges, we needed to convert them into roughly the same scale of others, creating categories.
It is important to place attention on how forming these groups, since it's not amenable to have unbalanced categories, of quite different cardinalities. Our results of this process are reported in the tables below:



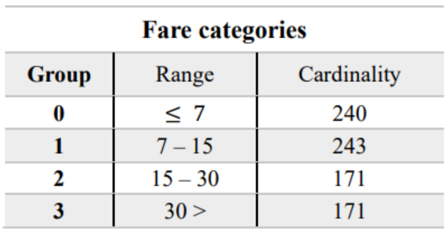
```
<class 'pandas.core.frame.DataFrame'>
Index: 825 entries, 0 to 888
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  825 non-null    int64
 1   Survived     825 non-null    int64
 2   Pclass       825 non-null    int64
 3   Sex          825 non-null    int64
 4   Age          825 non-null    int32
 5   SibSp        825 non-null    int64
 6   Parch        825 non-null    int64
 7   Fare         825 non-null    int32
 8   Embarked     825 non-null    int64
dtypes: int32(2), int64(7)
memory usage: 90.3 KB
```

Figure 9 - Data conversion



Age categories

Group	Range	Cardinality
0	≤ 18	112
1	18 – 25	155
2	25 – 35	191
3	35 - 50	138
4	50 >	229



Fare categories

Group	Range	Cardinality
0	≤ 7	240
1	7 – 15	243
2	15 – 30	171
3	30 >	171

Since the feature ‘PassengerId’ and ‘Embarked’ could not be strictly correlated with a person survival probability, we dropped them from the dataset, in order to avoid misleading in the training process.
Finally, as it is probable that, in a family group, only one member would pay for everyone, having then a higher ‘Fare’ “wrongly” associated, it can be of interest to create a new feature, called ‘Fare_per_person’, to assign the cost of the ticket to each component in a family. The relatives have been calculated as the sum of ‘SibSp’ and ‘Parch’ values.

3. Model development and evaluation

We divided our dataset into Training and Test set, respectively with the 80% and 20% of samples. To generate the data for training and testing, we duplicated the sets and removed the label from one of the two, while maintaining just the label variable in other. In this way, we formed two groups both for training and testing, one with unlabelled data and one with the reference tags. Then, we trained several machine learning models and compare their results. Specifically, we tried Random Forest, Decision Tree, Support Vector Machine and Stochastic Gradient Descent algorithms.

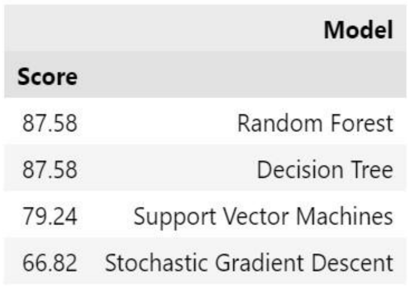
The Decision Tree learning algorithm is a machine learning technique that aims to construct a predictive model in the form of a tree-like structure. It emulates the decision-making process by breaking down complex problems into a series of simpler decisions based on input features. The algorithm identifies the most informative feature that best separates the data based on a certain criterion (such as Entropy or Gini impurity) and recursively repeats this process for each subset of data created by the splits, forming a tree structure. Once the decision tree is constructed, it can be used for both classification and regression tasks. For classification, the tree predicts the class label of a new sample by following the path from the root to a leaf node. However, Decision Tree can be prone to overfitting if not properly controlled, problem that can be addressed by adopting techniques like pruning or ensemble methods such as Random Forest.

The Random Forest algorithm is an ensemble learning technique that combines multiple decision trees to create a robust and accurate predictive model, improving the generalization of the model and mitigating the risk of overfitting at the same time. It builds a "forest" of decision trees, where each tree is trained on a random subset of the training data. The random sampling of data helps to introduce diversity in the trees and reduces the variance of the model. At each node, a subset of features is randomly selected as candidate for splitting, which helps in capturing different aspects of the data. The predictions of all the decision trees are aggregated and, for classification tasks, the class that receives the higher number of votes across the trees is selected as the final prediction (majority voting).

Support Vector Machine (SVM) aims to find the optimal hyperplane that best separates the data into different classes. During the training phase, SVM learns the optimal hyperplane by solving a quadratic optimization problem to maximize the margin between the separating surface and the nearest data points of each class, while minimizing the classification error. Once the best decision boundaries are determined, new samples can be

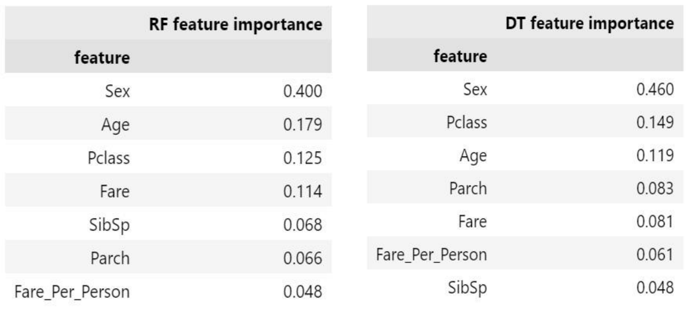
classified based on their position with respect to the hyperplane. SVM is robust against overfitting and is less affected by irrelevant features in the data.
The main objective of Stochastic Gradient Descent (SGD), instead, is to minimize a given loss function by iteratively adjusting the model parameters in the direction of steepest descent, until a stopping criterion is met. SGD randomly selects a mini-batch of samples at each iteration and computes the gradient based on that subset only. This characteristic introduces randomness into the optimization process, which can result in faster convergence and more efficient updates. In classification problems, the loss function should be a measure of the discrepancy between the predicted class probabilities and the true class labels in the mini-batch. SGD is more sensitive to the choice of hyperparameters, such as the learning rate and batch size, and it may also converge to a suboptimal solution due to the noise introduced by the random mini-batches.

Figure 10 reports the performances achieved in these four cases and it can be seen that the best results have been obtained with Decision Tree and Random Forest. Consequently, the subsequent analyses have been carried out using these two algorithms only. An accuracy of 87.58% is acceptable considering the small amount of data available.



	Model
Score	
87.58	Random Forest
87.58	Decision Tree
79.24	Support Vector Machines
66.82	Stochastic Gradient Descent

Figure 10 - Accuracy of the trained models



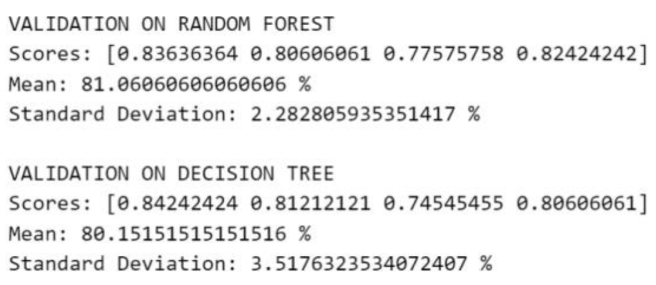
	RF feature importance
feature	
Sex	0.400
Age	0.179
Pclass	0.125
Fare	0.114
SibSp	0.068
Parch	0.066
Fare_Per_Person	0.048

	DT feature importance
feature	
Sex	0.460
Pclass	0.149
Age	0.119
Parch	0.083
Fare	0.081
Fare_Per_Person	0.061
SibSp	0.048

Figure 11 - Feature importance

After the learning algorithm selection, we checked the features importance for the training process. Sklearn measures a feature's importance by looking at how much the tree nodes that use that feature, reduce impurity on average. It computes this score automatically for each feature after the training phase, and it normalizes the results. As all the remaining features turned out to be quite important for the training of the model (Figure 11), we decided not to eliminate any other variable from the dataset.

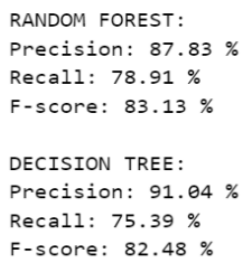
Then, we performed K-Fold Cross Validation on the trained models using 4 folds, in order to assess their performance and generalization abilities. Indeed, K-Fold Cross Validation helps estimate how well the model will perform on unseen data by simulating its performance on K-subsets, randomly partitioned from the training data. One fold is held out as a validation set while the model is trained on the remaining folds. This process is repeated until every subset acted once as an evaluation fold.
As it can be seen from Figure 12, our Random Forest model had an average accuracy of 81.06% with a standard deviation of 2.28%, while the model built with the Decision Tree algorithm model had an average accuracy of 80.16% with a standard deviation of 3.52%, which are still good results.



```
VALIDATION ON RANDOM FOREST
Scores: [0.83636364 0.80606061 0.77575758 0.82424242]
Mean: 81.06060606060606 %
Standard Deviation: 2.282805935351417 %

VALIDATION ON DECISION TREE
Scores: [0.84242424 0.81212121 0.74545455 0.80606061]
Mean: 80.15151515151516 %
Standard Deviation: 3.5176323534072407 %
```

Figure 12 - Results of K-Fold Cross Validation

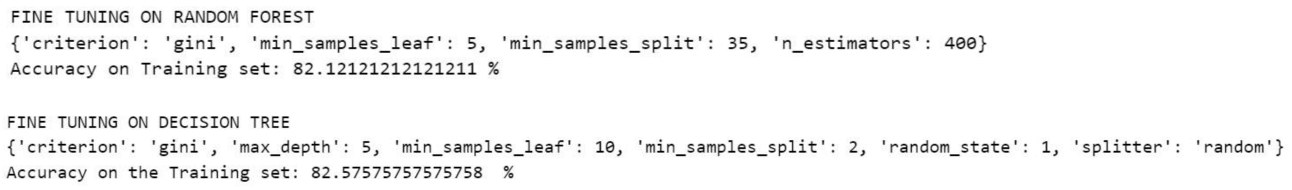
As last check, we also studied the Precision and Recall scores of our models.
Precision is the measure of the model's accuracy in correctly predicting positive instances. It is calculated as the ratio of true positives to the sum of all the positive predictions. A high precision value indicates that the model has a low rate of false positives. The Recall, instead, measures the model's ability to correctly identify positive cases from the actual positive instances. It is calculated as the ratio of true positives to the sum of true positives and false negatives, and it represents the proportion of actual positives that the model correctly identifies. Indeed, a high recall value indicates that the model has a low rate of false negatives. The combination of precision and recall is called F-score and provides a balanced measure of the model's ability to correctly identify positive data, avoiding both false positives and false negatives.



```
RANDOM FOREST:
Precision: 87.83 %
Recall: 78.91 %
F-score: 83.13 %

DECISION TREE:
Precision: 91.04 %
Recall: 75.39 %
F-score: 82.48 %
```

Figure 13 - Precision, Recall and F-score

Finally, we tried fine-tuning the models using the GridSearchCV tool of scikit-learn, adjusting their hyperparameters to improve performances. Respectively, we obtained an 82.12% and 82.57% of accuracy for Random Forest and Decision Tree (Figure 14).



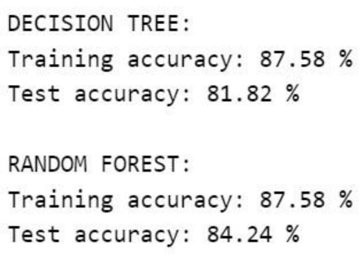
```
FINE TUNING ON RANDOM FOREST
{'criterion': 'gini', 'min_samples_leaf': 5, 'min_samples_split': 35, 'n_estimators': 400}
Accuracy on Training set: 82.12121212121211 %

FINE TUNING ON DECISION TREE
{'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 10, 'min_samples_split': 2, 'random_state': 1, 'splitter': 'random'}
Accuracy on the Training set: 82.57575757575758  %
```

Figure 14 - Fine-tuning results

The decreasing in the accuracy can found explanation in several factors, as, first of all, the interaction between hyperparameters. The performance of a model can be affected by possible interactions between different hyperparameters that, if not captured, could lead to suboptimal results. Additionally, if the dataset is small or noisy, as in this case, fine-tuning the model with GridSearchCV may not lead to significant improvements. In such cases, indeed, the model may not have enough information to generalize better, and hyperparameter tuning may not make a noticeable difference. For these reasons, we maintained the models previously trained.

4. Prediction

As final step, we made predictions on the testing data and evaluated the accuracy over the test set, which resulted pretty good, specifically **84.24%** and **81.82%** of accuracy for Random Forest and Decision Tree (Figure 15). Random Forest results to be the best choice, and we concluded by computing further evaluations on such model only.



```
DECISION TREE:
Training accuracy: 87.58 %
Test accuracy: 81.82 %

RANDOM FOREST:
Training accuracy: 87.58 %
Test accuracy: 84.24 %
```

Figure 15 - Accuracy of the prediction

In particular, we obtained a Precision of **86.27%,** a Recall of **69.84%** and a F-score of **77.19%** over the test set.

Moreover, we also calculated the Confusion Matrix over both the training and test sets for Random Forest.

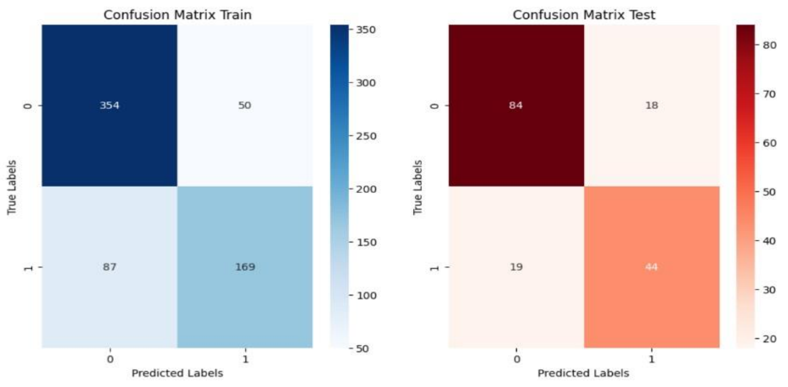


Figure 16 - Confusion matrices

5. Conclusions and recommendations

To conclude, the objective of the project was to analyse the 'Titanic' dataset in order to predict the survival rate of a passenger. We accomplished such task implementing the following steps:

- ❖ Check for missing data and elimination of outliers
- ❖ Data preprocessing, eliminating useless features ('PassengerId', 'Embarked'), creating new ones ('Fare_per_person'), building categories (for 'Age' and 'Fare') and transforming the data
- ❖ Separate the dataset over Train and Test sets
- ❖ Train the model using different learning algorithms and evaluate the results (the Random Forest leads to the best performances)
- ❖ Predict the survival rate over the test set and evaluate its accuracy

The final design has the following performances:

	Train	**Test**
Precision (%)	87.83	86.27
Recall (%)	78.91	69.84
F-score (%)	83.13	77.19
K-Fold Cross Validation (%)	81.06 ± 2.28	
Accuracy (%)	**87.58**	**84.24**

During the development of the project we performed multiple tests, which, however, didn't lead to an improvement of the performances, as it is shown in the table below.

	With Outliers	**With Data Augmentation**	**Without Parch and SibSp catergories**
Train accuracy	89.03 %	83.03 %	85.30 %
Validation accuracy	83.10 %	73.30 %	80.0 %
Test accuracy	78.10 %	69.60 %	81.21 %

In particular, since our dataset was not particularly rich, we tried to maintain also the outliers for the training of the model, but this caused overfitting, as the accuracy over the train set improved while the one evaluated on the test set decreased. For this reason, we also tried to augment the data (eliminating the outliers). Nevertheless, also this approach turned out to be unsuccessful, probably because we added the data randomly, bringing confusion during the learning phase.
Furthermore, we checked the performances of the model eliminating the 'Parch' and 'SibSp' features, as we had introduced 'Fare_per_person' that includes in itself such information. However, this changing too lowered the overall accuracy, se we decided to keep both of them.

In spite of the results we obtained, there is still of course, room for improvement.
For instance, it could be done a more extensive feature engineering, acting in particular on the amount of data provided and their quality. In fact, a fulfilling data augmentation would for sure improve the results, as the algorithm would have more samples on which computing the training.
Moreover, more complex learning algorithm could be tested. For example we believe a MLP (Multi Layer Perceptron) neural network would achieve very good performances.
The MLP consists of multiple layers (at least one input, hidden and output layer) of artificial neurons organized in a specific way. Each neuron in a given layer is connected to all neurons in the previous and next layers. Each connection between neurons is associated with a weight, which represents the importance of the connection. Additionally, each neuron has an activation function that introduces non-linearity into the model. The more complex structure of such method gives it the ability of learning and discovering any hidden pattern in the data, solving any learning task (wherever it is simple or complicated).
However, in order to achieve good performances, MLP requires a large amount of data and a lot of computational burden. For these reasons, we decided to avoid a deep learning approach, preferring other machine learning algorithms.