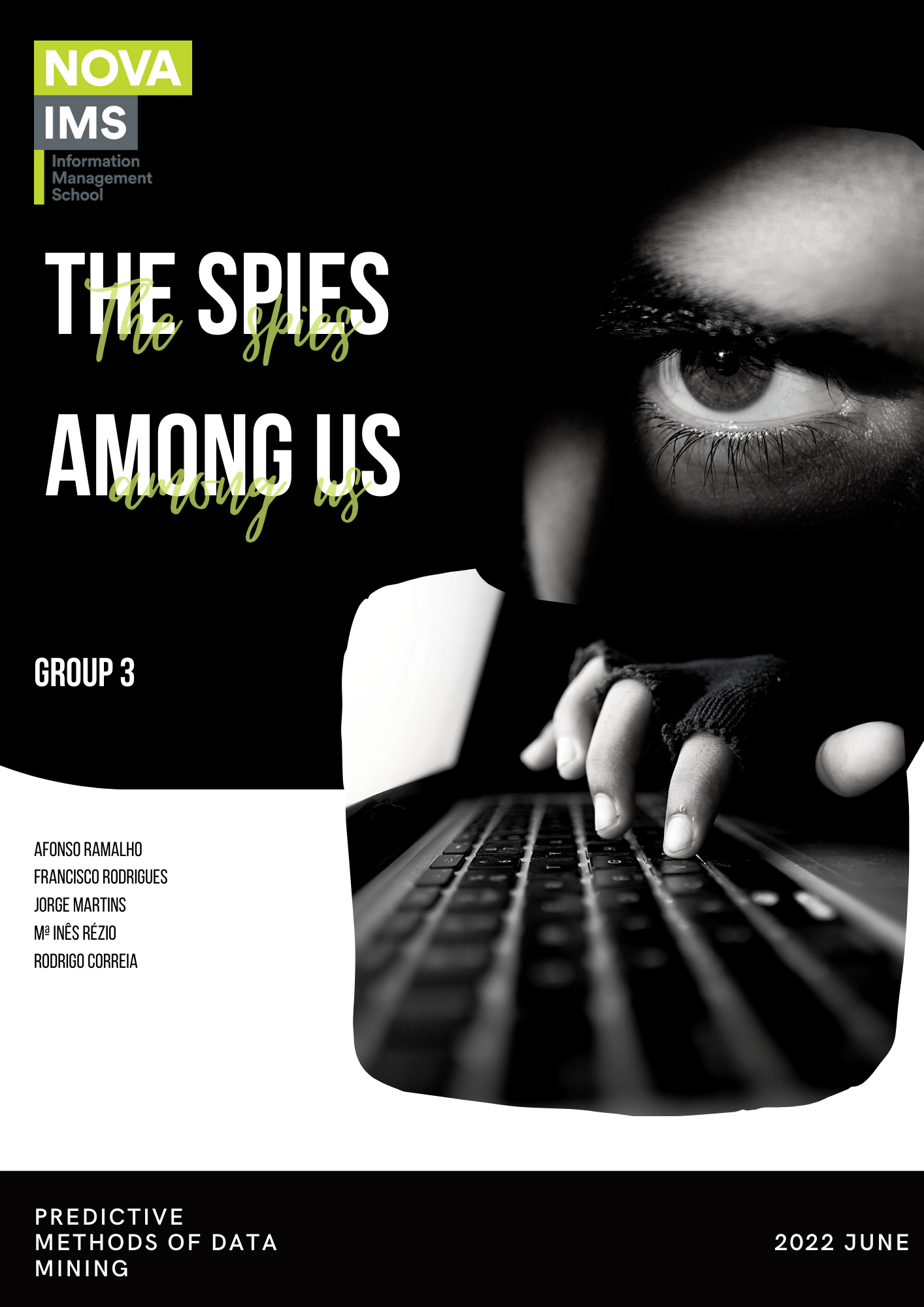
**nnnnnnnnnnnnñ**

**Index**

[Abstract 3](#_Toc106218399)

[1. Introduction 3](#_Toc106218400)

[2. Background 4](#_Toc106218401)

[2.1. Data Pre-Processing 4](#_Toc106218402)

[*2.1.1. KNN Imputer 4*](#_Toc106218403)

[*2.1.2. Clipping Outliers 4*](#_Toc106218404)

[2.2. Feature Selection and Engineering 4](#_Toc106218405)

[*2.2.1. MinMax Scaler 4*](#_Toc106218406)

[*2.2.2 RFE 5*](#_Toc106218407)

[2.3 Machine Learning Algorithms 5](#_Toc106218408)

[*2.3.2 SVC 5*](#_Toc106218409)

[*2.3.3 Gradient Boosting Classifier 6*](#_Toc106218410)

[*2.3.4 Passive Aggressive Classifier 6*](#_Toc106218411)

[*2.3.5 Gaussian Naïve Bayes 7*](#_Toc106218412)

[*2.3.6 AdaBoost 7*](#_Toc106218413)

[*2.3.7 MLP Classifier 8*](#_Toc106218414)

[*2.3.8 Bernoulli Naïve Bayes 8*](#_Toc106218415)

[*2.3.9 Decision Tree Classifier 9*](#_Toc106218416)

[*2.3.10 Random Forest Classifier 9*](#_Toc106218417)

[*2.3.11 Stacking Classifier 9*](#_Toc106218418)

[3. Methodology 10](#_Toc106218419)

[3.3 Exploratory Data Analysis 10](#_Toc106218420)

[*3.3.1 Check Records 10*](#_Toc106218421)

[*3.3.2 Summary Statistics 10*](#_Toc106218422)

[*3.3.3 Charts 11*](#_Toc106218423)

[*3.3.3.1 Univariate Analysis 12*](#_Toc106218424)

[*3.3.3.2 Bivariate Analysis 12*](#_Toc106218425)

[3.4 Data Pre-Processing 13](#_Toc106218426)

[*3.4.1 Train/Test Split 13*](#_Toc106218427)

[*3.4.2 Missing and Duplicate Data 13*](#_Toc106218428)

[3.5 Feature Selection and Engineering 18](#_Toc106218429)

[*3.5.1 Data Normalization 18*](#_Toc106218430)

[*3.5.2 Feature Selection 19*](#_Toc106218431)

[3.6 Building Models 20](#_Toc106218432)

[4. Results 35](#_Toc106218433)

[5. Conclusion 38](#_Toc106218434)

[6. References 39](#_Toc106218435)

# Abstract

The objective of this project is to predict the number of spies present in the United States.

We have as our main goal, to achieve the highest possible F1 score in the predictions of the test dataset.

We will describe the entirety of the process, from the data exploration phase to building of the models and analyzing the final results. Techniques like missing values imputation, clipping outliers, feature selection and engineering will be applied

In the end we will conclude what are the outcomes of this project, if we achieved our goal and what did we learn from it.

# Introduction

The president of the USA was told that some foreign forces spies are infiltrating into the USA to steal information, which is their most precious asset.

Having spies infiltrating in the country causes majors setbacks to the economy and consequent loss of trust among citizens. For that is necessary to act and develop a solution to “control” the entering of spies in the country.

On the premises that there are some groups of people more likely to be a spy, it is our job to build a predictive model that answer the question: *“Which citizens should be placed under close surveillance?”*

For that we will build ML models, using the data provided in which we had access to the citizens gender, age, area of residence, their household income, their household size, their level of satisfaction, if they are foreign citizen, if they are social, their mobile usage, their occupation, if they did military service or have any political participation.

As our primary goal we wish to develop the best performance model.

# Background

## Data Pre-Processing

### **KNN Imputer**

In order to deal with missing values of discrete variables, we use the *KNN Imputer*. This imputation method uses the *KNN* algorithm to find the average of the neighbors’ values to fill the respective missing value. The number of neighbors can be chosen, with the default number being 5.

### **Clipping Outliers**

To deal with the outliers of the discrete variables we used the *Winsorize* method.

It’s a process of substituting the extreme values to reduce the impact of outliers on computations or findings derived with that data. This method allows the values corresponding to outliers to be limited to pre-established values.

## Feature Selection and Engineering

### **MinMax Scaler**

In order to normalize the data, we used the *MinMax Scaler*. This method places the data in a range between 0-1. In addition to giving a normal distribution to the variables to be studied, it will also suppress the effects of outliers in the data. It has the following formula:

A picture containing text

Description automatically generated

Equation 1 - MinMax Scaler formula

### **RFE**

To choose the variables to be used we applied the *Recursive Feature Elimination* method. This method is efficient, since it selects the columns that are more relevant, in the training dataset in predicting the target variable.

It’s a wrapper-type feature selection algorithm that works by searching for a subset of features in the training dataset, starting with all of them and successfully deleting them until the target number remains.

It is an iterative method, whereby it is successively being improved iteration after iteration, and will rank the most important features and then re-fit them. This process will happen until a certain number of features are left.

## Machine Learning Algorithms

* + 1. ***KNN***

The *K-Nearest Neighbours*, mostly knows as *KNN*, is a non-parametric supervised machine learning algorithm that estimates the likelihood that a data point will be a member of one class. The logic to choose the class of each point, is its nearest neighbours.

It can be used either for regression or classification problems. To determine the distance between points it can use measures such as Euclidean distance, Manhattan distance, among others. We are also able to define the *‘k’* value on how many neighbours will be checked to determine the classification of a specific point.

### **SVC**

The *SVC*, *Support Vector Classifier*, is a vastly used supervised machine learning algorithm that is known for its kernel trick to support non-linear input spaces.

This algorithm returns a multidimensional hyperplane that divides our data in different classes. The returned hyperplane aims to better divide our data, minimizing the errors. So, the main idea is to reach a maximum marginal hyperplane which divides the datasets into classes in an optimized way.

To find the better hyperplane there are some parameters that we can tune like the kernel to be used in the algorithm, the gamma, which is the kernel coefficient, the *‘C’* which is the strength of the regularization parameter, among others.

### **Gradient Boosting Classifier**

The *GradientBoosting* is an algorithm specifically used for classification problems, that combine weak learning models to create strong predictions. This type of algorithms became very popular due to its effectiveness at classifying complex datasets.

We can implement the *GB* in four steps: fit the model, tune the parameters, make predictions, and interpret results.

This model can be seen as the *AdaBoost* (explained later) combined with weighted minimizations. The objective is to minimize the loss, this is, the difference between the actual value and the predicted one.

This type of model settles on two necessary parts: a week learner and an additive component. The weak learner that the model uses are decision trees, and once they return real values, as new learners are added to the model, the output of the trees can be joined together to correct errors in predictions. The additive component comes from the necessity to maintain the values of the trees fixed when we add other trees to the model, avoiding manipulation of the other trees.

### **Passive Aggressive Classifier**

*Passive-Aggressive* algorithms are often used for large-scale learning. It’s one of the few "online learning algorithms". With this type of model, the input data comes in sequence, and it’s updated incrementally, rather than in batches. This is useful when the sheer size of the data makes it computationally impossible to train on the entire dataset. We can simply say that an OL algorithm takes a training sample, updates the classifier, and then discards the sample.

This algorithm works like a Perceptron, because it does not require a learning rate, but it requires a regularization parameter. The way of work of this algorithm is simple. It is called *Passive-Aggressive* because it uses passive in a way that if the prediction is correct, it keeps the current model, but also is aggressive because if the prediction is wrong, then the model is changed. In terms of important parameters, the model uses the *‘C’* that is the regularization parameter, and applies some punishment if the prediction is incorrect, the *‘max\_iter’* that is maximum number of iterations that the model makes over the training data and the *‘tol’* that is the stopping criterion.

### **Gaussian Naïve Bayes**

*Gaussian Naïve Bayes* is a probabilistic request computation considering applying Bayes' theory with strong opportunity assumptions. With respect to arrange, opportunity suggests the likelihood that the presence of one worth of a part doesn't affect the presence of another (as opposed to independence in probability speculation). Straightforward suggests the use of a notion that the features of a thing are independent of one another. Concerning AI, straightforward Bayes classifiers are known to be significantly expressive, adaptable, and reasonably exact, yet their show debilitates rapidly with the advancement of the readiness set. Different components add to the result of simple Bayes classifiers. Most very, they require no tuning of the limits of the request model, they scale well with the size of the arrangement enlightening record, and they can without a doubt manage steady components.

### **AdaBoost**

The *Adaptive Boosting* algorithm, mostly know just as *AdaBoost*, is a boosting method which is used as an ensemble model. At each iteration, the weights are rearranged by giving higher weights to the incorrectly classified instances, converting these learners into stronger ones.

To have a better view of how this model works, let’s give a more detailed explanation. First, it builds a simple classification model, the model only needs to be slightly better than random chance (each record is used in the algorithm with equal case weights). Records correctly classified have their case weights reduced and records incorrectly classified have their case weights increased, and a new simple model is built.

The records that are difficult to classify keep getting their weights increased until, hopefully, they are finally classified correctly. Boosting is often repeated several times. The final predictions are made based on a weighted average/sum of the predictions from all the models.

This model can have its parameters tunned regarding to the learning rate, the number of estimators, the algorithm that support calculation of class probabilities, etc.

### **MLP Classifier**

The *Multi-Layer Perceptron Classifier* is a NN algorithm with multiple layers. These layers are called hidden layers, in addition to the input and output layer.

This type of algorithm doesn’t require linear signed functions. Arbitrary functions such as the logistics, sigmoid or hyperbolic tangents may be used in different nodes of the hidden layer and output layer. When classification errors, some type of “feedback” is required. This is achieved with the use of backpropagation algorithms, which contain two main phases, the forward and backward phase.

The *MLP* is considered a very powerful algorithm due to its ability to capture arbitrary functions such as decision boundaries of arbitrary shapes or non-contiguous class distributors. This model can have its parameters tuned regarding the hidden layer sizes, the maximum iterations, the activation function, the solver for weight optimization, between other parameters.

### **Bernoulli Naïve Bayes**

This algorithm implements *Naïve Bayes* training and classification algorithms for data distributed according to multivariate Bernoulli distributions; that is, several features may exist, but each is assumed to be a binary-valued variable. As a result, samples must be represented as binary-valued feature vectors; if given any other type of data, the algorithm instance may binarize it.

### **Decision Tree Classifier**

The *Decision Tree* algorithm is a supervised learning algorithm. The decision tree approach may also be utilized to solve regression and classification issues. By learning simple decision rules inferred from prior data, the purpose of utilizing a decision tree is to develop a training model that can be used to predict the class or value of the target variable.

We start from the root of the tree when using decision trees to forecast a class label for a record. The values of the root attribute and the record's attribute are compared. We follow the branch that corresponds to that value and jump to the next node based on the comparison.

### **Random Forest Classifier**

A *Random Forest* is a ML technique for solving classification and regression problems. It makes use of ensemble learning, which is a technique for solving complicated problems by combining multiple classifiers.

Many decision trees make up a random forest. Bagging or bootstrap aggregation are used to train the “forest” formed by the random forest method.

The algorithm determines the outcome based on the decision trees predictions. It forecasts by averaging the output of various trees. The precision of the result improves as the number of trees grows.

### **Stacking Classifier**

Stacking is an ensemble method that combines different ML models, those being a combination of heterogeneous weak learners. It can be used of classification or regression problems and can combine simple models with other ensemble or bagging models.

It consists of two layers of estimators. The first layer includes the baseline models that will generate output predictions and a second layer, where a meta-classifier/regressor takes all those outputs as inputs and generate the final predictions.

Stacking tends to be very efficient and one of the best performing methods, as it takes advantage of combining “different points of view”.

# Methodology

## Exploratory Data Analysis

### **Check Records**

The first step was to establish the consistency of the records. In this sense we started by dropping the *'ID'* column, as it was redundant, since for the Kaggle competition we would only use the *'ID\_ORIGINAL'*.

We analyzed the columns, understanding their type (categorical or numerical) and a general sense of possible missing values.

A picture containing table

Description automatically generated

Figure 1 -The-Spies-Among-Us dataset shape

### **Summary Statistics**

Analyzing the descriptive statistics is a fundamental step, was it helps to understand is the data is skewed and needs to be normalized

We plotted the statistics, like the mean, mode, standard deviation, etc.

We point out that the average age of the people observed is around 43.57, and the average of *‘Household\_Income’* is higher than the median, therefore having a left-skewed distribution.

We also observe that the level of satisfaction of the people observed is quite low, around 2.6.

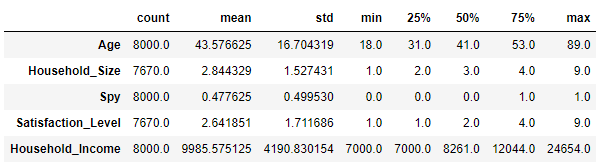
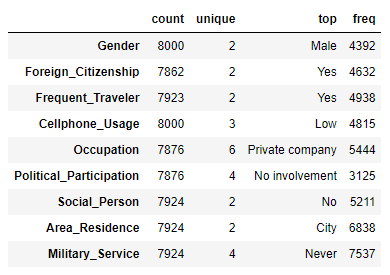
Noting that military service is primarily *‘Never’*, and cellphone use is *‘Low’* in most observations.

Figure 2 - Descriptive statistics for categorical variables

Figure 3 - Descriptive statistics for numerical variables

### **Charts**

Plots and Graphs are the visual evidence that helps us to get real insight into the data with ease.

For this purpose, we have established a division between univariate and bivariate analysis. The first helps us to have a clear perception of the distributions and patterns presented by the graphs, the second helps us to understand the relationships between variables.

### **Univariate Analysis**

For the categorical variables, we start by presenting count plots. These help us understand the number of observations per variable. From these we can extract some interesting insights.

Firstly, we find that the observations register more citizens with foreign nationality (58%) and that around 61.6% travel frequently.

In the vast majority of cases, mobile usage is low, around 60%, compared to high use (15%).

Regarding occupations, people tend to work for private companies, representing 68%. Few records of students, only 3.6%.

With regard to political participation, two categories stand out clearly: *‘No involvement’* and *‘Unknown’*, with percentages of 39.2% and 30.8% respectively.

Finally, about 65.2% of the people consider themselves to be non-social, about 85.2% live in the city and the overwhelming majority, about 94%, have no military service.

For the numerical variables we construct box plots that help us understand their distribution and whether they contain outliers.

We verify that the *‘Household\_*Size’ variable has a normal distribution.

Concerning the *‘Household\_Income’* and the *‘Satisfaction\_Level’*, we found that both variables present a left-skewed distribution, since the mean values of both variables are substantially higher than the median values. All three present outliers.

### **Bivariate Analysis**

For the bivariate analysis we built a correlation matrix and a pair plot.

We found that there are some relationships, as is the case of *‘Satisfaction\_Level’* with *‘Household\_Size’* with a Spearman coefficient of about -0.8 and the relationship between *‘Foreign\_Citizenship‘* and *‘Cellphone\_Usage\_Low’* with a Spearman coefficient of -0.7.

Regarding the numerical variables, it is not possible to extract relevant insights from these obtained relationships.

## Data Pre-Processing

### **Train/Test Split**

Before any pre-processing was done, we split the dataset into two, one for training and one for validation. The split was 80/20 respectively.

After the splitting we began to pre-process the training dataset, as well as applying the same steps on the validation dataset, and later in the test dataset.

### **Missing and Duplicate Data**

Regarding missing values, we noticed that there were some variables with missing data, those being *‘Foreign\_Citizenship’*, *‘Frequent\_Traveler’*, *‘Household\_Size’*, *‘Satisfaction\_Level’*, *‘Occupation’*, *‘Political\_Participation’*, *‘Social\_Person’*, *‘Area\_Residence’* and *‘Military\_Service’*.

We calculated the percentage of missing values per feature, to understand the impact of missing values, where we highlight *‘Household\_Size’* and *‘Satisfaction\_Level’* columns, both discrete and with the highest percentage of missing values, around 4%.

As far as duplicate values are concerned, we can say that these were not present in the dataset.

To treat the missing data in the features we created a new *DataFrame* with only the numerical features and to then impute using the *KNN Imputer* the discrete variables *‘Household\_Size’* and *‘Satisfaction\_Level’*. We used this method that fills the missing values with the mean value from five nearest neighbors. We used this imputer fitted to the training to transform the validation dataset.

With regard to the missing values of the categorical variables, we filled them in using the mode.

* + 1. ***Dummy variables***

After dealing with the missing values, we created dummies variables for the categorical features.

We got new 22 columns from the original 14.

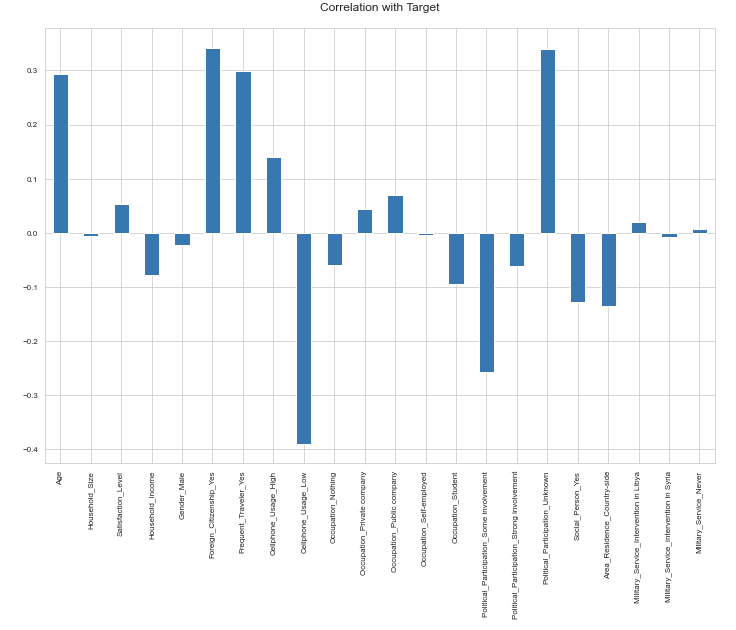
****An analysis of the correlation of each feature with the target *‘Spy’* was made.

Figure 4 - Bar plot of the features' correlation with the target variable

* + 1. ***Outliers***

The last step of the Data Pre-Processing phase was the checking for outliers in the training and validation dataset. We found outliers in three variables: *‘Household\_Size’*, *‘Satisfaction\_Level’* and *‘Household\_Income’*.

We used a process of clipping the outliers using the percentiles of the features, this meaning that we used a certain percentile value to clip every value above to that specific value. Most of the outliers were above the 97th percentile.

Graphical user interface, diagram

Description automatically generated**Diagram

Description automatically generated**

Figure 6 - Training features after outliers’ treatment

Figure 5 - Outliers for training features

In order to understand better the correlations between variables, we created a pair plot to observe the relations between numerical columns and also performed a Spearman correlation and plotted it into a heatmap to observe the relations between all variables.

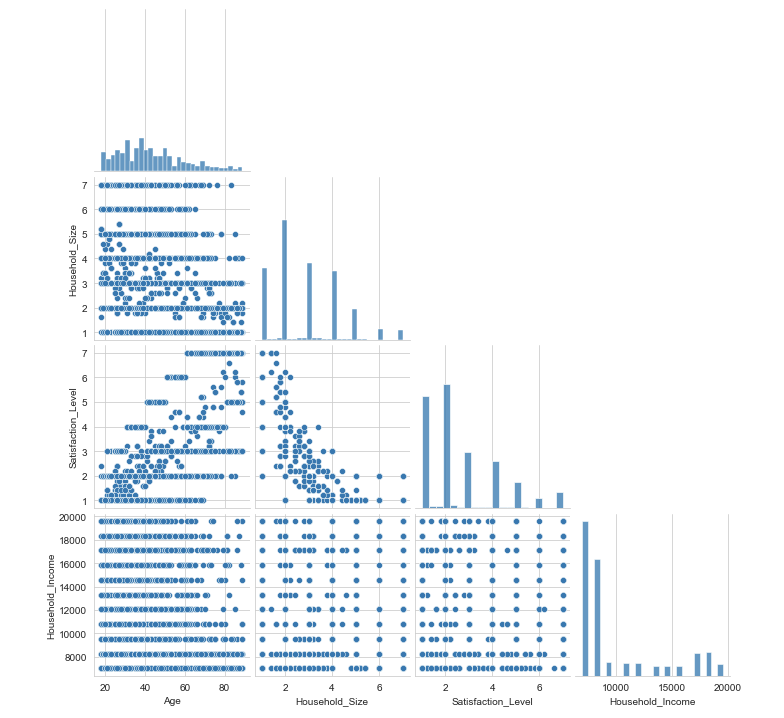


Figure 7 - Pair plot of the numerical features

We can visualize that no numerical feature presents a insightful relationship.

Graphical user interface, chart

Description automatically generated

Figure 8 - Spearman Correlaction Matrix

It’s possible to observe a highly negative correlation between the variables *‘Satisfaction\_Level’* and *‘Household\_Size’*, and also a high positive correlation between the first variable and *‘Age’* feature. Another high negative correlation occurs between the dummy variables *‘Foreign\_Citizenship\_Yes’* and *‘Cellphone\_Usage\_Low’*.

## Feature Selection and Engineering

Next, we initiated the Feature Selection and Engineering phase.

Chart, bar chart

Description automatically generatedFirst, we try to understand if the variable *‘Cellphone\_Usage\_High’* has an inverse relation with the variable *‘Cellphone\_Usage\_Low’*.

Figure 9 - Relation between dummy variables of 'Cellphone\_Usage'

### **Data Normalization**

Before applying any models to the data, we needed to scale the datasets in order to be able to apply certain models that require normalized data. For that we applied the *MinMaxScaler*.

### **Feature Selection**

Next, we started this process of selecting the features.

For that we used a *LogisticRegressionCV* and a *RandomForest* to be our estimators to feed to the *RFE*.

Chart

Description automatically generatedThe first model allows us to do the fit to the training data with 5-fold cross-validation. The accuracy score was 0.746875 for both training and validation datasets.

Figure 10 - LogisticRegressionCV Feature Importance

Chart, bar chart

Description automatically generatedThe random forest was initiated with a *‘max\_depth’*  of 8 and then fitted to the training data. The accuracy scores were 0.77421875 and 0.76125 for train and validation datasets respectively.

Figure 11 - RandomForest Feature Importance

We proceeded to feed the algorithms to the *RFE*, using the version with cross-validation built-in, that allows us once again to fit the data using a 5-fold cross-validation.

After running the model with both estimators, the one with the *LR* returned 11 features and the one with the *RF* returned 16 features.

Table

Description automatically generated with medium confidenceThe *RFE* with the *RF* was the chosen one to provide the features to train the models with, as the models improved their scores using this combination.

Figure 12 - Selected Features

## Building Models

For each model that we will present next, we did four steps: hyperparameters tuning if we found it necessary to improve the scores, fit and score the model, plot a confusion matrix to visualize the precision and recall and build a classification report to get a more in-depth analysis of the evaluation metrics.

Chart, treemap chart

Description automatically generatedFor the *GaussianNB* model, no parameter tuning was made, and the scores were 0.72234 for the training dataset and a score of 0.7175 for the validation dataset. These results were not promising, and no further optimization was made.

Figure 13 - Confusion Matrix for GaussianNB

Chart, square

Description automatically generatedRegarding the *KNN* model, we also did not perform any parameter tuning as we obtained a score of 0.803 for the training dataset and a posterior score of 0.725 for the validation dataset. Once again, the results weren’t promising and so we didn’t try improving the scores.

Figure 14 - Confusion Matrix for KNN

Square

Description automatically generatedAfter the *KNN* algorithm we proceeded to create the *Bernoulli Naïve Bayes* model, whose score for the training dataset was 0.7334 and the score for the validation dataset was 0.72875. No tuning of the parameters was made, as once again the results didn’t show any potential.

Figure 15 - Confusion Matrix for BernoulliNB

When it comes to the next model, the *Decision Tree Classifier*, we were able to visualize the difference in the score using different depths.

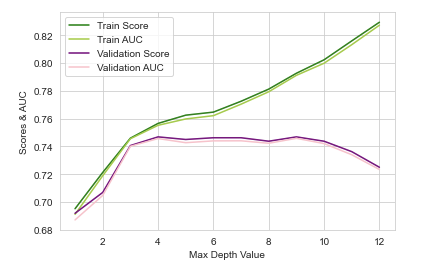
****The score of the training data set was 0.7678 whereas the score of the validation data set was 0.74125.

Figure 16 - Line plot for the evolution of scores with different depths

We did a variation of this same model, this using the *‘entropy’* as criteria. The score of the training data set was 0.7671875 in contrast to that of the valid data set which was 0.743.

Since the *Decision Tree Classifier* using *'entropy'* as a function to measure the quality of a split provided better results on the test data without performing significantly worse on training compared to the one using *'gini'*, it will be the model of choice. That said, we proceed to create the confusion matrix for the decision tree model.

Chart

Description automatically generated

Figure 17 - Confusion Matrix for DecisionTreeClassifier

A second run with the *Decision Tree* model using the *Grid Search* to do a model’s parameters selection was made, where we tested *‘max\_depth’*, *‘criterion’*, *‘min\_samples\_split’*, *‘min\_samples\_leaf’*. The grid search method used 5-fold cross-validation, fitting a total of 900 candidates, but it didn’t significantly improve the score of the decision tree model.

We also tried to improve the decision tree model using the ensemble method of bagging.

**Chart, treemap chart

Description automatically generated**So, we created a *Bagging Decision Tree*. The score for the training data set was 0.7729 and the score for the validation data set was 0.75625. Unlike in the *DecisionTreeClassifier*, the *BaggingClassifier* using the decision tree with the *‘entropy’* as criteria performed worse than the one using *‘gini’* as the function to define the quality of the split.

Figure 18 - Confusion Matrix for BaggingClassifier

**Chart, treemap chart, square

Description automatically generated**Moving to the next model built, the *RandomForestClassifier*, the score of the training data set was 0.97453 and the score of the validation data set was 0.7293. Until this moment was the one with the worst overfitting, but one with the biggest potential to be the best performing algorithm.

Figure 19 - Confusion Matrix for RandomForestClassifier

We also applied a grid search to this model, testing the same first three parameters as the grid search with decision tree and another one, the *‘n\_estimators’*. The total of candidates fitted was 1215.

Chart, treemap chart, square

Description automatically generatedWe can say that the training score was worse, scoring 0.7939, but the validation score did improve, going from 0. 7293 to 0.76.

Figure 20 - Confusion Matrix for GS RandomForestClassifier

The *AdaBoostClassifier* score for the training data set was 0.7498, while the score for the validation data set was 0.7418. The algorithm was also improved using the grid search method, but the increase in the scores wasn’t impressive.

Figure 21 - Confusion Matrix for GS AdaBoostClassifier

Chart, square

Description automatically generated

**Chart, square

Description automatically generated**The next model to be built was the *SVC*, which was tested with some parameters, like *‘C’*, which is regularization parameter, allowing to control the fitting of the model. It should be noted that it has a training data set score of 0.72859 and a validation data set score of 0.723.

Figure 22 - Confusion Matrix for SVC

The *GradientBoostingClassifier* is the next model to be built, its score for the training data set was 0.773 and its score for the validation data set was 0.75875, proving to be one of the best performing models as we see later in the Results section. It was built using the parameter *‘learning\_rate’* to 0.09, as we did some prior tests to understand which value was the best.

**Chart, treemap chart, square

Description automatically generated**The variation using *GridSearchCV* was also built, with a score of 0.78156 for the training data set and a score of 0.764 for the validation data set. Some of parameters tested were the *‘n\_estimators’*, *‘min\_samples\_leaf’* and *‘min\_samples\_split’*.

Figure 22 - Confusion Matrix for GS Gradient Booster

Chart, square

Description automatically generatedThe next model is the *MLPClassifier*, and the score for the training data set for this model was 0.7664 and the score for the validation data set was 0.74125. It uses some modified parameters like the activation function being *‘tanh’* or using a *‘adaptive’* learning rate.

Figure 22 - Confusion Matrix for MLP

Chart, treemap chart

Description automatically generatedSecond-to-last built was the *PassiveAggressiveClassifier*. This was a model that we decided to try as it seems a valid option for our case, but the results showed a different perspective. The score of the training data set was 0.7198 and the score of the validation data set was 0.71375. Values quite low, and only compared to the first models built.

Figure 23 - Confusion Matrix for Passive-Aggressive

The final model built was the *StackingClassifier*. This ensemble method allows us to stack multiple heterogeneous models and make a better prediction, as it’s based on several “opinions”. The estimators used as the baseline models were the *RandomForest* using *GridSearchCV*, and both *GradientBoosting* algorithms, the one with the *GridSearchCV* and the one without. The final estimator was a *LogisticRegression* using cross-validation.

Figure 24 - Confusion Matrix for Stacking

Chart, treemap chart

Description automatically generatedThe result was 0.7878 for the training and 0.7625 for the validation dataset.

# Results

This chapter aims to present the results that the group obtained from the analysis of the different algorithms used regarding the *f1 score*, *accuracy* score, *precision* score, *recall* score, all for training and validation datasets and the scores obtained on the Kaggle competition for the test dataset.

Between Bayesian, instance-based classifiers, decision trees, ensembles methods and NNs, the group used eleven different models, including some that were not teach during classes, like the *PassiveAggressiveClassifier* or the *StackingClassifier*.

Table

Description automatically generatedThe results tend to differ one from another, and at the end the algorithm with better validation *f1 score* is the *GradientBoosterClassifier* with grid search and the one with better training *f1 score* is the *RandomForestClassifier* with grid search.

Figure 25 - General Classification Report

We also conclude that the ones with the best recall were the ones more useful to our case, like *GaussianNB* but did not perform the best overall.

Chart, histogram

Description automatically generatedBelow we can also check the results on a bar chart with the accuracy scores for validation in ascending order, that can help us on the analysis.

Figure 26 - Models’ validation accuracies

However, when we check the results on the Kaggle competition the scores are a bit different. Here the *GradientBoosting* and the *RandomForest* with grid search were the best models, returning both a score of 0.72335.





Looking to the table of the models, we can also observe that some models suffer from overfitting, mainly *KNN* and *RandomForest*. When we have training scores differ substantially from the validation ones, it means that the model “memorizes” the training dataset so well that is not able to correctly classify new data.

To summarize the analysis, the *GradientBoosting* and the *RandomForest* with grid search were the models that had better scores in Kaggle.

However, if we pretend to pay more attention to the proportion of correctly positive events from all the events identified as positive, we must look to the *precision* and here the best model is the *PassiveAggressiveClassifier*. On the other hand, if we look to the proportion of events identified as positive from all the true positive events, we must look to the *recall* and here the strongest model is the *GaussianNB*. The group considered the *recall* a very important metric, once it is of the maximum importance to correctly identify the biggest number of true spies.

Another good metric is the *f1* *score* since it gave us the balance between *precision* and *recall* and here the strongest model is the *GradientBoosting* with grid search. The *accuracy* is not the “perfect” measure in our case, as the cost between a *False Positive* and a *False Negative* is not equal, as a *FP* can be very costly, this is, having a true spy not identified can bring terrible consequences, as identified in the being of this report.

# Conclusion

To conclude, in this project we created a “pipeline”, containing all phases of a real-life Data Mining project, starting with an EDA, Pre-Processing, Feature Engineering, building and training ML models and finally deploying those models onto a test case, with the purpose of predicting in a set of people with certain physical and social characteristics, which ones would be spies.

Our dataset had some inconsistencies, missing values, errors, and outliers that required some transformations and fine-tuning in order to proceed and guaranteed that the models would perform their best.

In terms of feature engineering, we decided not to create any new variables since we decided that the original variables were sufficient to develop our algorithms.

On the feature selection phase, the use of the RFE with two different models as well as a *SelectKBest* using mutual information as the scoring function to confirm which were the most important features, allowed us to reduce the possible overfitting from the redundant variables and therefore improve the performance of our models.

We have to note that our best performing models, *GradientBoosting* and *RandomForest* with grid search performed the best in test dataset without clipping the outliers, but the overall improvement with this technique in the remaining models by was noticeable.

In the end both of the models referred above scored the best in the competition (0.72335) and were also top performers during training, 0.75149 for *GradientBoosting* and 0.77489 for *RandomForest*. The stacking of both models, using the *StackingClassifier* did not perform better in the competition compared to each model alone, which we were not expecting.

# References

[1] - *The Spies Among Us*. (2022). Kaggle. Retrieved from https://www.kaggle.com/competitions/the-spies-among-us/leaderboard

[2] - Beheshti, N. (2022). *Cross-Validation and Grid Search*. Towards Data Science. Retrieved from https://towardsdatascience.com/cross-validation-and-grid-search-efa64b127c1b

[3] - Browlee, J. (2021, January). *Random Oversampling and Under sampling for Imbalanced Classification*. Machine Learning Mastery. Retrieved from https://Machinelearningmastery.Com/Random-Oversampling-and-Undersampling-for-Imbalanced-Classification/.

[4] - *Confusion Matrix*. (2022). scikit-learn. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion\_matrix.html

[5] - Gupta, M. (2021, June). *ML | Feature Scaling - Part 2*. Geeksforgeeks. Retrieved from https://www.geeksforgeeks.org/ml-feature-scaling-part-2/

[6] - Kaul, S. (2020, May). *Deeply Explained Cross-Validation in ML/AI*. Medium. Retrieved from https://medium.com/analytics-vidhya/deeply-explained-cross-validation-in-ml-ai-2e846a83f6ed

[7] - *For outliers treatment: clipping, winsorizing or removing?* (2020). StackExchange. Retrieved from https://datascience.stackexchange.com/questions/65802/for-outliers-%20treatment-clipping-winsorizing-or-removing

[8] - Bronshtein, A. (2017, May). *Train/Test Split and Cross Validation in Python*. Towards Data Science. Retrieved from https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6