NOVA IMS Information Management School

Predictive Methods of Data Mining
2021/2022

THE SPIES AMONG US

June 2022

Group 13
Anastasia Nica M20210516
Duarte Empis M20210373
João Pestana M20210551
Maureen Rachel Tibbe M20210366
Teresa Portugal Ramos M20210561

Abstract

The purpose of this project is to predict which citizens should be placed under close surveillance based on their characteristics. By other words, to predict the citizens that should be considered as spies based on their characteristics. For this, it was provided a dataset containing a set of variables like, for example, age, gender and occupation, describing 8000 different citizens.

The first step was to understand the data by looking at the descriptive statistics (for the numerical and categorical variables). Also, some preprocessing tasks were performed like set an index, looking for duplicated values, incoherencies, missing values, outliers and performing some feature engineering. In order to improve the models, it was also analyzed if there were any variables correlated.

After the treatment and transformation process, it was needed to perform Feature Selection, by leveraging the RFE and feature importance techniques, we selected the best variables to use in the predictive models.

In order to find the model that best predicts the target variable, eight different models were applied: Random Forest, Gradient Boosting, XGB Model, Adaboost, Bagging with Extra Trees, Gaussian Naïve Bayes, Neural Networks and Voting Classifier, the last one being the one with the best performance results.

Table of Contents

Abstract ..... **I**

List of Figures ..... **V**

List of Tables ..... **VI**

1 Introduction **1**

2 Methodology **1**

2.1 Dataset and Variables Description 2

2.2 Descriptive Statistics 3

3 Data Preparation **5**

3.1 Initial Preparation 5

3.2 Splitting the Data 5

3.3 Missing Values 6

3.4 Outliers 8

3.4.1 Categorical Variables 8

3.4.2 Numerical Variables 9

3.5 Feature Engineering – Dummy Variables 11

3.6 Feature Engineering – Data Transformation/New Variables 12

3.7 Feature Engineering – Correlations 12

4 Models **13**

4.1 Feature Selection 13

4.2 Hyperparameter Tuning Strategy 15

4.3 Random Forest 16

4.3.1 Hyperparameter Tuning 16

4.3.2 Model Evaluation 16

4.4 Gradient Boosting 17

4.4.1 Hyperparameter Tuning 17

4.4.2 Model Evaluation 17

4.5 XGB Model 18

4.5.1 Hyperparameter Tuning 18

4.5.2 Model Evaluation 18

4.6 Adaboost 18

4.6.1 Hyperparameter Tuning 18

4.6.2 Model Evaluation 19

4.7 Bagging with Extra Trees 19

4.7.1 Hyperparameter Tuning 19

4.7.2 Model Evaluation 21

4.8 Gaussian Naïve Bayes 21

4.8.1 Hyperparameter Tuning 21

4.8.2 Model Evaluation 21

4.9 Neural Networks 22

4.9.1 Hyperparameter Tuning 22

4.9.2 Model Evaluation 22

4.10 Voting Classifier 23

4.10.1 Hyperparameter Tunning 23

4.10.2 Model Evaluation 23

5 Model Comparison 24

6 Conclusion 26

7 References 27

8 Appendix 31

8.1 Appendix 1: Correlation Matrix Heatmap 31

8.2 Appendix 2: Gradient Boosting Theoretical Introduction 32

8.3 Appendix 3: XGBoost Theoretical Introduction 32

8.4 Appendix 4: Bagging with Extra Trees Theoretical Introduction............. 33

8.5 Appendix 5: Voting Classifier Theoretical Introduction........................... 33

List of Figures

Figure 1: Value counts for the categorical variables 8

Figure 2: Box plots for the numerical variables 9

Figure 3 - Box plots for "Satisfaction_Level" with the log transformation 9

Figure 4 - Value count for the variable "Satisfaction_Level" 9

Figure 5: Histograms of the variables "Household_Size", "Age" and "Household_Income", respectively 10

Figure 6 - Box plots for the other numerical variables with the log transformation 10

Figure : Feature Importance by variable 15

List of Tables

Table 1: Variables Description ... 3
Table 2: Descriptive statistics for the Numerical Variables ... 4
Table 3: Descriptive statistics for the Categorical Variables ... 4
Table 4: Missing Values ... 6
Table 5: Wrapper method RFE results ... 14
Table 6: Random Forest parameters ... 16
Table 7: Random Forest results ... 17
Table 8: Gradient Boosting parameters ... 17
Table 9: Gradient Boosting results ... 17
Table 10: XGB parameters ... 18
Table 11: XGB results ... 18
Table 12: Adaboost parameters ... 19
Table 13: Adaboost results ... 19
Table 14: Extra Trees parameters ... 20
Table 15: Extra Trees and Bagging parameters ... 20
Table 16: Extra Trees and Bagging final parameters (using the default values for the Bagging) ... 21
Table 17: Bagging with Extra Trees results ... 21
Table 18: Gaussian Naïve Bayes results ... 22
Table 19: Neural Networks parameters ... 22
Table 20: Neural Networks results ... 22
Table 21: Voting Classifier parameters ... 23
Table 22: Voting Classifier results ... 23
Table 23: Accuracy, Precision and Recall results ... 24
Table 24: Accuracy Calculation ... 24

Table 25: Precision Calculation 24

Table 26: Recall Calculation 25

Table 27: F1 Score results 25

1 Introduction

Nowadays, more than ever, information is the most valuable but also vulnerable asset for governmental institutions. As we know, this is a major concern for the President of the United States of America due to the fact that if such information is accessed by the wrong person, major setbacks to the economy might happen as well as the loss of trust among the citizens.

Therefore, the US Defense Minister contacted our team and provided us the data of 8000 citizens to build a predictive model to find out which citizens should be placed in close surveillance and prove to be a threat to the American State.

Thereby, our elite team of data scientists will perceive the awareness of this phenomenon and understand to what extent there are structured ways to anticipate threats and prevent risks related to the security and collective privacy of citizens and the State, through the analysis of the profile of citizens registered in the database who are most likely to espionage.

2 Methodology

This project follows the CRISP-DM methodology, which stands for Cross-Industry Standard Process for Data Mining, a popular and robust methodology for increasing the success of DM projects. It includes six sequential, yet iterative phases to guide the data mining efforts

In the first phase, Business Understanding, we defined the Data Mining problem that we were handling: answering the question “Which citizens should be placed under close surveillance?”.

After identifying the goal to achieve, the data needs to be analyzed (2nd phase - Data Understanding). In this step it is important to get familiarized with the dataset. How many citizens are present in the dataset, which attributes (columns) do we have for each of them, datatypes of our data (numerical or categorical data) and analyzing the descriptive statistics. In this phase, some quality issues can already be identified.

Then, after the first insights about the data, some actions should be performed in order to transform and clean the data (3rd phase – Data Preparation). In this step, we will detect missing values and replace them following the best approach, evaluate if there are any extreme values to be considered as outliers. In this step we will also create Dummy variables, create new features and look for correlations between the variables.

As the fourth step, some modelling techniques will be applied (4th phase – Modelling). First, we will choose the variables to use in our models, it might exist features that are not useful in the prediction of the target, then we will run different algorithms / models. For each model is required to tune the parameters.

After performing the different algorithms, we need to evaluate the different model results (5th phase – Evaluation). The performance of the model is measured by the capability to predict the target variable that represents the goal initially defined.

At last, the 6th phase – Deployment, can be as simple as implementing a repeatable data mining process (the model) to be used in the future for the same goal (with the same dataset structure).

By following all this steps, knowing that moving back and forward may happen, we believe that we will achieve better results in this project, as this approach will serve as support for the project and will help us to lead the way.

2.1 Dataset and Variables Description

The entire dataset provided contains the information of 8000 citizens who may or may not be spies with further additional details illustrated in 16 different columns in a CSV file.

The dataset holds 16 variables in total, consisting of:

- 5 int
- 2 float
- 9 categorical variables

Variable Class	Variable Name	Variable Type	Description
Numerical	ID	int64	The unique identifier of the citizen
	ID_Original	int64	Another unique identifier of the citizen
	Age	int64	The age (in years) of the citizen Area of residence
	Household_Income	int64	The household income
	Spy	int64	If the citizen has been identified as a spy (assumes the values: 0 or 1)
	Household_Size	float64	Number of people in the citizen's household including himself
	Satisfaction_Level	float64	The satisfaction level with the standard of living in the country (assumes values from 0 to 9)
Categorical	Gender	object	The gender of the citizen (assumes the values: female or male)
	Area_Residence	object	Area of residence (assumes the values: city or country-side)
	Foreign_Citizenship	object	If the citizen has foreign citizenship (assumes the values: yes or no)
	Frequent_Traveler	object	If the citizen is a frequent traveler (travels more than twice a year)
	Social_Person	object	If the citizen is seen as social (assumes the values: yes or no)
	Cellphone_Usage	object	Cell phone usage level of the citizen (assumes the values: low, average or high)
	Occupation	object	The type of occupation the citizen has (assumes the values: nothing, student, self-employed, government, private or public company)
	Military_Service	object	If the citizen has completed military service (assumes the values: intervation in Libya, intervation in Syria, intervation in Iraq or Never"
	Political_Participation	object	Political participation level (considering activities such as voting in elections, (assumes the values: No involvement, Some involvment, Strong involvment, Unknown)

Table 1: Variables Description

2.2 Descriptive Statistics

In this step, we did a deeper analysis of the descriptive statistics for each variable to understand their behavior and distribution.

Regarding the descriptive statistics, we reached the following conclusions for the numerical variables:

- It is noticeable that the highest standard deviation is in the Household Income variable, meaning its values are more dispersed comparing to the mean. In this specific case the range of the represented values is extremely high due to the nature of the variable (we are dealing with the total income per household). If the data is more spread out, that means for this variable, we will have to look for outliers and find a way to treat them.
- The age variable also presents a considerable standard deviation; however, it is unlikely

to find outliers since the maximum age in the dataset is 89 years.

- The variables Household Size and Satisfaction Level have the same number of missing values which might be an indication of correlation of these variables and will be analyzed in the following.
- In the case of the spy variable the 50% quartile assumes the value of 0 which means that at least half of our individuals are considered as no spies.

	count not null	count null	mean	std	min	25%	50%	75%	max
Age	8.000	0	44	17	18	31	41	53	89
Household_Size	7.670	330	3	2	1	2	3	4	9
Spy	8.000	0	0	0	0	0	0	1	1
Satisfaction_Level	7.670	330	3	2	1	1	2	4	9
Household_Income	8.000	0	9.986	4.191	7.000	7.000	8.261	12.044	24.654

Table 2: Descriptive statistics for the Numerical Variables

- The variable “Military_Service” seems to have a high skewness given that 95% of the variable assumes the value “Never”.
- The variable “Area_Residence” seems to have a high skewness given that 86% of the variable assumes the value “City”.
- The variable “Gender” seems to be the variable, which is closest to a normal distribution, given that approximately half of the values are “Male” and the other half “Female”.

	count not null	count null	unique	most frequent	freq
Gender	8.000	0	2	Male	4.392
Foreign_Citizenship	7.862	138	2	Yes	4.632
Frequent_Traveler	7.923	77	2	Yes	4.938
Cellphone_Usage	8.000	0	3	Low	4.815
Occupation	7.876	124	6	Private company	5.444
Political_Participation	7.876	124	4	No involvement	3.125
Social_Person	7.924	76	2	No	5.211
Area_Residence	7.924	76	2	City	6.838
Military_Service	7.924	76	4	Never	7.537

Table 3: Descriptive statistics for the Categorical Variables

3 Data Preparation

3.1 Initial Preparation

Once we are already aware of the variables in our dataset and have analyzed the descriptive statistics, the next step is to prepare our dataset. We started by deleting the “ID” column, as it does not give us any relevant information, and then set the “ID_ORIGINAL” column as our dataset index.

After this, we looked for duplicated values in our dataset and there was not any.

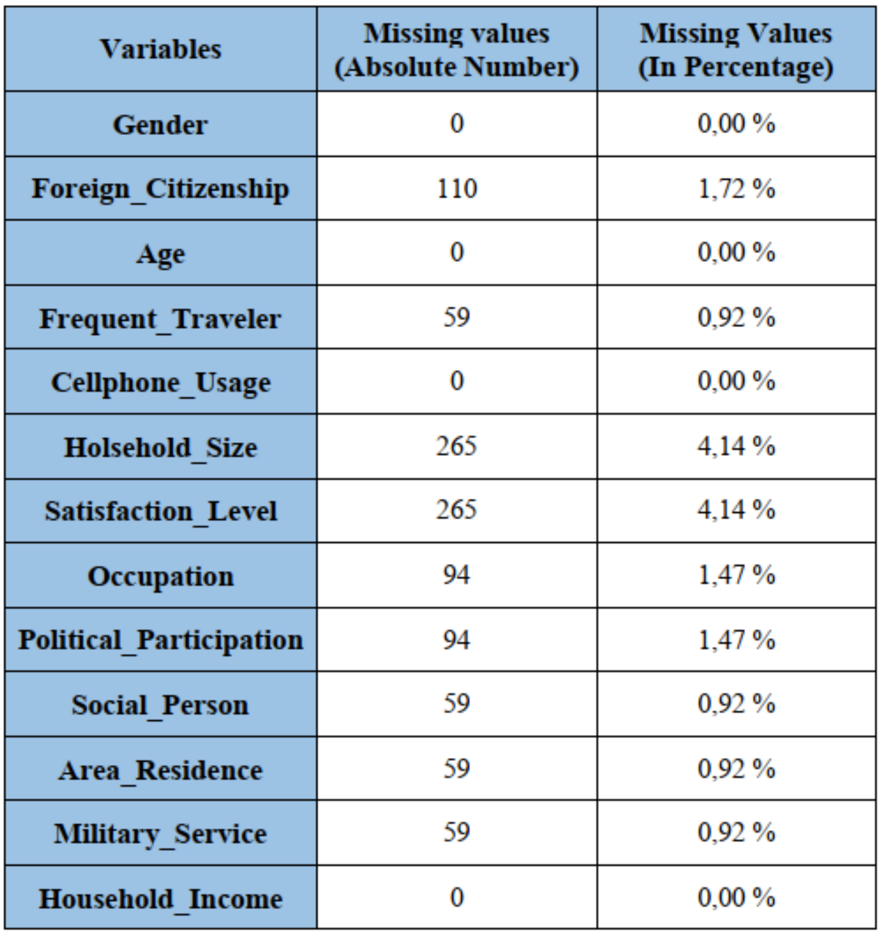
The last step for this initial preparation is checking if there is any incoherence in our dataset. After analyzing all the values for each variable, we can conclude that there is not any incoherence. So, we can now split the data.

3.2 Splitting the Data

In order to have a predictive model, we need to have more than one dataset. This is, we need to split the data so that we can train the model and then validate and test it. Before splitting it, we had to isolate the target variable “Spy”. After this, we can split the dataset into a training dataset and a validation dataset, using a proportion of 80:20 and we used the stratify parameter so that the same proportion is used for the target.

3.3 Missing Values

The next step was the identification/treatment of Missing Values for the training dataset. In the identification phase, we started by finding the columns with empty records (Table 4).



Variables	Missing values (Absolute Number)	Missing Values (In Percentage)
Gender	0	0,00 %
Foreign_Citizenship	110	1,72 %
Age	0	0,00 %
Frequent_Traveler	59	0,92 %
Cellphone_Usage	0	0,00 %
Holsehold_Size	265	4,14 %
Satisfaction_Level	265	4,14 %
Occupation	94	1,47 %
Political_Participation	94	1,47 %
Social_Person	59	0,92 %
Area_Residence	59	0,92 %
Military_Service	59	0,92 %
Household_Income	0	0,00 %

Table 4: Missing Values

Secondly, we decided to look for rows without at least half of the information, in other words, since we have 13 predictors, in each row our threshold for missing values was 6. However, no rows met this condition and therefore none were deleted.

For the treatment phase, we opted by imputing the missing values using 2 algorithms: KNN and Random Forest. Because we have variables of different types, continuous (Satisfaction_Level and Household_Size) and categorical (the others), we approached this problem as a Regression or a Classification, respectively. Additionally, to choose the best algorithm and fine-tune its parameters we used, in case of the Regression, the metric MAE (Mean Absolute Error), and for the Classification, the average f1-score of each class.

We began with the imputation of the continuous variables. The Satisfaction_Level had good results in both algorithms, with the RandomForestRegressor having the lowest MAE, with the value equal to 0.001. The Household_Size had worse results, but we still achieved a MAE of 0.39 for the RandomForestRegressor, and since the error is significantly lower than 0.5, we considered it a useful result.

Then, we proceed to the categorical variables and in all of these we obtained the highest f1-score with the RandomForestClassifier algorithm. The Frequent_Traveler and the Political_Participation were the ones with the worst results, having a f1-score of around 0.75. The rest of the variables had much better results, starting with the Occupation and Social_Person, which achieved a f1-score of 0.83, and the rest a score of approximately 0.90.

For the validation and test datasets we once again applied the two phases – Identification and Treatment. We identified the columns that had missing values treatment and, using the models previously fitted with the training dataset, we imputed the values for each respective column.

3.4 Outliers

To look for outliers, in a unidimensional perspective, we decide to analyze the boxplot and the histogram for each of the variables, when we are dealing with a numerical variable.

In the case of the categorical variables, a different approach was needed since these variables are not possible to plot the distribution.

3.4.1 Categorical Variables

For each categorical variable we proceed to do a value count to understand if the possible values correspond to the expected possibilities.

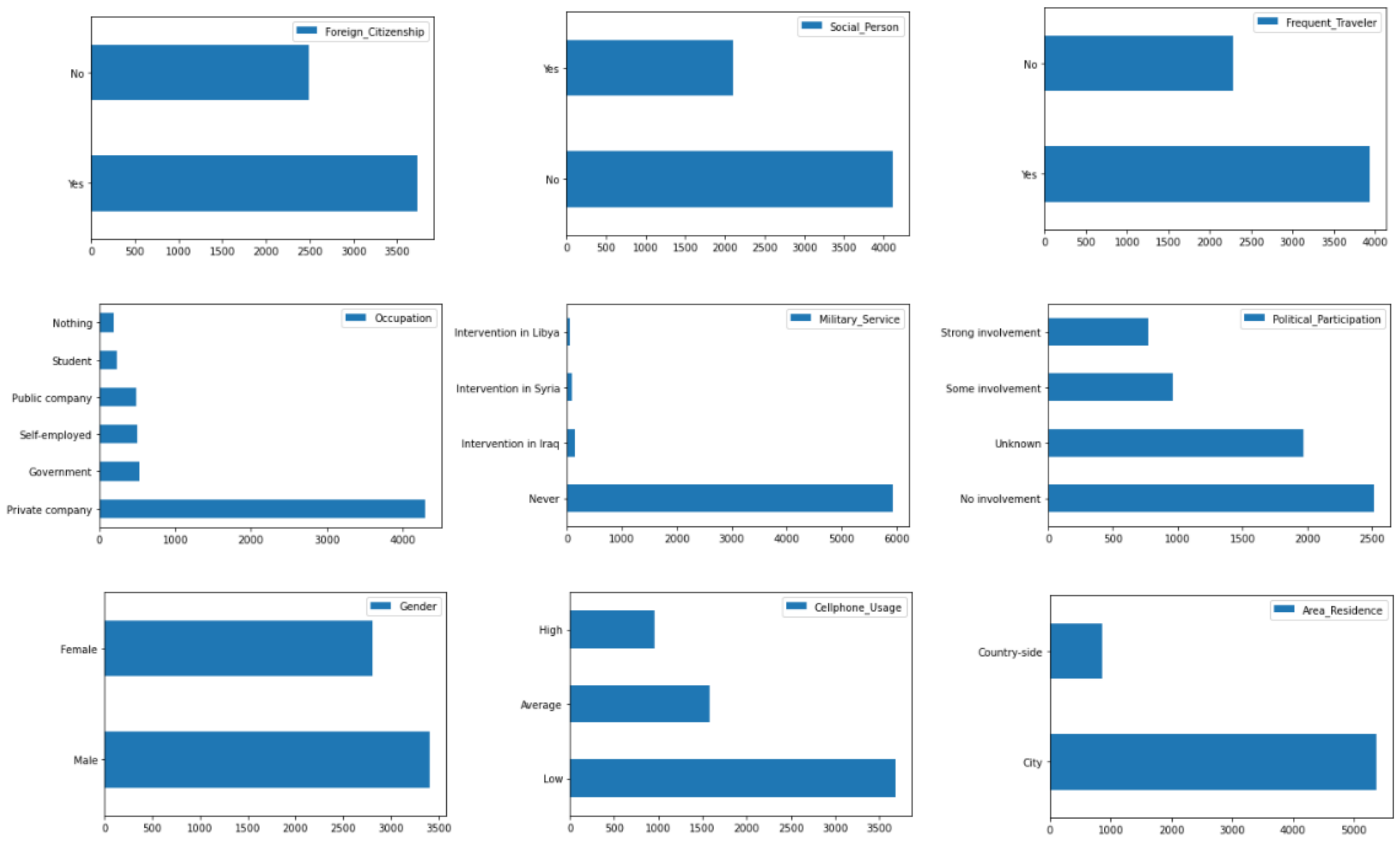


Figure 1: Value counts for the categorical variables

We concluded that, for each of these variables, the dataset only contains the expected outcomes. Therefore, no outliers were detected which was to expect due to the previous coherence check.

3.4.2 Numerical Variables

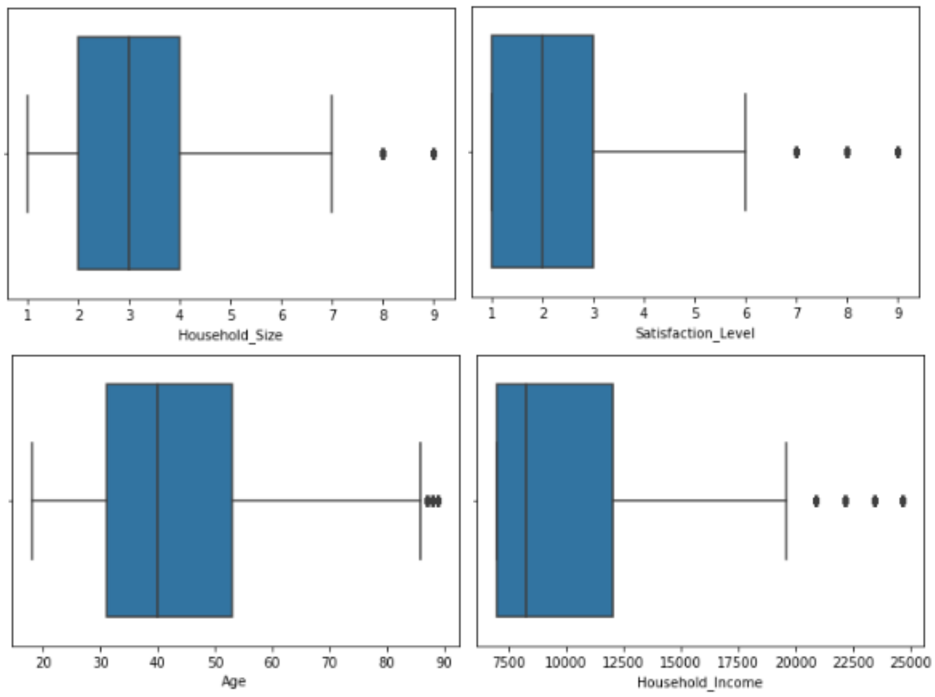


Figure 2: Box plots for the numerical variables

By looking at each boxplot of the numerical variables we conclude that all of them seem to have some outliers.

Regarding the variable "Satisfaction_Level", even with the boxplot showing us some outliers, we concluded that we wouldn't consider those data points as outliers, given that we are dealing with a variable that, in theory, should range from 0 to 9. After doing a value count of the variable we confirm that all the values for the variable were inside the expected interval. We decided to normalize the variable using the log function.

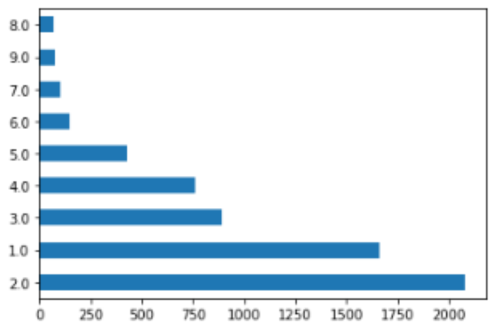


Figure 4 - Value count for the variable "Satisfaction_Level"

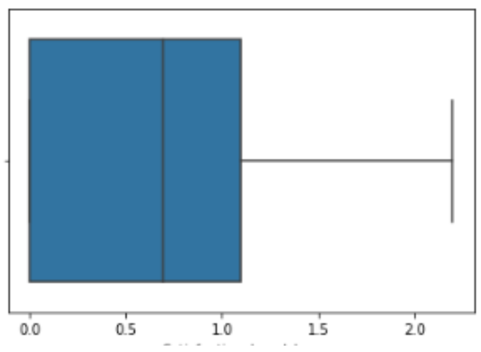


Figure 3 - Box plots for "Satisfaction_Level" with the log transformation

For, the other three numerical variables, we calculated the rule of the upper bound and realized that the total number of rows that were above this value represented 4.81% of the data which exceeds the 3% threshold. Therefore, we decided not to delete those rows and analyze the histogram of the variables.

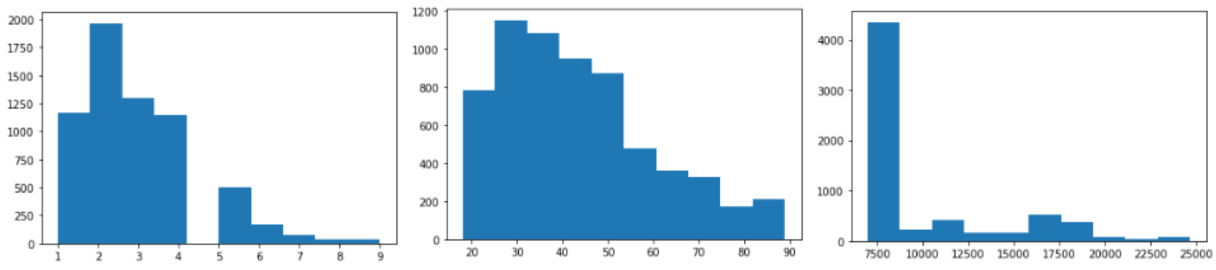


Figure 5: Histograms of the variables "Household_Size", "Age" and "Household_Income", respectively

All of these three variables presented positive skewness (skew>0). To correct the skewness, we decided to apply the log transformation in each of these variables. Like this we have all the numerical variables normalized. The boxplot was checked again, to see if we still have outliers, and we haven't found any extreme values.

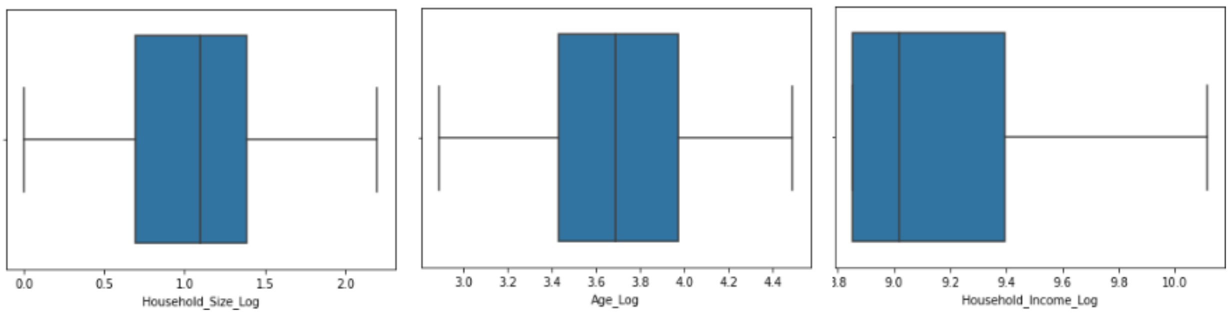


Figure 6 - Box plots for the other numerical variables with the log transformation

To ensure that the outliers' problem on these variables is solved, we decide to use the Isolation Forest algorithm, which partitions the dataset as a tree would do, identifying anomalies when, in a set of random trees, some of them produce shorter paths than others. Shorter branches generally mean outliers, while longer branches fit in the general picture of the selected sample. In practical terms, the random threshold defined between the minimum and maximum value of the variable will split the observations binarily: observations which don't fit the threshold will fall under one branch, while the ones that fit will fall under other.

We decided to perform this algorithm to the three numerical variables with outliers. Therefore, some parameters had to be initially defined, such as the contamination which was chosen as float in order to be able to set a corresponding threshold. Given this, the sum of the chosen contamination value for the three variables could not exceed 3-3,5% where the highest contamination value corresponds to the variable (Household_Income) with most outliers. After running the algorithm for these variables, we deleted 181 rows which translates to 2,828% of the training data. A new dataset without outliers was created which will be used in the modelling phase and holds 6.219 rows.

3.5 Feature Engineering – Dummy Variables

Since most models require numerical features, our next step was to transform the categorical variables in the training dataset into numerical ones.

To do so, we started with the variables that were binary ("Gender", "Foreign_Citizenship", "Frequent_Traveler", "Social_Person", "Area_Residence") and applied the One-hot encoding technique, which created one dummy variable per category and, to avoid the "dummy variable trap" (multi-collinearity), we deleted one of them.

For the variables "Military Service" and "Occupation", we also used the One-hot encoding, but, because they have more than two levels, the process of deleting one of the dummy variables was a bit different. To make this decision, we plotted the correlations between these variables and the target ("Spy") and decided to delete the least correlated.

The variable "Political_Participation" also had to receive a different treatment because, during our analysis, we identified the value "Unknown", which we considered a "Missing Not at Random", since we do not know the actual meaning of this category. To deal with this situation, we pondered two approaches: the One-hot encoding and the Target encoding. However, to avoid increasing the dimensionality of our dataset we opted for the second option.

Finally, for the "Cellphone_Usage", since the values of this variable ("Low", "Average" and "High") have a natural order, we decided to use the Label encoding technique, by replacing its values by 1, 2 and 3, respectively.

In order to keep the coherence between all the datasets, all of these transformations were then replicated for the validation and test datasets. It is also important to note, that the transformation of the variable "Political_Participation" was done with the Target Encoder, previously fitted with the training data.

3.6 Feature Engineering – Data Transformation/New Variables

Data transformation is one of the methods used to gain better insight into the data while allowing easier handling of the data, for example, by aggregating or combining attributes to create new features; it represents a process for enriching the dataset.

For this purpose, especially variables that take on many different values but can possibly be summarized under a single category were analyzed. In the case of the variable "Military_Service," it is irrelevant to know where a person served in the military; it is only important to know whether a person was active in the military service or not. A similar approach is adopted for the variable "Occupation". This leads to two new variables: first, "Public_Occupation" to distinguish whether a person works in the public sector or in the private sector and, second, "Employed" to distinguish between individuals who are employed and individuals who are not employed or hold a student status. In contrast, a slightly different transformation is performed with the variable "Household_Income". Based on the division with the variable "Household_Size" a new variable is derived, which holds information about how much income per person is accessible in the respective household (=Income_per_Person). Finally, the variable "Age" is split into different categories to provide instant access about the age group of an individual: [18, 28] = “Young”; [29, 39] = “Young_Adult”; [40, 50] = “Adult”; [51, 61] = “Adult_Senior”; [62, 72] = “Pre_Senior”; [73, +∞[= “Senior”.

In order to keep the coherence between all the datasets, all of these transformations were then replicated for the validation and test datasets.

3.7 Feature Engineering – Correlations

After feature engineering by creating dummy variables and new variables, it was needed to observe correlations between all variables. By using the Spearman Correlation, due to the fact that we are dealing with continuous and ordinal variables, 3 variables pairs that are highly correlated stand out, as can be analyzed by the correlation matrix displayed in Appendix 1:

- the variables with the strongest correlations are "Household_Size" and "Income_per_Person" with a correlation of -0.9.
- the variables "Household_Size" and "Satisfaction_Level" also show a high correlation of -0.8.
- the variables "Occupation_Private company" and "Political_Participation_No involvement" are significantly correlated as well with correlation of -0.8.

Therefore, in each of these correlated pair one variable must be deleted, specifically the one that is least correlated with the target variable ("Spy"):

- In the first case, "Household_Size" is the variable that is least correlated (0.02) with the target.
- In the second case, "Household_Size" is also the variable that is least correlated (0.02) with the target.
- In the last case, "Occupation_Private company" is the variable that is least correlated (0.04) with the target.

Consequently, the variables "Household_Size" and "Occupation_Private company" were deleted from both datasets (training and validation).

Additionally, to delete one of the variables regarding age sector in order to reduce the number of variables, we observe that the variable "Senior" is the least correlated with the target and subsequently is the one that was deleted. To keep the coherence between all the datasets, all of these transformations were also done for the validation and test datasets.

4 Models

4.1 Feature Selection

One challenge that we faced in this predictive model's phase was the high-dimensional data. In other words, data with many variables, not all of them useful in the prediction of the target and, therefore, with the potential of introducing redundancy and noise in our models. To overcome this challenge, we leveraged feature selection techniques to identify the best set of features.

Since our target is categorical, we started with the Wrapper method RFE, where we tried a different number of desired features and select the one with the best score. Additionally, in order to gain multiple perspectives, this process was done with more than one model to evaluate feature importance. The results were the follow:

	RFE with Random Forest	RFE with Gradient Boost	RFE with AdaBoost
Political_Participation	Keep	Keep	Keep
Age	Keep	Keep	Keep
Cellphone_Usage	Keep	Keep	Keep
Household_Income	Keep	Keep	Discard
Satisfaction_Level	Keep	Keep	Keep
Gender_Male	Keep	Keep	Keep
Foreign_Citizenship_Yes	Keep	Keep	Discard
Frequent_Traveler_Yes	Keep	Keep	Keep
Social_Person_Yes	Keep	Keep	Discard
Area_Residence_Country-side	Keep	Keep	Keep
Military_Service_Intervention in Iraq	Keep	Keep	Discard
Military_Service_Intervention in Libya	Discard	Discard	Keep
Military_Service_Intervention in Syria	Discard	Keep	Discard
Occupation_Government	Keep	Keep	Keep
Occupation_Nothing	Discard	Keep	Discard
Occupation_Public company	Keep	Keep	Keep
Occupation_Student	Keep	Keep	Keep
Military_Service	Keep	Discard	Keep
Public Occupation	Keep	Keep	Discard
Employed	Keep	Keep	Discard
Income_per_Household	Keep	Keep	Keep
Young	Keep	Keep	Discard
Young_Adult	Keep	Keep	Discard
Adult	Keep	Keep	Discard
Adult_Senior	Keep	Keep	Discard
Pre_Senior	Keep	Keep	Discard

Table 5: Wrapper method RFE results

Regarding the subset of variables to discard, we get different results by running RFE with different models. That is to say that, despite having some similarities, the 3 models recommend eliminating different sets of features. To complement this method, we decided to use another approach for the feature selection process, bearing in mind that the variables "Military_Service_Intervention_Lyria", "Military_Service_Intervention_Syria" and "Ocupation_Nothing" were the most selected to be discarded.

As for the other approach, we decided to leverage the feature importance method that is available on every model we implemented in our project (Random Forest, Extra Tree, Gradient Boosting, Adaboost and XGBoost). And we have done so, by averaging the importance of each of our variables. Then, to be able to compare the results, we plotted them in the following graphic:

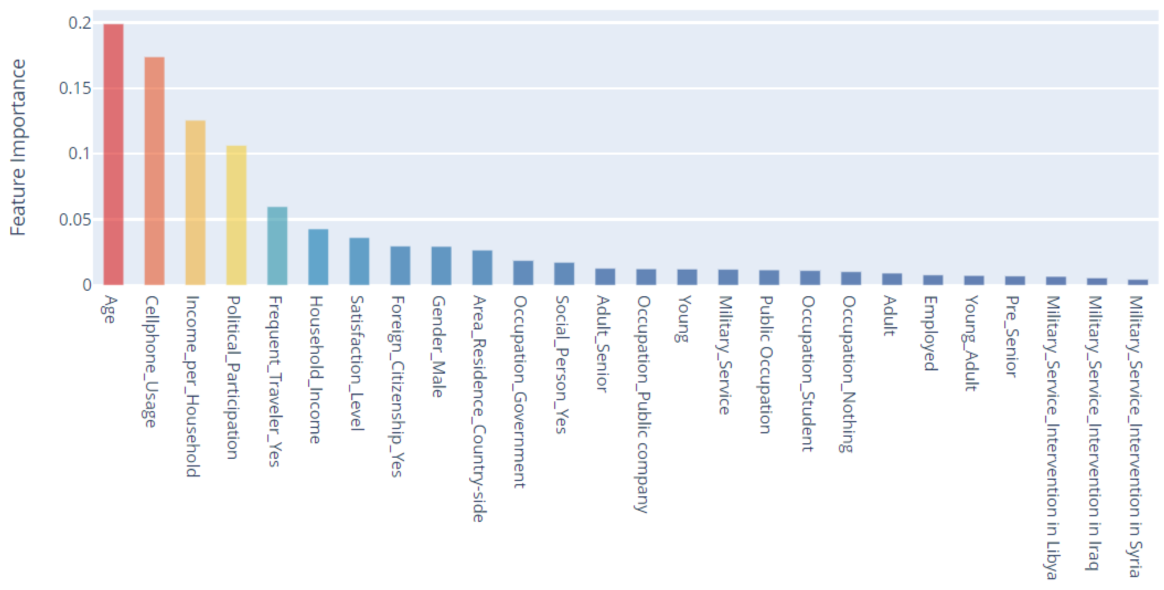


Figure 7: Feature Importance by variable

By analyzing the results of the two methods we decided to delete 4 variables:

- The ones regarding military service in Libya, Iraq and Syria, because both the RFEs and the feature importance consider them as the variables with the least added value
- The "Ocupation_Nothing" since 2 out of 3 RFE models discarded this variable and, when observing the feature importance graphic, we see that this variable is also ranked very low in terms of added value

After the feature selection process, we believe that the challenge of high-dimensional data was overcome, with the training dataset having now a total of 22 variables. To maintain the coherence, these same variables were also deleted in the validation dataset.

4.2 Hyperparameter Tuning Strategy

Due to the fact that we want to tune as many parameters as feasible with a large number of potential values, performing a grid search would require a large amount of computational effort. Consequently, as a new approach, we opted to first perform a random search with wide ranges for the selected parameters. The parameters identified as relevant to each model were chosen after intensive study and differ from model to model. For parameters that were not included in the tuning, the default value was already a good option in most cases. After we discovered the best values for each parameter through the first random search, the score for these best parameters was verified. In case of obtaining a good score, the ranges for potential values for each parameter were strongly limited based on the best parameters results from the first random search. In case the score was considered poor, the ranges were limited, but kept much larger. In other words, another random search was performed with limited ranges from the first random search using a different random state than the first random search. For each random search, 10.000 iterations were performed.

The majority of models follow this hyperparameter tuning strategy, although if this is not the case, it will be mentioned in the relevant chapter.

4.3 Random Forest

4.3.1 Hyperparameter Tuning

This model follows the hyperparameter tuning approach previously described and obtained the following best parameters after conducting two random searches and a grid search:

Parameter	Values Imputed for Grid Search	Grid Search Best Parameters
n_estimators	161; 180	161
criterion	"gini"; " entropy"	0,01; 0,2
max_depth	21; 18	21
max_features	"sqrt"; "log2"	"sqrt"
min_samples_leaf	5; 10	5
min_samples_split	0,004503435518628196; 0,0033897621388671254	0,003389762
max_samples	0,9858134718125645; 0,819801083169123	0,819801083
max_leaf_nodes	146; 129	129

Table 6: Random Forest parameters

4.3.2 Model Evaluation

The following table displays the respective mean values of each corresponding performance metric, calculated by each result after performing cross validation.

Dataset	Mean F1-Score	Mean Accuracy	Mean Precision	Mean Recall
Train	0,740468437	0,76171875	0,771316997	0,712130548
Validation	0,743241046	0,768125	0,788701336	0,705555556

Table 7: Random Forest results

4.4 Gradient Boosting

Please refer to the Appendix 2 for a brief theoretical introduction of this model.

4.4.1 Hyperparameter Tuning

Also, this model follows the hyperparameter tuning approach previously described and obtained the following best parameters after conducting two random searches and a grid search:

Parameter	Values Imputed for Grid Search	Grid Search Best Parameters
n_estimators	483; 594	594
learning_rate	0,013518562463528885; 0,007521306800445815	0.007521306800445815
max_depth	10; 12	12
max_features	"sqrt"; "log2"	"sqrt"
min_samples_leaf	11; 2	2
min_samples_split	146	146
max_leaf_nodes	80; 85	80

Table 8: Gradient Boosting parameters

4.4.2 Model Evaluation

The following table displays the respective mean values of each corresponding performance metric, calculated by each result after performing cross validation.

Dataset	Mean F1-Score	Mean Accuracy	Mean Precision	Mean Recall
Train	0,745933935	0,7646875	0,769957456	0,723576479
Validation	0,734596801	0,758125	0,770616121	0,705581355

Table 9: Gradient Boosting results

4.5 XGB Model

Please refer to the Appendix 3 for a brief theoretical introduction of this model.

4.5.1 Hyperparameter Tuning

Once again, this model follows the hyperparameter tuning approach previously described and obtained the following best parameters after conducting two random searches and a grid search:

Parameter	Values Imputed for Grid Search	Grid Search Best Parameters
n_estimators	991; 1955	1955
learning_rate	0,1392118091327114; 0,09881839229090249	0,098818392
max_depth	15; 22	22
gamma	8,381948282112184; 11,002058092439718	11,00205809
subsample	0,400015655249245; 0,8701697644164167	0,870169764
colsample_bytree	0,7522008329199338;0,47641794410341276	0,476417944
reg_alpha	0,6931130665141209; 0,2808954009032335	0,280895401
scale_pos_weight	0,8953355621208493; 1,3686676577191492	1,368667658

Table 10: XGB parameters

4.5.2 Model Evaluation

The following table displays the respective mean values of each corresponding performance metric, calculated by each result after performing cross validation.

Dataset	Mean F1-Score	Mean Accuracy	Mean Precision	Mean Recall
Train	0,749077028	0,75015625	0,719852644	0,781151118
Validation	0,741436883	0,74375	0,716510697	0,770992432

Table 11: XGB results

4.6 Adaboost

4.6.1 Hyperparameter Tuning

The Adaboost model follows the hyperparameter tuning approach previously described with a slight change and obtained the following best parameters after conducting not only two, but three random searches and a grid search:

Parameter	Values Imputed for Grid Search	Grid Search Best Parameters
n_estimators	144;148;118	148
learning_rate	0,3681981033316215; 0,4972718149160548; 0,44590143466533533	0,497271815
algorithm	"SAMME"; "SAMME.R"	"SAMME"

Table 12: Adaboost parameters

4.6.2 Model Evaluation

The following table displays the respective mean values of each corresponding performance metric, calculated by each result after performing cross validation.

Dataset	Mean F1-Score	Mean Accuracy	Mean Precision	Mean Recall
Train	0,744395192	0,75421875	0,739297769	0,749749687
Validation	0,726221686	0,74125	0,733611926	0,72124183

Table 13: Adaboost results

4.7 Bagging with Extra Trees

Please refer to the Appendix 4 for a brief theoretical introduction of this model.

4.7.1 Hyperparameter Tuning

For this model, a different approach was used for hyperparameter tuning, since there are significantly more parameters to be considered and even a random search with large ranges would require excessive computational effort. Initially, only the parameters of the Extra Trees model were tuned. However, this step follows the approach previously explained, meaning that two random searches were performed, first with a very large range and then with a smaller one based on the best parameters of the first random search. Finally, to find the best parameters from these two random searches, a grid search was performed with two values per parameter (one from each random search). All results are displayed in the table below.

Parameter	Values Imputed for Grid Search	Grid Search Best Parameters
n_estimators	116; 113	113
criterion	"entropy"; "gini"	"entropy"
max_depth	16	16
max_features	"sqrt"; "log2"	"sqrt"
min_samples_leaf	6	6
min_samples_split	0,006926076572967543; 0,004218085439517377	0,006926077
max_samples	0,7905554144086123; 0,8022758386587436	0,802275839
max_leaf_nodes	228; 213	213

Table 14: Extra Trees parameters

Subsequently, the best parameters resulting from the grid search were used as base estimator parameters. For the remaining parameters, in other words the parameters of the bagging model, various values were inserted to perform another grid search which along with the results can be derived from the table below.

Parameter	Values Imputed for Grid Search	Grid Search Best Parameters
base_estimator__n_estimators	113	113
n_estimators	10;50;70;90;113;120;150	120
base_estimator__criterion	"entropy"	"entropy"
base_estimator__max_depth	16	16
base_estimator__max_features	"sqrt"	"sqrt"
max_features	0,05; 0,1; 0,3; 0,5; 0,7; 0,9; 1	0,9
base_estimator__min_samples_leaf	6	6
base_estimator__min_samples_split	0,006926077	0,006926077
base_estimator__max_samples	0,802275839	0,802275839
max_samples	0,05; 0,1; 0,3; 0,5; 0,7; 0,9; 1	0,9
base_estimator__max_leaf_nodes	213	213

Table 15: Extra Trees and Bagging parameters

However, we have observed that by using the default parameters of the bagging model results in better scores, hence it was opted to use the default values for the model evaluation, as shown in the following table.

Parameter	Best Parameters
base_estimator__n_estimators	113
n_estimators	10
base_estimator__criterion	"entropy"
base_estimator__max_depth	16
base_estimator__max_features	"sqrt"
max_features	1
base_estimator__min_samples_leaf	6
base_estimator__min_samples_split	0,006926077
base_estimator__max_samples	0,802275839
max_samples	1
base_estimator__max_leaf_nodes	213

Table 16: Extra Trees and Bagging final parameters (using the default values for the Bagging)

4.7.2 Model Evaluation

The following table displays the respective mean values of each corresponding performance metric, calculated by each result after performing cross validation.

Dataset	Mean F1-Score	Mean Accuracy	Mean Precision	Mean Recall
Train	0,733746148	0,754375	0,760387924	0,709190976
Validation	0,733665479	0,758125	0,773425912	0,701633987

Table 17: Bagging with Extra Trees results

4.8 Gaussian Naïve Bayes

4.8.1 Hyperparameter Tuning

For the Gaussian Naïve Bayes model no hyperparameters tuning was performed, we only trained our model with the datasets and calculate the performance metrics.

4.8.2 Model Evaluation

The following table displays the respective mean values of each corresponding performance metric, calculated by each result after performing cross validation.

Dataset	Mean F1-Score	Mean Accuracy	Mean Precision	Mean Recall
Train	0,716667528	0,71375	0,679602134	0,758260325
Validation	0,709227389	0,705	0,671132121	0,753973168

Table 18: Gaussian Naïve Bayes results

4.9 Neural Networks

4.9.1 Hyperparameter Tuning

The Neural Networks model follows a different approach, since we only applied a Grid Search in order to find the best parameters for the model. It is important to mention that we applied the Standard Scaler to normalize the dataset in order to get better results. We achieved the following best parameters after conducting the Grid Search:

Parameter	Values Imputed for GridSearch	Grid Search Best Parameters
hidden_layers_size	(50,50,50); (100,)	(50,50,50)
activation	tanh; relu	relu
solver	sgd; adam	adam
learning_rate_init	0.0001; 0.001; 0.01; 0.1	0.0001
random_state	410	410

Table 19: Neural Networks parameters

4.9.2 Model Evaluation

The following table displays the respective mean values of each corresponding performance metric, calculated by each result after performing cross validation.

Dataset	Mean F1-Score	Mean Accuracy	Mean Precision	Mean Recall
Train	0,733019203	0,74984375	0,747661732	0,72690088
Validation	0,708739027	0,73125	0,734262852	0,687220502

Table 20: Neural Networks results

4.10 Voting Classifier

Please refer to the Appendix 5 for a brief theoretical introduction of this model.

4.10.1 Hyperparameter Tunning

For our Voting Classifier we used as base estimators the Random Forest, Gradient Boosting, Adabooost, XGBoost and Bagging previously tuned. Therefore, we just had to find the best voting strategy and the optimal weight for each of the base estimators. To do so, we did a grid search with multiple set of weights and the two votings, achieving the following best parameters:

Parameter	Grid Search Best Parameters
voting	"hard"
weights	(0.8365295434068388, 5.056025332840687, 0.5068537851666784, 1.2815500484532998, 0.660958087255565)

Table 21: Voting Classifier parameters

4.10.2 Model Evaluation

The following table displays the respective mean values of each corresponding performance metric, calculated by each result after performing cross validation.

Dataset	Mean F1-Score	Mean Accuracy	Mean Precision	Mean Recall
Train	0,7532666306238089	0,75453125	0,724195172	0,785075896
Validation	0,741895618	0,7425	0,712543768	0,77621259

Table 22: Voting Classifier results

5 Model Comparison

First, in order to choose the best model, we decided to analyze the accuracy, precision and recall metrics for the training dataset.

	Mean Accuracy	Mean Precision	Mean Recall
Gradient Boosting	0,76469	0,76996	0,72358
Random Forest	0,76172	0,77132	0,71213
Voting Classifier	0,75453	0,72420	0,78508
Bagging with Extra Trees	0,75438	0,76039	0,70919
Adaboost	0,75422	0,73930	0,74975
XGB Model	0,75016	0,71985	0,78115
Neural Networks	0,74984	0,74766	0,72690
Gaussian Naive Bayes	0,71375	0,67960	0,75826

Table 23: Accuracy, Precision and Recall results

All models presented a reasonably accuracy, in the training dataset for the 7 models, between 71,37 % and 76,47%, being the Gradient Boosting the one with highest accuracy followed by the Random Forest and then the Voting Classifier. These values give us the percentage of correct predictions made by a model.

		PREDICTED	
		Negative (not a spy)	**Positive (spy)**
ACTUAL	**Negative (not a spy)**	TN	FP
	Positive (spy)	FN	TP

Table 24: Accuracy Calculation

The precision results, for all the models, are between 67,96% and 77,13%. This metric computes the percentage of how many times an individual is an actual spy in the total amount of “being a spy” predictions. The model with the highest precision is the Random Forest followed by the Gradient Boosting and then the Bagging with Extra Trees.

		PREDICTED	
		Negative (not a spy)	**Positive (spy)**
ACTUAL	**Negative (not a spy)**	TN	FP
	Positive (spy)	FN	TP

Table 25: Precision Calculation

Regarding the recall, is more important than the precision because there is a high cost associated when a person is incorrectly predicted as not being a spy. Given this, it is important to analyze the recall results. Recall is the ratio of correctly predicted positive observations to all observations in actual class. The results are between 69,97% and 78,51%. The Voting Classifier has the best result in this metric, followed by the XGB model and then the Gaussian Naïve Bayes.

		PREDICTED	
		Negative (not a spy)	**Positive (spy)**
ACTUAL	**Negative (not a spy)**	TN	FP
	Positive (spy)	FN	TP

Table 26: Recall Calculation

Lastly, to choose the model that best predicts the target variable, we consider the F1 score more reliable, as there is a major downside to predict false negatives. Analyzing Figure 27, we can see that there is no significant overfitting/underfitting and the best model is the Voting Classifier with 75,33%.

Also, important to refer that, when submitting the different models in Kaggle, the Voting Classifier was also the model with the best results.

	Train F1-Score	**Validation F1-Score**	**Overfitting/Underfitting**
Voting Classifier	0,75327	0,74190	0,01137
XGB Model	0,74908	0,74144	0,00764
Gradient Boosting	0,74593	0,73460	0,01134
Adaboost	0,74440	0,72622	0,01817
Random Forest	0,74047	0,74324	-0,00277
Bagging with Extra Trees	0,73375	0,73367	0,00008
Neural Networks	0,73302	0,70874	0,02428
Gaussian Naive Bayes	0,71667	0,70923	0,00744

Table 27: F1 Score results

6 Conclusion

Concluding, the aim for this project was to develop a predictive model that would help us identify people who are spies and put them under close surveillance. Therefore, several steps were applied, starting by analyzing the provided dataset and the variables that would reveal information about each person.

After getting familiar with the dataset, the data needed to be prepared for the predicting models. Hence, we had to treat the missing values, look for incoherencies, and detect outliers, to get more accurate predictions.

Regarding the feature engineering we had to transform categorical variables into numerical by using one-hot, target and label encoding. We created 10 new variables that could help us analyze the data. Finally, by observing the correlations between all variables, we deleted the ones that would increase overfitting.

In order to avoid overfitting and increase metrics´ results, we proceeded with the feature selection. Here, we combined 3 different models from RFE that gave us the features to discard. If 2/3 models considered a feature unimportant, we discarded it. Additionally, we computed the feature importance method, averaging the importance of each variable for every predictive model we implemented in our project. That way, we were sure we had to delete 4 variables.

Finally, we could start implementing the predictive models. Grid search requires a lot of time and a high computational effort. So, we did a very thorough approach, by doing for most of the models, 2 random searches and a grid search. There might be other ways for hyperparameter tuning, but we believe this was the best approach we could use.

We decided to use F1-Score as the most important metric to evaluate the models. This because, in its calculation, recall is needed and, for our problem, it is an important metric. There would be a downside if the model predicts an actual spy as not being a spy. It would put USA's president in jeopardy.

Now, focusing in the proper f1-scores, the model with best results was XGB (0,74908). However, most of the models had similar scores, excepting the Gaussian Naïve Bayes that had the poorest result (0,71667). Having this, we felt like it was a great idea to use the Voting Classifier that will ensemble all the models we used with the best parameters. For this model, we did not use the Neural Networks, as all models should go through the same treatment, and only this model needed to have normalized data.

7 References

Accuracy, Precision, Recall F1 Score: Interpretation of Performance Measures (2016). Available at: https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/ [Accessed: 10 June, 2022]. Exsilio Solutions.

Anomaly detection using Isolation Forest – A Complete Guide (2021). Available at: https://www.analyticsvidhya.com/blog/2021/07/anomaly-detection-using-isolation-forest-a-complete-guide/ [Accessed: 30 April, 2022]. Analytics Vidhya.

Aznar, Pablo (2020). *What is the difference between Extra Trees and Random Forest?* Available at: https://quantdare.com/what-is-the-difference-between-extra-trees-and-random-forest/ [Accessed: 01 June, 2022]. QuantDare.

Brownlee, Jason (2016). *A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning*. Available at: https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/ [Accessed: 26 May, 2022]. Machine Learning Mastery.

— (2020a). *How to Develop an Extra Trees Ensemble with Python*. Available at: https://machinelearningmastery.com/extra-trees-ensemble-with-python/ [Accessed: 12 May, 2022]. Machine Learning Mastery.

— (2020b). *How to Develop an Extra Trees Ensemble with Python*. Available at: https://machinelearningmastery.com/extra-trees-ensemble-with-python/ [Accessed: 12 May, 2022]. Machine Learning Mastery.

Caetano, Nuno, Paulo Cortez, and Raul Laureano (2014). “Using data mining for predic- tion of hospital length of stay: an application of the CRISP-DM methodology”. In: *International Conference on Enterprise Information Systems*. Springer, pp. 149–166.

Clarke, Matt (2021). *How to use the Isolation Forest model for outlier detection*. Available at: https://practicaldatascience.co.uk/machine-learning/how-to-use-the-isolation-forest-model-for-outlier-detection [Accessed: 30 April, 2022]. Practical Data Science.

CRISP-DM Help Overview (2021). Available at: https://www.ibm.com/docs/en/spss-modeler/18.2.0?topic=dm-crisp-help-overview [Accessed: 08 June, 2022]. IBM.

Ergün, Cansu (2020). *How XGBoost Handles Sparsities Arising From of Missing Data? (With an Example)*. Available at: https://medium.com/hypatai/how-xgboost-handles-sparsities-arising-from-of-missing-data-with-an-example-90ce8e4ba9ca [Accessed: 15 May, 2022]. Medium.

F1 Score vs. Accuracy: Which Should You Use? (2021). Available at: https://www.statology.org/f1-score-vs-accuracy/ [Accessed: 10 June, 2022]. Statology.

Gupta, Alind (2020). *ML | Extra Tree Classifier for Feature Selection*. Available at: https://www.geeksforgeeks.org/ml-extra-tree-classifier-for-feature-selection/ [Accessed: 26 May, 2022]. Geeks for Geeks.

Hotz, Nick (2022). *What is CRISP DM?* Available at: https://www.datascience-pm.com/crisp-dm-2/ [Accessed: 08 June, 2022]. Data Science Process Alliance.

Kumar, Ajitesh (2020). *Hard vs Soft Voting Classifier Python Example*. Available at: https://vitalflux.com/hard-vs-soft-voting-classifier-python-example/ [Accessed: 22 May, 2022]. Data Analytics.

Kumar, Satyam (2021). *Use Voting Classifier to improve the performance of your ML model: Essential guide to Voting Classifier Ensemble*. Available at: https://towardsdatascience.com/use-voting-classifier-to-improve-the-performance-of-your-ml-model-805345f9de0e

[Accessed: 22 May, 2022]. Towards Data Science.

Morde, Vishal (2019). *XGBoost Algorithm: Long May She Reign!* Available at: https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d [Accessed: 15 May, 2022]. Towards Data Science.

Moro, Sergio, Raul Laureano, and Paulo Cortez (2011). "Using data mining for bank direct marketing: An application of the crisp-dm methodology". In: *EUROSIS-ETI*

Nadali, Ahmad, Elham Naghizadeh Kakhky, and Hamid Eslami Nosratabadi (2011). "Evaluating the success level of data mining projects based on CRISP-DM methodology by a Fuzzy expert system". In: *2011 3rd International Conference on Electronics Com- puter Technology*. Vol. 6. IEEE, pp. 161–165.

Saini, Anshul (2021). *Gradient Boosting Algorithm: A Complete Guide for Beginners*. Available at: https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-beginners/ [Accessed: 26 May, 2022]. Analytics Vidhya.

Shung, Koo Ping (2018). *Accuracy, Precision, Recall or F1?* Available at: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9 [Accessed: 10 June, 2022]. Towards Data Science.

sklearn.ensemble.AdaBoostClassifier (2022). Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html [Accessed: 11 May, 2022]. scikit learn.

sklearn.ensemble.BaggingClassifier (2022). Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html [Accessed: 11 May, 2022]. scikit learn.

sklearn.ensemble.ExtraTreesClassifier (2022). Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html [Accessed: 11 May, 2022]. scikit learn.

sklearn.ensemble.GradientBoostingClassifier (2022). Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html [Accessed: 11 May, 2022]. scikit learn.

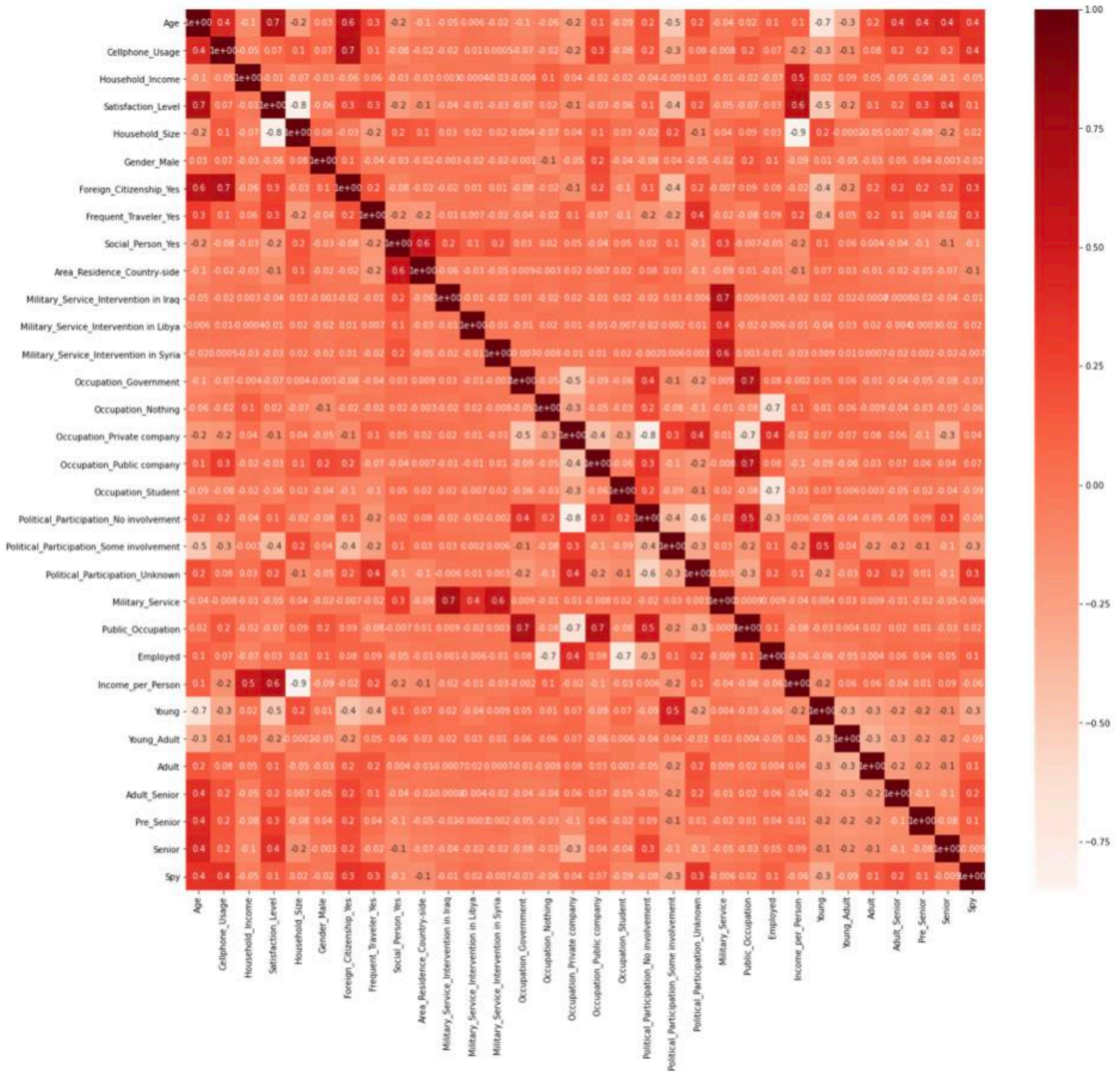
sklearn.ensemble.RandomForestClassifier (2022). Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html [Accessed: 11 May, 2022]. scikit learn.

Wade, Corey (2020). *Getting Started with XGBoost in scikit-learn*. Available at: https://towardsdatascience.com/getting-started-with-xgboost-in-scikit-learn-f69f5f470a97 [Accessed: 11 May, 2022]. Towards Data Science.

What is the CRISP-DM methodology? (2022). Available at: https://www.sv-europe.com/crisp-dm-methodology/ [Accessed: 08 June, 2022]. Smart Vision Europe.

8 Appendix

8.1 Appendix 1: Correlation Matrix Heatmap



8.2 Appendix 2: Gradient Boosting Theoretical Introduction

Gradient Boosting refers to a machine learning algorithm classified as an ensemble method and based on decision trees. Generally, the overall model is iteratively improved by minimizing the residuals. Thereby, it becomes evident that each decision tree is based on the previous ones. After a new decision tree is added, classifications are again estimated from a random sample and residuals are calculated. With the help of these, a next decision tree grows, which in turn is incorporated into the overall model with the learning rate. The lower the learning rate is chosen, the larger the number of decision trees to be estimated should be chosen. In the literature recommendation values between 0.001 and 0.1 can be found. In a nutshell, it can be stated that with gradient boosting the errors of the previous decision trees are accounted for by further decision trees. With the help of the further emerging trees, the classifications are corrected and improved.

8.3 Appendix 3: XGBoost Theoretical Introduction

XGBoost stands for Extreme Gradient Boosting and it is a decision-tree-based ensemble ML algorithm that uses a gradient boosting framework. Even tough, XGBoost and Gradient Boosting Machines (GBMs) are both ensemble tree methods that apply the principle of boosting weak learners, XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements.

Regarding the systems optimization, XGBoost approaches the process of sequential tree building using parallelized implementation. Additionally, unlike GBM, where tree pruning stops once a negative loss is encountered, XGBoost grows the tree up to max_depth and then prunes backward until the improvement in the loss function is below a threshold. Lastly, this algorithm has been designed to efficiently reduce computing time and allocate an optimal usage of memory resources.

However, the biggest advantages of XGBoost are in its algorithmic enhancements. Here, XGBoost offers additional regularization parameters to control the complexity of the model, which helps to avoid overfitting. It also employs the Sparsity-aware Split Finding algorithm, with this XGBoost handles sparsities in data, such as the presence of missing data and one-hot encoded values.

8.4 Appendix 4: Bagging with Extra Trees Theoretical Introduction

This method has a lot in common with the Random Forest, both are composed of a large number of decision trees, where the final decision is obtained taking into account the prediction of every tree. In the case of a classification problem by majority voting, and in the case of a regression problem by doing the arithmetic mean. In both methods the growing tree procedure is the same, when selecting the partition of each node, both models randomly choose a subset of features.

However, there are some main differences between the models:

- Extra Trees model does not perform bootstrap aggregation as Random Forest does. In simple words, takes a random subset of data without replacement.
- When splitting the nodes, Random Forest chooses the optimal split while Extra Trees choose it randomly. Important to refer that Extra Trees adds randomization however reaches optimization. For this reason, extra Trees is much faster, because choosing randomly the optimal split requires much less effort than calculating the optimal one (like Random Forest does).

On one hand, using the whole original sample instead of a bootstrap replica will reduce bias. On the other hand, however, choosing the split point randomly of each node will reduce variance.

In this project it was opted to use the Extra Trees Model as a base estimator for the Bagging algorithm, standing for Bootstrap Aggregation as an ensemble method.

The main idea of bagged trees is that rather than depending on a single tree, you are depending on many trees, which allows to leverage the insight of many models.

8.5 Appendix 5: Voting Classifier Theoretical Introduction

Voting classifier is a machine learning estimator that trains various base models and predicts by aggregating the findings of each base estimator based on the highest majority of voting. As such, instead of creating separate dedicated models and finding the accuracy for each of them, we create a single model which trains the base estimators and predicts the output based on their combined majority of voting for each output class.

Voting Classifier supports two types of voting strategy:

In Hard Voting, the predicted output class is a class with the highest majority of votes. Suppose three classifiers predicted the output class (A, A, B), so here the majority predicted A as output.

Hence A will be the final prediction.

In Soft Voting, the output class is the prediction based on the average of probability given to that class. Suppose given some input to three models, the prediction probability for class A = (0.30, 0.47, 0.53) and B = (0.20, 0.32, 0.40). So, the average for class A is 0.4333 and B is 0.3067, the winner is class A because it had the highest probability averaged by each classifier.