Data Science and Machine Learning

Elaborated on:

28-12-2023

By:

Gabriel Lopes

Assinado por: Gabriel José Marques Pessoa Lopes
Identificação: BI15885955
Data: 2023-12-28 às 23:48:53

Index

List of Figures v
1 Introduction 1
2 Descriptive analysis 2
2.1 Choice of analysis Software 2
2.2 First impressions in SPSS 3
2.2.1 Measure field 4
2.2.2 Value and labels 5
2.2.3 Labels 5
2.2.4 Result 5
2.3 Missing values 6
2.3.1 Verify the existence of missing values 6
2.3.2 Missing values configurarion tool 8
2.4 Descriptive statistics 8
2.5 Histogram 9
2.6 Variable recodification 10
2.7 Bar graph 11
2.8 Custom table 13
2.9 Boxplot 14
3 Exploratory data analysis 15
3.1 Independence tests 15
3.1.1 Pearsons's chi-squared test (x2 test) 15
3.1.2 Fisher's exact test 15
3.1.3 likelihood ratio test 16
3.2 Association tests 16
3.3 Crosstabs table 16
3.3.1 Select cases 17
3.3.2 Layer (groups) 19
3.4 Correlation measures 20
3.4.1 Normality assumptions 21
3.4.2 Pearson test 22
3.4.3 Spearman test 24
4 Parametric and non-parametric tests 24
4.1 Normality tests 25
4.2 Student's T-test 27
4.2.1 Normality assumptions 27
4.2.2 scale and categorical variables test 28

4.3 One-way ANOVA test 29
4.3.1 Normality assumptions 29
4.3.2 scale and categorical variables test 29
4.3.3 Curiosity 31
4.4 Mann-Whitney test 32
4.5 Kruskal-Wallis test 33
5 Exploratory data analysis – Python 34
5.1 Data acquisition 35
5.2 Descriptive statistics 36
5.3 Missing values 38
5.4 Normalization 39
5.5 Binning 41
5.6 Inferential statistics 42
6 Orange 44
6.1 Supervised Learning 44
6.1.1 Dataset importation 45
6.1.2 Predictions for the categorical target variable 46
6.1.3 Test and Score for the categorical target variable 47
6.1.4 Predictions for the scale target variable 51
6.1.5 Test and Score for the scale target variable 52
6.2 Unsupervised learning 54
6.2.1 k-means 54
6.2.2 Silhouette metric 58
6.2.3 Hierarchical clustering 59
6.2.4 DBSCAN 62
6.2.5 PCA 64
6.2.6 Feature ranking 65
7 Python – Supervised learning 67
7.1 Linear regression 67
7.1.1 needed packages 67
7.1.2 Data analysis and manipulation 67
7.1.3 Data plotting 68
7.1.4 Train and test dataset creation 69
7.1.5 Modelling and plot outputs 70
7.1.6 Evaluation 71
7.2 KNN 72
7.2.1 needed packages 72

7.2.2 Data visualization and analysis 72
7.2.3 Train and Test split 74
7.2.4 Training and prediction 75
7.2.5 Accuracy evaluation 75
7.2.6 Accuracy for different values for K 76
7.3 Decision Tree 77
7.3.1 needed packages 77
7.3.2 Data visualization and analysis 77
7.3.3 Decision tree set up 78
7.3.4 Modelling and prediction 79
7.3.5 Evaluation and visualization 79
7.4 Logistic Regression 81
7.4.1 needed packages 81
7.4.2 Data visualization and analysis 81
7.4.3 Train and test split and modelling 83
7.4.4 Evaluation with the Jaccard index 83
7.4.5 Evaluation with a confusion matrix 84
7.4.6 Evaluation with precision, recall and F1-score metrics 85
7.4.7 Evaluation with Log loss 86
7.5 SVM 86
7.5.1 needed packages 86
7.5.2 Data visualization and analysis 87
7.5.3 Train and test split 88
7.5.4 Modelling 88
7.5.5 Evaluation with the Jaccard index 88
7.5.6 Evaluation with a confusion matrix 89
7.5.7 Evaluation with precision, recall and F1-score metrics 90
8 Python – Unsupervised learning 91
8.1 K-Means 91
8.1.1 Needed packages 91
8.1.2 Data visualization and pre-processing 91
8.1.3 Modelling and insights 92
8.2 Hierarchical Clustering 94
8.2.1 Needed packages 94
8.2.2 Feature selection and normalization 95
8.2.3 Clustering with Scipy 95
8.2.4 Clustering with scikit-learn 97

8.3 DBSCAN 98
8.3.1 Needed packages 98
8.3.2 Data visualization 99
8.3.3 Visualization 99
8.3.4 Clustering 100
8.3.5 Visualization of clusters based on an aspect 100
8.3.6 Clustering based on several aspects 101
9 Conclusion 102
10 References 103

List of Figures

Figure 1 - SPSS first impressions 4
Figure 2 - SPSS Variable View 4
Figure 3 - Value field configuration 5
Figure 4 - Label field configuration 5
Figure 5 - Completed dataset's first configuration 6
Figure 6 - Missing Values analysis 7
Figure 7 - Missing (scale) Values Analysis result 7
Figure 8 - Missing (ordinal) Values Analysis result 7
Figure 9 - *Missing Values* button 8
Figure 10 - *Missing Values* configuration window 8
Figure 11 - Descriptive statistics configuration 9
Figure 12 - Descriptive statistics on days between entrance and arrival results 9
Figure 13 – Histogram on days between entrance and arrival 10
Figure 14 - Histogram on days between entrance and arrival depending on repeated (or not) guest 10
Figure 15 - Variable recodification 11
Figure 16 - Bar graph configuration 12
Figure 17 - Bar graph (number of cases) 12
Figure 18 - Bar graph (% of cases) 13
Figure 19 - Custom tables configuration 13
Figure 20 - Custom tables configuration (values to display) 14
Figure 21 - Custom table 14
Figure 22 - Boxplot configurations 14
Figure 23 - Boxplot 15
Figure 24 - Degrees of association 16
Figure 25 - Crosstabs table configuration 16
Figure 26 – Crosstabs table (and tests) results 17
Figure 27 - Selected cases condition 18
Figure 28 – Crosstab with the Selected cases condition 18
Figure 29 - Crosstabs layer configuration 19
Figure 30 - Crosstabs layer result (count) 19
Figure 31 - Crosstabs layer result (tests) 20
Figure 32 – *Explore* tool configuration 21
Figure 33 – *Normality tests* 22
Figure 34 - Pearson test configuration 23
Figure 35 - Pearson test result 23
Figure 36 - Spearman test configuration 24
Figure 37 - Spearman test result 24
Figure 38 - Parametric vs Non-parametric tests 25
Figure 39 - Normality tests configuration 26
Figure 40 - Normality tests results 26
Figure 41 - Asymmetry measures configuration 27
Figure 42 - Asymmetry measures results 27
Figure 43 - Normality tests (T-test) 28
Figure 44 - T-test configuration 28
Figure 45 - T-test results 28

FIGURE 46 - NORMALITY TESTS (ANOVA) 29
FIGURE 47 - ANOVA TEST CONFIGURATION I 30
FIGURE 48 - ANOVA TEST CONFIGURATION II 30
FIGURE 49 - ANOVA TEST RESULT 31
FIGURE 50 - ANOVA POST HOC TEST RESULT 31
FIGURE 51 - VARIABLE *ARRIVAL_DATE_MONTH* RECODING 32
FIGURE 52 - ANOVA TEST WITH THE DATE TYPE VARIABLE RECODED TO STRING/INT 32
FIGURE 53 - MANN-WHITNEY TEST CONFIGURATIONS 33
FIGURE 54 - MANN-WHITNEY TEST RESULTS 33
FIGURE 55 - KRUSKAL-WALLIS TEST CONFIGURATION 34
FIGURE 56 - KRUSKAL-WALLIS TEST RESULTS 34
FIGURE 57 - DATASET DOWNLOAD AND DISPLAY 35
FIGURE 58 - HEADER REMOVAL 36
FIGURE 59 - HEADER INSERTION 36
FIGURE 60 - HEADER REPLACEMENT AND INDEX COLUMN CREATION 36
FIGURE 61 - ALL VARIABLES' SAMPLE 37
FIGURE 62 - CATEGORICAL VARIABLES' SUMMARY 37
FIGURE 63 - OTHER SUMMARIES 37
FIGURE 64 - HISTOGRAM, CATEGORY DISPLAY AND BOXPLOT (RESPECTIVELY) 38
FIGURE 65 - MISSING VALUES COUNT 38
FIGURE 66 - MISSING VALUES REPLACEMENT 39
FIGURE 67 - DATA TYPES AND DATA TYPE CHANGE (RIGHT) 40
FIGURE 68 - COLUMN NORMALIZATION 40
FIGURE 69 - VARIABLE BINNING 41
FIGURE 70 - HISTOGRAM 41
FIGURE 71 - CORRELATION SCATTERPLOT 42
FIGURE 72 - PIVOT TABLE, HEATMAP AND MEAN BY GROUP (RESPECTIVELY) 43
FIGURE 73 - ANOVA ONE-WAY TEST IN PYTHON 43
FIGURE 74 - IMPORT DATASET OPTIONS 46
FIGURE 75 - CATEGORICAL TARGET PREDICTIONS SCHEME 46
FIGURE 76 - CATEGORICAL TARGET PREDICTIONS RESULTS 47
FIGURE 77 - CATEGORICAL TARGET TEST AND SCORE SCHEME 47
FIGURE 78 - CATEGORICAL TARGET TEST AND SCORE CROSS VALIDATION 48
FIGURE 79 - CATEGORICAL TARGET TEST AND SCORE TEST ON TRAIN DATA 48
FIGURE 80 - CONFUSION MATRIX EXAMPLE (KNN) 49
FIGURE 81 - ROC ANALYSIS (ALL ALGORITHMS) 49
FIGURE 82 – SCATTER PLOT HIGH ACCURACY ALGORITHM EXAMPLE 50
FIGURE 83 - SCATTER PLOT LOWER ACCURACY ALGORITHM EXAMPLE 50
FIGURE 84 - TARGET SCALE VARIABLE SELECTION 51
FIGURE 85 - SCALE TARGET PREDICTIONS SCHEME 52
FIGURE 86 – SCALE TARGET PREDICTIONS RESULTS 52
FIGURE 87 - SCALE TARGET TEST AND SCORE SCHEME 53
FIGURE 88 - SCALE TARGET TEST AND SCORE TEST ON TRAIN DATA 53
FIGURE 89 - SCALE TARGET TEST AND SCORE CROSS VALIDATION 54
FIGURE 90 - SCALE TARGET TEST AND SCORE DISTRIBUTION EXAMPLE (DECISION TREE) 54
FIGURE 91 – K-MEANS SCHEME 55
FIGURE 92 - K-MEANS CLUSTER CONFIGURATION (2 CLUSTERS) 55
FIGURE 93 - K-MEANS SCATTERPLOT (2 CLUSTERS) 56

FIGURE 94 - K-MEANS BOXPLOT (2 CLUSTERS) 56
FIGURE 95 - K-MEANS CLUSTER CONFIGURATION (5 CLUSTERS) 57
FIGURE 96 – K-MEANS SCATTERPLOT (5 CLUSTERS) 57
FIGURE 97 – K-MEANS BOXPLOT (5 CLUSTERS) 58
FIGURE 98 - K-MEANS SILHOUETTE SCHEME 59
FIGURE 99 - SILHOUETTE METRIC 59
FIGURE 100 – DENDROGRAM 60
FIGURE 101 - CLUSTER SELECTION (FOR PLOTTING) 61
FIGURE 102 – HIERARCHICAL CLUSTERING SCATTER PLOT 62
FIGURE 103 – HIERARCHICAL CLUSTERING BOXPLOT (1 CLUSTER) 62
FIGURE 104 - DBSCAN SCHEME 63
FIGURE 105 - 100 CORE NEIGHBORS (DBSCAN) 63
FIGURE 106 - 50 CORE NEIGHBORS (DBSCAN) 63
FIGURE 107 – PCA SCHEME 64
FIGURE 108 - PCA MINIMUM COMPONENTS 64
FIGURE 109 - PCA DATA TABLE 65
FIGURE 110 - SCATTER PLOT (PCA) 65
FIGURE 111 - FEATURE RANKING SCHEME 66
FIGURE 112 - FEATURE RANK BOXPLOT 66
FIGURE 113 - FEATURE RANK DISTRIBUTIONS 66
FIGURE 114 – LIBRARIES IMPORTS AND ALIASES 67
FIGURE 115 - DATA ACQUISITION AND READING 67
FIGURE 116 - DATA STATISTICAL ANALYSIS 68
FIGURE 117 - DISPLAY SPECIFIC COLUMNS 68
FIGURE 118 – HISTOGRAM PLOTTING 69
FIGURE 119 - LINEAR REGRESSION PLOTTING 69
FIGURE 120 – LINEAR REGRESSION TRAIN AND TEST DATASET SETTING 70
FIGURE 121 – LINEAR REGRESSION MODELLING 71
FIGURE 122 - LINEAR REGRESSION MODEL EVALUATION 72
FIGURE 123 - KNN NEEDED PACKAGES 72
FIGURE 124 - KNN DATA VISUALIZATION 73
FIGURE 125 – KNN COLUMN VISUALIZATION 73
FIGURE 126 – KNN'S DATASET'S COLUMNS AND DATA NORMALIZATION 74
FIGURE 127 - KNN TRAIN AND TEST SPLIT 74
FIGURE 128 - KNN TRAIN AND DATA PREDICTION 75
FIGURE 129 - ACCURACY EVALUATION FOR K=4 AND K=6 75
FIGURE 130 – ACCURACY FOR DIFFERENT VALUES FOR K 76
FIGURE 131 - KNN RESULTS AND BEST VALUE FOR K 76
FIGURE 132 – DECISION TREE NEEDED PACKAGES 77
FIGURE 133 – DECISION TREE DATA VISUALIZATION 77
FIGURE 134 - PRE-PROCESSING 78
FIGURE 135 – DECISION TREE SET UP 78
FIGURE 136 - DECISION TREE TRAIN SET SHAPE 78
FIGURE 137 - DECISION TREE TEST SET SHAPE 78
FIGURE 138 - DECISION TREE MODELLING 79
FIGURE 139 - DECISION TREE PREDICTION 79
FIGURE 140 - DECISION TREE EVALUATION 79
FIGURE 141 - ACCURACY CALCULATION WITHOUT *SKLEARN* 79

FIGURE 142 - DECISION TREE VISUALIZATION....80
FIGURE 143 – LOGISTIC REGRESSION NEEDED PACKAGES....81
FIGURE 144 – LOGISTIC REGRESSION DATA VISUALIZATION....81
FIGURE 145 - LOGISTIC REGRESSION DATA PREPROCESSING....82
FIGURE 146 - DATA ARRAYS CREATION AND NORMALIZATION....82
FIGURE 147 - LOGISTIC REGRESSION TRAIN/TEST SPLIT....83
FIGURE 148 - LOGISTIC REGRESSION MODELLING....83
FIGURE 149 - JACCARD INDEX FOR LOGISTIC REGRESSION EVALUATION....83
FIGURE 150 - LOGISTIC REGRESSION EVALUATION WITH A CONFUSION MATRIX....85
FIGURE 151 - LOGISTIC REGRESSION EVALUATION METRICS....86
FIGURE 152 - LOGISTIC REGRESSION EVALUATION WITH LOG LOSS....86
FIGURE 153 - SVM NEEDED PACKAGES....87
FIGURE 154 - SVM DATA VISUALIZATION....87
FIGURE 155 - SVM DATASET SCATTER PLOT....87
FIGURE 156 - SVM MODELLING....88
FIGURE 157 - JACCARD INDEX FOR SVM EVALUATION....88
FIGURE 158 - SVM EVALUATION WITH A CONFUSION MATRIX....90
FIGURE 159 - SVM EVALUATION METRICS....90
FIGURE 160 – K-MEANS NEEDED PACKAGES....91
FIGURE 161 – K-MEANS DATA VISUALIZATION AND PRE-PROCESSING....92
FIGURE 162 - K-MEANS DATA NORMALIZATION....92
FIGURE 163 - K-MEANS MODELLING....92
FIGURE 164 - K-MEANS INSIGHTS....93
FIGURE 165 - K-MEANS DISTRIBUTION....93
FIGURE 166 - K-MEANS 3D REPRESENTATION OF DISTRIBUTION....94
FIGURE 167 - HIERARCHICAL CLUSTERING NEEDED PACKAGES....95
FIGURE 168 - HIERARCHICAL CLUSTERING FEATURE SELECTION AND NORMALIZATION....95
FIGURE 169 – HIERARCHICAL CLUSTERING MATRIX'S EUCLIDEAN DISTANCE....96
FIGURE 170 - HIERARCHICAL CLUSTERING CLUSTER DETERMINATION....96
FIGURE 171 - HIERARCHICAL CLUSTERING DENDROGRAM....96
FIGURE 172 - HIERARCHICAL CLUSTERING MATRIX CREATION....97
FIGURE 173 - HIERARCHICAL CLUSTERING FORMING....97
FIGURE 174 - HIERARCHICAL CLUSTERING ADDED FIELD....97
FIGURE 175 - HIERARCHICAL CLUSTERING....98
FIGURE 176 - DBSCAN NEEDED PACKAGES....99
FIGURE 177 - DBSCAN DATA VISUALIZATION....99
FIGURE 178 - DBSCAN VISUALIZATION....99
FIGURE 179 - DBSCAN CLUSTERING....100
FIGURE 180 - DBSCAN CLUSTERS BASED ON AN ASPECT....100
FIGURE 181 - CLUSTERING BASED ON SEVERAL ASPECTS....101

1 Introduction

ML is a subset of artificial intelligence that focuses on the development of algorithms and models that enable computers to learn from and make predictions or decisions based on data. The presented document will approach this thematic focusing on the use of several software applications to perform descriptive, exploratory and inferential data analysis as well.

Descriptive data analysis is the process of summarizing and presenting data in a meaningful way. It involves calculating basic statistics (*e.g.* mean, median, mode, standard deviation, etc) and creating visual representations, such as charts, graphs, and tables to help understand the main features of a dataset.

Exploratory Data Analysis is an approach of data analysis to summarize their main characteristics, often with visual methods. It's used to understand the underlying structure, patterns, and relationships in the data. Techniques like scatter plots, histograms, and correlation matrices are often used in EDA.

Inferential data analysis involves making inferences or predictions about a population based on a sample of data. It uses statistical techniques to draw conclusions and generalize. This approach may be used to determine if a certain treatment is effective for an entire population based on a study conducted on a sample of individuals.

All these subjects will be discussed and displayed on the next topics of this document. In addition to this explanation, the practical experimentation in different kind of platforms will also be presented and explain. These platforms include SPSS for descriptive and exploratory, Jupyter Notebooks to use the *Python* language and Orange (3) for a more graphical way of inferential data analysis.

2 Descriptive analysis

Descriptive statistics are a set of measures and techniques used to summarize and describe the important characteristics, features, and patterns of a dataset. These statistics help to provide a clear and concise summary of the main aspects of the data, making it easier to understand and interpret.

In descriptive analysis there are three types of statistical variables, **continuous** (real numerical values such as heigh, age, salary), **nominal** (no possible order in its representation like name, sex or hair colour) and **ordinal** (represents a quality that can be ordered based on its results, for example, the months of the year or days of the week). Is important to verify and group each type of variable for the available tests given that, usually, they have criteria based on which type of variable.

2.1 Choice of analysis Software

SPSS statistics software is an application developed by IBM for data management, advanced and multifactorial analysis and even criminal investigation. It possesses a wide range of statistical procedures such as descriptive statistics as well as regression, ANOVA, factor and cluster analysis. It supports several data file formats as well, including Excel, CSV and other common spreadsheet formats. This platform provides a variety of tools for data cleaning and preparation, in particular, to handle missing data, variable recodification and/or transformation, etc…

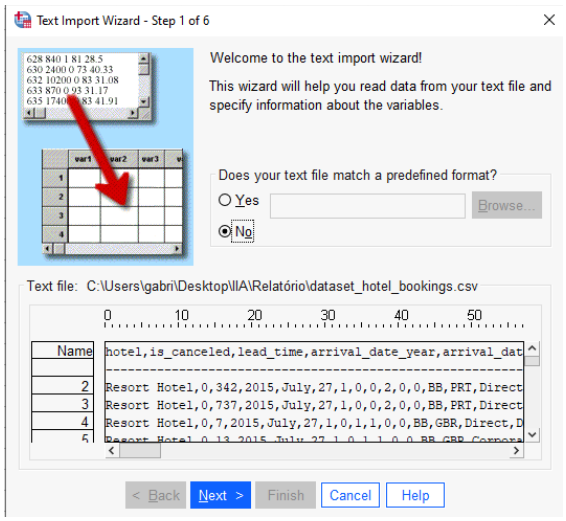
In addition to this software developed by IBM, there are similar applications such as Ubisurvey, SISTAT, SAS. In comparison. The least user friendly should be SAS and Ubisurvey while SISTAT and SPSS provide a more intuitive interface for someone with little to no experience in statistics. Although the SPSS can turn to be costly for individuals or small organizations, it provides a clear and interpretable output of tables and graphs, while Ubisurvey's output is primarily in the former of survey results. However while SISTAT provides clear and organized reports SAS outputs a detailed and customizable report. In terms of capabilities, although in majority, any of these softwares provide a variety of tools for basic and advanced statistical operations, the known as the most capable is sure the SAS software.

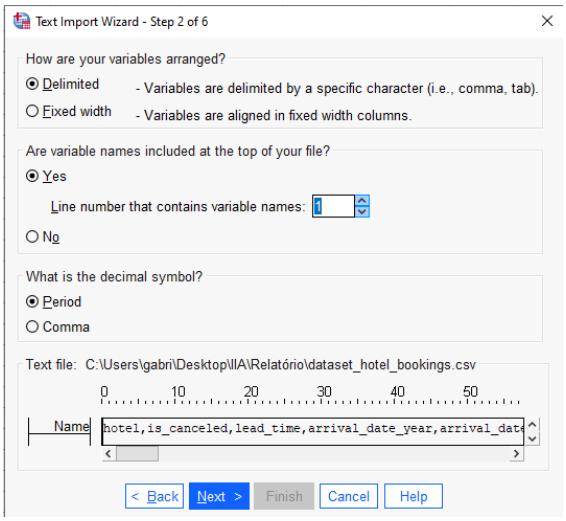
The comparison between the approached softwares can be summarized by the following table:

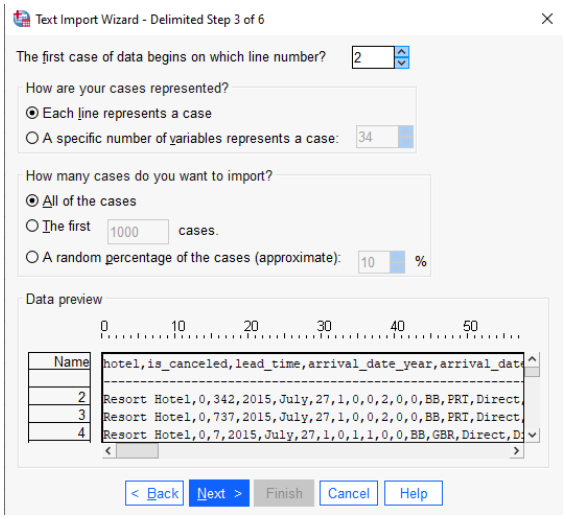
	Easiness	Use	Capabilities	Output
SPSS	High	Fields that require statistical analysis such as social sciences, psychology	Wide range of basic and advanced statistical and data manipulation tools	Clear and interpretable tables and graphs
Ubisurvey	Medium	Survey research and data collection	Provides features to various types of surveys and basic reporting	Primarily in the form of survey results and basic summary statistics.
SISTAT	High	statistical analysis and data visualization	Wide range of statistical tests and procedures and data visualization tools	Clear and organized output
SAS	Low	Data management, advanced analytics and business intelligence	Powerful data manipulation and advanced statistical procedures	Detailed and customizable reports

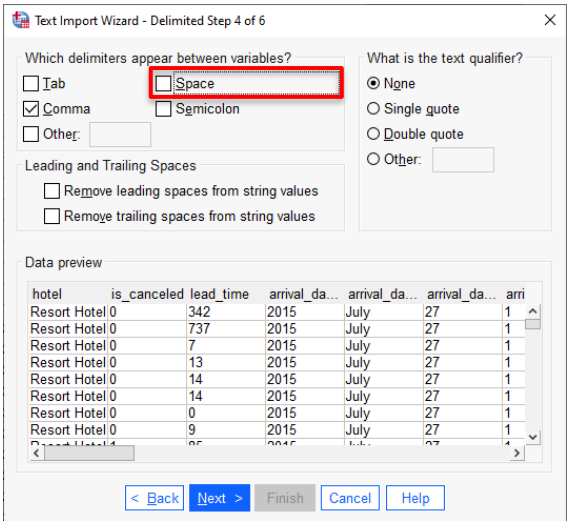
2.2 First impressions in SPSS

First, to open the required dataset on SPSS, after accessing *File>Open File* and selecting the dataset a series of sequential windows will appear. The first one asks if the file has a predefined format, since that in the example it does not is only needed to proceed by clicking *Next*. Then, it will be required to specify how the variables are arranged if their names are included and at which line and what is the decimal symbol (a period or a comma). With the following windows, usually the predefined configurations will be correct, so proceeding with clicking next until finish will not be significant.









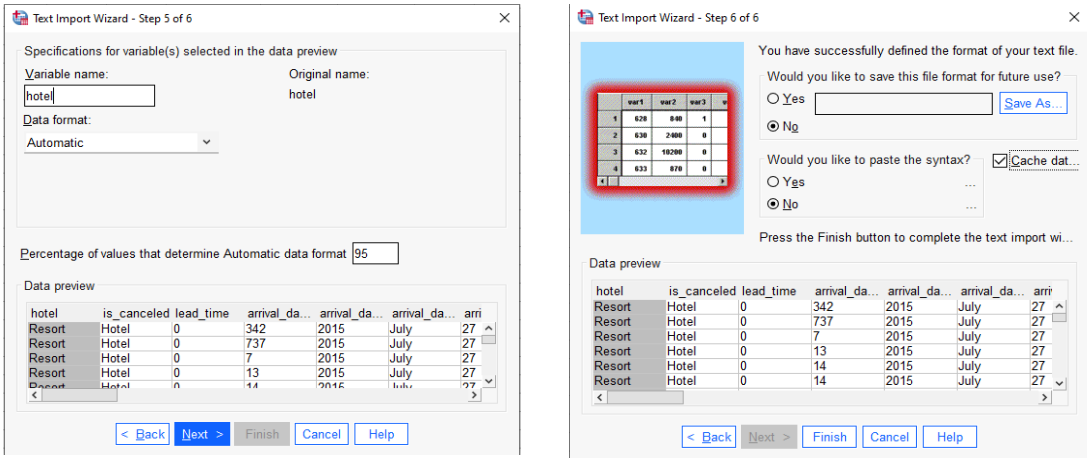


Figure 1 - SPSS first impressions

2.2.1 Measure field

In order to analyse statistics more effectively, it is necessary to classify the type of variables in the dataset to be used. Variable types, in SPSS, can be configured in the *Variable View* environment, that usually is open by default. Then, to change each variable type, simply right-click each line in the *Measure* column and it will appear two or three options, scale, ordinal and nominal. Is very important to select the correct type for each variable since it will affect many of the tests that will be done on the data.

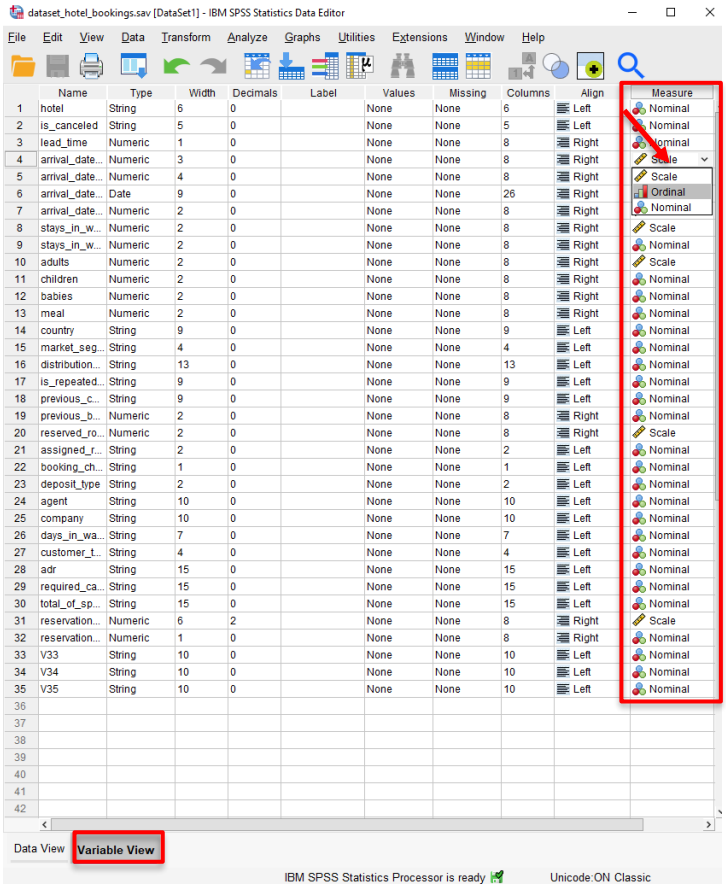


Figure 2 - SPSS Variable View

2.2.2 Value and labels

The *Value* feature, in SPSS, is particularly useful for categorical data with numeric codes that might not be intuitive to interpret. For example, for the *is_canceled* variable, there are two possible values, *1* or *0* for cancelled or not cancelled, respectively, in the Value field is possible to add a label to these values.

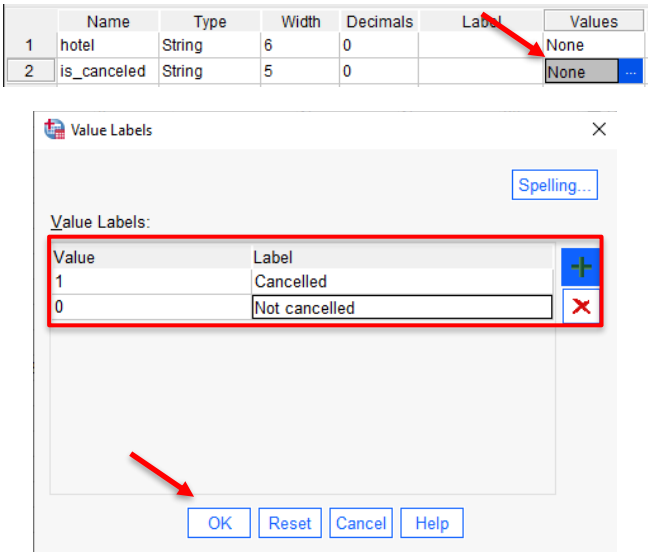


Figure 3 - Value field configuration

2.2.3 Labels

The *Label* field, in SPSS, has a similar goal as the *Value* field, to further inform the user of a variable. However, the given description is relative to the variable purpose and should be descriptive of it, providing a better understanding of the variable objective on the dataset.

	Name	Type	Width	Decimals	Label
1	hotel	String	6	0	
2	is_canceled	String	5	0	
3	lead_time	Numeric	1	0	Days between entering and arrival

Figure 4 - Label field configuration

2.2.4 Result

At the end of the *Measure*, *Value* and *Labels* fields configuration, the dataset is greatly complete and informative providing any user with a easier comprehension of each variable objective and characteristics. After these configurations, all variables should have at least a label and the correct measure type, some of the nominal variables should possess a Value label as well. With the completed configurations, the dataset should have the following aspect:

dataset_hotel_bookings.sav [DataSet2] - IBM SPSS Statistics Data Editor

File Edit View Data Transform Analyze Graphs Utilities Extensions Window Help

	Name	Type	Width	Decimals	Label	Values	Missing	Columns	Align	Measure
1	hotel	String	12	0	if is a Resort or City Hotel	None	None	12	Left	Nominal
2	is_canceled	Numeric	1	0	if a booking was cancelled or not	{0, }...	None	8	Right	Nominal
3	lead_time	Numeric	3	0	Days between entering and arrival	None	None	8	Right	Scale
4	arrival_date...	Numeric	4	0	Year of arrival date	None	None	8	Right	Scale
5	arrival_date...	Date	9	0	Month of arrival date	None	None	20	Right	Ordinal
6	arrival_date...	Numeric	2	0	Day of arrival date	None	None	8	Right	Ordinal
7	arrival_date...	Numeric	2	0	Week number for arrival date	None	None	8	Right	Ordinal
8	stays_in_w...	Numeric	2	0	Weekend nights stayed/booked	None	None	8	Right	Scale
9	stays_in_w...	Numeric	2	0	Week nights stayed/booked	None	None	8	Right	Scale
10	adults	Numeric	2	0	Number of adults in the hotel	None	None	8	Right	Scale
11	children	Numeric	2	0	Number of children in the hotel	None	None	8	Right	Scale
12	babies	Numeric	2	0	Number of babies in the hotel	None	None	8	Right	Scale
13	meal	String	9	0	Meal type	{BB, Bed and Breakfast}...	None	9	Left	Nominal
14	country	String	4	0	Country of origin	None	None	4	Left	Nominal
15	market_seg...	String	13	0	Market segment designation	{TA, Travel Agent}...	None	13	Left	Nominal
16	distribution...	String	9	0	Booking distribution channel	{TA, Travel Agent}...	None	9	Left	Nominal
17	is_repeated...	Numeric	1	0	If a guest is a repeating guest	{0, Not repeated guest}...	None	8	Right	Nominal
18	previous_c...	Numeric	2	0	Previous bookings cancelled	None	None	8	Right	Scale
19	previous_b...	Numeric	2	0	Previous bookings not cancelled	None	None	8	Right	Scale
20	reserved_ro...	String	1	0	Code of room type reserved	None	None	1	Left	Nominal
21	assigned_r...	String	1	0	Code of room type assigned	None	None	1	Left	Nominal
22	booking_ch...	Numeric	2	0	Number of changes made to the...	None	None	8	Right	Scale
23	deposit_type	String	10	0	If the customer made a deposit	None	None	10	Left	Nominal
24	agent	String	4	0	Travel agency ID	None	None	4	Left	Nominal
25	company	String	4	0	Company ID that made the book...	None	None	4	Left	Nominal
26	days_in_wa...	Numeric	3	0	Number of days in the waiting list	None	None	8	Right	Scale
27	customer_t...	String	15	0	Type of booking	None	None	15	Left	Nominal
28	adr	Numeric	6	2	Average Daily Rate	None	None	8	Right	Scale
29	required_ca...	Numeric	1	0	Car parking spaces required pe...	None	None	8	Right	Scale
30	total_of_sp...	Numeric	1	0	Special requests made by the cl...	None	None	8	Right	Scale
31	reservation...	String	9	0	Reservation last status	None	None	9	Left	Nominal
32	reservation...	String	10	0	Date at which the last status wa...	None	None	10	Left	Nominal
33	VAR00001	Numeric	8	2		None	None	8	Right	Nominal
34	VAR00002	Numeric	8	2		None	None	8	Right	Unknown
35	VAR00003	Numeric	8	2		None	None	8	Right	Unknown

Figure 5 - Completed dataset's first configuration

2.3 Missing values

2.3.1 Verify the existence of missing values

Cases with missing values pose an critical challenge because typical modelling procedures simply discard these cases from the analysis. When a missing value is not dependent on other values, the method of listwise deletion is usually adopted and safe, as long as the missing values are less than 5% of the number of cases.

According to SPSS' documentation *the Missing Values option provides two sets of procedures for handling missing values*. It supports *Multiple Imputation* procedures to analyse patterns of missing data geared toward subsequent multiple Imputation of missing values and *Missing Value Analysis* that provides a different set of descriptive tools for missing data analysis, it also includes single imputation methods.

To analyse the missing values question, first, their missingness should be examined by using *Missing Value Analysis*, in the *Analyse* tab, to analyse patterns of missing data and determine the necessity for multiple imputation. Then, the missing imputed data should be multiplied by using *Impute Missing Data Values*. Finally, a procedure that supports multiple imputation data should be used to analyse the complete data.

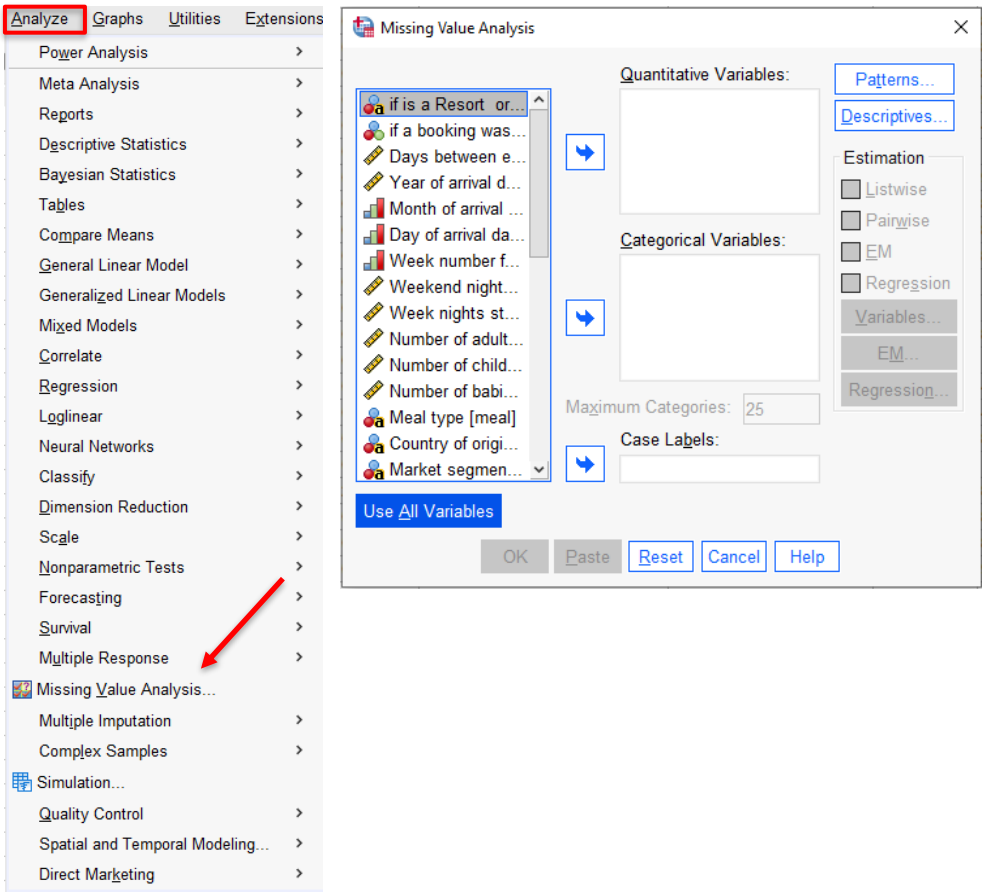


Figure 6 - Missing Values analysis

Univariate Statistics

	N	Mean	Std. Deviation	Missing Count	Missing Percent	No. of Extremes[a] Low	No. of Extremes[a] High
lead_time	119390	104.01	106.863	0	.0	0	6069
arrival_date_year	119390	2016.16	.707	0	.0	0	0
stays_in_weekend_nights	119390	.93	.999	0	.0	0	3458
stays_in_week_nights	119390	2.50	1.908	0	.0	0	3354
adults	119390	1.86	.579	0	.0	403	78
children	119386	.10	.399	4	.0	0	8590
babies	119390	.01	.097	0	.0	0	917
previous_cancellations	119390	.09	.844	0	.0	0	433
previous_bookings_not_canceled	119390	.14	1.497	0	.0	0	1165
booking_changes	119390	.22	.652	0	.0	0	5375
days_in_waiting_list	119390	2.32	17.595	0	.0	0	2806
adr	119390	101.8311	50.53579	0	.0	1962	4694
required_car_parking_spaces	119390	.06	.245	0	.0	0	7416
total_of_special_requests	119390	.57	.793	0	.0	0	2877

a. Number of cases outside the range (Mean - 2*SD, Mean + 2*SD).

Figure 7 - Missing (scale) Values Analysis result

Univariate Statistics

	N	Missing Count	Missing Percent
is_canceled	119390	0	.0
arrival_date_month	119390	0	.0
is_repeated_guest	119390	0	.0

Figure 8 - Missing (ordinal) Values Analysis result

Because no values were counted as missing, there is no need for multiple imputation.

2.3.2 Missing values configurarion tool

In the case of existing missing values, two approaches should be considered:

- **For numerical values** - After accessing the *Variable View* tab, on the right-hand side button of the cell to configure a Missing Values dialogue box will appear. Next, in *Discrete Missing Values*, values like 999 should be entered only in the first box. Finally, in the *Values field*, the same process should be carried out, however, the dialogue box differs graphically, so in *Value Field* the value 999 is entered and in *Label Field* the value *No* is entered. To add this label to the file, *Add* can be used.
- **For strings** - The process is similar for both numeric values and strings, however the value "NR", for example, can be entered in the first Discrete Missing Values text box, bearing in mind that *Missing Values* for strings are case sensitive. After validating this change with the Ok button, the *Values field* can be changed to NR as the value and No as the answer.

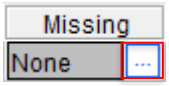


Figure 9 - *Missing Values* button

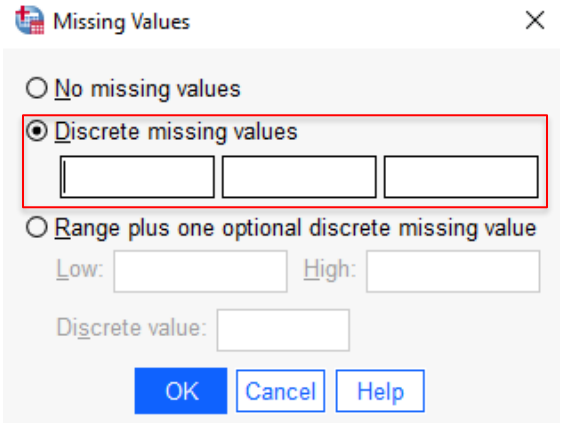


Figure 10 - *Missing Values* configuration window

2.4 Descriptive statistics

To display and interpret descriptive statistics, SPSS, provides a tool present in the *Analyse > Descriptive Statistics > Frequencies* tab. Is important to select what values should be displayed, this configuration can be made on the *Statistics* option. After selecting the desirable field, different will be calculated:

- **Central Tendency** – Corresponding to the mean, median and mode, are highlighted in red in the figure upfront;
- **Dispersion** – Values such as range, variance and standard deviation are highlighted as blue. The coefficient of variation can be calculated by the formula *100*std. Deviation/mean*;
- **Distribution** – Can be used to measure the values distribution with *Kurtose* and their symmetry with *skewness*. These values are highlighted as green.

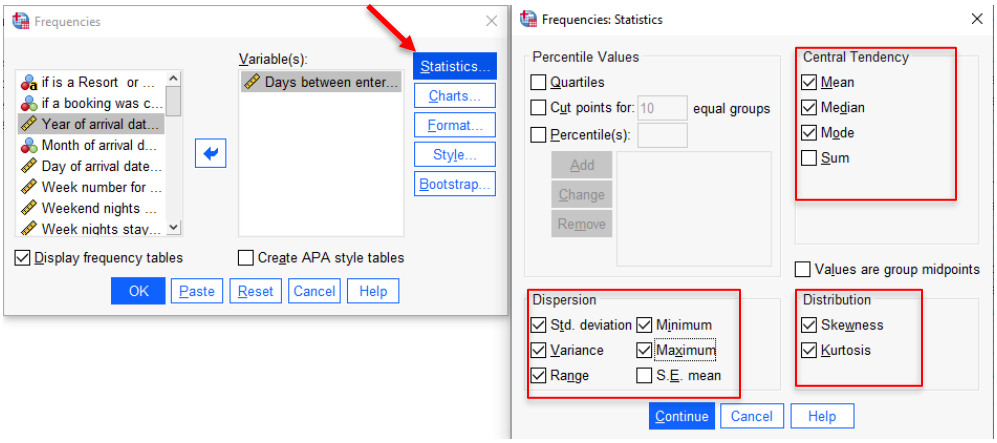


Figure 11 - Descriptive statistics configuration

Statistics

Days between entering and arrival

N	Valid	119390
	Missing	0
Mean		104.01
Median		69.00
Mode		0
Std. Deviation		106.863
Variance		11419.722
Skewness		1.347
Std. Error of Skewness		.007
Kurtosis		1.696
Std. Error of Kurtosis		.014
Range		737
Minimum		0
Maximum		737

Figure 12 - Descriptive statistics on days between entrance and arrival results

2.5 Histogram

To create an histogram with a scale variable, the same one chosen before, SPSS provides the *Graphs > Legacy Dialogs > Histogram* tool. Is possible to verify that the resulting histogram has a almost-like exponential decrease between the lowest number of days between entrance and arrival (0) and highest (737), this means that, in majority, customers have few or no days between these two in stances. However, the mean, because of the discrepancy of values and the outliers, is 104 days which is significant.

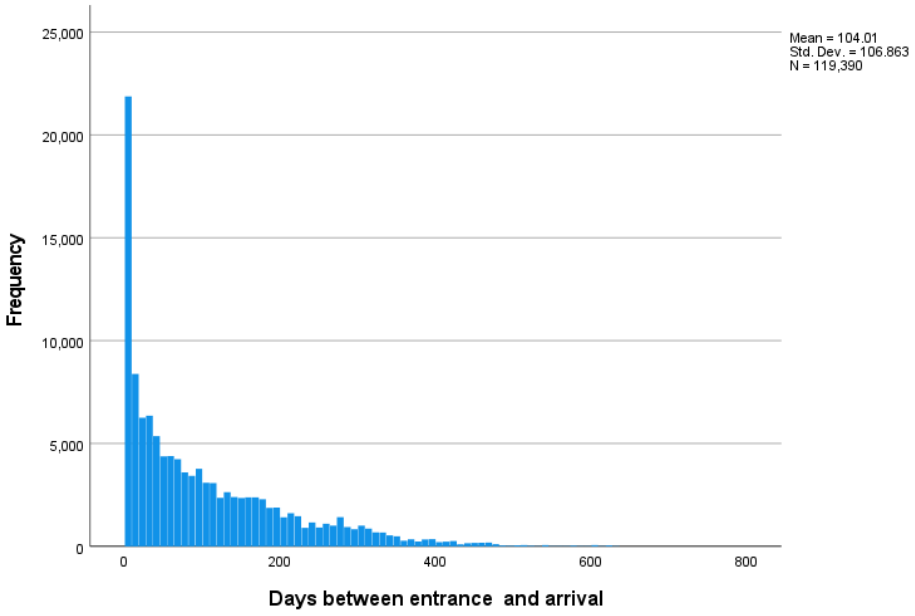


Figure 13 – Histogram on days between entrance and arrival

For this procedure, is possible to make an histogram with the scale variable depending on another variable, for this example, of the categorical type. For example, the repeated costumers had, on average, a lower number of fays between entrance and arrival.

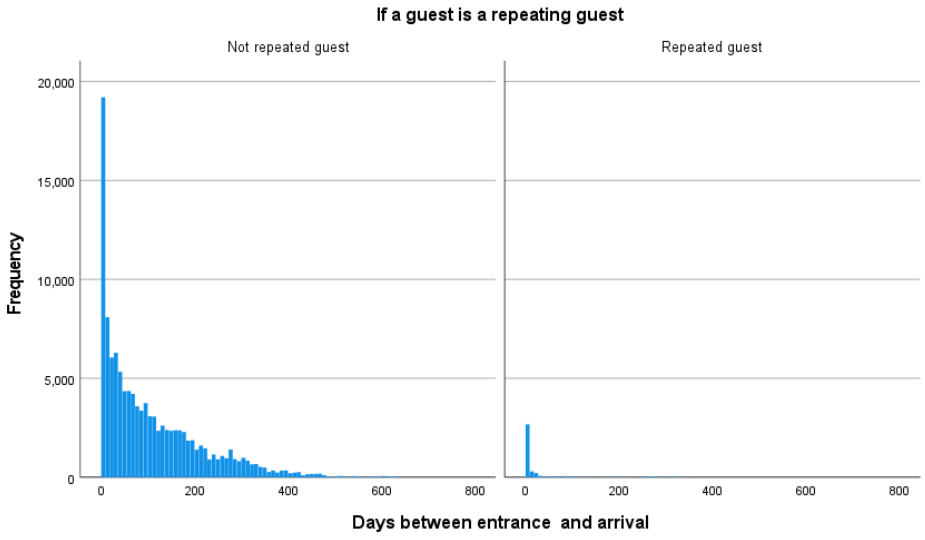


Figure 14 - Histogram on days between entrance and arrival depending on repeated (or not) guest

2.6 Variable recodification

SPSS has the ability to change the values of a variable with the objective of recodification. Variable recodification can be done for various reasons such as simplification of complex data, preparation for analysis or to make it mor interpretable. To recode a variable into a new one, being categorical but separated by ranges of values, SPSS provides the *Transform > Recode into Different Variables* tab.

Since the variable's minimum value is zero, the maximum 737 and the required range is three intervals. The range must be equal to the maximum value divided by three, so the ranges vary in 245. Is important that, after creating the new variable, the value labels are added to it (*Value labels > + > ok)*.

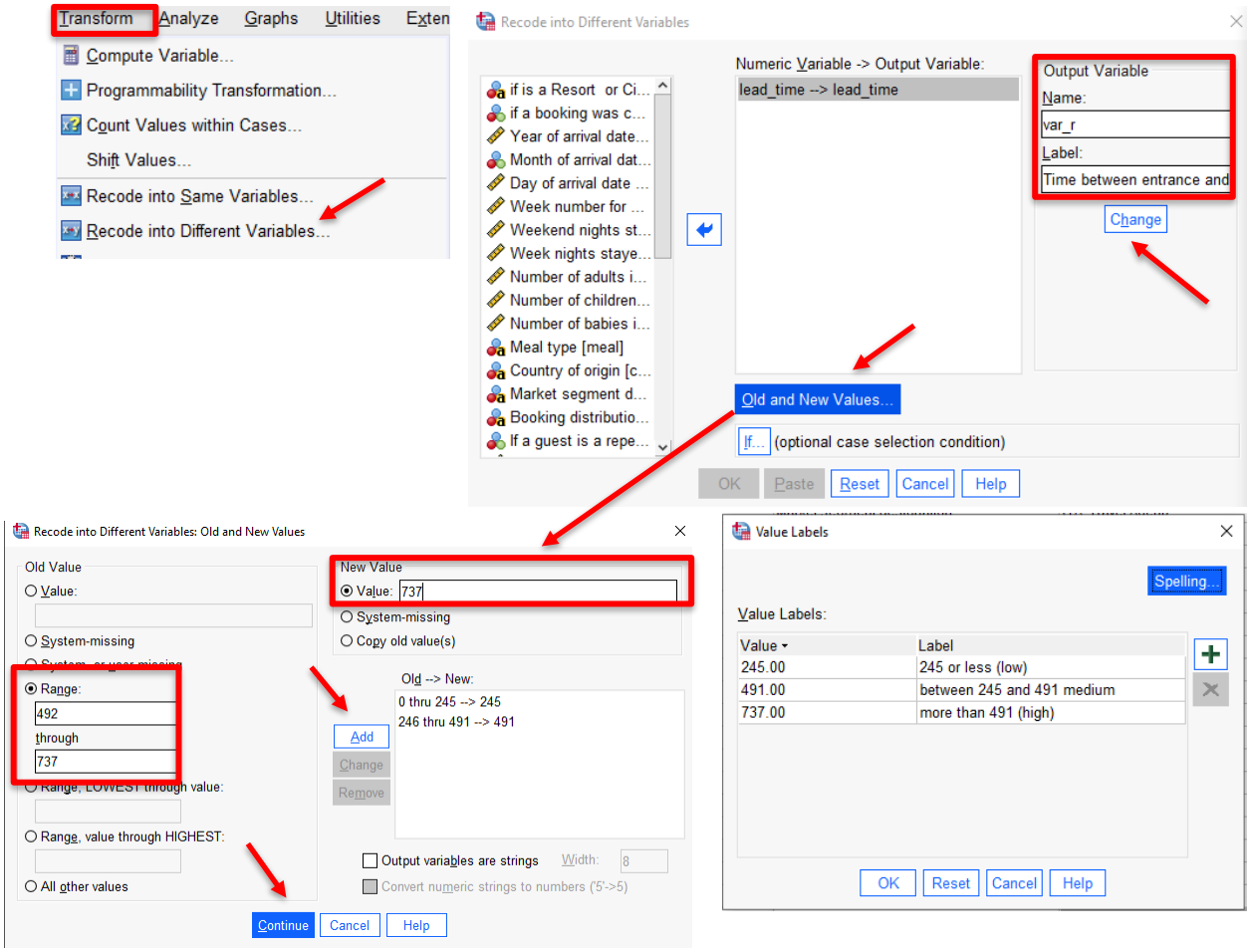


Figure 15 - Variable recodification

2.7 Bar graph

Is also possible to create another type of graphs, for example, bar graphs. A bar graph, created through *Graphs > Legacy Dialogs > Bar > Simple*, with two categorical variables is very intuitive. On the window displayed by this tab, is important to select which variables to represent but what they represent as well, that is, if the representation should be in number of cases, % of cases, etc. To visualize more than one representation multiple graphs need to be created (they will appear on the same window).

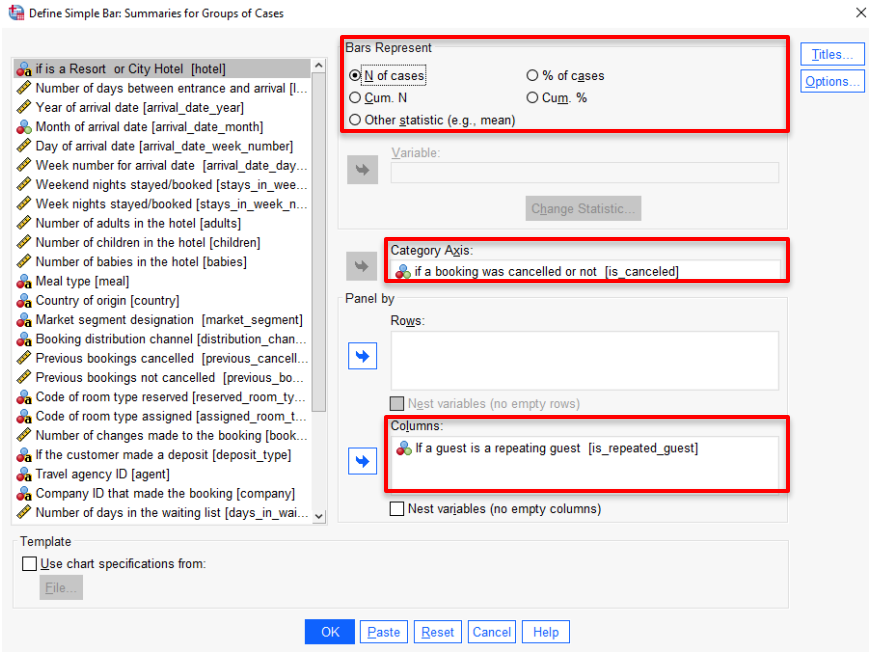


Figure 16 - Bar graph configuration

Figure 17 - Bar graph (number of cases)

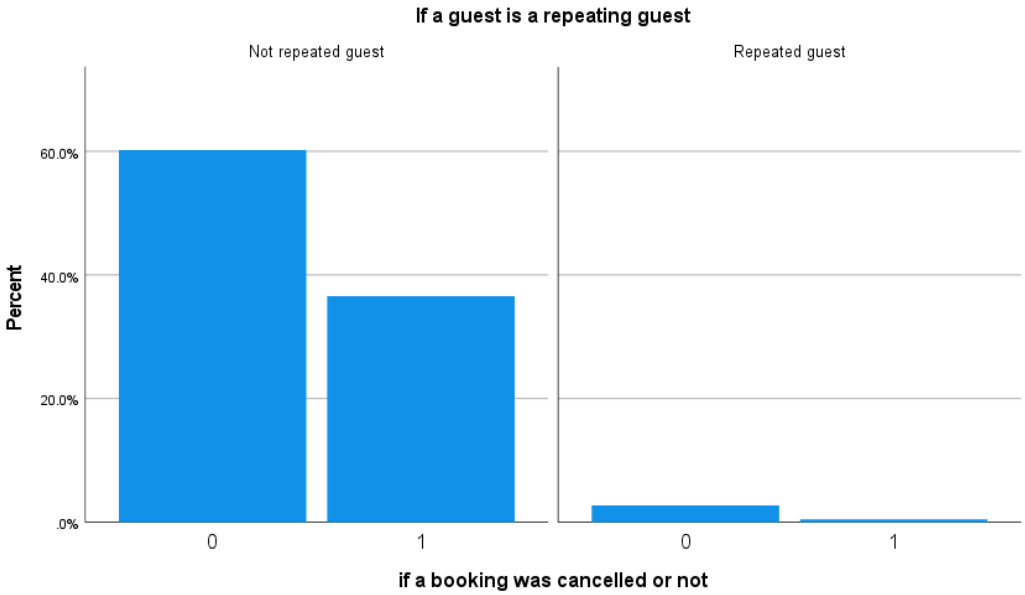


Figure 18 - Bar graph (% of cases)

2.8 Custom table

Is possible to create a table and display values such as minimum, maximum, mean and standard deviation, this tool is called *Custom Table* and is accessed through the *Analyse > Tables > Custom Tables* tab. By opening this tab is important to select one or more variables, in this case a scale variable (Number of days between entrance and arrival) was selected in depending on a categorical variable (If booking was cancelled). Is important to check, in *Summary Statistics*, if the desired values to be displayed are selected.

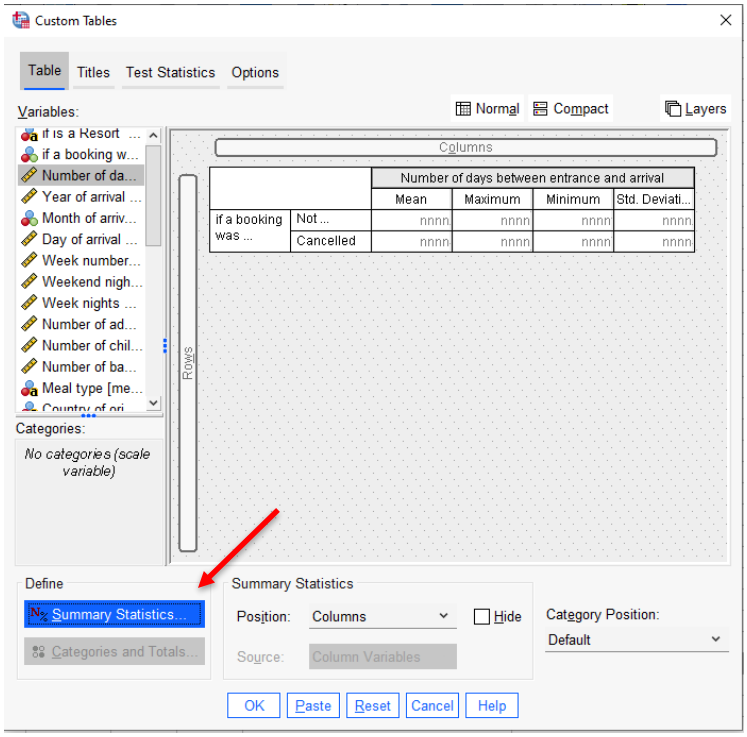


Figure 19 - Custom tables configuration

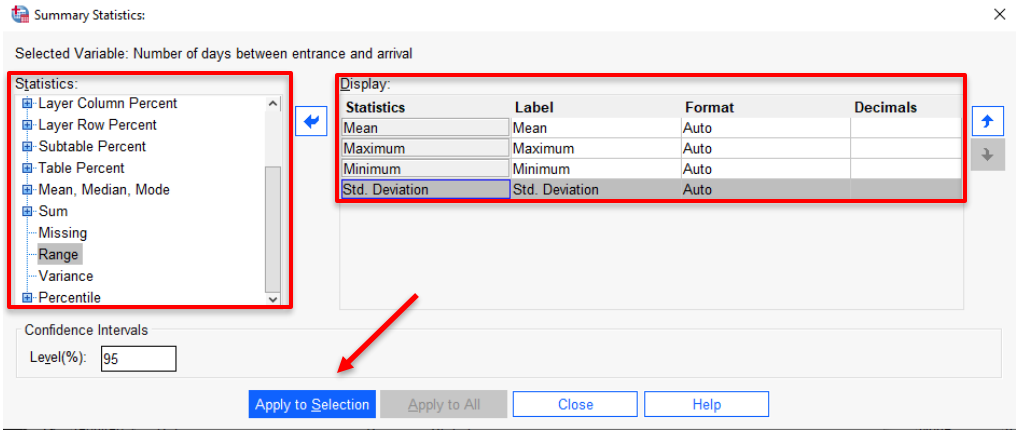


Figure 20 - Custom tables configuration (values to display)

		Number of days between entrance and arrival			
		Mean	Maximum	Minimum	Standard Deviation
if a booking was cancelled or not	0	80	737	0	91
	Not cancelled	.	.	.	.
	Cancelled	.	.	.	.
	1	145	629	0	119

Figure 21 - Custom table

2.9 Boxplot

Another way to represent the previously analysed data, is in a boxplot. A provides a summary of a graphical representation of a dataset. It displays critical statistics such as minimum and maximum, as well as first, second and third quartile. To create a boxplot in SPSS, with *Graphs > Legacy Dialogs > Boxplot*, by selecting an independent variable and another for the category axis, SPSS will display a table and a boxplot representing the selected data. With the results is possible to verify that the interquartile range is on the lower side of the values (days), meaning that a bigger quantity of bookings with a lower temporal interval between arrival and entrance were less cancelled as well.

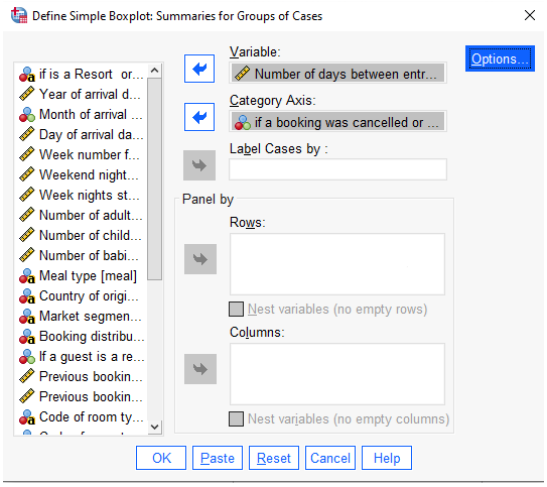


Figure 22 - Boxplot configurations

if a booking was cancelled or not

Case Processing Summary

	if a booking was cancelled or not	Cases					
		Valid		Missing		Total	
		N	Percent	N	Percent	N	Percent
Number of days between entrance and arrival	0	75166	100.0%	0	0.0%	75166	100.0%
	1	44224	100.0%	0	0.0%	44224	100.0%

Number of days between entrance and arrival

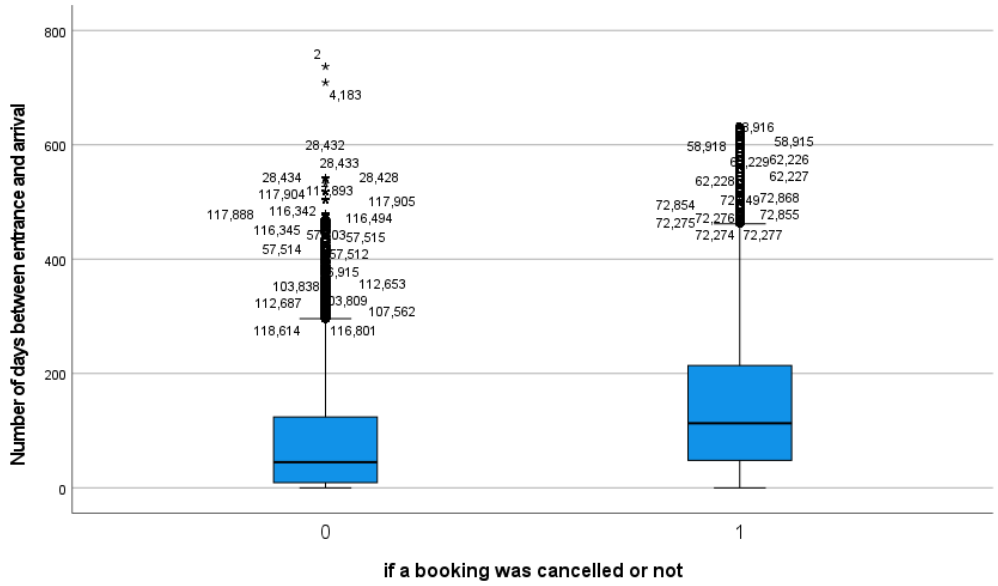


Figure 23 - Boxplot

3 Exploratory data analysis

3.1 Independence tests

Independence tests, in statistics, are used to measure the relation of different values, even evaluate their independence. There are many types of these tests, chi-squared, fisher, and likelihood, being the most popular.

3.1.1 Pearsons's chi-squared test (x2 test)

Is a non-parametric test that verifies the independence of two categorical variables. However, it doesn't give a relation degree, the p-value (sig) only is significant if is bigger or smaller than 0.05. If this value is lower than, the variables are related and other tests are required to measure the level of dependency.

It has two assumptions to comply to, there must be less than 20% of cell with the expected value lower than 5 and no cell must have a value lower than 1. In other words, the least number of values that a cell should have is 5, unless there are less than 20 of them with that amount. To solve these assumptions there may be an attempt to aggregate or to create smaller classes.

3.1.2 Fisher's exact test

In the case of chi-squared not being applicable, for example, the sample being too small, this test provides a more computationally intensive alternative. Most commonly used for 2x2 tables.

3.1.3 likelihood ratio test

The Likelihood ratio test (LRT) tests the fit between two nested models, usually a simpler such as a null hypothesis and a more complex one like an alternative hypothesis. This statistical hypothesis test is used for likelihood-based estimation. In other words, the LRT tests, between two models (one complex and one simple) which one explains the data better. This way, the higher likelihood value, the better the model explains the data.

3.2 Association tests

Since the independence tests, like chi-squared, only show if two variables are independent, in the case of being dependent association tests must be used. Association measures help on the evaluation of the relative intensity of a statistically significant relationship. There are three popular tests for nominal/ordinal type variables, Phi, Cramer's V and Contingency coefficient. Each of these tests give results between 0 and 1 where 1 is a perfect relationship and 0 is no relation at all.

Value	Association degree
0.9+	Very high
0.7 – 0.9	High
0.5 – 0.7	Moderate
0.3 – 0.5	Weak
0 – 0.3 (positive or negative)	insignificant

Figure 24 - Degrees of association

3.3 Crosstabs table

To measure the kind of relationship between two categorical variables, in SPSS, crosstabs can be applied through the *Analyse > Descriptive Statistics > Crosstabs* tab. By doing independence and association tests on the results the dependence between the two variables can be interpreted. Is important to, in the *crosstabs* window, select which tests to do in the *Statistics button*.

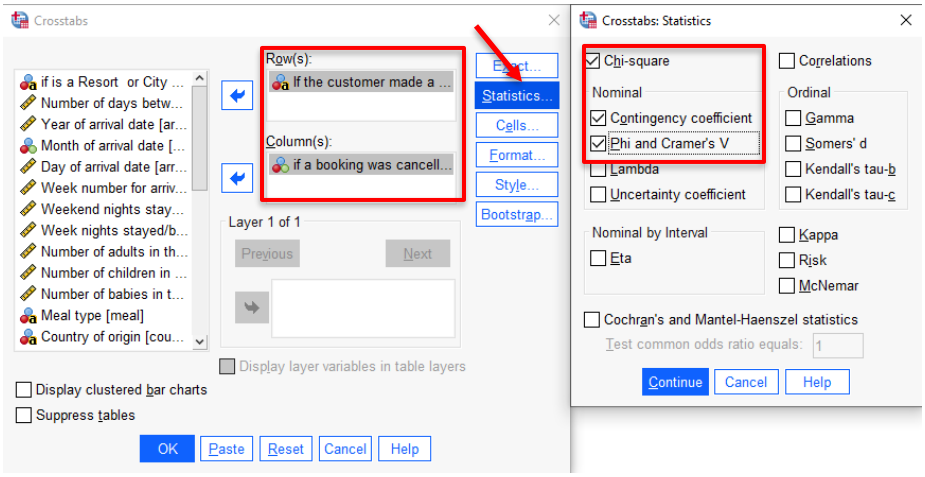


Figure 25 - Crosstabs table configuration

After the configurations, the results of the test will be displayed. As interpretable by the figure below, the selected variables are dependent since the chi-square value is high. However, their relation is weak since its value is between 0.3 and 0.5.

If the customer made a deposit * if a booking was cancelled or not Crosstabulation

Count

		if a booking was cancelled or not		
		0	1	Total
If the customer made a deposit	No Deposit	74947	29694	104641
	Non Refund	93	14494	14587
	Refundable	126	36	162
Total		75166	44224	119390

Chi-Square Tests

	Value	df	Asymptotic Significance (2-sided)
Pearson Chi-Square	27677.329[a]	2	.000
Likelihood Ratio	31268.621	2	.000
N of Valid Cases	119390		

a. 0 cells (0.0%) have expected count less than 5. The minimum expected count is 60.01.

Symmetric Measures

		Value	Approximate Significance
Nominal by Nominal	Phi	.481	.000
	Cramer's V	.481	.000
	Contingency Coefficient	.434	.000
N of Valid Cases		119390	

Figure 26 – Crosstabs table (and tests) results

3.3.1 Select cases

In SPSS the *Select Cases* tool (*Data > Selected Cases > If condition is satisfied*) provides several methods for subgroup of cases selection based on criteria that includes variables and complex expressions. To use a conditional expression to select case, if the result of the conditional expression is true, the case is selected. If the result is false or missing, the case is not selected.

It is also possible to select random samples of cases. These criteria include Variable values and ranges, Date and time ranges, Case (row) numbers, Arithmetic expressions, Logical expressions and Functions. To use all cases the filer *case* can be turned off.

For example, is possible to obtain the results of the previous subtopic but satisfying a condition, condition that, in this case is that the entry date year must be bigger than 2015 and the results differ, in number of cases of course, but statistically as well the relation degree turning out lower for the more recent years.

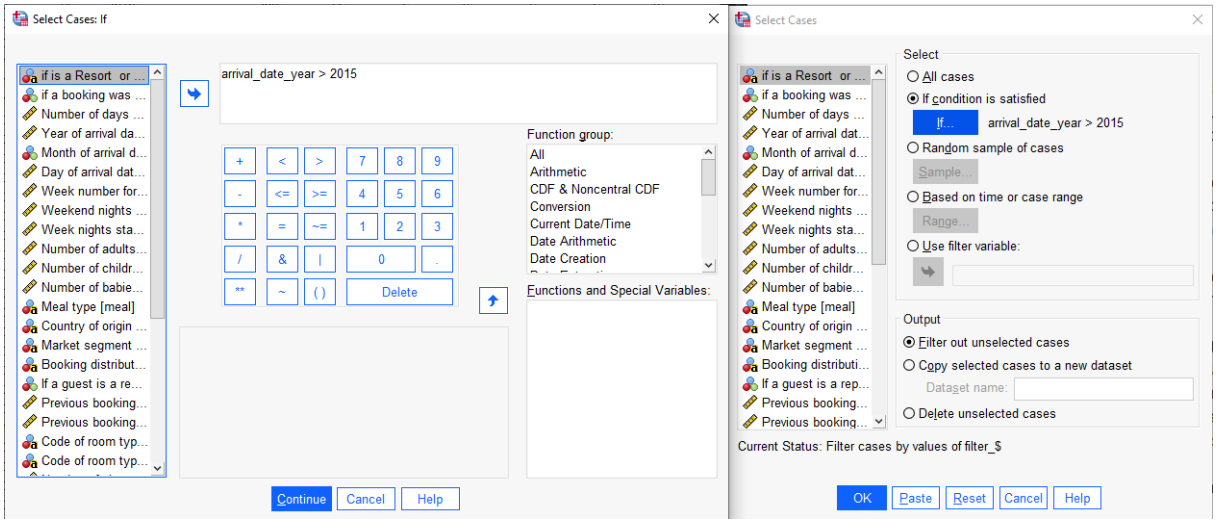


Figure 27 - Selected cases condition

If the customer made a deposit * if a booking was cancelled or not Crosstabulation

Count

		if a booking was cancelled or not		
		0	1	Total
If the customer made a deposit	No Deposit	61106	25210	86316
	Non Refund	84	10836	10920
	Refundable	122	36	158
Total		61312	36082	97394

Chi-Square Tests

	Value	df	Asymptotic Significance (2-sided)
Pearson Chi-Square	20394.108[a]	2	.000
Likelihood Ratio	22982.816	2	.000
N of Valid Cases	97394		

a. 0 cells (0.0%) have expected count less than 5. The minimum expected count is 58.53.

Symmetric Measures

		Value	Approximate Significance
Nominal by Nominal	Phi	.458	.000
	Cramer's V	.458	.000
	Contingency Coefficient	.416	.000
N of Valid Cases		97394	

Figure 28 – Crosstab with the Selected cases condition

3.3.2 Layer (groups)

In the SPSS' crosstabs tool is also possible to add another variable with the goal of creating a layer, a subgroup to divide the results of the table. With the layer applied is comprehensible that the results now are divided into two subgroups, one for resort hotels and another for city hotels. This way, is possible to verify that the previous variable relation is bigger for city hotels and smaller for resort hotels in comparative to the results without this division.

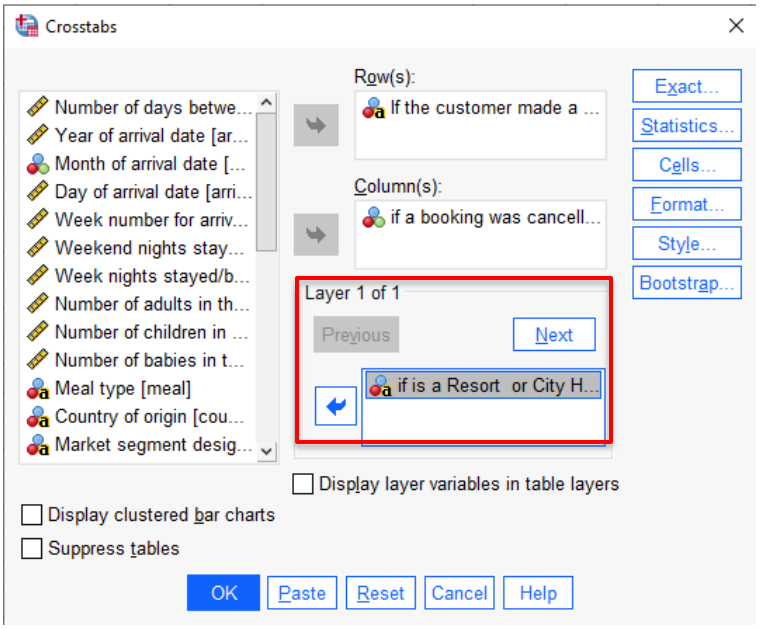


Figure 29 - Crosstabs layer configuration

If the customer made a deposit * if a booking was cancelled or not * if is a Resort or City Hotel Crosstabulation

Count

if is a Resort or City Hotel			if a booking was cancelled or not 0	1	Total
City Hotel	If the customer made a deposit	No Deposit	46198	20244	66442
		Non Refund	24	12844	12868
		Refundable	6	14	20
	Total		46228	33102	79330
Resort Hotel	If the customer made a deposit	No Deposit	28749	9450	38199
		Non Refund	69	1650	1719
		Refundable	120	22	142
	Total		28938	11122	40060
Total	If the customer made a deposit	No Deposit	74947	29694	104641
		Non Refund	93	14494	14587
		Refundable	126	36	162
	Total		75166	44224	119390

Figure 30 - Crosstabs layer result (count)

Chi-Square Tests

if is a Resort or City Hotel		Value	df	Asymptotic Significance (2-sided)
City Hotel	Pearson Chi-Square	21325.698[b]	2	.000
	Likelihood Ratio	25723.971	2	.000
	N of Valid Cases	79330		
Resort Hotel	Pearson Chi-Square	4174.258[c]	2	.000
	Likelihood Ratio	3885.153	2	.000
	N of Valid Cases	40060		
Total	Pearson Chi-Square	27677.329[a]	2	.000
	Likelihood Ratio	31268.621	2	.000
	N of Valid Cases	119390		

a. 0 cells (0.0%) have expected count less than 5. The minimum expected count is 60.01.

b. 0 cells (0.0%) have expected count less than 5. The minimum expected count is 8.35.

c. 0 cells (0.0%) have expected count less than 5. The minimum expected count is 39.42.

Symmetric Measures

if is a Resort or City Hotel			Value	Approximate Significance
City Hotel	Nominal by Nominal	Phi	.518	.000
		Cramer's V	.518	.000
		Contingency Coefficient	.460	.000
	N of Valid Cases		79330	
Resort Hotel	Nominal by Nominal	Phi	.323	.000
		Cramer's V	.323	.000
		Contingency Coefficient	.307	.000
	N of Valid Cases		40060	
Total	Nominal by Nominal	Phi	.481	.000
		Cramer's V	.481	.000
		Contingency Coefficient	.434	.000
	N of Valid Cases		119390	

Figure 31 - Crosstabs layer result (tests)

3.4 Correlation measures

Correlation is the degree to which two variables are linearly related. It assists the understanding of how changes in a variable affect other variables. There are various correlation measures, however, for this topic the most relevant are the Pearson and Spearman correlations.

The Pearson correlation is the most commonly used type of correlation and, by giving values from -1 to 1 where 0 is no correlation at all, -1 is a negative and 1 is a linear relationship. The Pearson correlation coefficient requires a normal data distribution and is suitable for tests between quantitative variables.

The Spearman correlation assesses the strength and direction of a monotonic relationship. In contrary to Pearson id does not assume a linear relationship, making it more suitable for ordinal or continuous variables. It does not impose restrictions on the sample distribution.

3.4.1 Normality assumptions

As already mentioned, the Pearson test requires that certain assumptions are fulfilled such as that the variables follow a normal distribution, there is a linear relationship between the variables and that the variance of one variable is the same for all variables.

To verify these assumptions, SPSS provides the *Analyze > Descriptive Statistics > Explore* tool. After this process, there will be displayed three values in addition to the normality tests. *Case Processing Summary* points out the number of cases that have complete data (Valid N) and the number of missing cases. The *Descriptives* table displays the total number of cases (N), a mean, a std. deviation, a maximum and minimum, as well as the percentiles. The last given result, *Stem-and-Leaf Plots* is the visual representation of the dataset's distribution. Each data point is divided into the first digit(s) (stem) and the last digit(s).

The normality null hypothesis assumes that data comes from a normal distribution. Given the normality tests results, is possible to verify that every tested variable may not have normal distribution since all values have a p-value < 0.05. This way, the null hypothesis is rejected, suggesting that the data is probably not normally distributed. The normality can also be observed with histograms.

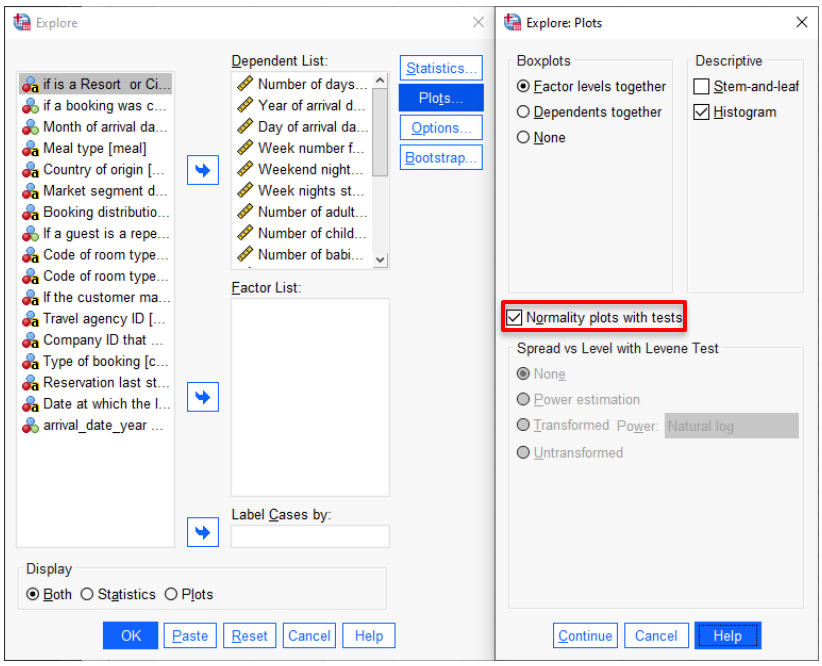


Figure 32 – *Explore* tool configuration

Tests of Normality

	Kolmogorov-Smirnov[a]		
	Statistic	df	Sig.
Number of days between entrance and arrival	.165	119386	.000
Year of arrival date	.247	119386	.000
Day of arrival date	.053	119386	.000
Week number for arrival date	.075	119386	.000
Weekend nights stayed/booked	.259	119386	.000
Week nights stayed/booked	.203	119386	.000
Number of adults in the hotel	.402	119386	.000
Number of children in the hotel	.531	119386	.000
Number of babies in the hotel	.525	119386	.000
Previous bookings cancelled	.487	119386	.000
Previous bookings not cancelled	.506	119386	.000
Number of changes made to the booking	.481	119386	.000
Number of days in the waiting list	.522	119386	.000
Average Daily Rate	.085	119386	.000
Car parking spaces required per client	.538	119386	.000
Special requests made by the client	.353	119386	.000
Number of days between entrance and arrival (range)	.522	119386	.000

a. Lilliefors Significance Correction

Figure 33 – *Normality tests*

3.4.2 Pearson test

The existence of correlations between the quantitative variables of the dataset can be identified through *Analyse > Correlate > Bivariate*. By using this tool, SPSS will carry out with the selected (Pearson) test and display the correlation values. Knowing that that the values of 1 and -1 are, respectively, a perfect positive or negative linear relationship and 0 is no relation at all, the results given by SPSS can be analysed and conclusions can be withdrawn from them. Giving that, in the results, the highest values for Pearson positive values are no bigger than 0.20 and the negative no smaller than -0.20, there are no significant linear relationships known.

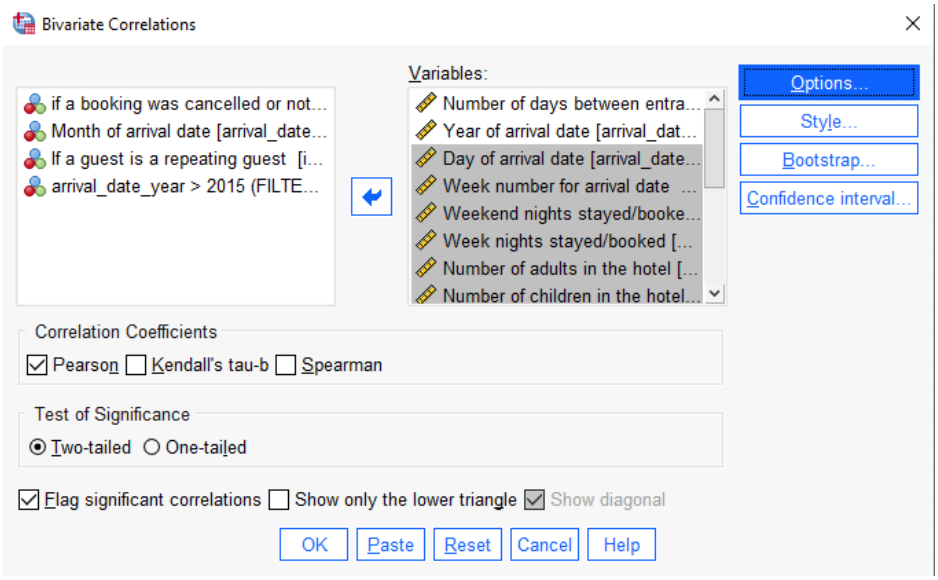


Figure 34 - Pearson test configuration

Correlations

		Number of days between entrance and arrival	Year of arrival date	Day of arrival date	Week number for arrival date	Weekend nights stayed/booked	Week nights stayed/booked	Number of adults in the hotel	Number of children in the hotel	Number of babies in the hotel	Previous bookings cancelled	Previous bookings not cancelled	Number of changes made to the booking	Number of days in the waiting list	Average Daily Rate	Car parking spaces required per client	Special requests made by the client
Number of days between entrance and arrival	Pearson Correlation	1	.040**	.127**	.002	.086**	.166**	.120**	-.038**	-.021**	.086**	-.074**	.000	.170**	-.063**	-.116**	-.096**
	Sig. (2-tailed)		<.001	.000	.433	<.001	.000	.000	<.001	<.001	<.001	<.001	.959	.000	<.001	.000	<.001
	N	119390	119390	119390	119390	119390	119390	119390	119386	119390	119390	119390	119390	119390	119390	119390	119390
Year of arrival date	Pearson Correlation	.040**	1	-.541**	.000	.021**	.031**	.030**	.055**	-.013**	-.120**	.029**	.031**	-.056**	.198**	-.014**	.109**
	Sig. (2-tailed)	<.001		.000	.939	<.001	<.001	<.001	<.001	<.001	.000	<.001	<.001	<.001	.000	<.001	.000
	N	119390	119390	119390	119390	119390	119390	119390	119386	119390	119390	119390	119390	119390	119390	119390	119390
Day of arrival date	Pearson Correlation	.127**	-.541**	1	.067**	.018**	.016**	.026**	.006	.010**	.036**	-.021**	.006	.023**	.076**	.002	.026**
	Sig. (2-tailed)	.000	.000		<.001	<.001	<.001	<.001	.057	<.001	<.001	<.001	.057	<.001	<.001	.507	<.001
	N	119390	119390	119390	119390	119390	119390	119390	119386	119390	119390	119390	119390	119390	119390	119390	119390
Week number for arrival date	Pearson Correlation	.002	.000	.067**	1	-.016**	-.028**	-.002	.015**	.000	-.027**	.000	.011**	.023**	.030**	.009**	.003
	Sig. (2-tailed)	.433	.939	<.001		<.001	<.001	.588	<.001	.937	<.001	.917	<.001	<.001	<.001	.003	.290
	N	119390	119390	119390	119390	119390	119390	119390	119386	119390	119390	119390	119390	119390	119390	119390	119390
Weekend nights stayed/booked	Pearson Correlation	.086**	.021**	.018**	-.016**	1	.499**	.092**	.046**	.018**	-.013**	-.043**	.063**	-.054**	.049**	-.019**	.073**
	Sig. (2-tailed)	<.001	<.001	<.001	<.001		.000	<.001	<.001	<.001	<.001	<.001	<.001	<.001	<.001	<.001	<.001
	N	119390	119390	119390	119390	119390	119390	119390	119386	119390	119390	119390	119390	119390	119390	119390	119390
Week nights stayed/booked	Pearson Correlation	.166**	.031**	.016**	-.028**	.499**	1	.093**	.044**	.020**	-.014**	-.049**	.096**	-.002	.065**	-.025**	.068**
	Sig. (2-tailed)	.000	<.001	<.001	<.001	.000		<.001	<.001	<.001	<.001	<.001	<.001	.485	<.001	<.001	<.001
	N	119390	119390	119390	119390	119390	119390	119390	119386	119390	119390	119390	119390	119390	119390	119390	119390
Number of adults in the hotel	Pearson Correlation	.120**	.030**	.026**	-.002	.092**	.093**	1	.030**	.018**	-.007*	-.108**	-.052**	-.008**	.231**	.015**	.123**
	Sig. (2-tailed)	.000	<.001	<.001	.588	<.001	<.001		<.001	<.001	.020	<.001	<.001	.004	.000	<.001	.000
	N	119390	119390	119390	119390	119390	119390	119390	119386	119390	119390	119390	119390	119390	119390	119390	119390
Number of children in the hotel	Pearson Correlation	-.038**	.055**	.006	.015**	.046**	.044**	.030**	1	.024**	-.025**	-.021**	.049**	-.033**	.325**	.056**	.082**
	Sig. (2-tailed)	<.001	<.001	.057	<.001	<.001	<.001	<.001		<.001	<.001	<.001	<.001	<.001	.000	<.001	<.001
	N	119386	119386	119386	119386	119386	119386	119386	119386	119386	119386	119386	119386	119386	119386	119386	119386
Number of babies in the hotel	Pearson Correlation	-.021**	-.013**	.010**	.000	.018**	.020**	.018**	.024**	1	-.008**	-.007*	.083**	-.011**	.029**	.037**	.098**
	Sig. (2-tailed)	<.001	<.001	<.001	.937	<.001	<.001	<.001	<.001		.010	.024	<.001	<.001	<.001	<.001	<.001
	N	119390	119390	119390	119390	119390	119390	119390	119386	119390	119390	119390	119390	119390	119390	119390	119390
Previous bookings cancelled	Pearson Correlation	.086**	-.120**	.036**	-.027**	-.013**	-.014**	-.007*	-.025**	-.008**	1	.153**	-.027**	.006*	-.066**	-.018**	-.048**
	Sig. (2-tailed)	<.001	.000	<.001	<.001	<.001	<.001	.020	<.001	.010		.000	<.001	.040	<.001	<.001	<.001
	N	119390	119390	119390	119390	119390	119390	119390	119386	119390	119390	119390	119390	119390	119390	119390	119390

Figure 35 - Pearson test result

3.4.3 Spearman test

To execute a spearman test, the process is similar (*Analyse* > *Correlate* > *Bivariate*), the difference is in the configuration window. Instead if checking the *Pearson* test box, the *Spearman* must be selected. Is important to not forget that the *Spearman* test is suitable for ordinal variables. The results did not appoint to any kind of linear relationship since there are no negative correlation values lower than -0.10 and no positive correlation values bigger than 0.10.

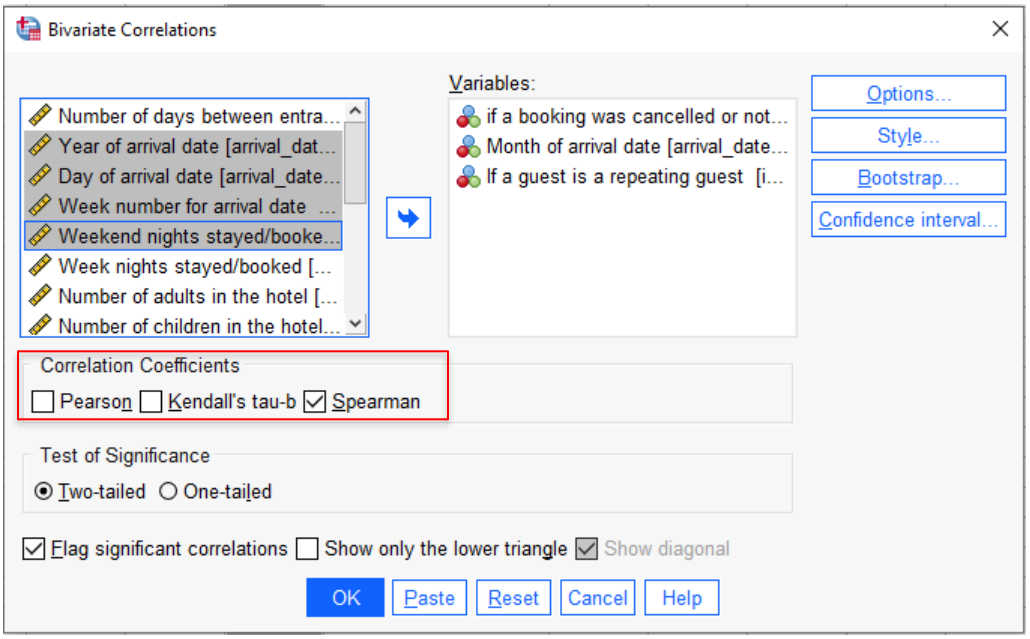


Figure 36 - Spearman test configuration

Correlations

			if a booking was cancelled or not	Month of arrival date	If a guest is a repeating guest
Spearman's rho	if a booking was cancelled or not	Correlation Coefficient	1.000	.010**	-.085**
		Sig. (2-tailed)	.	<.001	<.001
		N	119390	119390	119390
	Month of arrival date	Correlation Coefficient	.010**	1.000	-.030**
		Sig. (2-tailed)	<.001	.	<.001
		N	119390	119390	119390
	If a guest is a repeating guest	Correlation Coefficient	-.085**	-.030**	1.000
		Sig. (2-tailed)	<.001	<.001	.
		N	119390	119390	119390

**. Correlation is significant at the 0.01 level (2-tailed).

Figure 37 - Spearman test result

4 Parametric and non-parametric tests

The type of statistical operations approached in the previous topics, is what is called of descriptive statistics. This type if statistical analysis, provides tools to describe a sample. However, there is the Inferential type statistics which analyses a sample of a population to make a statement about it.

The inferential statistics offer two types of tests, parametric and non-parametric. Parametric tests possess assumptions about the population such as that the values are normally distributed in the population. In addition to normality distribution, these tests assume that the variables on the dataset are of the scale or ordinal, with at least 7 levels, type. Non-parametric tests do not assume anything about a population.

Between parametric and non-parametric tests, are different types of tests for different types of situations. For each parametric test, there is a non-parametric alternative, shown in the following table:

Situation	Parametric	Non-parametric
Describe one group	Mean	Median
Compare one group to a hypothetical value	1-sample t-test	Wilcoxon test (or chi-square)
Compare two unpaired groups	2-sample unpaired t-test	Mann-Whitney test
Compare two paired groups	2-sample paired t-test	Wilcoxon test
Compare three or more unmatched groups	One-way ANOVA	Kruskal-Wallis test (or chi-square)
Predict values from measures	Linear or nonlinear regression	Nonparametric regression

Figure 38 - Parametric vs Non-parametric tests

4.1 Normality tests

To verify a sample normal distribution there are several steps to make:

1. **Normality tests** – Guaranty normality (H0 is that the distribution is normal)
    a. For more than 50 cases the Kolmogorov–Smirnov test is used;
    b. For less than 50 cases the Shapiro–Wilk method is adopted.
2. **Asymmetry measures** – Slightly guaranties normality
    a. Asymmetry coefficient/Skewness test – Values between -3 and 3 are accepted as symmetric;
    b. Flatness/sharpness coefficient/Kurtosis – Values close to 3 are acceptable.
3. **Central Limit Theorem** – Decent guarantee
    a. If a sample has more than 30 cases, normality is assumed.

To perform these tests, on SPSS, this software provides two tools. Through ***Analyse > Descriptive > Explore > Plots > Normality plots with tests*** is possible to test the normality of the sample through the step 1. approach. To calculate the asymmetry measures, SPSS offers the ***Analyse > Descriptive > Freq > Stat*** tab.

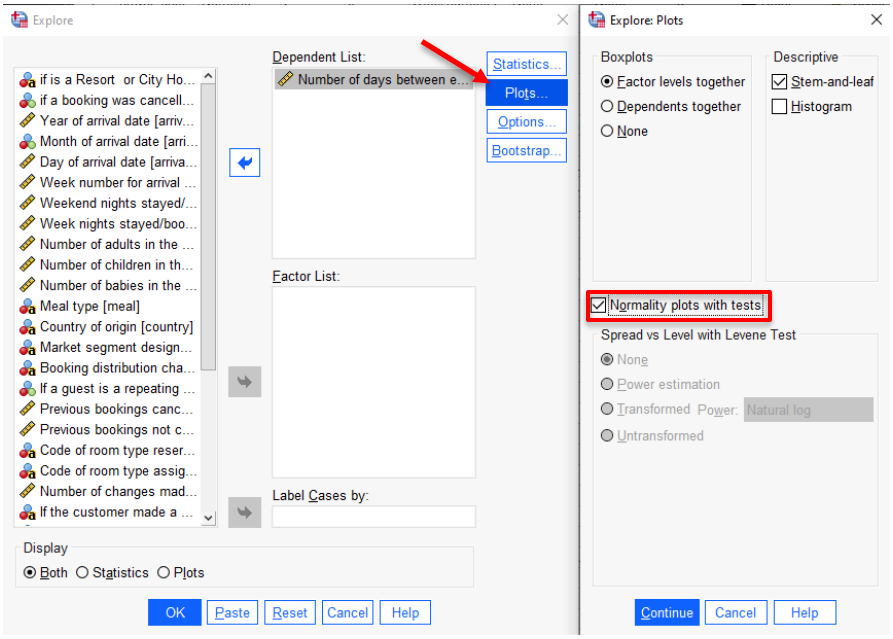


Figure 39 - Normality tests configuration

Descriptives

			Statistic	Std. Error
Number of days between entrance and arrival	Mean		104.01	.309
	95% Confidence Interval for Mean	Lower Bound	103.41	
		Upper Bound	104.62	
	5% Trimmed Mean		93.74	
	Median		69.00	
	Variance		11419.722	
	Std. Deviation		106.863	
	Minimum		0	
	Maximum		737	
	Range		737	
	Interquartile Range		142	
	Skewness		1.347	.007
	Kurtosis		1.696	.014

Tests of Normality

	Kolmogorov-Smirnov[a]		
	Statistic	df	Sig.
Number of days between entrance and arrival	.165	119390	.000

a. Lilliefors Significance Correction

Figure 40 - Normality tests results

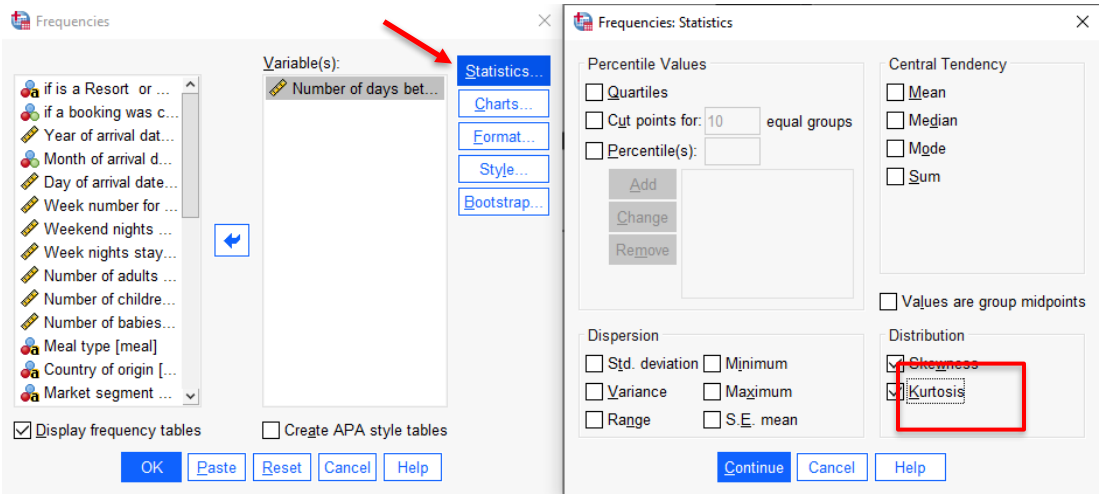


Figure 41 - Asymmetry measures configuration

Statistics

Number of days between entrance and arrival

N	Valid	119390
	Missing	0
Skewness		1.347
Std. Error of Skewness		.007
Kurtosis		1.696
Std. Error of Kurtosis		.014

Figure 42 - Asymmetry measures results

4.2 Student's T-test

The Student's T-test consists of hypotheses test to determine if there is a significant difference between the mean of two groups. It's generally used in the case of existing two independent groups to compare their means. There are two types of T tests, one for when the samples are independent and other for when they're paired. While the first focuses on a two separate groups, the latter is used when there is one group that is measured twice, in different conditions.

4.2.1 Normality assumptions

Before doing a parametric test, as already know, certain assumptions must be satisfied. One of which is the normality distribution of the sample. So, first, normality tests, as shown on the previous subtopic, need to be performed on the sample chosen for the test (***Analyse > Descriptive > Explore > Plots > Normality plots with tests***).

As displayed on the following figure, both chosen variables have KS values of 0.165 and 0.408, which represents the probability of the null hypothesis being rejected. Since the values are relatively low, it can be assumed that the sample has normal distribution.

Tests of Normality

	Kolmogorov-Smirnov[a]		
	Statistic	df	Sig.
Number of days between entrance and arrival	.165	119390	.000
if a booking was cancelled or not	.408	119390	.000

a. Lilliefors Significance Correction

Figure 43 - Normality tests (T-test)

4.2.2 scale and categorical variables test

Given that the normality tests were completed, the parametric test can now be performed. By accessing ***Analyse > Compare Means > Independent-sample T test*** a series of values will be displayed for both variables. Since there are two independent variables being used, there should be a test variable and a grouping variable selected. Is important, as well, to select the values for each group of the variables in ***Define Groups…*** .

Given the results, the Levene test null hypothesis gives the uniformity of the variances. If *sig* > 0.05 there is uniformity. Also, it is also displayed several other values that can be used to interpret the sample.

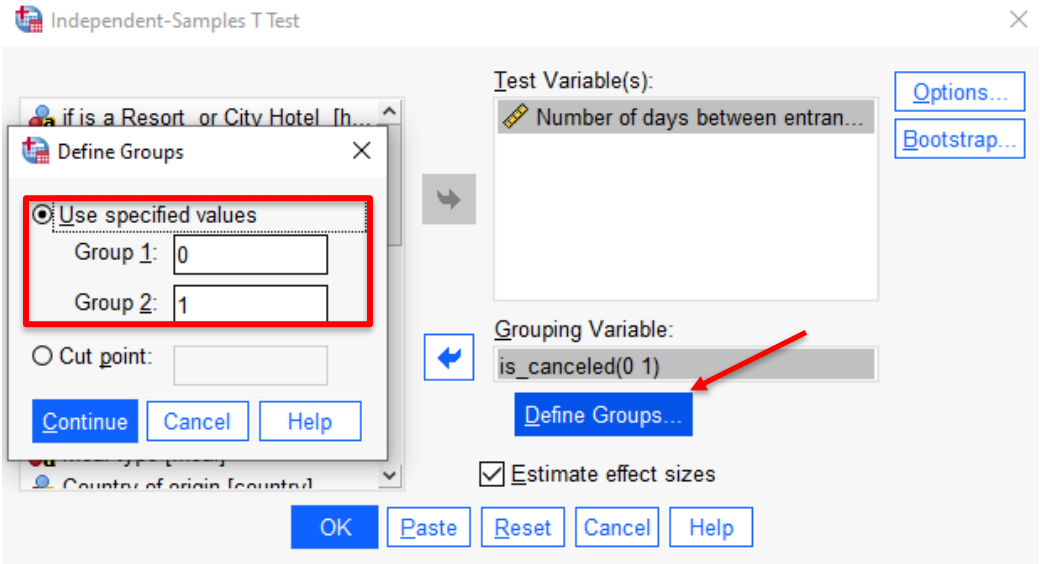


Figure 44 - T-test configuration

Group Statistics

	if a booking was cancelled or not	N	Mean	Std. Deviation	Std. Error Mean
Number of days between entrance and arrival	0	75166	79.98	91.110	.332
	1	44224	144.85	118.625	.564

Independent Samples Test

		Levene's Test for Equality of Variances		t-test for Equality of Means							
						Significance				95% Confidence Interval of the Difference	
		F	Sig.	t	df	One-Sided p	Two-Sided p	Mean Difference	Std. Error Difference	Lower	Upper
Number of days between entrance and arrival	Equal variances assumed	4437.728	.000	-105.935	119388	.000	.000	-64.864	.612	-66.064	-63.664
	Equal variances not assumed			-99.075	74936.126	.000	.000	-64.864	.655	-66.147	-63.581

Figure 45 - T-test results

4.3 One-way ANOVA test

One-way Analysis of Variance (ANOVA) is a statistical test to determine if there are significant differences between the means of three or more independent groups. Should be used with a continuous variable as the dependent variable and a categorical with more than three levels as the dependent variable.

After conducting a One-way ANOVA test and finding a significant result, there should be used a post hoc test. These tests, that include Tukey's Honestly Significant Difference (HSD), Bonferroni, and Scheffé tests, are used to identify which specific groups are different. These tests adjust for the increased probability of Type I errors (false positives) that can occur when conducting multiple pairwise comparisons.

Another test type that should be considered is the Homogeneity of Variance Test as well as the calculation of descriptive statistics. The first allows the checking of the homogeneity of variances assumption while the latter are used to summarize and describe the main features of a dataset.

4.3.1 Normality assumptions

As usual, is required to test the normality of a sample to perform parametric tests on it. The results are displayed in the following figure:

Tests of Normality

		Kolmogorov-Smirnov[a]		
	Month of arrival date	Statistic	df	Sig.
Number of days between entrance and arrival	JANUARY	.277	5929	.000
	FEBRUARY	.252	8068	.000
	MARCH	.210	9794	.000
	APRIL	.120	11089	.000
	MAY	.129	11791	.000
	JUNE	.107	10939	.000
	JULY	.107	12661	.000
	AUGUST	.130	13877	.000
	SEPTEMBER	.170	10508	.000
	OCTOBER	.163	11160	.000
	NOVEMBER	.225	6794	.000
	DECEMBER	.224	6780	.000

Figure 46 - Normality tests (ANOVA)

4.3.2 scale and categorical variables test

As already known, when a sample variable has more than three groups, the One way Anova test should be used. To perform this test on the selected scale and categorical variables, the SPSS provides the ***Analyse > Compare Means > One-way Anova*** tool. The Post-hoc tests can be selected by accessing *Post Hoc* as well as *Descriptive* and *Homogeneity and variance test* can be configured in the *Options* tab.

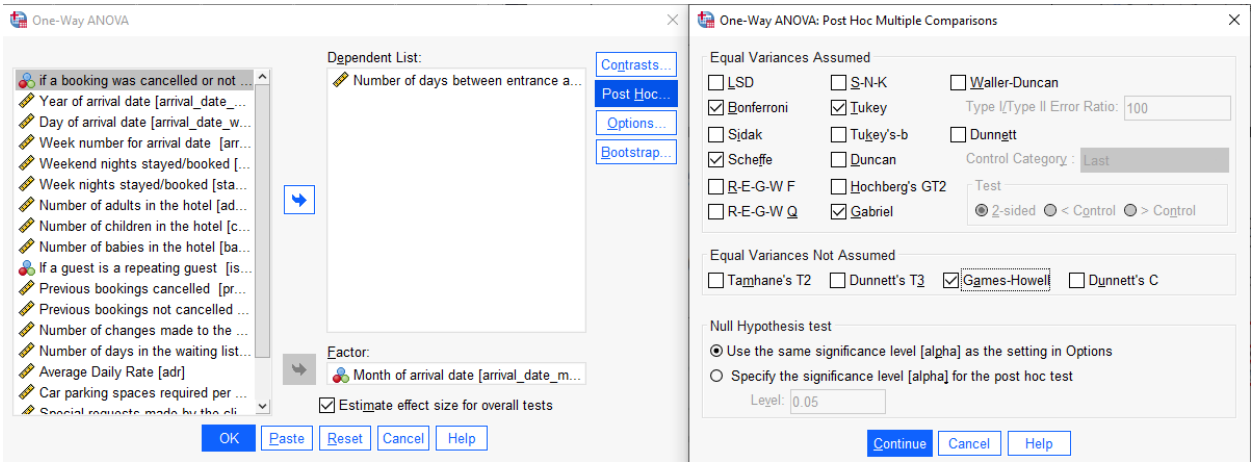


Figure 47 - Anova test configuration I

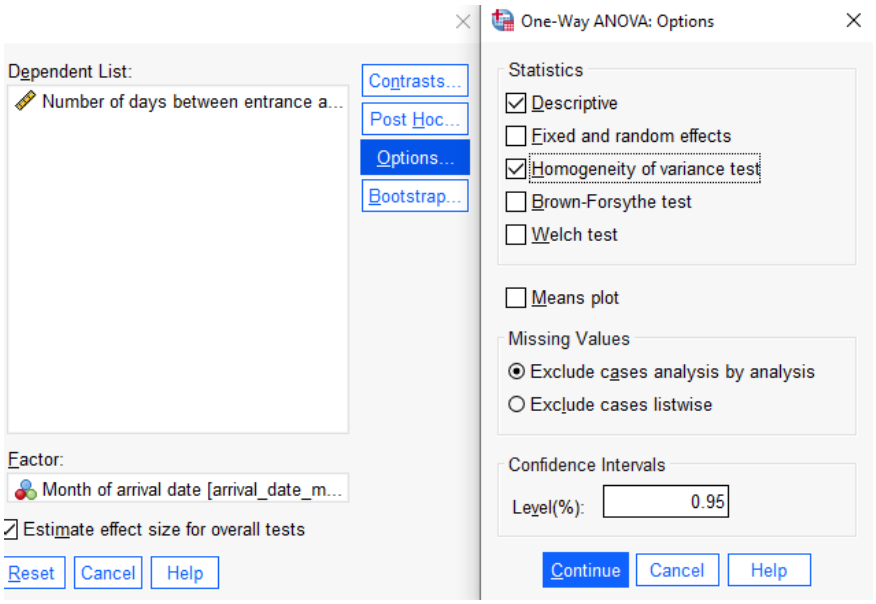


Figure 48 - Anova test configuration II

Tests of Homogeneity of Variances

		Levene Statistic	df1	df2	Sig.
Number of days between entrance and arrival	Based on Mean	1038.439	11	119378	.000
	Based on Median	795.463	11	119378	.000
	Based on Median and with adjusted df	795.463	11	107664.301	.000
	Based on trimmed mean	1024.032	11	119378	.000

ANOVA

Number of days between entrance and arrival

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	110592726.31	11	10053884.210	958.027	.000
Within Groups	1252796405.1	119378	10494.366		
Total	1363389131.4	119389			

ANOVA Effect Sizes[a]

		Point Estimate	95% Confidence Interval Lower	Upper
Number of days between entrance and arrival	Eta-squared	.081	.078	.084
	Epsilon-squared	.081	.078	.084
	Omega-squared Fixed-effect	.081	.078	.084
	Omega-squared Random-effect	.008	.008	.008

a. Eta-squared and Epsilon-squared are estimated based on the fixed-effect model.

Figure 49 - Anova test result

Post Hoc Tests

Multiple Comparisons

Dependent Variable: Number of days between entrance and arrival

	(I) Month of arrival date	(J) Month of arrival date	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval Lower Bound	Upper Bound
Tukey HSD	JANUARY	FEBRUARY	-2.550	1.752	.952	-8.28	3.18
		MARCH	-24.809*	1.686	<.001	-30.32	-19.30
		APRIL	-49.513*	1.648	<.001	-54.90	-44.13
		MAY	-72.336*	1.631	<.001	-77.67	-67.01
		JUNE	-82.937*	1.652	<.001	-88.34	-77.54
		JULY	-91.013*	1.612	<.001	-96.28	-85.74
		AUGUST	-75.828*	1.589	<.001	-81.02	-70.63
		SEPTEMBER	-91.365*	1.664	<.001	-96.80	-85.93
		OCTOBER	-77.685*	1.646	<.001	-83.06	-72.30
		NOVEMBER	-31.271*	1.821	<.001	-37.22	-25.32
		DECEMBER	-28.251*	1.821	<.001	-34.20	-22.30

Figure 50 - Anova Post Hoc test result

4.3.3 Curiosity

Despite the selected variable being of the *Date* type, the values will be interpreted as numeric for ANOVA. However, if preferred, the variable used, *arrival_date_month* can be converted to a string type variable and recoded to int values to suffer the same test explained before. Independently of the approach the results are the same, however, this process would be required for a ordinal variable with string values (*e.g.* values such as "male" and "female" would require the recoding to 0 and 1). This recodification can be observed in the following figure:

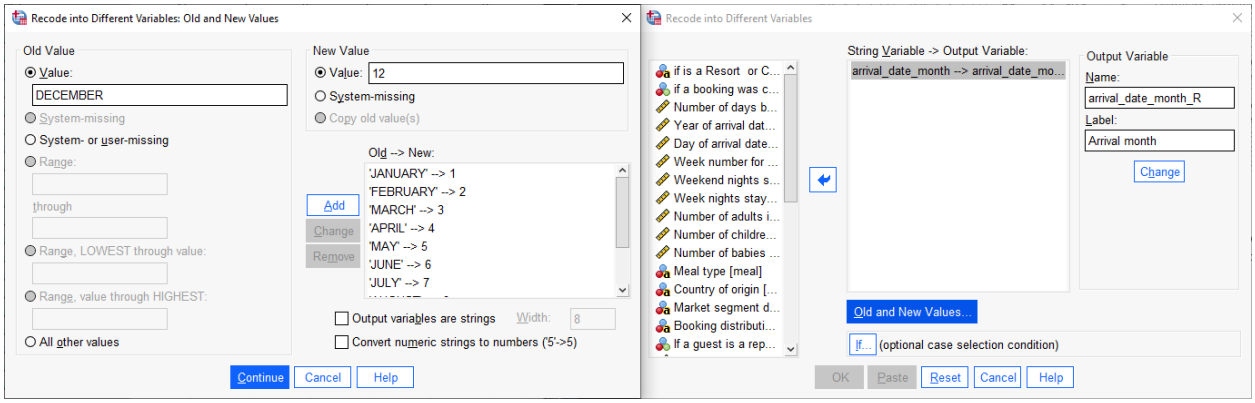


Figure 51 - Variable *arrival_date_month* recoding

Multiple Comparisons

Dependent Variable: Number of days between entrance and arrival

	(I) Month of arrival date	(J) Month of arrival date	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval Lower Bound	95% Confidence Interval Upper Bound
Tukey HSD	JANUARY	FEBRUARY	-2.550	1.752	.952	-8.28	3.18
		MARCH	-24.809*	1.686	<.001	-30.32	-19.30
		APRIL	-49.513*	1.648	<.001	-54.90	-44.13
		MAY	-72.336*	1.631	<.001	-77.67	-67.01
		JUNE	-82.937*	1.652	<.001	-88.34	-77.54
		JULY	-91.013*	1.612	<.001	-96.28	-85.74
		AUGUST	-75.828*	1.589	<.001	-81.02	-70.63
		SEPTEMBER	-91.365*	1.664	<.001	-96.80	-85.93
		OCTOBER	-77.685*	1.646	<.001	-83.06	-72.30
		NOVEMBER	-31.271*	1.821	<.001	-37.22	-25.32
		DECEMBER	-28.251*	1.821	<.001	-34.20	-22.30

Figure 52 - ANOVA test with the date type variable recoded to string/int

4.4 Mann-Whitney test

The Mann-Whitney test is used to determine the difference between two independent groups based on their ranked data. This test requires that the data is ordinal and the samples should be independent. The null hypothesis assumes that there is no difference between the data distribution.

To produce the results of this test, SPSS provides the ***Analyse > Nonparametric tests > Legacy Dialogs > 2 independent samples*** tool. In the window displayed, it will appear different types of tests, one of them being Mann-Whitney. Is possible to make SPSS to provide descriptive statistics as well, on the *Options* tab.

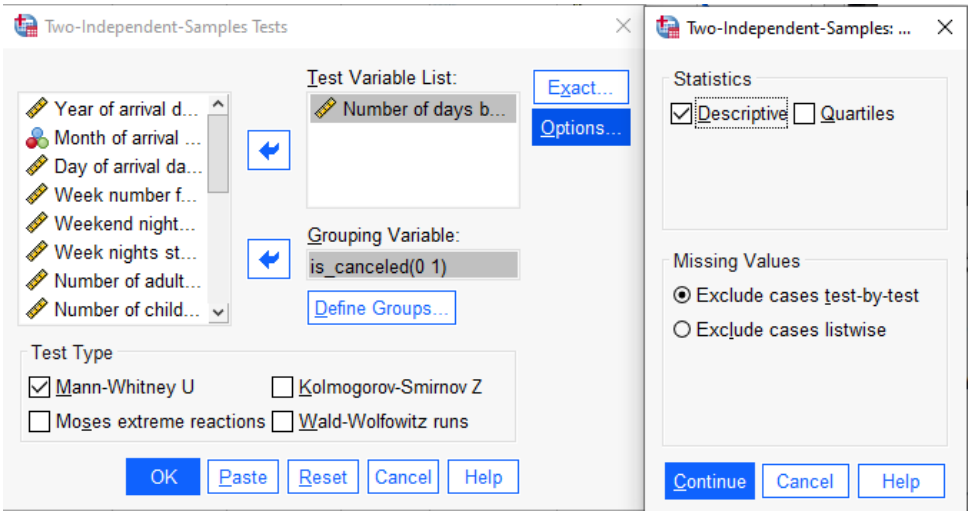


Figure 53 - Mann-Whitney test configurations

Mann-Whitney Test

Ranks

	if a booking was cancelled or not	N	Mean Rank	Sum of Ranks
Number of days between entrance and arrival	0	75166	51325.82	3857956278.5
	1	44224	73921.16	3269089466.5
	Total	119390		

Test Statistics[a]

	Number of days between entrance and arrival
Mann-Whitney U	1032954917.5
Wilcoxon W	3857956278.5
Z	-109.406
Asymp. Sig. (2-tailed)	.000

a. Grouping Variable: if a booking was cancelled or not

Figure 54 - Mann-Whitney test results

4.5 Kruskal-Wallis test

The Kruskal-Wallis test is an extension of the Mann-Whitney test for more than two independent groups. Is used to determine the difference between three or more independent groups. The null hypothesis also assumes that there is no difference between the groups' distribution.

To display the results of this test, SPSS provides the ***Analyse > Nonparametric tests > Legacy Dialogs > k independent samples*** tool. In the window displayed, it will appear different types of tests, one of them being Kruskal-Wallis. Is possible to make SPSS to provide descriptive statistics as well, on the *Options* tab.

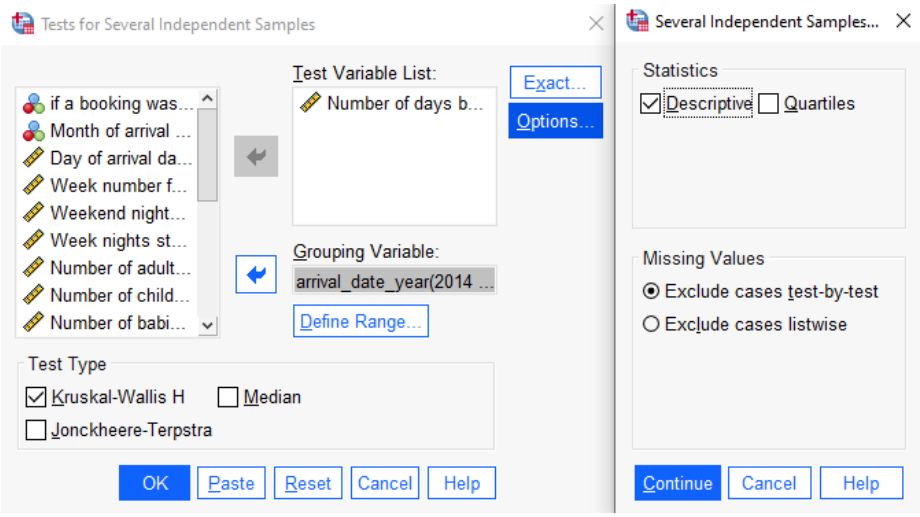


Figure 55 - Kruskal-Wallis test configuration

Descriptive Statistics

	N	Mean	Std. Deviation	Minimum	Maximum
Number of days between entrance and arrival	119390	104.01	106.863	0	737
Year of arrival date	119390	2016.16	.707	2015	2017

Kruskal-Wallis Test

Ranks

	Year of arrival date	N	Mean Rank
Number of days between entrance and arrival	2015	21996	56604.76
	2016	56707	59185.95
	2017	40687	62076.57
	Total	119390	

Test Statistics[a,b]

	Number of days between entrance and arrival
Kruskal-Wallis H	383.563
df	2
Asymp. Sig.	<.001

a. Kruskal Wallis Test

b. Grouping Variable: Year of arrival date

Figure 56 - Kruskal-Wallis test results

5 Exploratory data analysis – Python

Python is a high-level programming language known for its simplicity, readability, and vast ecosystem of libraries and frameworks. It supports various programming paradigms, including procedural, object-oriented, and functional programming. This programming language is a popular choice for statistical analysis and data science due to several available libraries (*e.g. NumPy*, *Pandas*, *Matplotlib*, etc…).

For this topic, the data analysis will no longer be done through SPSS but trough a script in python that aims to acquire the data and perform descriptive and inferential statistical operations as well as solve problems like missing values, normalization and binning.

5.1 Data acquisition

First, for the dataset to be read and kept in a dataframe, the *Pandas* library provides the ***pd.read_csv(path)*** function. This function returns the data set to a variable (named *df*). Then, the first 15 lines (5 is set by default) can be displayed by the ***df.head()*** function and its size can be obtained with ***df.shape***.

```
In [7]: #Libraries
        import pandas as pd
        import matplotlib. pyplot as plt
        import seaborn as sns
        import numpy as np

        #reads and displays the first 15 lines of the sample
        df = pd.read_csv('csv.csv')
        df.head(15)
```

Out[7]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month
0	Resort Hotel	0	342	2015	July	27	1
1	Resort Hotel	0	737	2015	July	27	1
2	Resort Hotel	0	7	2015	July	27	1
3	Resort Hotel	0	13	2015	July	27	1
4	Resort Hotel	0	14	2015	July	27	1
5	Resort Hotel	0	14	2015	July	27	1
6	Resort Hotel	0	0	2015	July	27	1
7	Resort Hotel	0	9	2015	July	27	1
8	Resort Hotel	1	85	2015	July	27	1
9	Resort Hotel	1	75	2015	July	27	1
10	Resort Hotel	1	23	2015	July	27	1
11	Resort Hotel	0	35	2015	July	27	1
12	Resort Hotel	0	68	2015	July	27	1
13	Resort Hotel	0	18	2015	July	27	1
14	Resort Hotel	0	37	2015	July	27	1

15 rows × 32 columns

```
In [8]: df.shape
Out[8]: (119390, 32)
```

Figure 57 - Dataset download and display

The first line of the dataset, if it contains the headers, can be removed with ***df = pd.read_csv('your_file.csv', header=None, skiprows=1)***. This process can be reversed. In the case of the dataset not containing headers, these can be created with ***headers = ["column1","column2"] > df.columns = ["column1","column2"]***. The headers' names can be displayed with ***df.columns***. These headers can be renamed with ***df.rename(columns={'column_old': 'column_new'}, inplace=True)***, for example, the first column, for being an index was renamed as "Index" with ***df.index.name = 'Index'***.

```
In [11]: df = pd.read_csv('csv.csv', header=None, skiprows=1)
         df.head()
```

Out[11]:

	0	1	2	3	4	5	6	7	8	9	...	22	23	24	25	26	27	28	29	30	31
0	Resort Hotel	0	342	2015	July	27	1	0	0	2	...	No Deposit	NaN	NaN	0	Transient	0.0	0	0	Check-Out	2015-07-01
1	Resort Hotel	0	737	2015	July	27	1	0	0	2	...	No Deposit	NaN	NaN	0	Transient	0.0	0	0	Check-Out	2015-07-01
2	Resort Hotel	0	7	2015	July	27	1	0	1	1	...	No Deposit	NaN	NaN	0	Transient	75.0	0	0	Check-Out	2015-07-02
3	Resort Hotel	0	13	2015	July	27	1	0	1	1	...	No Deposit	304.0	NaN	0	Transient	75.0	0	0	Check-Out	2015-07-02
4	Resort Hotel	0	14	2015	July	27	1	0	2	2	...	No Deposit	240.0	NaN	0	Transient	98.0	0	1	Check-Out	2015-07-03

5 rows × 32 columns

Figure 58 - Header removal

```
In [13]: headers = ["hotel","is_canceled", "lead_time","arrival_date_year","arrival_date_month","arrival_date_w

         df.columns = headers

         df.columns

Out[13]: Index(['hotel', 'is_canceled', 'lead_time', 'arrival_date_year',
                'arrival_date_month', 'arrival_date_week_number',
                'arrival_date_day_of_month', 'stays_in_weekend_nights',
                'stays_in_week_nights', 'adults', 'children', 'babies', 'meal',
                'country', 'market_segment', 'distribution_channel',
                'is_repeated_guest', 'previous_cancellations',
                'previous_bookings_not_canceled', 'reserved_room_type',
                'assigned_room_type', 'bookig_changes', 'deposit_type', 'agent',
                'company', 'days_in_waiting_list', 'customer_type', 'adr',
                'required_car_parking_spaces', 'total_of_special_requests',
                'reservation_status', 'reservation_status_date'],
               dtype='object')
```

Figure 59 - Header insertion

```
In [14]: df.rename(columns={'hotel':'hotel_type'}, inplace=True)

         df.index.name='Index'

         df.head(5)
```

Out[14]:

Index	hotel_type	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_
0	Resort Hotel	0	342	2015	July	27	
1	Resort Hotel	0	737	2015	July	27	
2	Resort Hotel	0	7	2015	July	27	
3	Resort Hotel	0	13	2015	July	27	
4	Resort Hotel	0	14	2015	July	27	

5 rows × 32 columns

Figure 60 - header replacement and index column creation

5.2 Descriptive statistics

As previously approached, descriptive statistics contain three types of variables and are a set of measures and techniques used to summarize and describe the important characteristics, features, and patterns of a dataset. Although these statistical operations were seen from the SPSS perspective, these can be performed with the use of python as well. A sample can be summarized in various ways:

- Each column can be summarized by mean, standard deviation and quartiles with the ***df.describe(include="all")*** function. The calling of this method displays various values regarding the sample's variables such as mean, frequency, etc.
- A summary of only the categorical values can be displayed with ***df.describe(include=['object'])***.
- The remaining variables can be displayed with ***df[['column1', 'column2']].describe()***.
- Values like max and frequency can be observed with the ***df['column'].max()*** and ***df['column'].value_counts()***, respectively.
- A new dataframe containing only the values of the frequency found by the previous process can be created with ***df['column'].value_counts().to_frame()***.

```
In [15]: df.describe(include="all")
```

Out[15]:

	hotel_type	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_d
count	119390	119390.000000	119390.000000	119390.000000	119390	119390.000000	
unique	2	NaN	NaN	NaN	12	NaN	
top	City Hotel	NaN	NaN	NaN	August	NaN	
freq	79330	NaN	NaN	NaN	13877	NaN	
mean	NaN	0.370416	104.011416	2016.156554	NaN	27.165173	
std	NaN	0.482918	106.863097	0.707476	NaN	13.605138	
min	NaN	0.000000	0.000000	2015.000000	NaN	1.000000	
25%	NaN	0.000000	18.000000	2016.000000	NaN	16.000000	
50%	NaN	0.000000	69.000000	2016.000000	NaN	28.000000	
75%	NaN	1.000000	160.000000	2017.000000	NaN	38.000000	
max	NaN	1.000000	737.000000	2017.000000	NaN	53.000000	

Figure 61 - All variables' sample

```
In [16]: df.describe(include=['object'])
```

Out[16]:

	hotel_type	arrival_date_month	meal	country	market_segment	distribution_channel	reserved_room_type	assigned_r
count	119390	119390	119390	118902	119390	119390	119390	
unique	2	12	5	177	8	5	10	
top	City Hotel	August	BB	PRT	Online TA	TA/TO	A	
freq	79330	13877	92310	48590	56477	97870	85994	

Figure 62 - Categorical variables' summary

```
In [19]: df[['adults']].describe()
```

Out[19]:

	adults
count	119390.000000
mean	1.856403
std	0.579261
min	0.000000
25%	2.000000
50%	2.000000
75%	2.000000
max	55.000000

```
In [26]: df['adults'].max()
Out[26]: 55

In [27]: df['adults'].value_counts()
Out[27]: 2     89680
         1     23027
         3      6202
         0       403
         4        62
         26        5
         27        2
         20        2
         5         2
         40        1
         50        1
         55        1
         6         1
         10        1
         Name: adults, dtype: int64
```

```
In [29]: df['adults'].value_counts().to_frame()
```

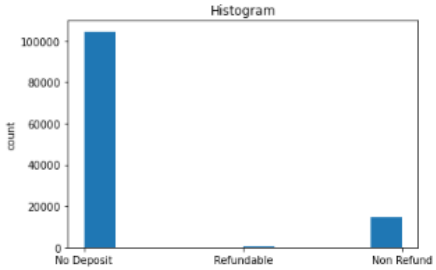
Out[29]:

	adults
2	89680
1	23027
3	6202
0	403
4	62
26	5
27	2
20	2
5	2
40	1
50	1
55	1
6	1
10	1

Figure 63 - Other summaries

This dataframe can be displayed, for example, on a histogram with ***plt.pyplot.hist(df["column"]) > plt.pyplot.xlabel("column") > plt.pyplot.ylabel("count") > plt.pyplot.title("Histogram")***. The existent categories can be displayed with ***df['column'].unique()***. A boxplot can be created as well with ***sns.boxplot(x="column1", y="column2", data=df)***.

```
In [38]: #import matplotlib.pyplot as plt above
         plt.hist(df[['deposit_type']])
         plt.ylabel("count")
         plt.title("Histogram")

Out[38]: Text(0.5, 1.0, 'Histogram')
```




```
In [39]: df['deposit_type'].unique()

Out[39]: array(['No Deposit', 'Refundable', 'Non Refund'], dtype=object)

In [42]: sns.boxplot(x="days_in_waiting_list", y="deposit_type", data=df)

Out[42]: <AxesSubplot:xlabel='days_in_waiting_list', ylabel='deposit_type'>
```

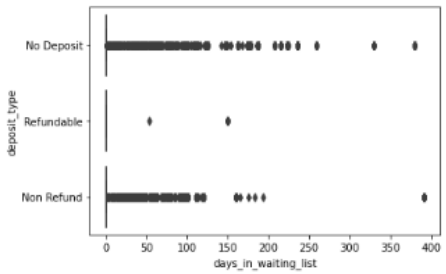


Figure 64 - Histogram, category display and boxplot (respectively)

5.3 Missing values

An important part of any statistical operations on a sample is to deal with the possible missing values. In python, there are various ways to solve these missing values. First, the missing values should be identified with either the ***isna()*** or the ***isnull()*** methods.

To practically demonstrate this process, the missing values quantity can be obtained with ***missing_data=df.isnull() > print (missing_data[column].value_counts())***, the output will display the number of not missing values as *False*. Then, the scale column value was replaced with *?* values, that will be treated as missing values, this can be done with ***df.replace('?', np.NaN, inplace = True)***.

```
In [20]: #Counts missing values in the sample
         missing_data = df.isnull()
         missing_data.head(5)
         #Displays the missing values in "children" - False represents the not missing values
         print (missing_data["children"].value_counts())

False    119386
True          4
Name: children, dtype: int64
```

Figure 65 - Missing values count

The missing values were then replaced by the mean of the selected column with ***avg_norm_loss = df["column"].astype("float").mean(axis=0) > df["column"].replace(np.nan, avg_norm_loss, inplace=True)***.

```
In [30]: #"Children" has 4 missing values
         df["children"].replace('?', np.NaN, inplace = True)
         print(df["children"])

         #replaces missing values with the mean
         avg_norm_loss = df["children"].astype("float").mean(axis=0)
         df["children"].replace(np.nan, avg_norm_loss, inplace=True)
         print(df["children"])

         #replaces missing values with the nan
         df["children"].replace(avg_norm_loss, np.nan, inplace=True)
         df.reset_index(drop=True, inplace=True)
         print(df["children"])

         #removes the missing values
         df.dropna(subset=["children"], axis=0, inplace=True)
         df.reset_index(drop=True, inplace=True)
         print(df["children"])
```

```
0         0.0
1         0.0
2         0.0
3         0.0
4         0.0
          ...
119381    0.0
119382    0.0
119383    0.0
119384    0.0
119385    0.0
Name: children, Length: 119386, dtype: float64
```

Figure 66 - Missing values replacement

Finally, these missing values were removed with ***df.dropna(subset=["column"], axis=0, inplace=True) df.reset_index(drop=True, inplace=True)***.

There are several approaches to solving missing values:

- After finding the missing values, they can be removed with ***dropna(axis="column", inplace=True***), the ***axis*** value can be omitted to remove the missing values in a line. After removing these values, the now empty cells can be replaced with values, for example with the column mean (***df['column'].fillna(df['column'].mean(), inplace=True)***) or any specific value (***df.fillna(0, inplace=True)***).
- Missing values can also be interpolated since the *Pandas* library provides the ***interpolate(inplace=True)***) method. In the case of being useful, a boolean mask can be created with the goal of performing specific operations. This mask must be attributed to a value resulting of the ***df['column'].isna()*** operation, followed by the replacement of the missing values by 0 with ***df[mask]=0***.
- Another approach, more complex, is by treating the missing values with Machine Learning models. With the *sklearn.impute* library, by importing *SimpleImputer*, ***imputer = SimpleImputer(strategy='mean')*** replaces the missing values by the mean. These values are then kept in a variable with ***X = imputer.fit_transform(X)***.

5.4 Normalization

The *Pandas* library provides various types of values to categorize the data used by its dataframes. These types are:

- Int64 – Positive or negative integer value composed by 64 bits;

- float64 – Positive or negative floating (decimal) value composed by 64 bits;
- bool – Boolean values, so *True* or *False*;
- datetime64 – Represents date and time;
- timedelta64 – Represents the difference between two datetimes;
- category – Can keep a limited number of values. Useful for a finite set of categories;
- object – Can store any type of value Python, usually strings or mix of different value types;
- string – Introduced after *Pandas 1.0.0*, represents strings more efficiently than object type.

The existent data types on the dataset, can be obtained through the ***df.dtypes*** method or ***df.info()***. However this automatic type setting can be incorrect or inaccurate. Which is the case for some variables on the dataset used for this report. So, if necessary, the data types can be converted with ***df[["column1", "column2", …]] = df[["column1", "column2"]].astype("*type*")***.

```
In [32]: #Displays sample's variables types and other info
         df.dtypes
         df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119386 entries, 0 to 119385
Data columns (total 32 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   hotel_type                      119386 non-null  object
 1   is_canceled                     119386 non-null  int64
 2   lead_time                       119386 non-null  int64
 3   arrival_date_year               119386 non-null  int64
 4   arrival_date_month              119386 non-null  object
 5   arrival_date_week_number        119386 non-null  int64
 6   arrival_date_day_of_month       119386 non-null  int64
 7   stays_in_weekend_nights         119386 non-null  int64
 8   stays_in_week_nights            119386 non-null  int64
 9   adults                          119386 non-null  int64
 10  children                        119386 non-null  float64
 11  babies                          119386 non-null  int64
 12  meal                            119386 non-null  object
 13  country                         118898 non-null  object
 14  market_segment                  119386 non-null  object
 15  distribution_channel            119386 non-null  object
 16  is_repeated_guest               119386 non-null  int64
 17  previous_cancellations          119386 non-null  int64
 18  previous_bookings_not_canceled  119386 non-null  int64
 19  reserved_room_type              119386 non-null  object
 20  assigned_room_type              119386 non-null  object
 21  bookig_changes                  119386 non-null  int64
 22  deposit_type                    119386 non-null  object
 23  agent                           103048 non-null  float64
 24  company                         6797 non-null    float64
 25  days_in_waiting_list            119386 non-null  int64
 26  customer_type                   119386 non-null  object
 27  adr                             119386 non-null  float64
 28  required_car_parking_spaces     119386 non-null  int64
 29  total_of_special_requests       119386 non-null  int64
 30  reservation_status              119386 non-null  object
 31  reservation_status_date         119386 non-null  object
dtypes: float64(4), int64(16), object(12)
memory usage: 29.1+ MB
```

```
In [33]: df[["is_canceled", "is_repeated_guest"]] = df[["is_canceled", "is_repeated_guest"]].astype("bool")
         df[["children"]] = df[["children"]].astype("int64")
         df[["agent", "company"]] = df[["agent", "company"]].astype("object")
         df[["reservation_status_date"]] = df[["reservation_status_date"]].astype("datetime64")
         df.dtypes
         df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119386 entries, 0 to 119385
Data columns (total 32 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   hotel_type                      119386 non-null  object
 1   is_canceled                     119386 non-null  bool
 2   lead_time                       119386 non-null  int64
 3   arrival_date_year               119386 non-null  int64
 4   arrival_date_month              119386 non-null  object
 5   arrival_date_week_number        119386 non-null  int64
 6   arrival_date_day_of_month       119386 non-null  int64
 7   stays_in_weekend_nights         119386 non-null  int64
 8   stays_in_week_nights            119386 non-null  int64
 9   adults                          119386 non-null  int64
 10  children                        119386 non-null  int64
 11  babies                          119386 non-null  int64
 12  meal                            119386 non-null  object
 13  country                         118898 non-null  object
 14  market_segment                  119386 non-null  object
 15  distribution_channel            119386 non-null  object
 16  is_repeated_guest               119386 non-null  bool
 17  previous_cancellations          119386 non-null  int64
 18  previous_bookings_not_canceled  119386 non-null  int64
 19  reserved_room_type              119386 non-null  object
 20  assigned_room_type              119386 non-null  object
 21  bookig_changes                  119386 non-null  int64
 22  deposit_type                    119386 non-null  object
 23  agent                           103048 non-null  object
 24  company                         6797 non-null    object
 25  days_in_waiting_list            119386 non-null  int64
 26  customer_type                   119386 non-null  object
 27  adr                             119386 non-null  float64
 28  required_car_parking_spaces     119386 non-null  int64
 29  total_of_special_requests       119386 non-null  int64
 30  reservation_status              119386 non-null  object
 31  reservation_status_date         119386 non-null  datetime64[ns]
dtypes: bool(2), datetime64[ns](1), float64(1), int64(15), object(13)
memory usage: 27.6+ MB
```

Figure 67 - Data types and data type change (right)

A column can be normalized by creating a new one from it, this column was created In a way that its value vary from 0 to 1. This process is possible with ***df['column'] = df['column']/df['column'].max()***.

```
#normalizes a variable to vary from 0 to 1
df['days_in_waiting_list (0 and 1)'] = df['days_in_waiting_list']/df['days_in_waiting_list'].max()
df[['days_in_waiting_list (0 and 1)']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119386 entries, 0 to 119385
Data columns (total 1 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   days_in_waiting_list (0 and 1)  119386 non-null  float64
dtypes: float64(1)
memory usage: 932.8 KB
```

Figure 68 - Column normalization

5.5 Binning

Binning is a data preprocessing technique used in data analysis and machine learning that involves dividing a continuous variable into a set of distinct intervals or "bins". This action is useful for simplification of continuous data, with dealing with 'noisy' (by reducing the impact of outliers) data and for handling non-linear relationships.

A column, resulting from the division of another can be created . The new and the old columns can be compared by displaying each other's lines. The following block will create three levels for the new column, *low, medium* and *high* and group the values from the original column in these sets with:

1. ***bins = np.linspace(min(df["column"]) > max(df["column"]), 4)***
2. ***group_names = ['Low', 'Medium', 'High']***
3. ***df['column_new']=pd.cut(df['column_old'], bins, labels=group_names, include_lowest=True)***
4. ***df[['column_old','column_new']].head(5)***

```
In [42]: #Creates three sets of values from another variable
         bins = np.linspace(min(df["days_in_waiting_list"]), max(df["days_in_waiting_list"]),4)
         group_names = ['Low', 'Medium','High']
         df["days_in_waiting_list_classes"] = pd.cut(df["days_in_waiting_list"], bins, labels=group_names,include_lowest=True )
         df[["days_in_waiting_list", "days_in_waiting_list_classes"]].head(5)
```

Out[42]:

	days_in_waiting_list	days_in_waiting_list_classes
0	0	Low
1	0	Low
2	0	Low
3	0	Low
4	0	Low

Figure 69 - Variable binning

By making an histogram of this variable is possible to observe the behaviour of the data. For the selected variable, the low values make up most of the variable's data. To be presented in a histogram graph in python, the following block of code is required:

1. ***plt.pyplot.hist(df["column"])***
2. ***plt.pyplot.xlabel("column")***
3. ***plt.pyplot.ylabel("count")***
4. ***plt.pyplot.title("Histogram")***

```
In [43]: #Creates a histogram from the created column
         plt.hist(df["days_in_waiting_list_classes"])
         plt.xlabel("Days in the waiting list by class")
         plt.ylabel("Values")
         plt.title("Days in the waiting list histogram")
Out[43]: Text(0.5, 1.0, 'Days in the waiting list histogram')
```

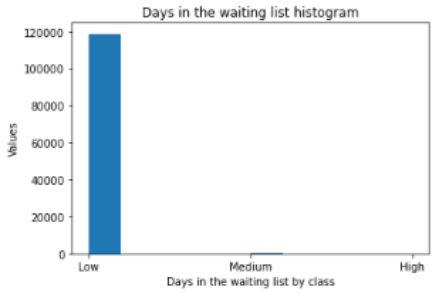


Figure 70 - Histogram

5.6 Inferential statistics

Inferential statistics is a branch of statistics that deals with making predictions, inferences, and generalizations about a population based on a sample of data taken from that population. In Python involves using the sample data to draw conclusions about a population based on the subset of the data while accounting for uncertainty and variability.

The correlation coefficient between the numerical columns on the dataset can be obtained through ***df.corr***. Then, the most correlated features can be used to produce a scatterplot with ***sns.regplot(x="column1", y="column2", data=df) > plt.ylim(0,)***. The columns found with both correlation and interest in knowing what type of correlation, were *lead_time* which represents the number of days between the entering date of the booking and the arrival date and *is_canceled* which represents if a booking was cancelled. The result of these two variables in a scatterplot was that when the time elapsed tends to be bigger, the more probable it is that a booking is going to be cancelled.

```
In [23]: #scatterplot of the correlation between two columns
         sns.regplot(x="lead_time", y="is_canceled", data=df)
         plt.ylim(0,)
Out[23]: (0.0, 1.2857488525876133)
```

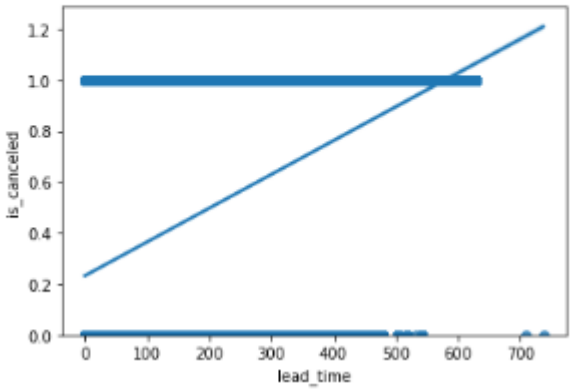


Figure 71 - Correlation scatterplot

A pivot table between two categorical and one scale variables can be presented with the code block:

1. ***df_new=df[['column1','column2','column3']]*** - creates a new data frame with three columns;
2. ***grouped=df_new.groupby(['column1','column2'],as_index=False).mean()*** – groups the data;
3. ***grouped_pivot=grouped.pivot(index='column1',columns='column2')*** – Creates pivot table;
4. ***grouped_pivot*** – Contains (and shows) resulting pivot table;

The created table can be displayed in a form of a heatmap with ***plt.pcolor(grouped_pivot, cmap='RdBu') > plt.colorbar() > plt.show()***.

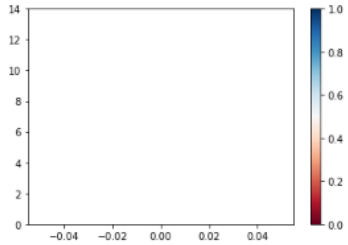
For each selected categorical variable, the scale variable mean can be presented with ***grouped_by_mean = df_groups.groupby(['column1'],as_index=False).mean()***. The group (***df.groups***) is created by ***df_groups = df[['column1','column2']]***.

For any unknown reason, the variables on the dataset are not being accounted for these tasks and only these tasks, so the output is as it follows for every other selected set of variables:

```
#shows a pivot table between a scale variable and two categorical variables
df_new = df[["adults","hotel_type","arrival_date_month"]]
grouped = df_new.groupby(["adults","hotel_type","arrival_date_month"],as_index=False).count()
grouped_pivot = grouped.pivot(index="adults",columns=["hotel_type","arrival_date_month"])
print(grouped_pivot)
```

```
Empty DataFrame
Columns: []
Index: [0, 1, 2, 3, 4, 5, 6, 10, 20, 26, 27, 40, 50, 55]
```

```
#Creates heatmap from the previous pivot table
plt.pcolor(grouped_pivot, cmap='RdBu')
plt.colorbar()
plt.show()
```




```
#Displays the scale variable's mean for each categorical variable
df_groups = df[["adults","hotel_type","arrival_date_month"]]
grouped_by_mean = df_groups.groupby(["adults","hotel_type","arrival_date_month"],as_index=False).mean()
print(grouped_by_mean)
```

```
     adults  hotel_type  arrival_date_month
0         0           0                   1
1         0           0                   2
2         0           0                   3
3         0           0                   4
4         0           0                   5
..      ...         ...                 ...
114      26           1                  10
115      27           1                   9
116      40           1                   9
117      50           1                   9
118      55           1                  10

[119 rows x 3 columns]
```

Figure 72 - Pivot table, heatmap and mean by group (respectively)

Finally, the ANOVA one-way test can be performed for the previously variables (scale variable as the dependent) with:

1. ***grouped=df_groups.groupby(["column1"])["column2"]***
2. ***groups = [grouped.get_group("name") for "name" in df["column"].unique()]***
3. ***f_val, p_val = stats.f_oneway(*groups)***
4. ***print(f"ANOVA F-value: {f_val}")***
5. ***print(f"P-value: {p_val}")***

```
# Performs the ANOVA test
import scipy.stats as stats

# Group by "arrival_date_month" and get the groups for "adults"
grouped = df.groupby(["arrival_date_month"])["adults"]

# Collect the groups in a list
groups = [grouped.get_group(month) for month in df["arrival_date_month"].unique()]

# Perform ANOVA
f_val, p_val = stats.f_oneway(*groups)

# Print the results
print(f"ANOVA F-value: {f_val}")
print(f"P-value: {p_val}")
```

```
ANOVA F-value: 268.3182614467674
P-value: 0.0
```

Figure 73 - ANOVA one-way test in python

6 Orange

Orange is an open-source data mining and machine learning software suite. It's designed to be a user-friendly tool for data exploration, visualization, preprocessing, and predictive modelling. This software is available to use on the Anaconda distribution, a platform that made available the *Jupyter Notebooks* used for data analysis in python, in the previous topic.

For this topic, the software Orange 3 will be used to perform machine learning tasks such as supervised and unsupervised learning. The difference between both these approaches is that through supervised learning, there is a train dataset that is used for the algorithm to find the different characteristics and what conclusions come from it to then make predictions to be tested by the second dataset. Unsupervised learning finds the characteristics and conclusions of the sample by itself.

6.1 Supervised Learning

As previously explained, the supervised learning type of algorithm requires two datasets. These were originated by the original dataset that's been being used on the different topics of this document. This original dataset was simply divided in two, on for training and other for testing. On the test dataset, a target categorical variable should be removed to make predictions about it. The selected variable was *is_canceled* that represents if a booking was cancelled or not.

Before proceeding with the predictions is important to grasp some notions about the different metrics that will be displayed in these type of tasks. There are five major metrics to consider when handling categorical type target variables:

- **AUC** – AUC or Area Under the Curve evaluates the performance of a classification algorithm. It represents the area under the ROC curve. ROC or Receiver Operating Characteristic is a graph that displays the rate between the true (sensibility) and false positives (specificity). The closer the AUC is to 1 the better. If the result is equal to 0.5 the algorithm is as good as an random choice;
- **Accuracy** – Measures the relation between correct and wrong classifications. The closer to 1, the better;
- **Precision** – Measures the relation between true positives and the total of predictions made. Important when the cost of a false positive is high;
- **Recall** – Measures the relation between true positives and the total of true instances (true positives and false negatives). Important when the cost of a false negative is high;
- **F1-Score** – Combines Precision and Recall. Useful for when the classes are not balanced. Is equal to 2 times the multiplication between Precision and Recall in relation to their sum, this is, 2*(P*R)/(P+R);

In contrary, when the target variable is of the scale type, the metrics are:

- **MSE** – The result of the sum of the squared differences between actual and predicted values and the division it by the total number of observations. Is sensitive to outliers because it squares the differences;
- **RMSE** – The square root of the MSE. provides a measure of the average magnitude of error between actual and predicted values.;
- **MAE** – A measure of the average absolute differences between actual and predicted values. Unlike MSE, MAE does not square the differences, so it gives equal weight to all errors regardless of their magnitude;

- **R2** – R-squared, also known as the coefficient of determination, measures the proportion of the variance in the dependent variable that is predictable from the independent variable(s). It is a relative measure of model fit. A higher R2 indicates a better fit;

To make predictions about a certain data sample, there are several supervised learning algorithms, between them:

- **KNN** – Classifies a new data point based on the majority class among its K nearest neighbors in the feature space. Can be used for both classification and regression tasks;
- **Decision Tree** – Splits the data based on features to create a tree-like structure. At each node, a decision is made based on a feature's value, eventually leading to a final decision or prediction at the leaf nodes;
- **Random Forest** – Combines multiple decision trees to improve accuracy and control overfitting. It creates a "forest" of trees by training each tree on a random subset of the data and features;
- **Logistic Regression** – Models the probability of a binary outcome as a function of predictor variables. It employs the logistic function to squash predictions between 0 and 1;
- **SVM** – Support Vector Machine, aims to find the hyperplane that best separates classes in a high-dimensional space. It maximizes the margin between classes and uses support vectors (data points closest to the hyperplane) for decision making;
- **Naive Bayes** – Probabilistic classifier based on *Bayes* theorem with the *naive* assumption of feature independence. It calculates the probability of a class given the features and selects the class with the highest probability;
- **Adaboost** – Adaptive Boosting is an ensemble learning method that combines multiple weak learners (*e.g.* decision trees with limited depth) to create a strong learner. It assigns more weight to misclassified samples to improve performance iteratively;

6.1.1 Dataset importation

Having this information, the machine learning practical tasks can be started. The Orange software allows the importation of a dataset, by selecting *File* on the *Data* tab. However, the autoclassification of the features of the dataset could be wrong so is important to verify these configurations. The target variable is selected on the window opened by this tool, in the *Role* column.

After importing both datasets and removing the target variable from the test sample. In the *Evaluate* tab, it will be displayed a *Predictions* option. In *Model*, the different available algorithms will be presented. The results can be analysed by connecting *Predictions* to the test sample.

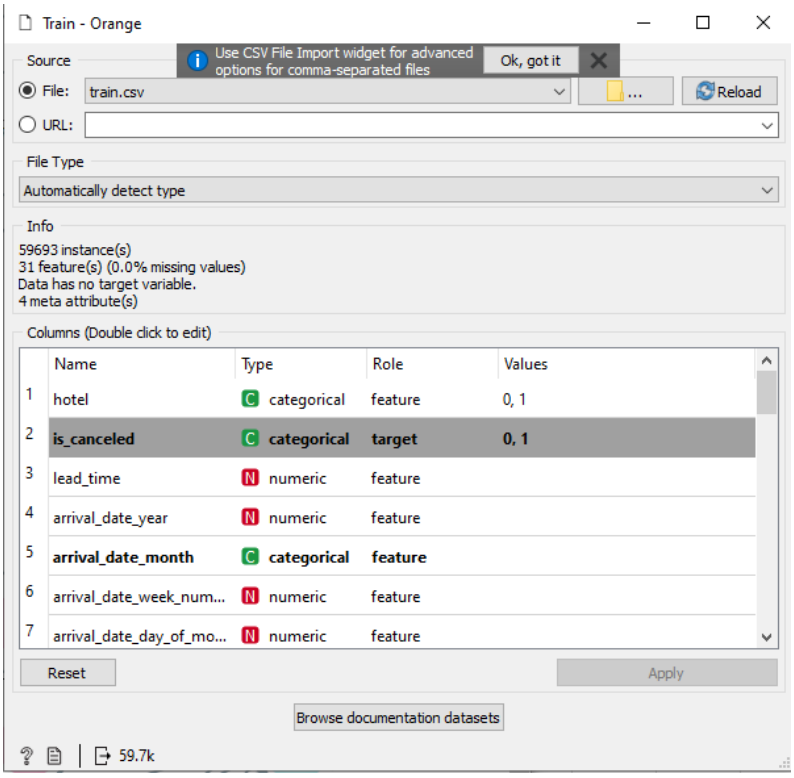


Figure 74 - Import dataset options

6.1.2 Predictions for the categorical target variable

After importing the dataset and selecting, as a target, the categorical variable to predict, the different algorithms will predict the values of the variable on the test dataset by using the same variable on the train dataset. The efficiency of each algorithm will be displayed by the *Predictions* tool by using different established metrics that were explained before such as AUC, Accuracy, Precision, Recall and F1-Score.

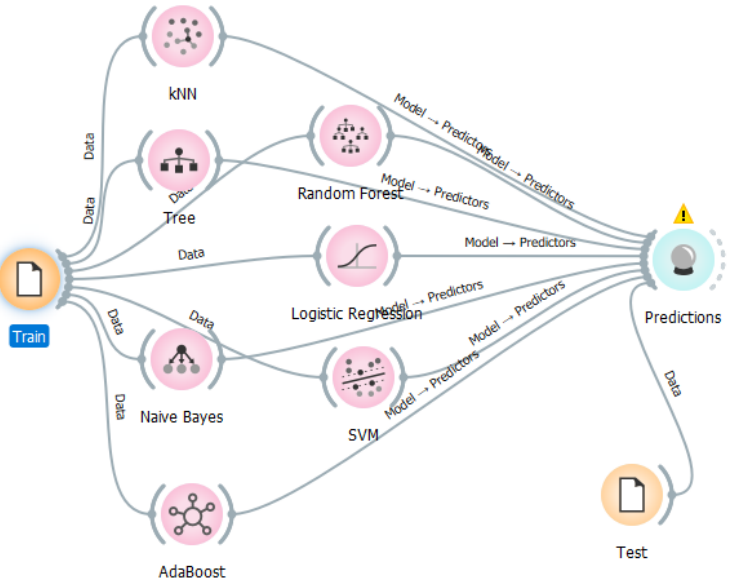


Figure 75 - Categorical target predictions scheme

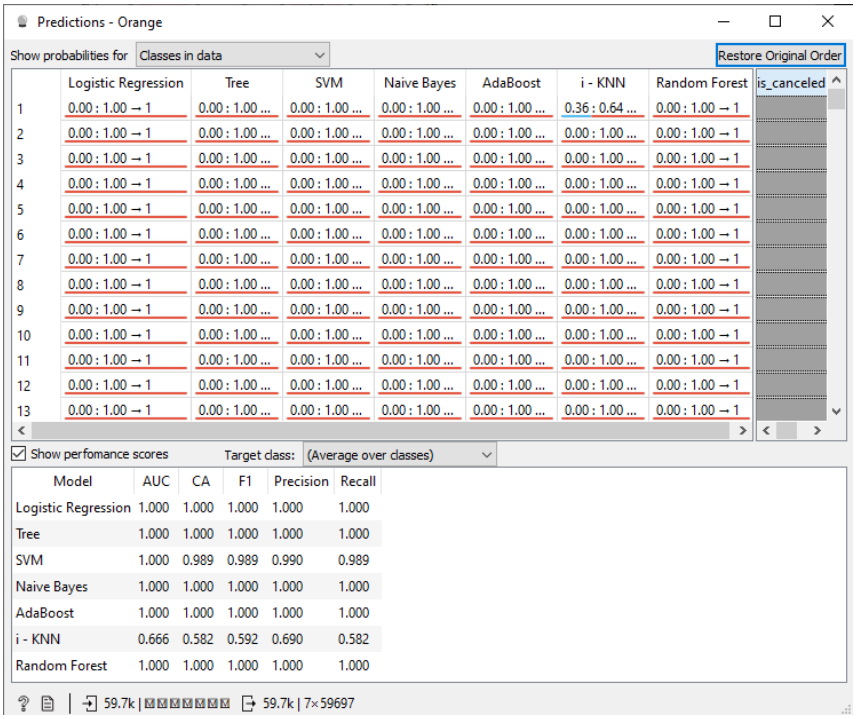


Figure 76 - Categorical target predictions results

6.1.3 Test and Score for the categorical target variable

The process to this type of test is similar and the metrics are the same. However, the results are displayed in a slightly different way. The goal of this test is to verify which algorithm had the best performance for the used dataset.

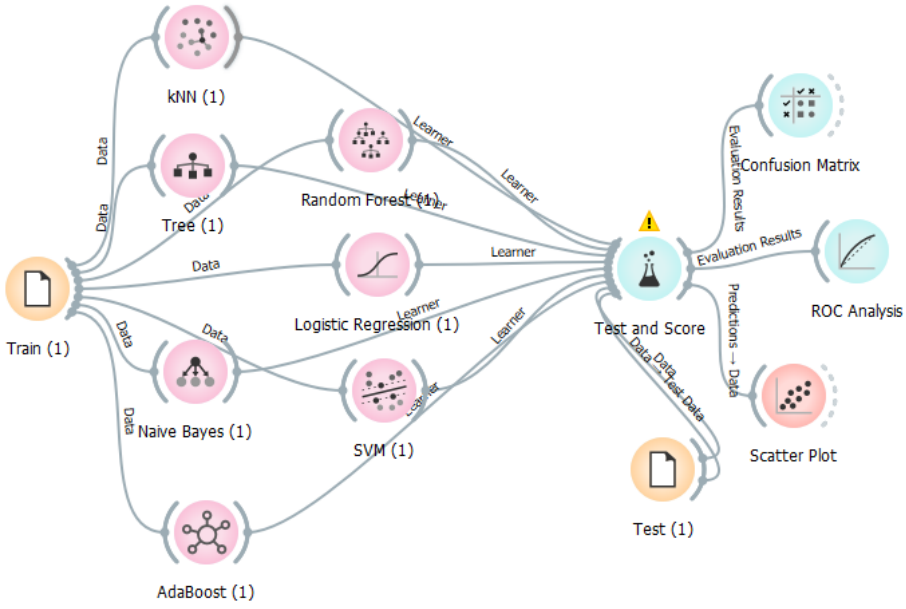


Figure 77 - Categorical target Test and Score scheme

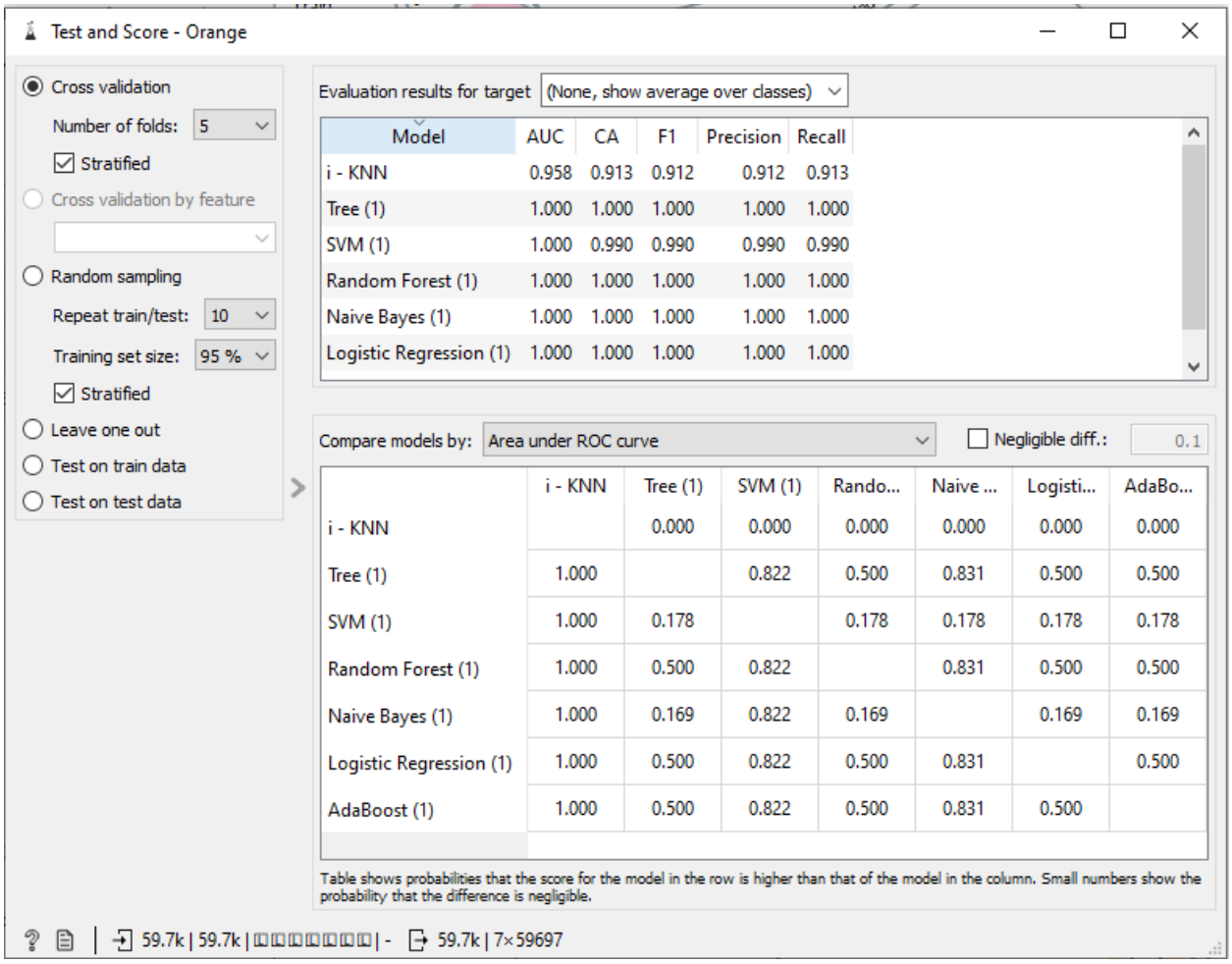


Figure 78 - Categorical target Test and Score cross validation

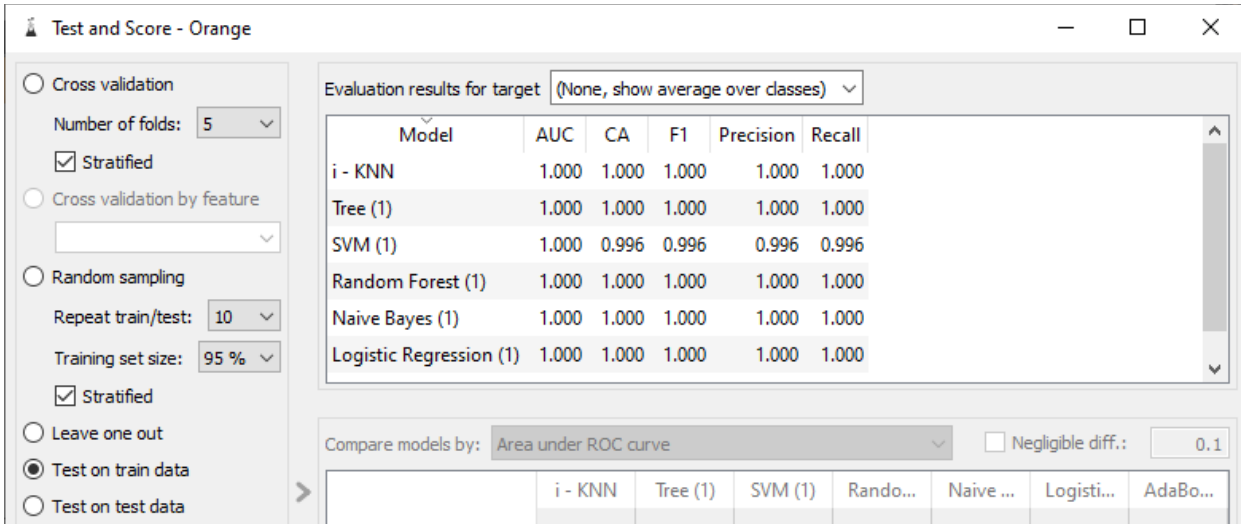


Figure 79 - Categorical target Test and Score test on train data

After analysing the scores of the different algorithms used, there are other types of data that can be analysed to further understand the behaviour of each algorithm. The Orange provides *Confusion Matrix*, *ROC Analysis* and *Scatter Plot*.

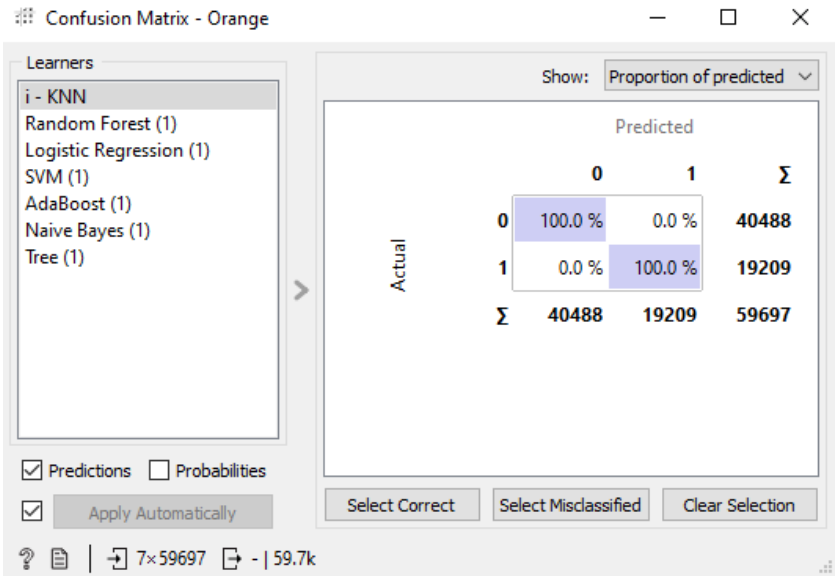


Figure 80 - Confusion Matrix example (KNN)

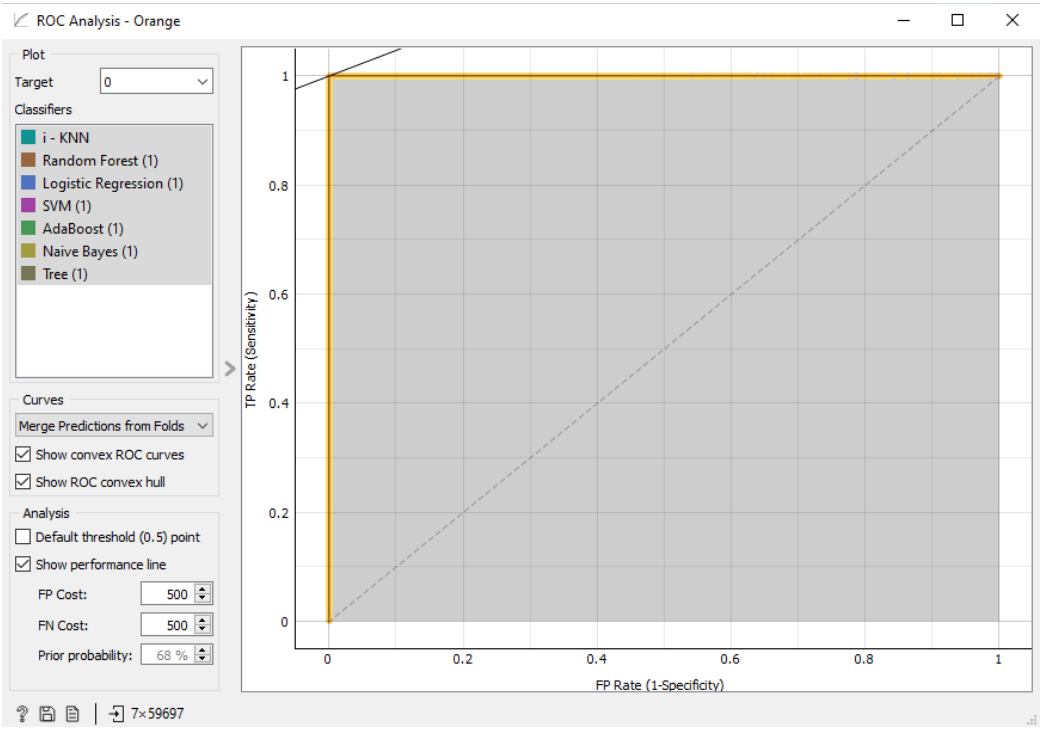


Figure 81 - ROC Analysis (all algorithms)

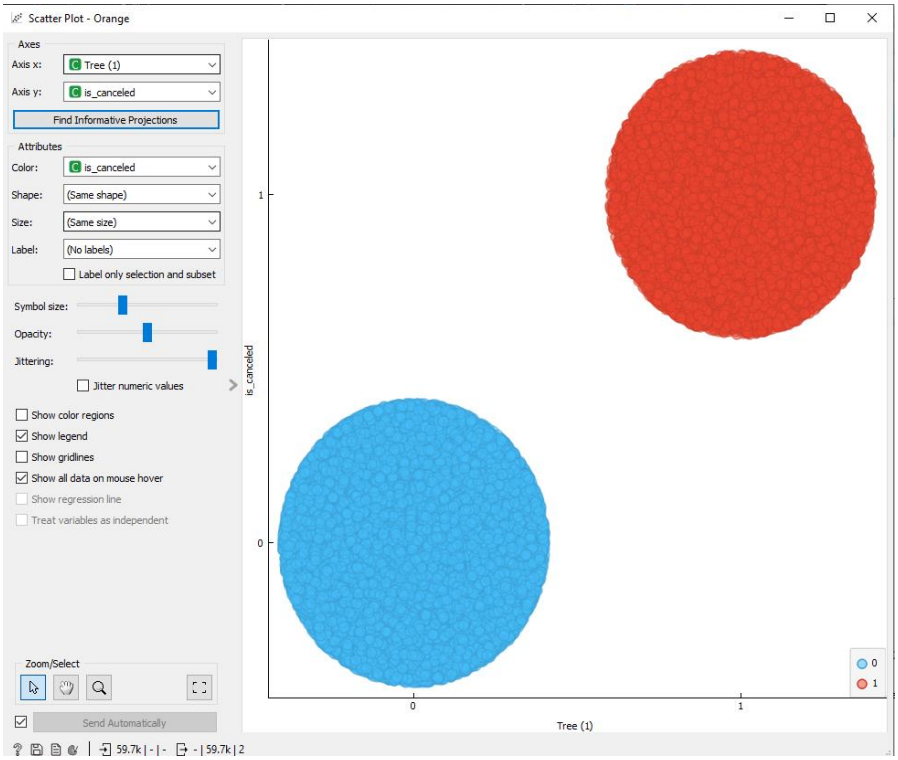


Figure 82 – Scatter plot high accuracy algorithm example

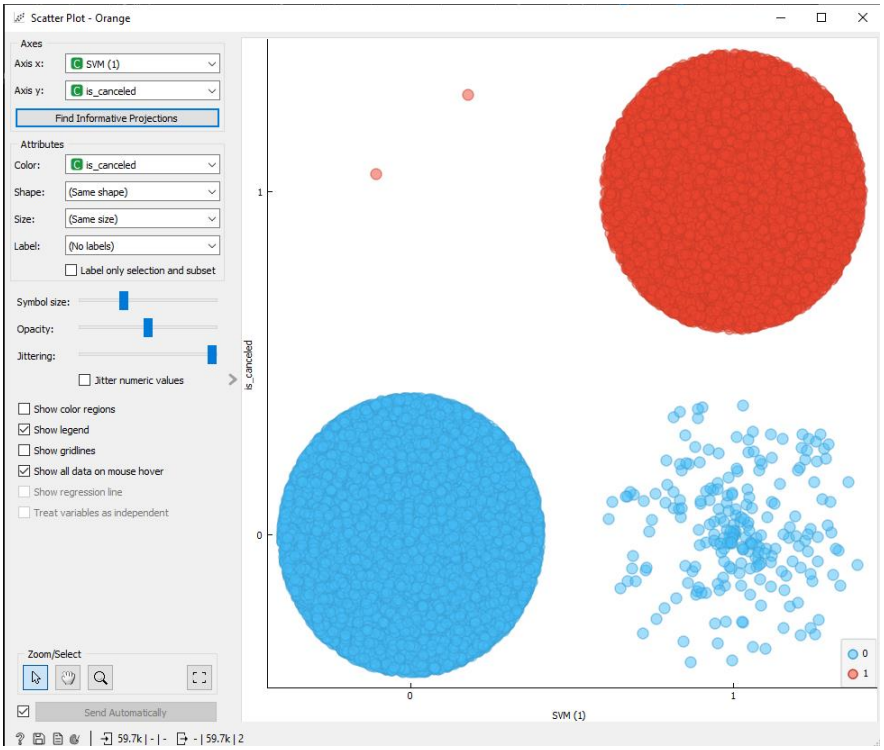


Figure 83 - Scatter plot lower accuracy algorithm example

6.1.4 Predictions for the scale target variable

For the prediction of the values of a scale variable, the process is similar. However the change of the target variable is critical.

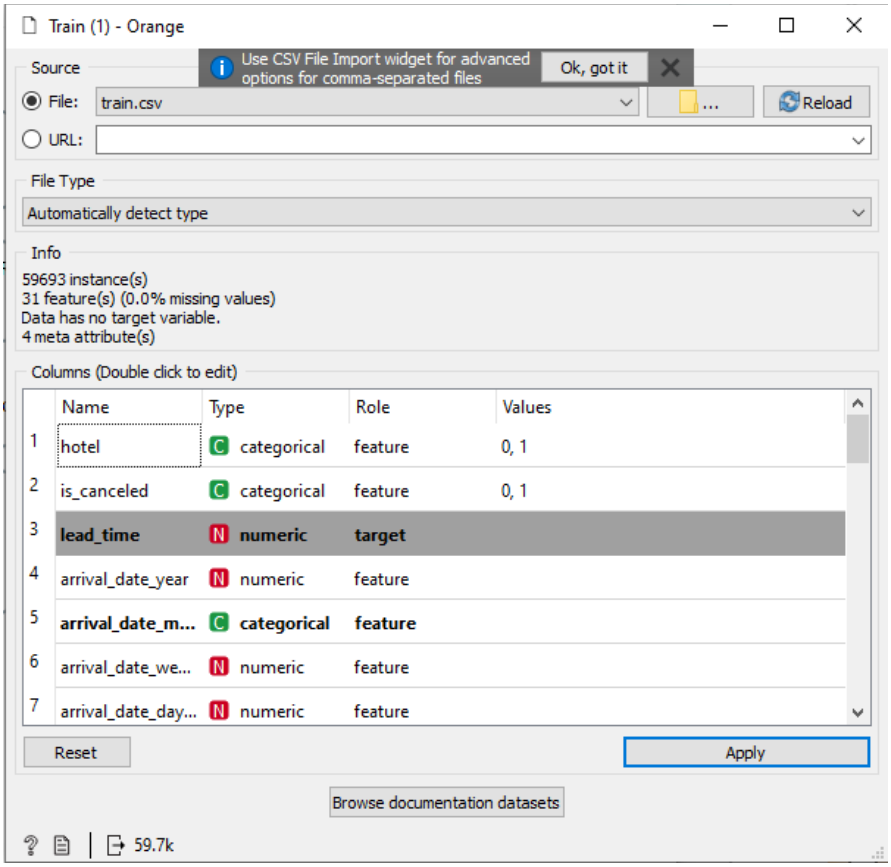


Figure 84 - Target scale variable selection

After selecting the scale variable as the target to predict, the different algorithms will predict the values of the variable and their efficiency will be displayed, by the *Predictions* tool, by using other established metrics such as MSE, RMSE, MAE and R2.

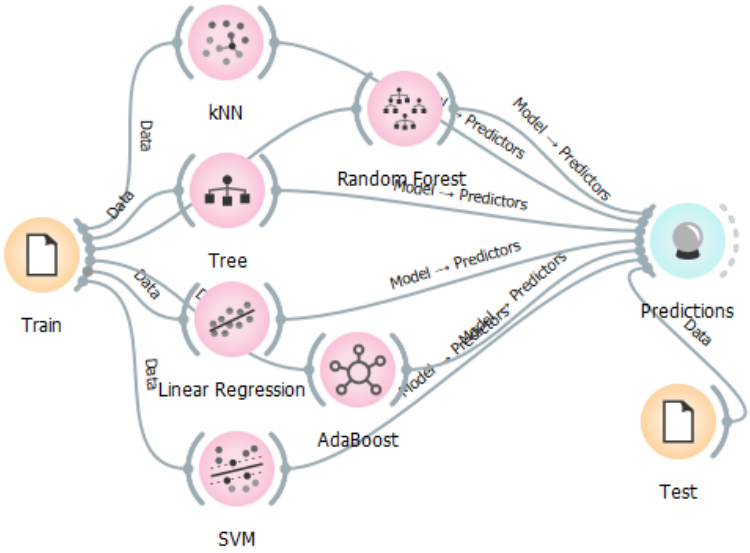
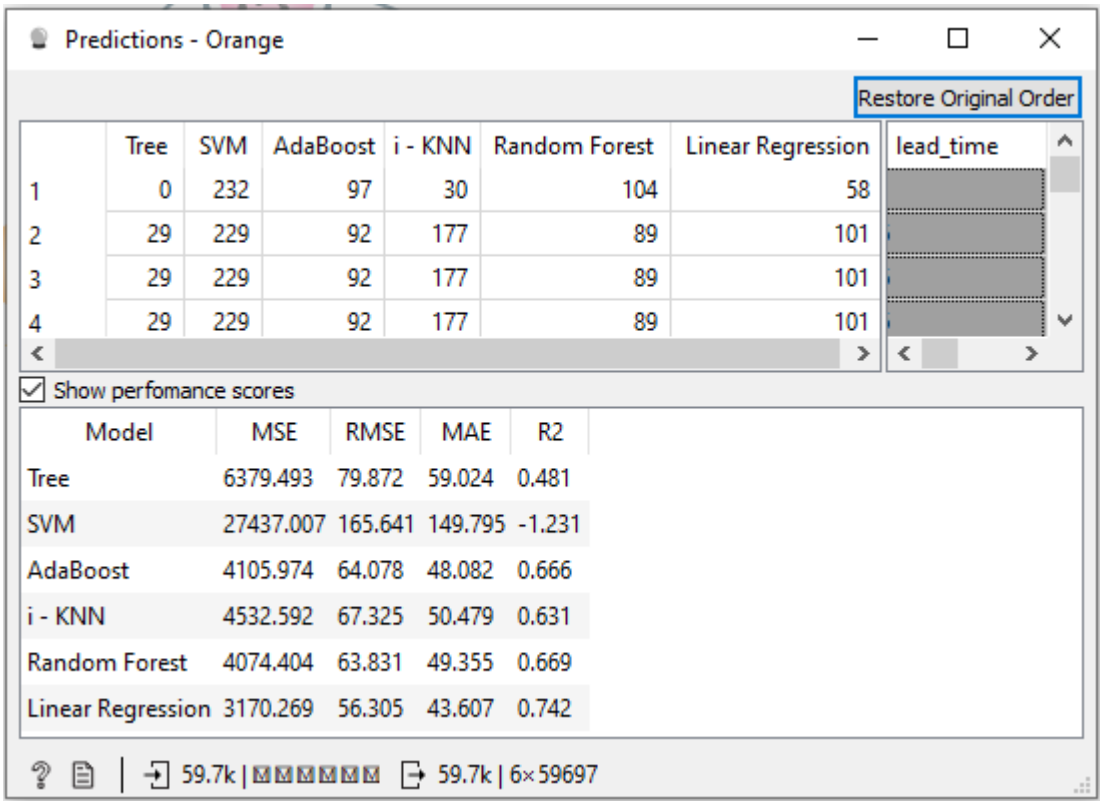


Figure 85 - Scale target predictions scheme



	Tree	SVM	AdaBoost	i - KNN	Random Forest	Linear Regression	lead_time
1	0	232	97	30	104	58	
2	29	229	92	177	89	101	
3	29	229	92	177	89	101	
4	29	229	92	177	89	101	

Model	MSE	RMSE	MAE	R2
Tree	6379.493	79.872	59.024	0.481
SVM	27437.007	165.641	149.795	-1.231
AdaBoost	4105.974	64.078	48.082	0.666
i - KNN	4532.592	67.325	50.479	0.631
Random Forest	4074.404	63.831	49.355	0.669
Linear Regression	3170.269	56.305	43.607	0.742

Figure 86 – Scale target predictions results

6.1.5 Test and Score for the scale target variable

The process to this type of test is similar and the metrics are the same. However, the results are displayed in a slightly different way. The goal of this test is to verify which algorithm had the best performance for the used dataset. Another different element to this subtopic is that a *Distributions* tool is added to the scheme with the goal of displaying further information of each algorithm.

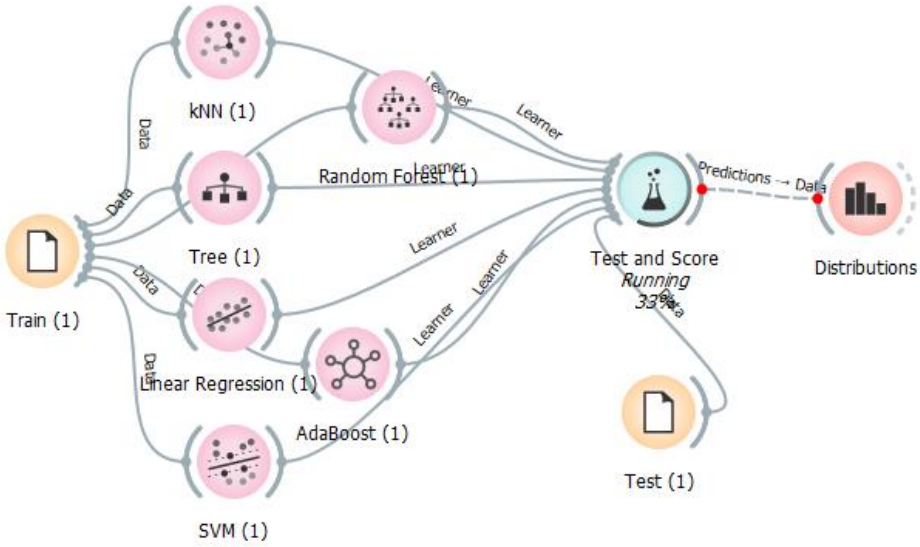


Figure 87 - Scale target Test and Score scheme

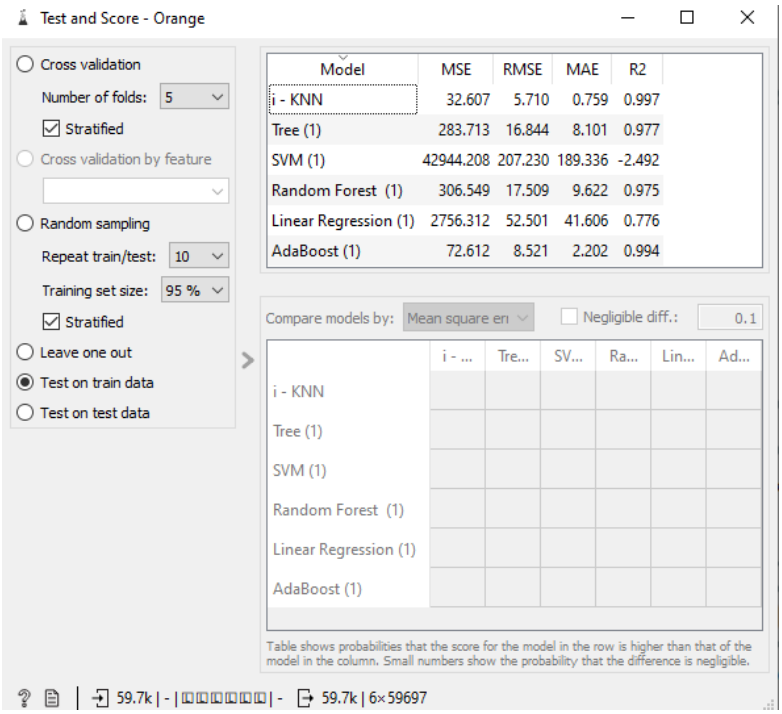


Figure 88 - Scale target Test and Score test on train data

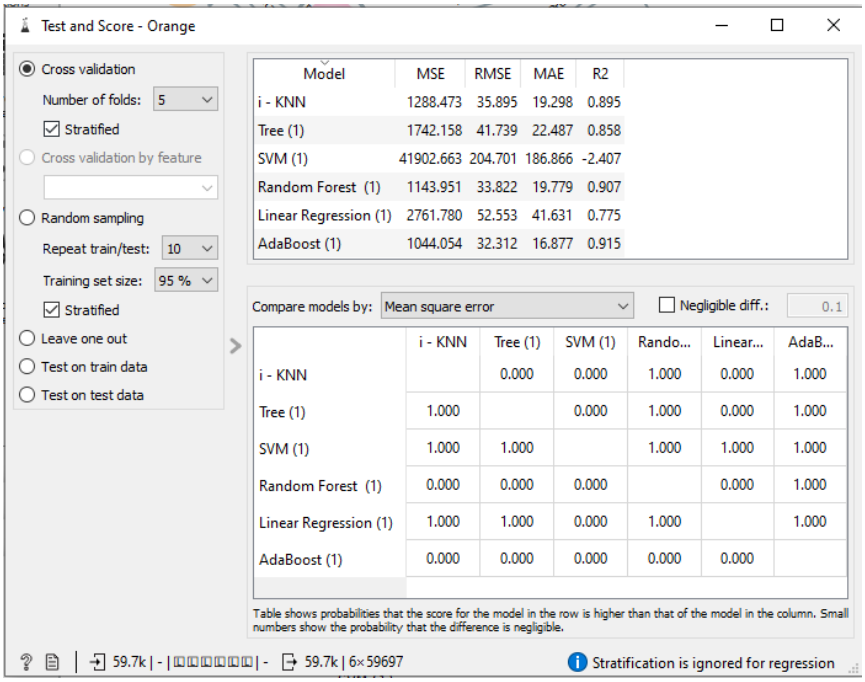


Figure 89 - Scale target Test and Score cross validation

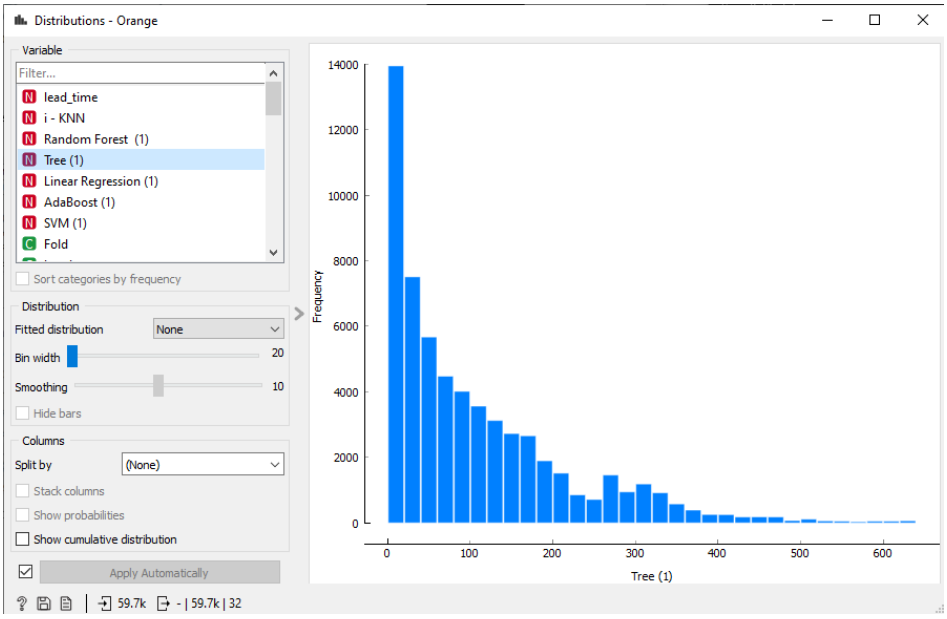


Figure 90 - Scale target Test and Score distribution example (decision tree)

6.2 Unsupervised learning

6.2.1 k-means

K-Means is a popular clustering algorithm used in unsupervised learning. By this approach, the dataset is partitioned into K distinct subgroups or clusters where each data point belongs to only one group. This algorithm assigns each data point to one of the groups based on its features, iteratively. This results on K clusters with feature similarities. Any type of variable can be used, however, numerical variables provide better outcomes since k-means calculates distance between values.

To display the process in k-means learning, the Orange provides the widget **K-Means**. First, the dataset is divided into two clusters and is analysed with a Scatter Plot (***widget Scatter Plot***) and a Boxplot (***widget Boxplot***). Then the same process is completed but with five clusters instead. Is important to define clusters number by clicking the K-means widget and setting the number of clusters to *Fixed* and the desired number. For the scatter plot two scale variables were used while for the boxplot, the same scale variable was used, this with a categorical type.

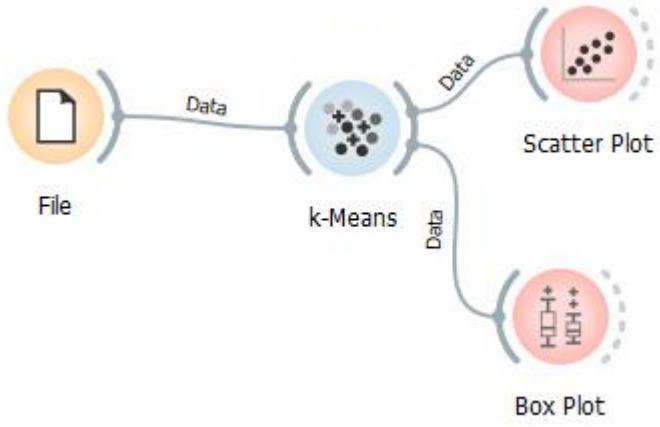


Figure 91 – K-means scheme

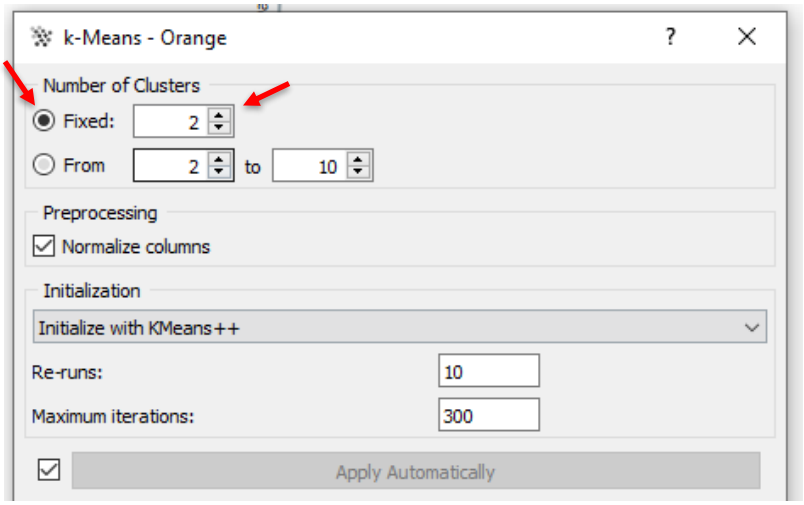


Figure 92 - K-means cluster configuration (2 clusters)

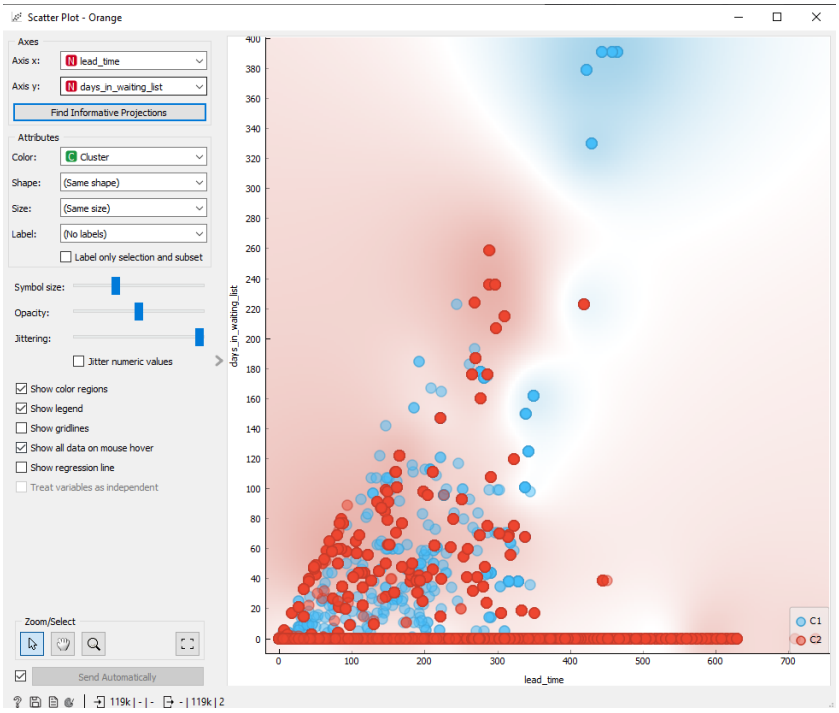


Figure 93 - K-Means scatterplot (2 clusters)

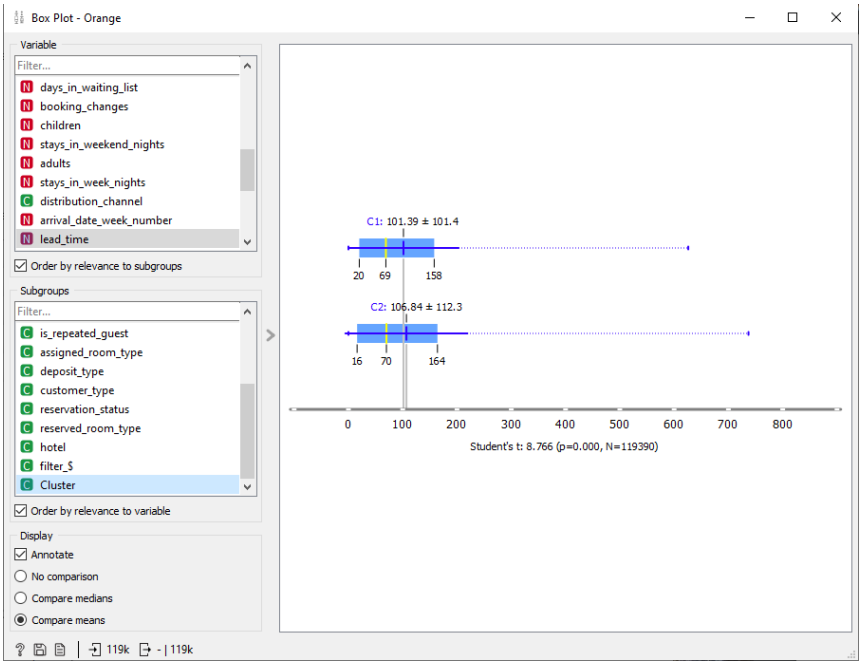


Figure 94 - K-Means boxplot (2 clusters)

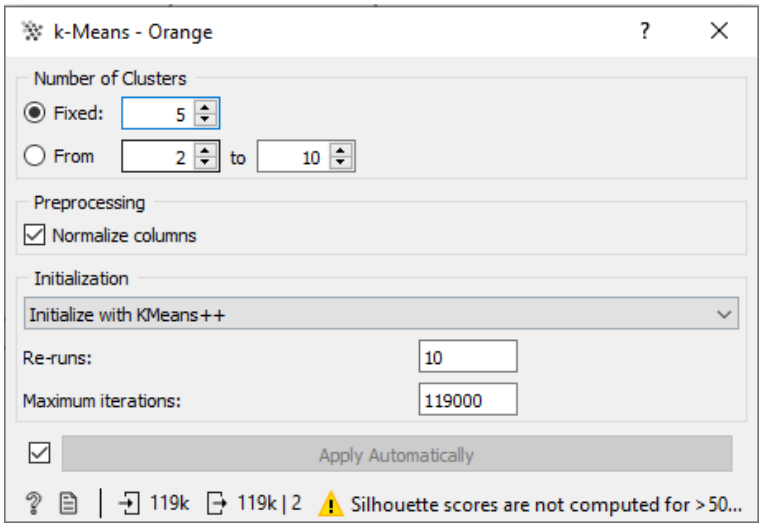


Figure 95 - K-means cluster configuration (5 clusters)

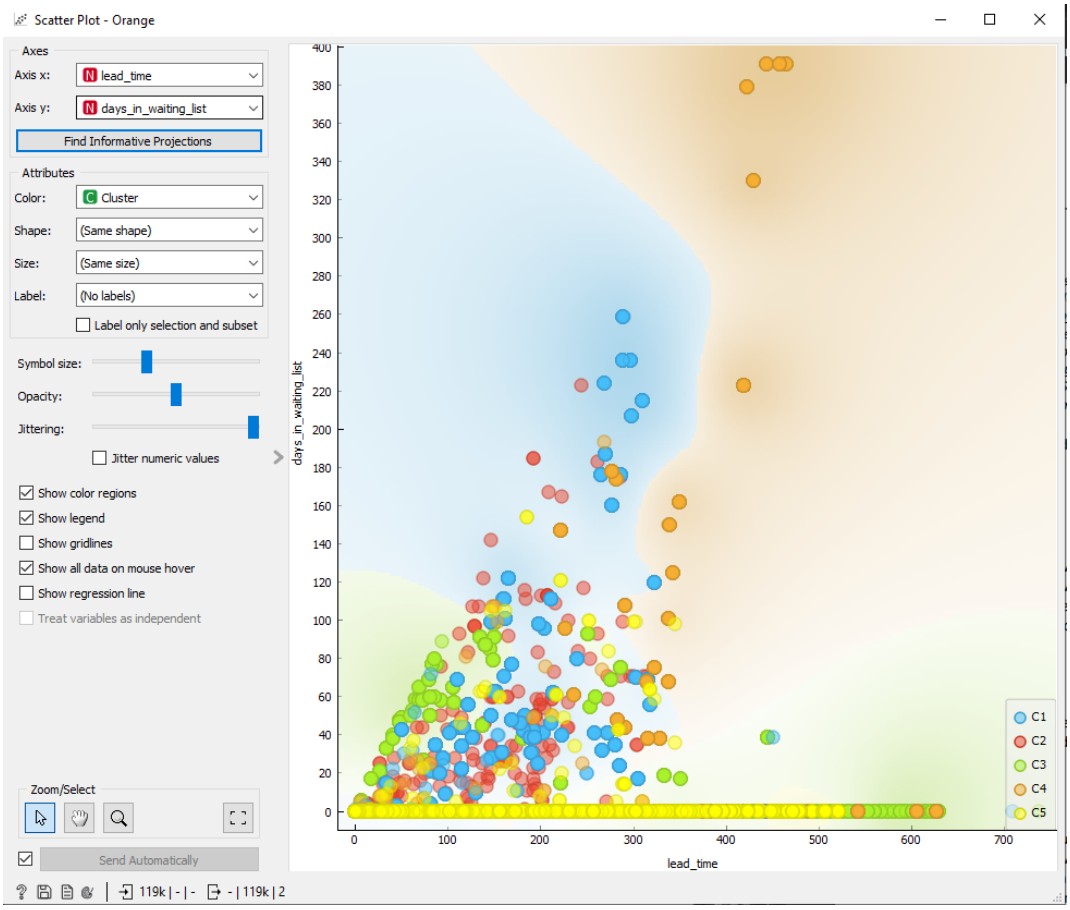


Figure 96 – K-means scatterplot (5 clusters)

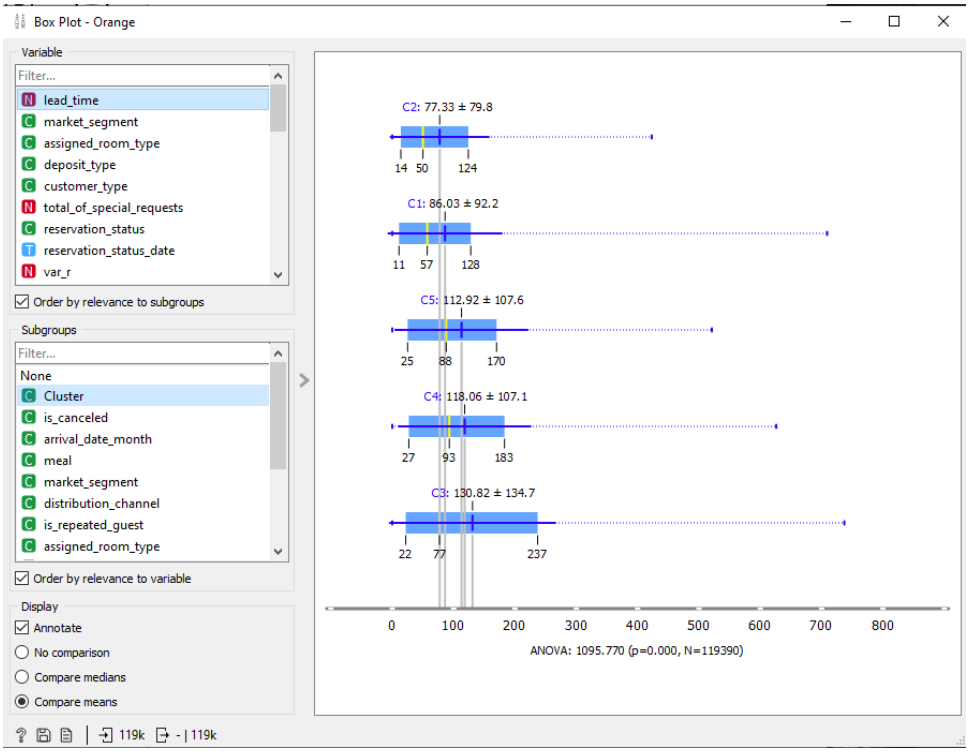


Figure 97 – K-Means boxplot (5 clusters)

6.2.2 Silhouette metric

The *Silhouette* metric is a measure of how similar an object is to its own cluster, comparing to other clusters. The silhouette value ranges from -1 and 1, where a high value indicates that the object is well matched to its own cluster. In the case of most objects having a high value, the clustering configuration is appropriate. If the contrary happens, the clustering configuration may have too many or too few clusters.

Through the ***Silhouette metric*** widget, the dataset can be divided into the optimal number of clusters. The result can be analysed with a Scatter Plot (***widget Scatter Plot***) and a Boxplot (***widget Boxplot***) as well. To utilize this metric, the dataset must have less than 5000 samples, in the case of having more, the dataset should be divided, by the ***Data Sampler*** widget. This way, by using only 4999 values, the Silhoutte metric displays the optimal number of clusters, in this case, being 2.

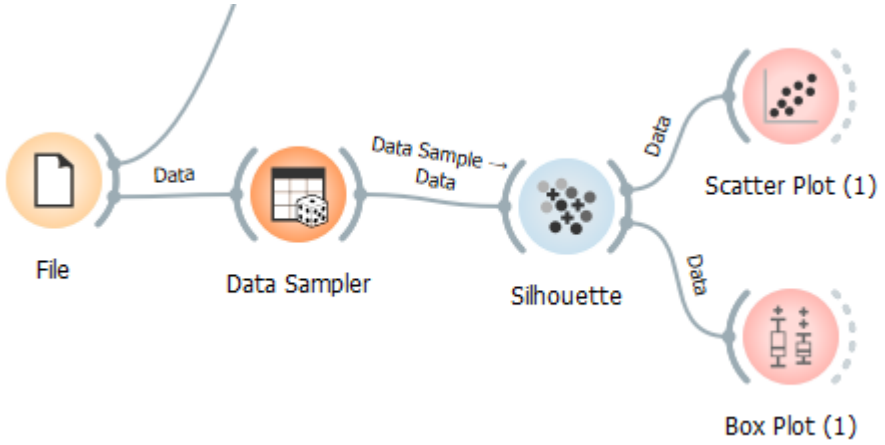


Figure 98 - K-means Silhouette scheme

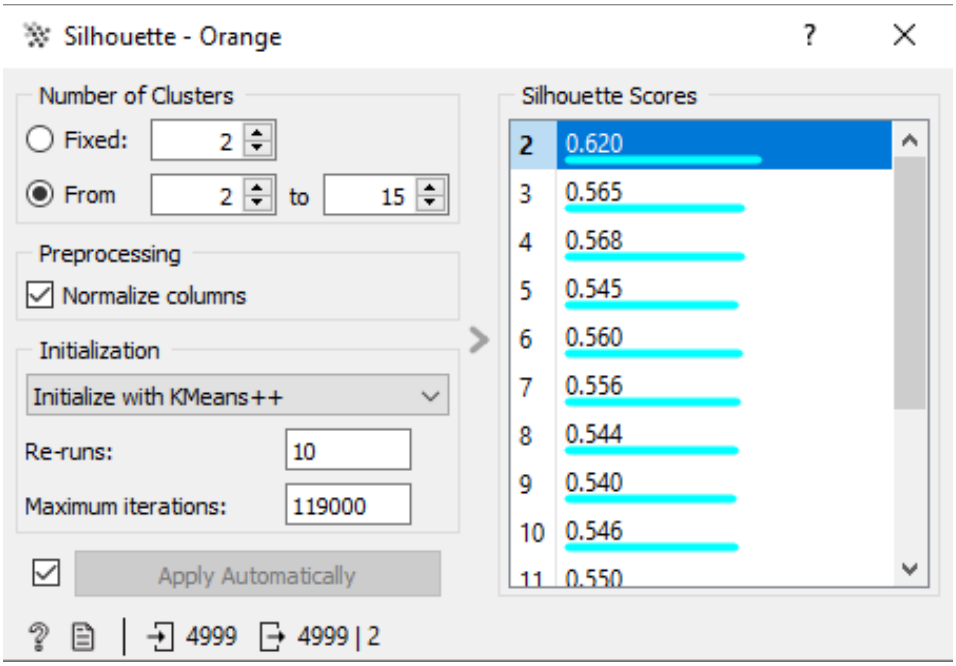


Figure 99 - Silhouette metric

6.2.3 Hierarchical clustering

Hierarchical clustering seeks to build an hierarchy of clusters through analysis. This algorithm starts by treating each data point as a cluster and proceeds by identifying the other closest cluster and merges them together. This process is repeated until it reaches a single cluster. This clustering can be done in a agglomerative (bottom-up) or divisive (top-down) approach.

To demonstrate this algorithm, a dendrogram can be obtained through ***Distances*** and ***Hierarchical Clustering*** widgets on Orange. Then, the elements of the dendrogram can be used in a Scatter Plot (***widget Scatter Plot***) and a Boxplot (***widget Boxplot***) for further analysis.

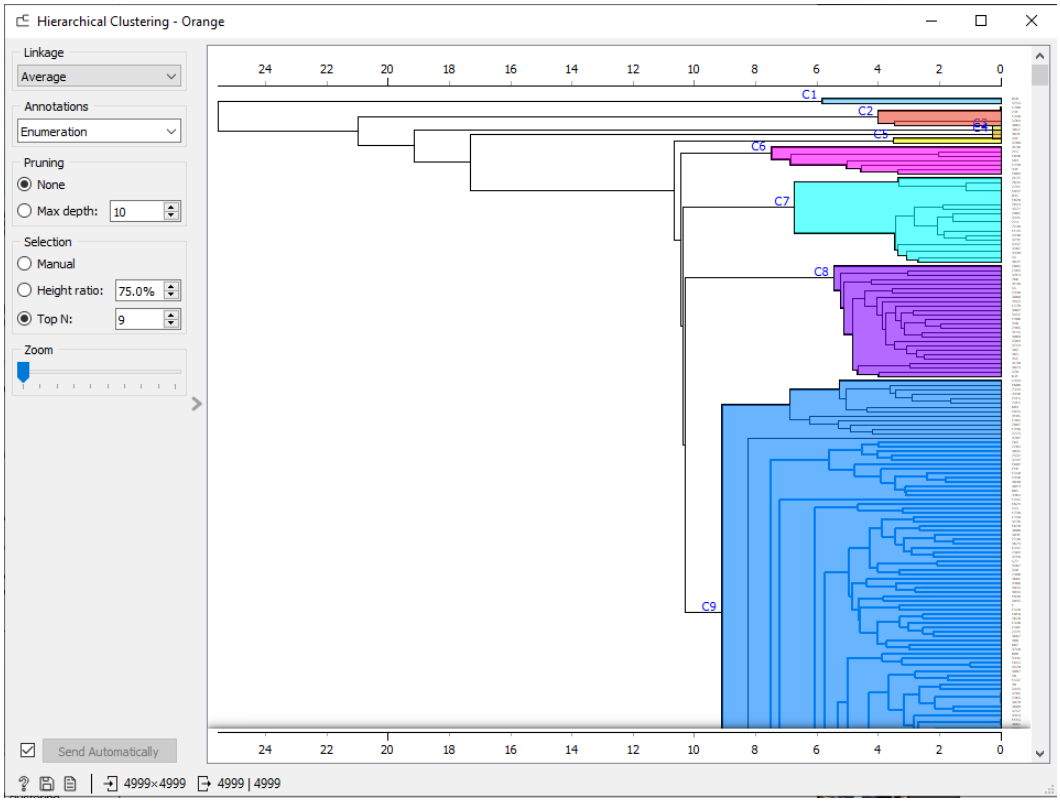


Figure 100 – Dendrogram

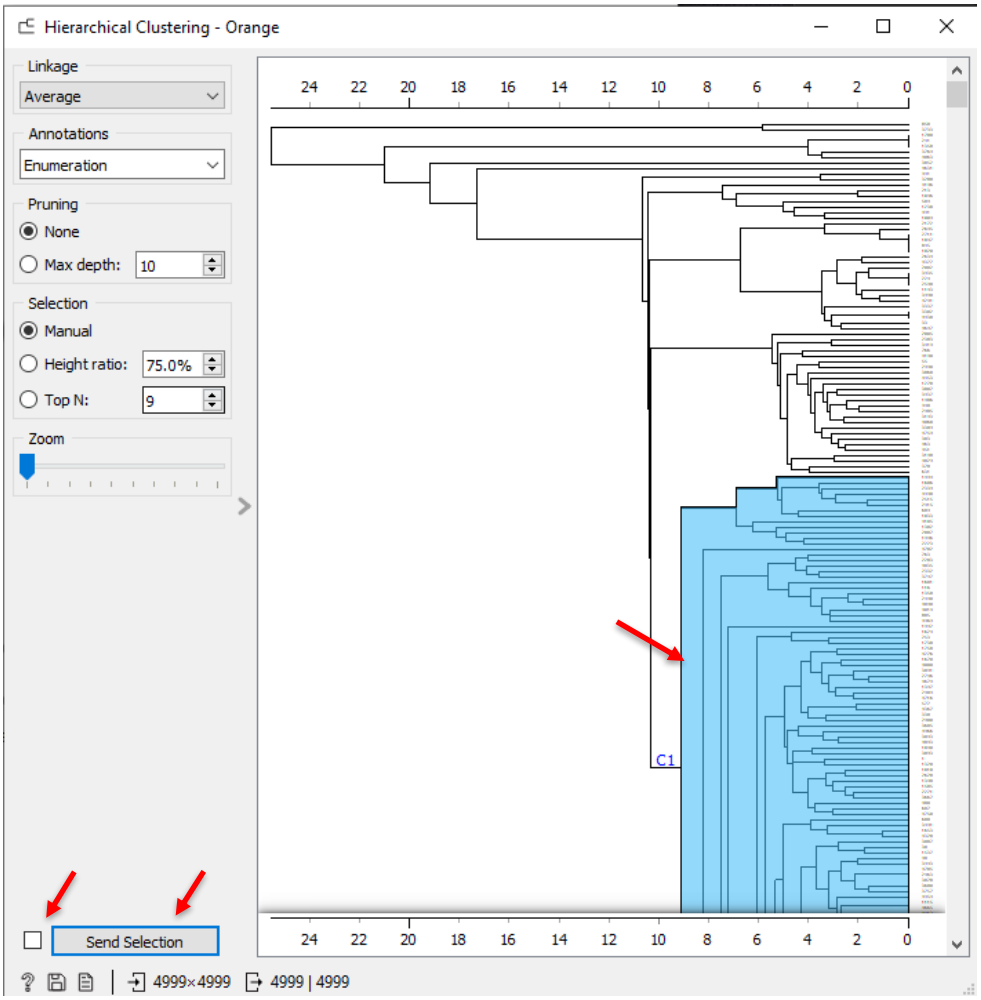


Figure 101 - Cluster selection (for plotting)

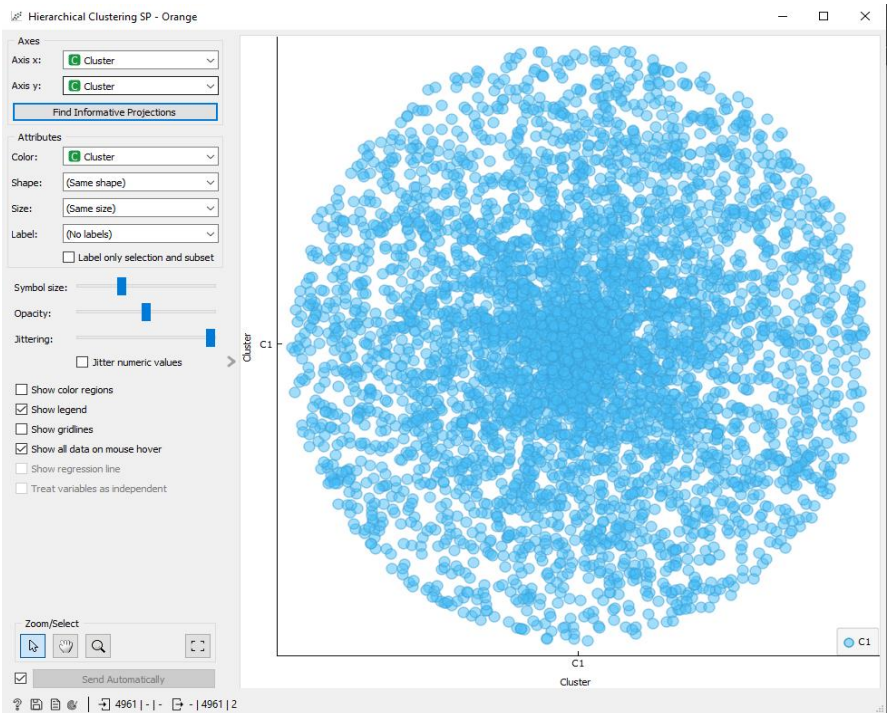


Figure 102 – Hierarchical Clustering scatter plot

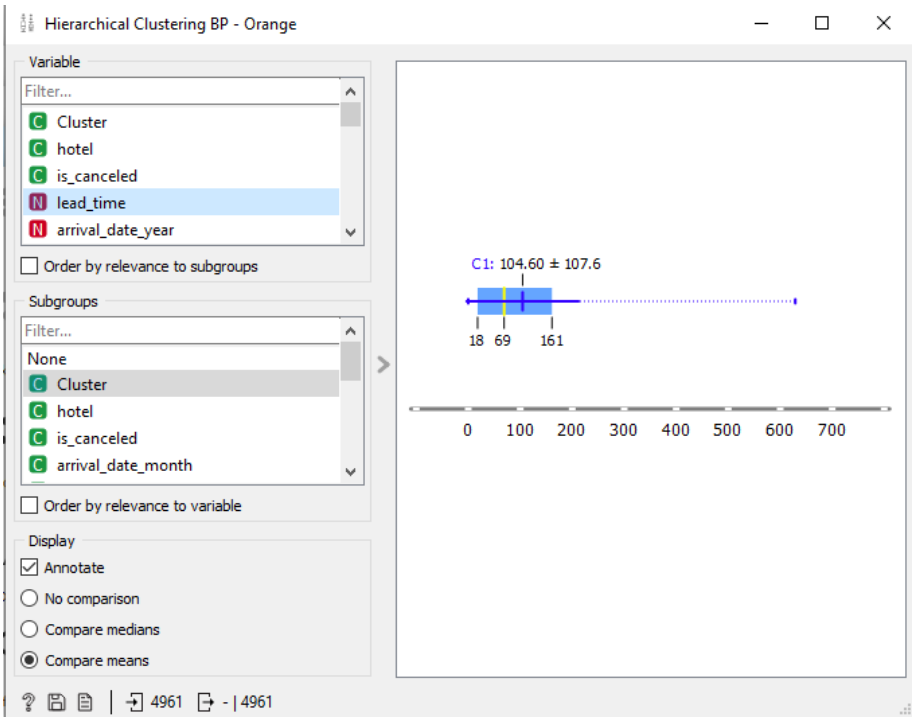


Figure 103 – Hierarchical Clustering boxplot (1 cluster)

6.2.4 DBSCAN

DBSCAN is a density-based clustering algorithm. It works by grouping points that are closely sorted together and by marking points that lie alone in low-density regions as outliers. For each point in a cluster,

the neighbourhood of a given radius must contain at least a minimum number of points. Opposed to k-means, DBSCAN does not require one to initially specify the number of clusters in the data.

The ***Neighbourhood Distance*** and the ***Core Point Neighbours*** can be defined to obtain any number of clusters, however, in this example, the number was two, so by changing the values, two clusters can be obtained, in this case, by reducing the core point neighbors by half. These clusters can be displayed with a Scatter Plot (***widget Scatter Plot***). Is important to note that if the dataset is composed of by a large number of samples, it should be reduced by a ***Data Sampler***, since the large data can crash the software.

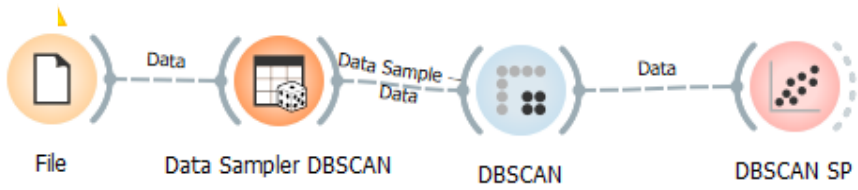


Figure 104 - DBSCAN scheme

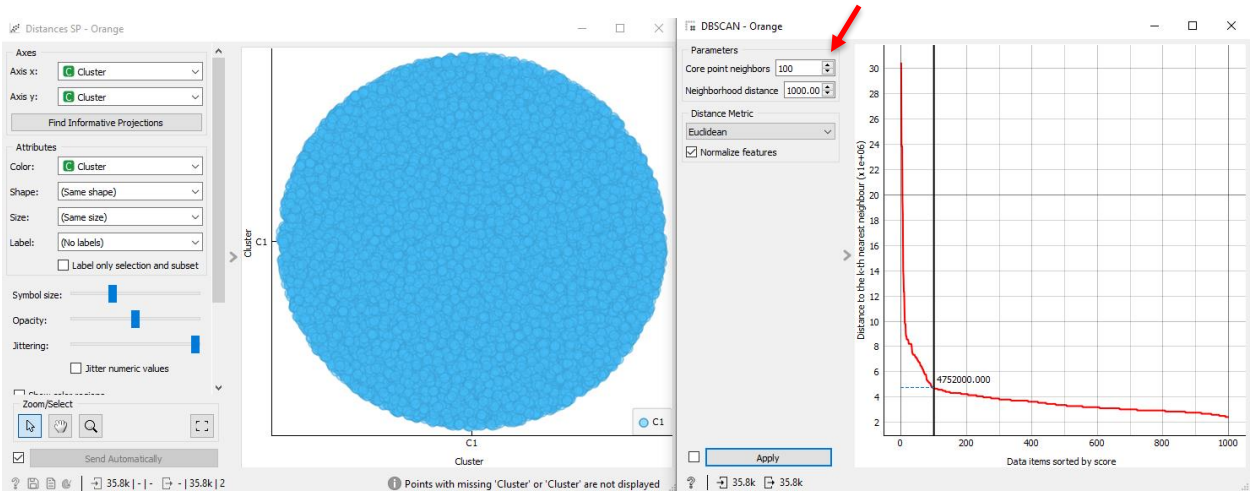


Figure 105 - 100 Core neighbors (DBSCAN)

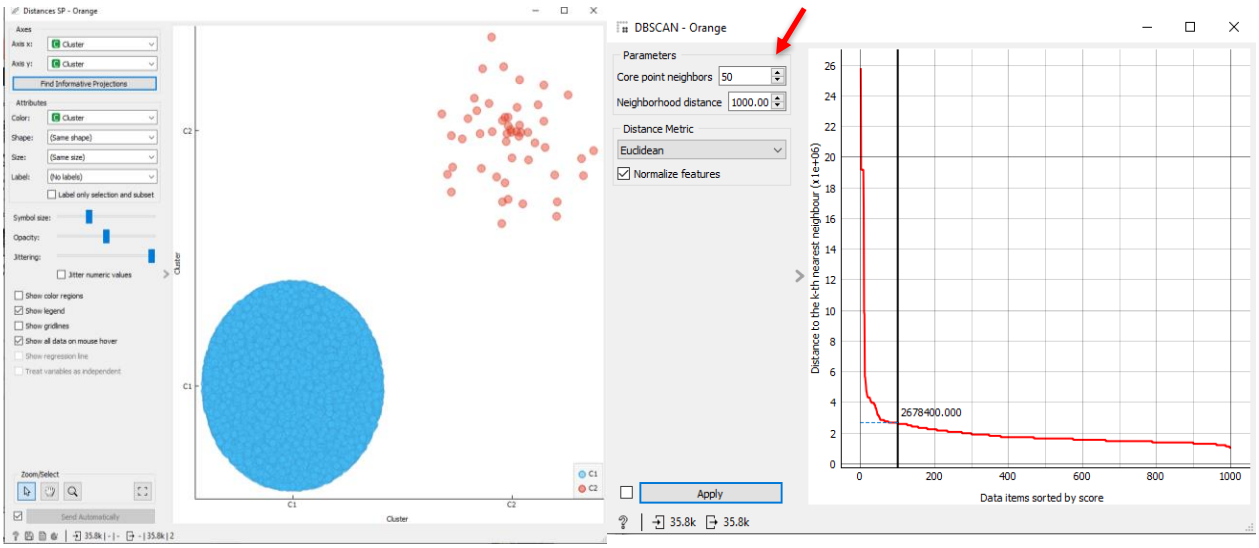


Figure 106 - 50 Core neighbors (DBSCAN)

6.2.5 PCA

PCA is a statistical procedure that aims to convert a set of observations of possibly correlated variables, into a set of values of linearly uncorrelated variables. These linearly uncorrelated variables are called principal components. The goal of PCA is to identify and quantify the patterns in the data. It reduces the dimensionality of the data by performing this transformation, which is uncorrelated, seeking to know the maximum amount of variance in the data.

The minimum number of principal components that can explain at least 70% of the target feature variance can be obtained with ***widget PCA***. To analyse the target variable, the dataset features' component composition and the variance explained by each component, a data table (***widget Data Table***) should be used. The results can be further investigated with a Scatter Plot (***widget Scatter Plot***).

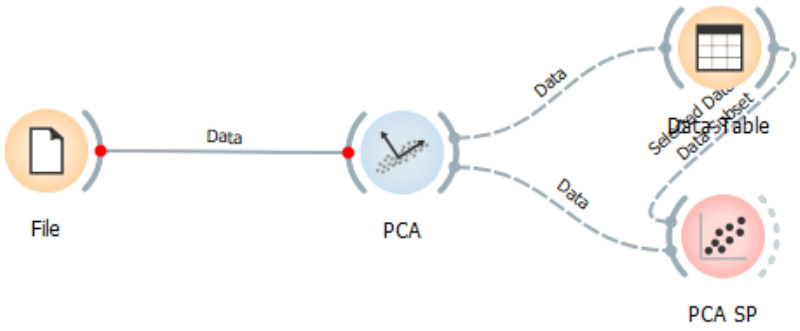


Figure 107 – PCA scheme

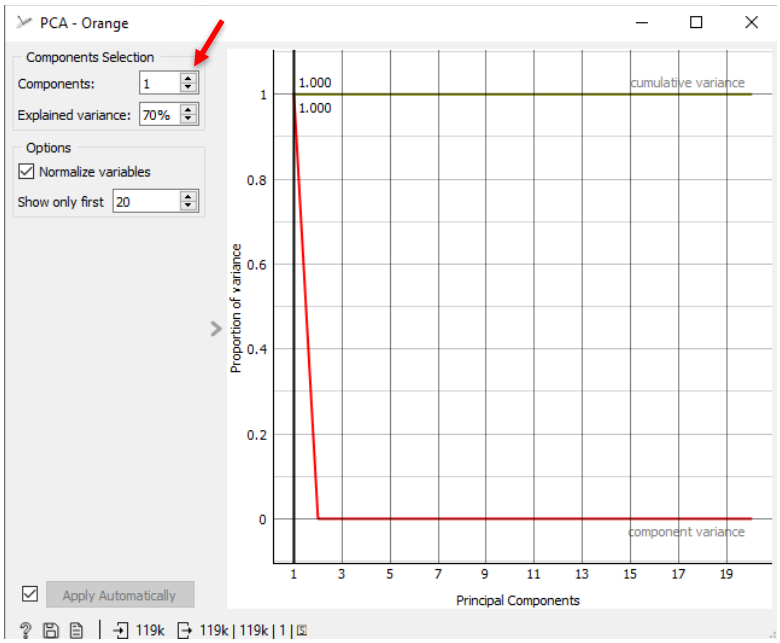


Figure 108 - PCA minimum components

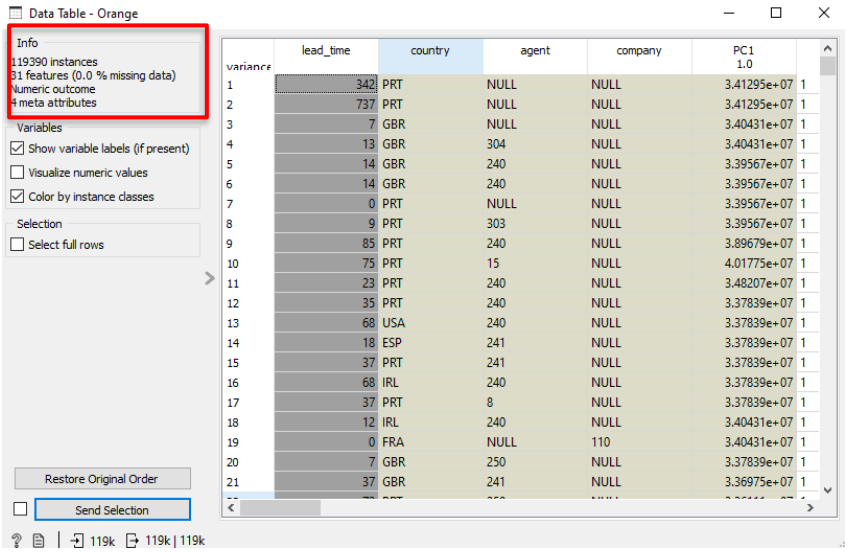


Figure 109 - PCA data table

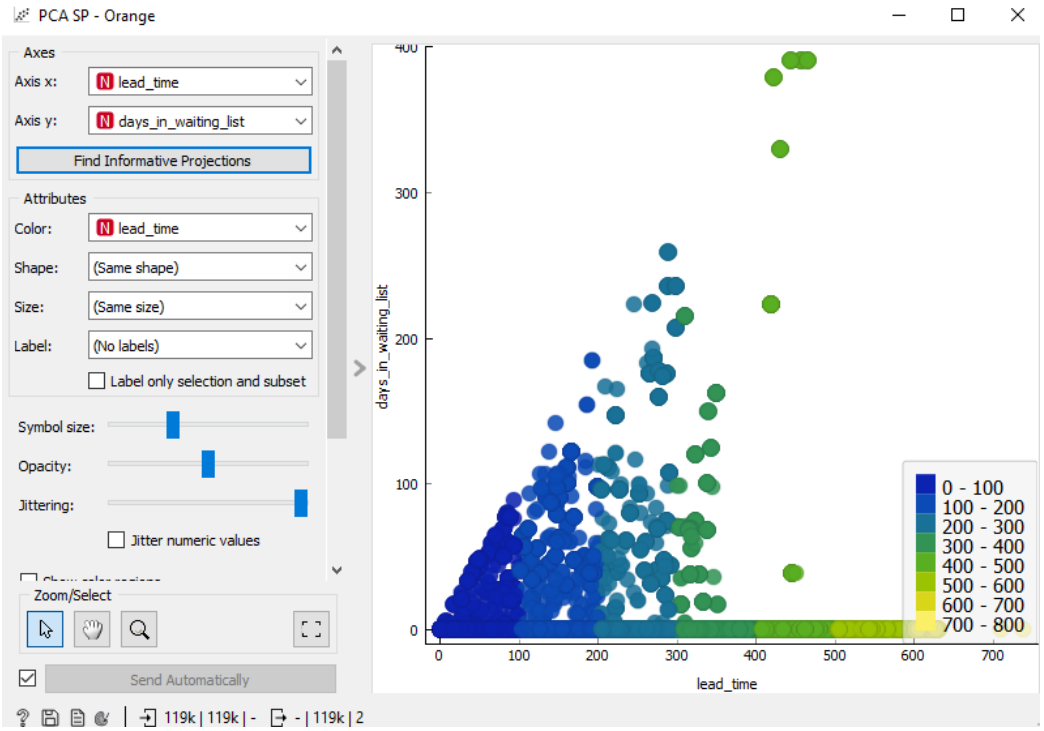


Figure 110 - Scatter Plot (PCA)

6.2.6 Feature ranking

By using the ***Rank*** widget and its metrics (*e.g.* information gain, Gini Decrease and ANOVA), the best three features to explain a categorical target, can be obtained. The features' behaviour, with or without subgrouping, can be displayed on a boxplot (***widget Boxplot***). The features distribution, with or without subgrouping, can be seen with a distribution graph (***widget Distributions***). Unfortunately, due to a software technical problem, the rank feature to identify the features that better explain a target was not feasible.

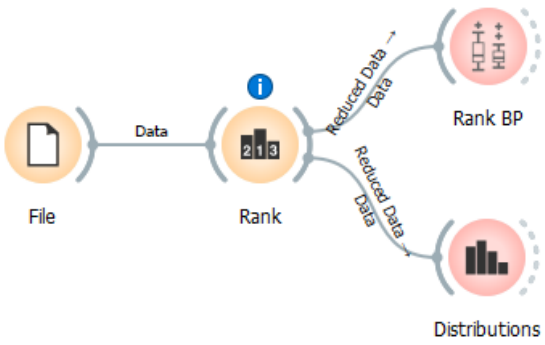


Figure 111 - Feature ranking scheme

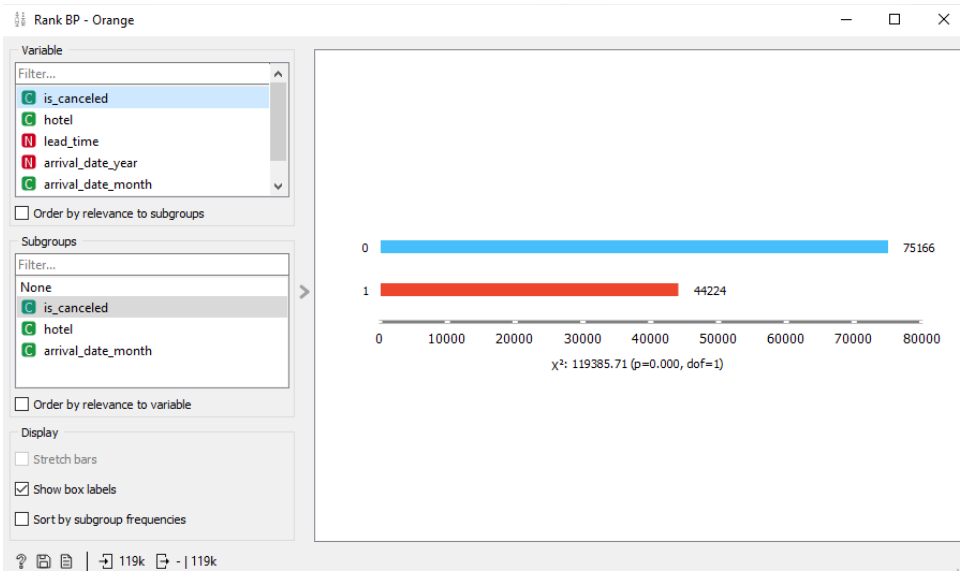


Figure 112 - Feature rank boxplot

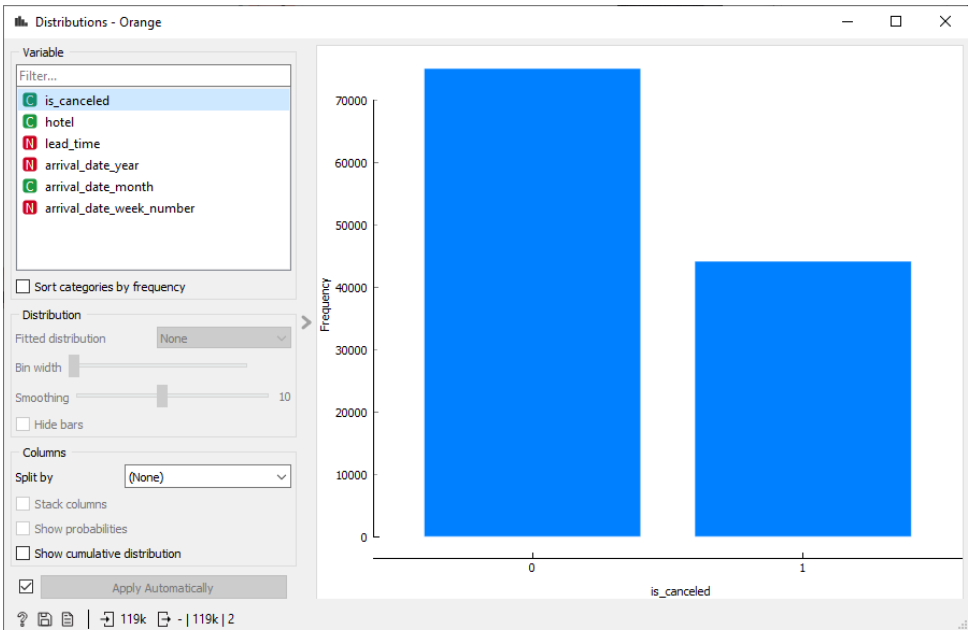


Figure 113 - Feature rank distributions

7 Python – Supervised learning

Python is a high level programming language, commonly used for data science due to its simplicity, flexibility and high number of libraries. These libraries include the *scikit-learn*, *TensorFlow* and *PyTorch* that offer extensive functionalities for building and training supervised unsupervised learning models.

7.1 Linear regression

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. The main goal is to predict the value of the dependent variable based on the values of the independent variables.

7.1.1 needed packages

Before creating or using a python program, is important to consider the needed libraries for the methods required for the script. Since the respective code, for this topic, regards mathematical operations and data file reading, the first libraries to consider are *matplotlib.pyplot* (data plotting), *pandas* (data manipulation), *pylab* (mathematical computations and plotting) and *numpy* (scientific computing). Each of these libraries should be imported with ***import*** *library_name*, to create an alias for these libraries they can be imported as ***import*** *library_name* ***as*** *alias_name*.

```
import matplotlib.pyplot as plt
import pandas as pd
import pylab as pl
import numpy as np
%matplotlib inline
```

Figure 114 – Libraries imports and aliases

7.1.2 Data analysis and manipulation

Before proceeding with the dataset learning methods, its data should be analysed and, if necessary, manipulated. With ***pd.read_csv(“****file_path/filename****”)*** the dataset data can be retrieved into a variable, in this case ***df***. Then, this dataset can be displayed with ***df.head()***, the *head* method will, by default, display the first five lines of the dataset, to alter this configuration a number can be inserted inside the *head()* method (*e.g. head(7)* will show the first 7 lines of the dataset).

```
df = pd.read_csv("dataset_hotel_bookings.csv")
# take a look at the dataset
df.head()
```

```
C:\Users\gabri\AppData\Local\Temp\ipykernel_11480\10599884
ption on import or set low_memory=False.
  df = pd.read_csv("dataset_hotel_bookings.csv")
```

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month
0	1	0	342	2015	7
1	1	0	737	2015	7
2	1	0	7	2015	7
3	1	0	13	2015	7
4	1	0	14	2015	7

5 rows × 35 columns

Figure 115 - Data acquisition and reading

The *describe()* method reads the dataset and displays various statistics such as mean, minimum and maximum value, data count, etc.

```
# summarize the data
df.describe()
```

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month
count	119390.000000	119390.000000	119390.000000	119390.000000	119390.000000
mean	0.335539	0.370416	104.011416	2016.156554	6.552483
std	0.472181	0.482918	106.863097	0.707476	3.090619
min	0.000000	0.000000	0.000000	2015.000000	1.000000
25%	0.000000	0.000000	18.000000	2016.000000	4.000000
50%	0.000000	0.000000	69.000000	2016.000000	7.000000
75%	1.000000	1.000000	160.000000	2017.000000	9.000000
max	1.000000	1.000000	737.000000	2017.000000	12.000000

8 rows × 24 columns

Figure 116 - Data statistical analysis

Specific columns can be displayed with ***variable = df[['column1', colum2, …, 'column_n']] > variable.head()***.

```
cdf = df[['hotel','lead_time','arrival_date_month','stays_in_weekend_nights', 'stays_in_week_nights']]
cdf.head(9)
```

	hotel	lead_time	arrival_date_month	stays_in_weekend_nights	stays_in_week_nights
0	1	342	7	0	0
1	1	737	7	0	0
2	1	7	7	0	1
3	1	13	7	0	1
4	1	14	7	0	2
5	1	14	7	0	2
6	1	0	7	0	2
7	1	9	7	0	2
8	1	85	7	0	3

Figure 117 - Display specific columns

7.1.3 Data plotting

To plot any columns of the dataset the method *hist()* can be used to create an histogram of such columns, with ***var = df[['column1', colum2, …, 'column_n']] > var.hist().*** Another data plotting type can be linear relations between two features. For this approach the ***plt.scatter(cdf.column1, cdf.column2, color = 'color_to_plot')*** is used. To apply labels to each axis, the methods ***plt.xlabel("x_label")*** and ***plt.ylabel("y_label")*** are used for the x and y axis, respectively. To display any plot, the method ***plt.show()*** is required.

```
viz = cdf[['hotel','lead_time','arrival_date_month','stays_in_weekend_nights', 'stays_in_week_nights']]
viz.hist()
plt.show()
```

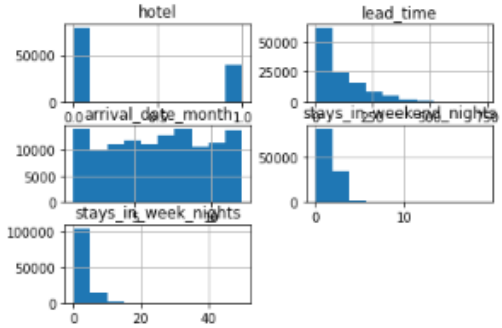


Figure 118 – Histogram plotting

```
plt.scatter(cdf.arrival_date_month, cdf.lead_time,  color='blue')
plt.xlabel("Arrival month")
plt.ylabel("Days between entrance and arrival")
plt.show()
```

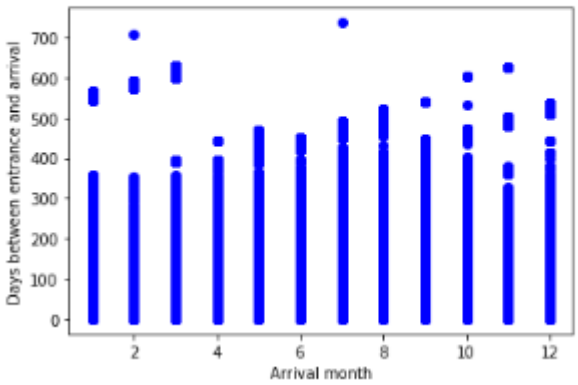


Figure 119 - Linear regression plotting

7.1.4 Train and test dataset creation

Train/Test Split involves splitting the dataset into training and testing sets respectively, which are mutually exclusive. After which, the algorithm uses the training set to model itself, to then be tested using the testing set. This will provide a more accurate evaluation on out-of-sample accuracy because the testing dataset is not part of the dataset that have been used to train the data. It is more realistic for real world problems.

To create both datasets, a 80-20 rule will be used to divide the datasets. 80% of the dataset is set to be the training set and the remaining is set to be used on testing the algorithm. This division is made with ***msk = np.random.rand(len(df)) < 0.8***, then the distribution is made by using ***train = cdf[msk]*** and ***test = cdf[~msk]***. Because the ~ represents the opposite of a variable, the test dataset is created with the remaining of the train dataset that had 80% (0.8) of the data. The train data distribution can be plotted with the already known methods.

```
msk = np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]

plt.scatter(train.stays_in_weekend_nights, train.lead_time,  color='blue')
plt.xlabel("Stays in weekend nights")
plt.ylabel("Days between entrance and arrival")
plt.show()
```

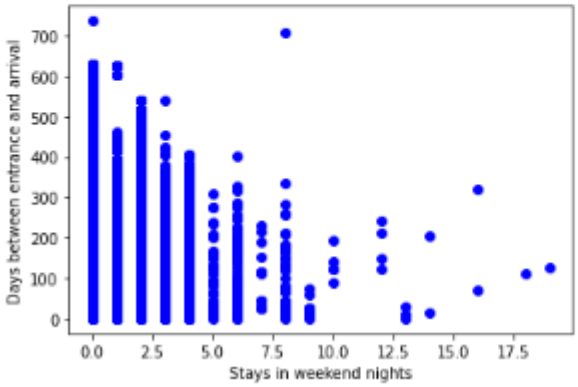


Figure 120 – Linear regression Train and Test dataset setting

7.1.5 Modelling and plot outputs

Linear Regression fits a linear model with coefficients B = (B1, ..., Bn) to minimize the 'residual sum of squares' between the independent x in the dataset, and the dependent y by the linear approximation. Coefficient and Intercept in the simple linear regression, are the parameters of the fit line. Given that it is a simple linear regression, with only 2 parameters, and knowing that the parameters are the intercept and slope of the line, *sklearn* can estimate them directly from the data. All of the data must be available to traverse and calculate the parameters. The results should be outputted as a plot.

To the operations of this subtopic, the *sklearn* library must be imported, more specifically the *linear_model* module. Then the regression model is allocated to an instance, with ***regr = linear_model.LinearRegression()***, before the dataset is divided to a train dependent and independent variable with ***train_x/train_y = np.asanyarray(train[['column']])*** , respectively. To conclude, the linear regression model is trained with ***regr.fit(train_x, train_y)***. Then the coefficients calculated are displayed with the use of the *print* method and the datasets regressed are displayed with a linear plot and intercepted with a red regression line.

```
from sklearn import linear_model
regr = linear_model.LinearRegression()
train_x = np.asanyarray(train[['stays_in_weekend_nights']])
train_y = np.asanyarray(train[['lead_time']])
regr.fit (train_x, train_y)
# The coefficients
print ('Coefficients: ', regr.coef_)
print ('Intercept: ',regr.intercept_)

plt.scatter(train.stays_in_weekend_nights, train.lead_time,  color='blue')
plt.plot(train_x, regr.coef_[0][0]*train_x + regr.intercept_[0], '-r')
plt.xlabel("Stays in weekend nights")
plt.ylabel("Days between entrance and arrival")
```

```
Coefficients:  [[9.22892125]]
Intercept:  [95.59215845]

Text(0, 0.5, 'Days between entrance and arrival')
```

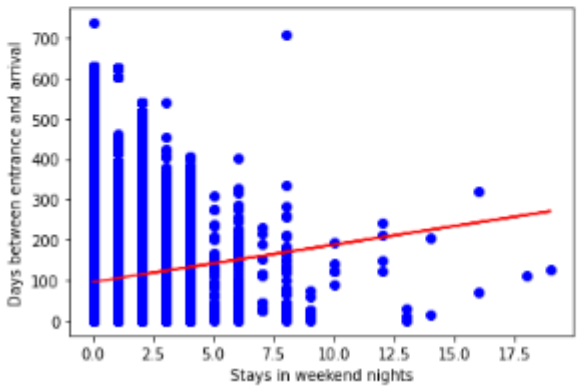


Figure 121 – Linear regression Modelling

7.1.6 Evaluation

The actual values and predicted values must be compared to calculate the accuracy of a regression model. Evaluation metrics provide a key role in the development of a model, as it provides insight to areas that require improvement.

There are different model evaluation metrics, but for this case the following metrics are used:

- Mean absolute error – Calculates the average error;
- Mean Squared Error (MSE) – The mean of the squared error. It's more popular than Mean absolute error because the focus is geared more towards large errors;
- Root Mean Squared Error (RMSE). - R-squared is not error but is a popular metric for accuracy. It represents how close the data is to the fitted regression line. The higher the R-squared, the better the model fits the data. Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse).

These metrics are applied after creating three variables to test and assigning them to a test feature for the x axis, y axis and to the prediction model, respectively. Then the metrics are calculated and displayed with the *print* method.

```
from sklearn.metrics import r2_score

test_x = np.asanyarray(test[['stays_in_weekend_nights']])
test_y = np.asanyarray(test[['lead_time']])
test_y_ = regr.predict(test_x)

print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y_ - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((test_y_ - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y_ , test_y) )
```

```
Mean absolute error: 84.45
Residual sum of squares (MSE): 11552.58
R2-score: -133.90
```

Figure 122 - Linear regression model evaluation

7.2 KNN

K-Nearest Neighbors is an algorithm for supervised learning. Where the data is 'trained' with data points corresponding to their classification. Once a point is to be predicted, it takes into account the 'K' nearest points to it to determine its classification.

7.2.1 needed packages

Before creating or using a python program, is important to consider the needed libraries for the methods required for the script. Each of these libraries should be imported with ***import*** *library_name*, to create an alias for these libraries they can be imported as ***import*** *library_name* ***as*** *alias_name*. The libraries for the python script respective to KNN are the following:

- ***itertools*** – Provides tools for creating efficient iterators (for loops).
- ***numpy*** – Essential for numerical computations and handling arrays.
- ***matplotlib.pyplot*** – Used for creating a wide variety of static, animated and interactive visualizations.
- ***matplotlib.ticker.NullFormatter*** – Disables tick labelling on plots.
- ***pandas*** – Ideal for data manipulation and analysis, especially with tabular data.
- ***matplotlib.ticker*** – Offers classes for tick locating and formatting on plots.
- ***sklearn.preprocessing*** – Provides tools for data preprocessing and feature scaling.

```
import itertools
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import pandas as pd
import numpy as np
import matplotlib.ticker as ticker
from sklearn import preprocessing
%matplotlib inline
```

Figure 123 - KNN needed packages

7.2.2 Data visualization and analysis

Before proceeding with the dataset learning methods, its data should be analysed and, if necessary, manipulated. With ***pd.read_csv("****file_path/filename****")*** the dataset data can be retrieved into a variable, in this case ***df***. Then, this dataset can be displayed with ***df.head()***, the *head* method will, by default, display the first five lines of the dataset, to alter this configuration a number can be inserted inside the *head()* method (*e.g. head(7)* will show the first 7 lines of the dataset).

```
df = pd.read_csv('dataset_hotel_bookings.csv')
df.head()
```

```
C:\Users\gabri\AppData\Local\Temp\ipykernel_21740\55142793
tion on import or set low_memory=False.
  df = pd.read_csv('dataset_hotel_bookings.csv')
```

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month
0	1	0	342	2015	7
1	1	0	737	2015	7
2	1	0	7	2015	7
3	1	0	13	2015	7
4	1	0	14	2015	7

5 rows × 35 columns

Figure 124 - KNN data visualization

Several other data can be retrieved from the imported dataset. The number of values of a certain column can be obtained with ***df.['column'].value_counts()***. An histogram can be obtained, as well, with the ***df.hist(column='column', bins=****number_of_bins****)***.

```
df['hotel'].value_counts()
```

```
0    79330
1    40060
Name: hotel, dtype: int64
```

```
df.hist(column='arrival_date_month', bins=50)
```

```
array([[<AxesSubplot:title={'center':'arrival_date_month'}>]],
      dtype=object)
```

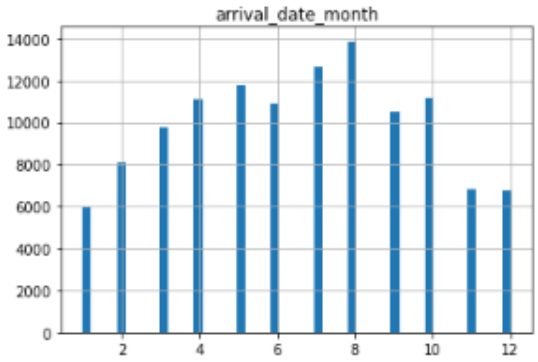


Figure 125 – KNN column visualization

The columns of the dataset can be displayed with ***df.columns***. These columns, or some of them, can be converted to a Numpy array to further practices. The created array can then be normalized with ***preprocessing.StandardScaler().fit(X).transform(X.astype(****type****)).***

```
df.columns
```

```
Index(['hotel', 'is_canceled', 'lead_time', 'arrival_date_year',
       'arrival_date_month', 'arrival_date_week_number',
       'arrival_date_day_of_month', 'stays_in_weekend_nights',
       'stays_in_week_nights', 'adults', 'children', 'babies', 'meal',
       'country', 'market_segment', 'distribution_channel',
       'is_repeated_guest', 'previous_cancellations',
       'previous_bookings_not_canceled', 'reserved_room_type',
       'assigned_room_type', 'booking_changes', 'deposit_type', 'agent',
       'company', 'days_in_waiting_list', 'customer_type', 'adr',
       'required_car_parking_spaces', 'total_of_special_requests',
       'reservation_status', 'reservation_status_date', 'var_r', 'filter_$',
       'arrival_month'],
      dtype='object')
```

To use scikit-learn library, we have to convert the Pandas data frame to a Numpy array:

```
X = df[['hotel', 'is_canceled','lead_time', 'arrival_date_year', 'arrival_date_month', 'days_in_waiting_list', 'adults','previous
X[0:5]
```

Normalize Data

Data Standardization give data zero mean and unit variance, it is good practice, especially for algorithms such as KNN which is based on distance of cases:

```
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
X[0:5]
```

```
array([[ 1.40722407, -0.76704049,  2.22705112, -1.63476794,  0.14479897,
        -0.1319236 ,  0.24789727, -0.09155477, -0.72069411],
       [ 1.40722407, -0.76704049,  5.9233847 , -1.63476794,  0.14479897,
        -0.1319236 ,  0.24789727, -0.09155477, -0.72069411],
       [ 1.40722407, -0.76704049, -0.90781407, -1.63476794,  0.14479897,
        -0.1319236 , -1.47844749, -0.09155477, -0.72069411],
       [ 1.40722407, -0.76704049, -0.85166723, -1.63476794,  0.14479897,
        -0.1319236 , -1.47844749, -0.09155477, -0.72069411],
       [ 1.40722407, -0.76704049, -0.84230942, -1.63476794,  0.14479897,
        -0.1319236 ,  0.24789727, -0.09155477,  0.54066585]])
```

Figure 126 – KNN's dataset's columns and data normalization

7.2.3 Train and Test split

Out of Sample Accuracy is the percentage of correct predictions that the model makes on data that that the model has NOT been trained on. Doing a train and test on the same dataset will most likely have low out-of-sample accuracy, due to the likelihood of being over-fit.

It is important that the models have a high out-of-sample accuracy, because the purpose of any model is to make correct predictions on unknown data. So, one way to improve accuracy is to use an evaluation approach called Train/Test Split.

Train/Test Split involves splitting the dataset into training and testing sets respectively, which are mutually exclusive. After which, the algorithm is trained with the training set and tested with the testing set. This will provide a more accurate evaluation on out-of-sample accuracy because the testing dataset is not part of the dataset that has been used to train the data. It is more realistic for real world problems.

The split can be accomplished by using the *train_test_split* module from the *sklearn.model_selection* library. Then, a target variable to predict is set as a ***y*** variable, then, a train and test split is set to each axis, ***X*** and ***y***, the array and the target. The test sample size is set to 0.2 to be equivalent to 20% of the dataset's data

```
from sklearn.model_selection import train_test_split

y = df['is_canceled']

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

Figure 127 - KNN train and test split

7.2.4 Training and prediction

By applying 4 for the number of k's, the train model was created with the ***KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)*** method. The dataset was then tested/have values predicted with ***yhat = neigh.predict(X_test)***.

Training

Lets start the algorithm with k=4 for now:

```
#Classifier implementing the k-nearest neighbors vote.
from sklearn.neighbors import KNeighborsClassifier
k = 4
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

```
KNeighborsClassifier(n_neighbors=4)
```

Predicting

we can use the model to predict the test set:

```
yhat = neigh.predict(X_test)
yhat[0:5]
```

```
array([1, 0, 0, 0, 0], dtype=int64)
```

Figure 128 - KNN train and data prediction

7.2.5 Accuracy evaluation

In multilabel classification, accuracy classification score function computes subset accuracy. This function is equal to the *jaccard_similarity_score* function. Essentially, it calculates how match the actual labels and predicted labels are in the test set.

So by using the module *metrics* from the *sklearn* library, the train and test sets accuracy is calculated with ***metrics.accuracy_score(y_train, neigh.predict(X_train)))*** and ***metrics.accuracy_score(y_test, yhat))***, respectively. The same process can be done for a different number of k's. For example, 6.

```
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.9751444844626853
Test set Accuracy:  0.9559845883239803
```

```
k = 6
neigh6 = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
yhat6 = neigh6.predict(X_test)
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh6.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat6))
```

```
Train set Accuracy:  0.9724013736493844
Test set Accuracy:  0.957827288717648
```

Figure 129 - Accuracy evaluation for k=4 and k=6

7.2.6 Accuracy for different values for K

K in KNN is the number of nearest neighbors to examine and it is supposed to be specified by an user. To choose between the more accurate K, all the K's must have their respective accuracy tested, then the K with the most accurate results, should be selected for the algorithm. In the following example, 10 Ks will be tested. First, the number of k is set, then, an array full of zeros must be created to store the mean accuracy for each k. Then, the same is done but for the standard deviation of the respective Ks.

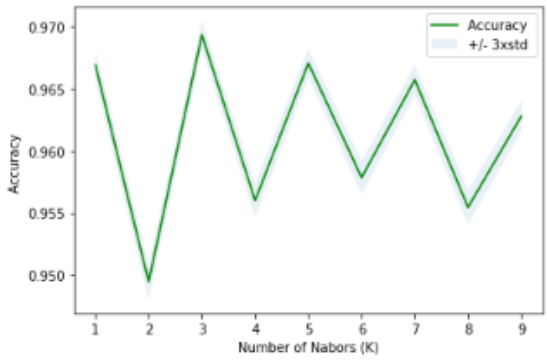
To test all the Ks, a loop is created to iterate over the values of n that represent the number of neighbours for the KNN algorithm. Inside this loop, the process of training and predicting, accuracy and its standard deviation calculating is repeated for each value of k.

Then, the result of the loop is presented to display which number of neighbours is optimal for the dataset. These results can be plotted and the best number of k can be also calculated with ***mean_acc.max()*** and ***mean_acc.argmax()+1***.

```
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];
for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)


    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc
```

Figure 130 – Accuracy for different values for K

```
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Nabors (K)')
plt.tight_layout()
plt.show()
```




```
print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```

```
The best accuracy was with 0.9693022866236704 with k= 3
```

Figure 131 - KNN results and best value for k

7.3 Decision Tree

Decision Trees are used for classification and regression undertakings. They handle decisions and their possible consequences as a tree-like structure of nodes where each node denotes a test on an attribute. Each branch represents an outcome of the test and each node holds a class label or a continuous value.

7.3.1 needed packages

Before creating or using a python program, is important to consider the needed libraries for the methods required for the script. Each of these libraries should be imported with ***import*** *library_name*, to create an alias for these libraries they can be imported as ***import*** *library_name* ***as*** *alias_name*. The libraries for decision tree algorithmic operations are the already known numpy and pandas, but also, the De*cisionTreeClassifier* module from the *sklearn.tree*.

```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
```

Figure 132 – Decision tree needed packages

7.3.2 Data visualization and analysis

Before proceeding with the dataset learning methods, its data should be analysed and, if necessary, manipulated. With ***pd.read_csv("****file_path/filename****")*** the dataset data can be retrieved into a variable, in this case ***df***. Then, this dataset can be displayed with ***df.head()***, the *head* method will, by default, display the first five lines of the dataset, to alter this configuration a number can be inserted inside the *head()* method (*e.g. head(7)* will show the first 7 lines of the dataset). To display the sample size (*e.g.* total size/column size/row size) the ***shape*** and ***size*** methods can be applied to the dataframe.

```
df = pd.read_csv("dataset_hotel_bookings.csv", delimiter=",")
df.head()
```

```
C:\Users\gabri\AppData\Local\Temp\ipykernel_24348\2718845732.
ption on import or set low_memory=False.
  df = pd.read_csv("dataset_hotel_bookings.csv", delimiter=",
```

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arriv
0	1	0	342	2015	7	
1	1	0	737	2015	7	
2	1	0	7	2015	7	
3	1	0	13	2015	7	
4	1	0	14	2015	7	

5 rows × 35 columns

```
rows, columns = df.shape
size = df.size
print("Number of rows:", rows)
print("Number of columns:", columns)
print("Total number of elements:", total_elements)
```

```
Number of rows: 119390
Number of columns: 35
Total number of elements: 4178650
```

Figure 133 – Decision tree data visualization

7.3.3 Decision tree set up

After verifying that the data to process is numerical, a target variable should be selected to predict using the decision tree algorithm.

```
X = df[['hotel', 'lead_time', 'arrival_date_year', 'arrival_date_month']].values
X[0:5]
```

```
array([[   1,  342, 2015,    7],
       [   1,  737, 2015,    7],
       [   1,    7, 2015,    7],
       [   1,   13, 2015,    7],
       [   1,   14, 2015,    7]], dtype=int64)
```

```
#Target variable
y = df["is_canceled"]
y[0:5]
```

```
0    0
1    0
2    0
3    0
4    0
Name: is_canceled, dtype: int64
```

Figure 134 - Pre-processing

After processing the data, the decision tree train/test split can be set up. To start, the ***train_test_split*** module from ***sklearn.cross_validation*** should be imported. Then, because, *train_test_split* will return 4 different parameters four variables will be created, *X_trainset, X_testset, y_trainset* and *y_testset*. The ***train_test_split*** method will require four parameters, two arrays with data required for the split, one value for the test size and another for the random state.

```
from sklearn.model_selection import train_test_split
X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_size=0.3, random_state=3)
```

Figure 135 – Decision tree set up

To print the shapes of both train and test sets, for each array, the ***shape*** method is used. However, to ensure that the dimensions of each set of arrays match, an if condition must be issued.

```
# Printing the shape of X_trainset and y_trainset
print("Shape of X_trainset:", X_trainset.shape)
print("Shape of y_trainset:", y_trainset.shape)

# Ensuring that the number of rows in X_trainset matches the length of y_trainset
if X_trainset.shape[0] == y_trainset.shape[0]:
    print("The dimensions of X_trainset and y_trainset match.")
else:
    print("The dimensions of X_trainset and y_trainset do not match.")
```

Figure 136 - Decision tree train set shape

```
# Printing the shape of X_testset and y_testset
print("Shape of X_testset:", X_testset.shape)
print("Shape of y_testset:", y_testset.shape)

# Ensuring that the number of rows in X_testset matches the length of y_testset
if X_testset.shape[0] == y_testset.shape[0]:
    print("The dimensions of X_testset and y_testset match.")
else:
    print("The dimensions of X_testset and y_testset do not match.")
```

Figure 137 - Decision tree test set shape

7.3.4 Modelling and prediction

An instance of the DecisionTreeClassifier can be created to verify the information gain of each node. Inside the classifier, the *criterion* should be specified to *entropy*.

```
drugTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
drugTree # it shows the default parameters
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

```
drugTree.fit(X_trainset,y_trainset)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

Figure 138 - Decision tree Modelling

To make predictions on the testing dataset and store them in a variable, the ***predict*** method is applied, then presented with a simple ***print*** method.

```
predTree = drugTree.predict(X_testset)

print (predTree [0:5])
print (y_testset [0:5])
```

```
[0 0 0 0 0]
19775    0
3094     1
114767   0
45361    0
78057    0
Name: is_canceled, dtype: int64
```

Figure 139 - Decision tree prediction

7.3.5 Evaluation and visualization

To proceed with the algorithm evaluation, the *metrics* module must be imported from the *sklearn* library. Then, with ***metrics.accuracy_score*** the accuracy of the decision tree is calculated.

```
from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_testset, predTree))
```

Figure 140 - Decision tree evaluation

The accuracy score can also be calculated without the *sklearn* library. The accuracy score is defined as the ratio of the number of correct predictions to the total number of predictions. In other words, it's the proportion of true results (both true positives and true negatives) among the total number of cases examined. This value can be calculated by comparing each element of the prediction array (predTree) with the corresponding element in the actual labels (y_testset), then counting the number of times the predicted label matches the actual label to, finally, divide this count by the total number of predictions to get the accuracy.

```
#Compare each element of the prediction array with the corresponding element in the actual labels
correct_predictions = sum(predTree == y_testset)
#Count the number of times the predicted label matches the actual label
total_predictions = len(y_testset)
#Divide this count by the total number of predictions to get the accuracy
accuracy = correct_predictions / total_predictions

print("DecisionTrees's Accuracy:", accuracy)
```

Figure 141 - Accuracy calculation without *sklearn*

To visualize the decision tree, for libraries/modules need to be imported, *StringIO* from *sklearn.externals.six* (used for in-memory text storage), *pydotplus* (used for visualizing decision trees), *matplotlib.image* (used for image processing) and *tree* from *sklearn* (used for decision tree models).

After importing the required libraries, the following processes are completed:

- ***dot_data = StringIO()*** – an in-memory buffer is initialized to store the decision tree's representation
- ***filename = "drugtree.png"*** – Sets the name of the file where the tree image will be saved.
- ***featureNames = my_data.columns[0:5]*** – Retrieves the first five column names from the dataset my_data to use as feature names in the tree.
- ***class_names = [str(name) for name in np.unique(y_trainset)]*** – Specifies the class names for the target variable in the decision tree model.
- ***targetNames = my_data["Drug"].unique().tolist()*** – Extracts unique values from the "Drug" column in my_data, for use as target class names in the tree.
- ***out=tree.export_graphviz(...)*** – Exports the decision tree model (drugTree) to a Graphviz format, stored in dot_data.
- ***graph = pydotplus.graph_from_dot_data(dot_data.getvalue())*** – Converts the Graphviz data from dot_data into a graph using pydotplus.
- ***graph.write_png(filename)*** – Saves the graph as a PNG image file with the specified filename.
- ***img = mpimg.imread(filename)***: Reads the saved PNG file into an array format suitable for plotting.
- ***plt.figure(figsize=(100, 200))***: Sets up the figure size for the plot.
- ***plt.imshow(img,interpolation='nearest')*** – Displays the image of the decision tree using Matplotlib. interpolation='nearest' is used to display the image without trying to interpolate between pixels if the display resolution is different from the image resolution.

```
from io import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
%matplotlib inline
```

```
dot_data = StringIO()
filename = "drugtree.png"
featureNames = df.columns[0:4].astype(str)
class_names = [str(name) for name in np.unique(y_trainset)]
targetNames = df["customer_type"].unique().tolist()
out = tree.export_graphviz(drugTree, feature_names=featureNames, out_file=dot_data, class_names=class_names)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png(filename)
img = mpimg.imread(filename)
plt.figure(figsize=(100, 200))
plt.imshow(img,interpolation='nearest')
```

```
<matplotlib.image.AxesImage at 0x1f6844814c0>
```

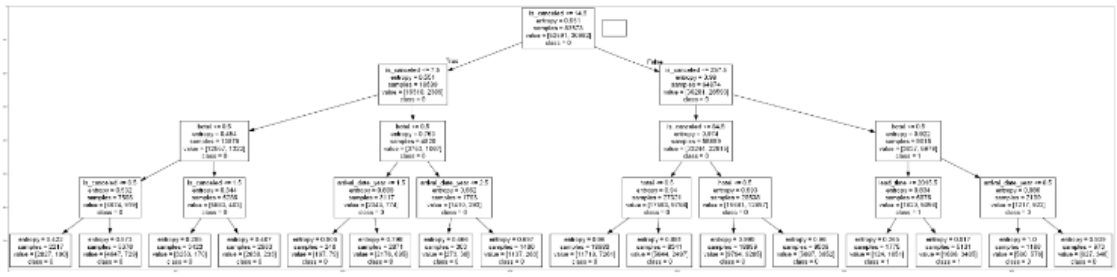


Figure 142 - Decision tree visualization

7.4 Logistic Regression

Logistic Regression is a variation of Linear Regression, useful when the observed dependent variable is categorical. It produces a formula that predicts the probability of the class label as a function of the independent variables. Logistic regression fits a special s-shaped curve by taking the linear regression and transforming the numeric estimate into a probability called sigmoid function

7.4.1 needed packages

Before creating or using a python program, is important to consider the needed libraries for the methods required for the script. Each of these libraries should be imported with ***import*** *library_name*, to create an alias for these libraries they can be imported as ***import*** *library_name* ***as*** *alias_name*. The libraries for the python script respective to Logistic Regression are the following:

- ***pandas*** – Ideal for data manipulation and analysis, especially with tabular data.
- ***pylab*** – Used for plotting and visualizing data.
- ***numpy*** – Essential for numerical computations and handling arrays.
- ***Scipy.optimize*** – Provides algorithms for function optimization, root finding, and curve fitting.
- ***sklearn.preprocessing*** – Provides tools for data preprocessing and feature scaling.
- ***matplotlib.pyplot*** – Used for creating a wide variety of static, animated and interactive visualizations.

```
import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
%matplotlib inline
import matplotlib.pyplot as plt
```

Figure 143 – Logistic regression needed packages

7.4.2 Data visualization and analysis

Before proceeding with the dataset learning methods, its data should be analysed and, if necessary, manipulated. With ***pd.read_csv("****file_path/filename****")*** the dataset data can be retrieved into a variable, in this case ***df***. Then, this dataset can be displayed with ***df.head()***, the *head* method will, by default, display the first five lines of the dataset, to alter this configuration a number can be inserted inside the *head()* method (*e.g. head(7)* will show the first 7 lines of the dataset).

```
df = pd.read_csv("dataset_hotel_bookings.csv")
df.head()
```

```
C:\Users\gabri\AppData\Local\Temp\ipykernel_23252\2769375820.py:1: DtypeWarning: Columns (10) have mixed types. Specify dtype o
ption on import or set low_memory=False.
  df = pd.read_csv("dataset_hotel_bookings.csv")
```

aiting_list	customer_type	adr	required_car_parking_spaces	total_of_special_requests	reservation_status	reservation_status_date	var_r	filter_$	arrival_month
0	Transient	0.0	0	0	Check-Out	2015-07-01	491	0	7
0	Transient	0.0	0	0	Check-Out	2015-07-01	737	0	7
0	Transient	75.0	0	0	Check-Out	2015-07-02	245	0	7
0	Transient	75.0	0	0	Check-Out	2015-07-02	245	0	7
0	Transient	98.0	0	1	Check-Out	2015-07-03	245	0	7

Figure 144 – Logistic regression data visualization

To select some of the features of the dataset for the modelling, the created ***df*** variable can be changed to contained these features with ***df = df[['feature1',...,'featureN']]***. Any value of the feature, if necessary, can be modified to another type with the ***astype(***type***)*** method. To check for the number of rows in the new dataframe and which columns it possesses, the ***df.shape*** and ***df.columns*** methods can be issued to a variable to be presented by a *print* method.

```
df = df[['hotel', 'is_canceled', 'lead_time', 'arrival_date_year', 'arrival_date_month', 'stays_
df['hotel'] = df['hotel'].astype('int')
df.head()
```

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	stays_in_weekend_nights	stays_in_week_nights
0	1	0	342	2015	7	0	0
1	1	0	737	2015	7	0	0
2	1	0	7	2015	7	0	1
3	1	0	13	2015	7	0	1
4	1	0	14	2015	7	0	2

How many rows and columns are in this dataset in total? What are the name of columns?

```
num_cols = df.columns
num_rows = df.shape[0]
print("Number of rows:", num_rows)
print("Columns:\n", num_cols)
```

Figure 145 - Logistic regression data preprocessing

Before splitting the dataset into a training and a testing dataset, is important to define the array of features to use for the learning algorithm. So, with ***X = np.asarray(df['column1',...,'columnN'])]*** a variable is declared to store the features to analyse. The same process must be done for the target feature. Data normalization is also important so the *preprocessing* module from the *sklearn* library is imported to implement the ***preprocessing.StandardScaler().fit(X).transform(X)*** method.

```
X = np.asarray(df[['hotel', 'lead_time', 'arrival_date_year', 'arrival_date_month',
X[0:5]
```

```
array([[   1,  342, 2015,    7,    0,    0],
       [   1,  737, 2015,    7,    0,    0],
       [   1,    7, 2015,    7,    0,    1],
       [   1,   13, 2015,    7,    0,    1],
       [   1,   14, 2015,    7,    0,    2]], dtype=int64)
```

```
y = np.asarray(df['is_canceled'])
y [0:5]
```

```
array([0, 0, 0, 0, 0], dtype=int64)
```

Also, we normalize the dataset:

```
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

```
array([[ 1.40722407,  2.22705112, -1.63476794,  0.14479897, -0.92889042,
        -1.31023993],
       [ 1.40722407,  5.9233847 , -1.63476794,  0.14479897, -0.92889042,
        -1.31023993],
       [ 1.40722407, -0.90781407, -1.63476794,  0.14479897, -0.92889042,
         -0.78620716],
       [ 1.40722407, -0.85166723, -1.63476794,  0.14479897, -0.92889042,
         -0.78620716],
       [ 1.40722407, -0.84230942, -1.63476794,  0.14479897, -0.92889042,
         -0.2621744 ]])
```

Figure 146 - Data arrays creation and normalization

7.4.3 Train and test split and modelling

Now that the arrays are created and the data normalized, the sample can be divided into a train and test with a 80-20 split. With the *LogisticRegression* and *confusion_matrix* modules from *sklearn.linear_model* and *sklearn.metrics*, respectively, the *LogisticRegression* method can be applied along with *fit* method to fit the model with the training set. Then, the prediction is done, using the test set, with the *predict* method. The *predict_proba* method returns estimates for all classes, ordered by the label of classes.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (95512, 6) (95512,)
Test set: (23878, 6) (23878,)
```

Figure 147 - Logistic regression Train/Test split

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
LR
```

```
LogisticRegression(C=0.01, solver='liblinear')
```

Now we can predict using our test set:

```
yhat = LR.predict(X_test)
yhat
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

predict_proba returns estimates for all classes, ordered by the label of classes. So, the firs
is probability of class 0, P(Y=0|X):

```
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

```
array([[0.67059728, 0.32940272],
       [0.5091822 , 0.4908178 ],
       [0.81335789, 0.18664211],
       ...,
       [0.66177768, 0.33822232],
       [0.70856478, 0.29143522],
       [0.62234178, 0.37765822]])
```

Figure 148 - Logistic regression modelling

7.4.4 Evaluation with the Jaccard index

The Jaccard Index, also known as the Jaccard similarity coefficient, is a statistic used for gauging the similarity and diversity of sample sets. Is defined as the size of the intersection divided by the size of the union of two sets. In the context of classification problems, it's used to compare the set of predicted labels with the set of true labels. That is, the result between Number of true positives divided by the sum of the false positives and false negatives.

To apply this coefficient, the *jaccard_similarity_score* module must be imported from the *sklearn.metrics* library so the *jaccard_similarity_score* method can be used. If the entire set of predicted labels for a sample strictly match with the true set of labels, then the subset accuracy is 1.0; otherwise it is 0.0.

```
from sklearn.metrics import jaccard_score
jaccard_score(y_test, yhat)
```

```
0.24796208530805688
```

Figure 149 - Jaccard index for logistic regression evaluation

7.4.5 Evaluation with a confusion matrix

Another way to evaluate the accuracy of a classifier is through a confusion matrix. For this, the *classification_report, confusion_matrix* modules must be imported from the *sklearn.metrics* library along with the *itertools* library. This way, a method can be created to use several tools provided by these packages.

The *plot_confusion_matrix* function:

1. **def plot_confusion_matrix(…) – definition:**
    a. *Cm* – The confusion matrix to be plotted.
    b. *Classes* – An array of class names.
    c. *normalize*: If set to True, the confusion matrix will be normalized.
    d. *Title* - Title of the plot.
    e. *Cmap* – Colormap used for the plot, defaulting to a blue color map.
2. **if normalize…else – normalization:**
    a. If normalize is True, each row of the confusion matrix is divided by the sum of that row (*e.g.* by the total number of instances in the true class), to show proportions instead of raw counts.
    b. The function prints the confusion matrix (either normalized or raw).
3. **Plotting:**
    a. The confusion matrix is displayed as an image where each grid cell's color intensity represents the number (or proportion, if normalized) of instances.
    b. ***plt.imshow()*** – Displays the confusion matrix as an image.
    c. ***plt.title(), plt.colorbar()*** – Adds a title and a color bar to the plot, respectively.
    d. ***plt.xticks(), plt.yticks()*** – The x-axis and y-axis are labelled with class names, and tick marks are set for each class.
4. **Loop to add text in Each Cell:**
    a. ***For i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1]))*** – Goes through each cell in the matrix to add text that shows the count (or proportion) in that cell.
    b. ***plt.text()*** **–** Adds the text to each cell. The color of the text is set to white if the background (the cell's value) is above a threshold (for better readability) and black otherwise.
5. **Final Plot Adjustments:**
    a. ***plt.tight_layout()*** – Adjusts subplot params for a nice fit.
    b. ***plt.xlabel, plt.ylabel*** – The labels for the axes are set to 'True label' and 'Predicted label', respectively.

Confusion matrix computation:

- Now that the previous function is defined, it can be called on the program to compute the confusion matrix for the evaluation of the algorithm accuracy. With the ***confusion_matrix(y_test, yhat, labels=[1,0])*** method the matrix is computed by comparing the *y_test* and *yhat* labels. Then, the floating-point numbers with a precision of 2 decimal places are displayed by the NumPy's ***np.set_printoptions*** method.

Confusion matrix plotting:

- After this process, the confusion matrix can be plotted with the *figure()* method and the previously created function ***plot_confusion_matrix***.

```
from sklearn.metrics import classification_report, confusion_matrix
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)


# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['is_canceled=1','is_canceled=0'],normalize= False,  title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[ 2616  6253]
 [ 1681 13328]]
```

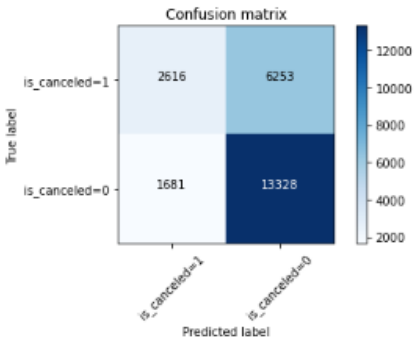


Figure 150 - Logistic regression evaluation with a confusion matrix

7.4.6 Evaluation with precision, recall and F1-score metrics

Based on the count of each section, the precision and recall can be calculated for each label. Precision is a measure of the accuracy provided that a class label has been predicted. It is defined by the division between the true positives and the sum of the true and false positives (TP / (TP + FP)). Recall, is the true positive rate and is defined by the division between the true positives and the sum of true positives and false negatives (TP / (TP + FN))

Then, is possible to calculate the F1 scores for each label based on the precision and recall of that label. The F1score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. It is a good way to show that a classifier has a good value

for both recall and precision. This way is possible to tell that the average accuracy for this classifier is the average of the f1-score for both labels.

These metrics are calculated with the ***classification_report*** method and displayed by a *print* function.

```
print (classification_report(y_test, yhat))
```

```
              precision    recall  f1-score   support

           0       0.68      0.89      0.77     15009
           1       0.61      0.29      0.40      8869

    accuracy                           0.67     23878
   macro avg       0.64      0.59      0.58     23878
weighted avg       0.65      0.67      0.63     23878
```

Figure 151 - Logistic regression evaluation metrics

7.4.7 Evaluation with Log loss

Log Loss, also known as logistic loss or cross-entropy loss, is a performance metric for evaluating the predictions of probabilities in classification tasks, especially in binary classification like logistic regression. It measures the uncertainty of the model's predictions based on how much they deviate from the actual labels.

To use this metric the *log_loss* module must be imported from the *sklearn.metrics* library, to be used as a method in ***log_loss(y_test, yhat_prob)***. Then, the logistic regression model is trained to make probability predictions on a test set, to evaluate the model's performance. ***LR2 = LogisticRegression(C=0.01, solver='sag').fit(X_train,y_train)*** creates and trains the model, ***yhat_prob2 = LR2.predict_proba(X_test)*** uses the trained model to make the predictions and ***print("LogLoss: : %.2f" % log_loss(y_test, yhat_prob2))*** prints the Log loss metric calculated.

```
from sklearn.metrics import log_loss
log_loss(y_test, yhat_prob)
```

```
0.6090828579787433
```

```
LR2 = LogisticRegression(C=0.01, solver='sag').fit(X_train,y_train)
yhat_prob2 = LR2.predict_proba(X_test)
print ("LogLoss: : %.2f" % log_loss(y_test, yhat_prob2))
```

Figure 152 - Logistic regression evaluation with log loss

7.5 SVM

7.5.1 needed packages

Before creating or using a python program, is important to consider the needed libraries for the methods required for the script. Each of these libraries should be imported with ***import*** *library_name*, to create an alias for these libraries they can be imported as ***import*** *library_name* ***as*** *alias_name*. The libraries for the python script respective to Logistic Regression are the following:

- ***pandas*** – Ideal for data manipulation and analysis, especially with tabular data.
- ***pylab*** – Used for plotting and visualizing data.
- ***numpy*** – Essential for numerical computations and handling arrays.

- ***Scipy.optimize*** – Provides algorithms for function optimization, root finding, and curve fitting.
- ***sklearn.preprocessing*** – Provides tools for data preprocessing and feature scaling.
- ***matplotlib.pyplot*** – Used for creating a wide variety of static, animated and interactive visualizations.

```
import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
%matplotlib inline
import matplotlib.pyplot as plt
```

Figure 153 - SVM needed packages

7.5.2 Data visualization and analysis

Before proceeding with the dataset learning methods, its data should be analysed and, if necessary, manipulated. With ***pd.read_csv("file_path/filename")*** the dataset data can be retrieved into a variable, in this case ***df***. Then, this dataset can be displayed with ***df.head()***, the *head* method will, by default, display the first five lines of the dataset, to alter this configuration a number can be inserted inside the *head()* method (*e.g. head(7)* will show the first 7 lines of the dataset).

```
df = pd.read_csv('dataset_hotel_bookings.csv')
df.head()
```

```
C:\Users\gabri\AppData\Local\Temp\ipykernel_21740\55142793
tion on import or set low_memory=False.
  df = pd.read_csv('dataset_hotel_bookings.csv')
```

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month
0	1	0	342	2015	7
1	1	0	737	2015	7
2	1	0	7	2015	7
3	1	0	13	2015	7
4	1	0	14	2015	7

5 rows × 35 columns

Figure 154 - SVM data visualization

Another way to analyse the data in the sample is to set an overall view of a certain target variable depending on other two. With the *plot* library, a scatter plot can be created with this information.

```
ax = df[df['is_canceled'] == 1][0:119000].plot(kind='scatter', x='days_in_waiting_list', y='total_of_special_requests',
                                               color='DarkBlue', label='Cancelled');
df[df['is_canceled'] == 0][0:119000].plot(kind='scatter', x='days_in_waiting_list', y='total_of_special_requests', color='red',
                                          label='Not cancelled', ax=ax);
plt.show()
```

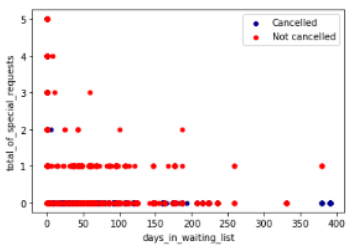


Figure 155 - SVM dataset Scatter plot

7.5.3 Train and test split

To create a train and test split, the process is equivalent to the already known by the previous topics. Two arrays are created, one for the independent features, another for the target variable. Then, a train and test split is set for each of the created arrays.

```
X = np.asarray(df[['hotel', 'lead_time', 'arrival_date_year', 'arrival_date_month', 'stays_in_weekend_nights',
y = np.asarray(df['is_canceled'])
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (95512, 6) (95512,)
Test set: (23878, 6) (23878,)
```

7.5.4 Modelling

The SVM algorithm offers a choice of kernel functions for performing its processing. Basically, mapping data into a higher dimensional space is called kernelling. The mathematical function used for the transformation is known as the kernel function, and can be of different types, such as, linear, polynomial, radial basis function or sigmoid. Each of these functions has its characteristics, pros and cons, and its equation, but as there's no way of knowing which function performs best with any given dataset. Given this, the default is going to be adopted by using the RBF (Radial Basis Function).

First, the *svm* module must be imported from the *sklearn* library to apply the *svm.SVC* method. After being fitted by the *fit* method, the model can be used to predict new values.

```
from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
yhat = clf.predict(X_test)
yhat [0:5]
```

Figure 156 - SVM modelling

7.5.5 Evaluation with the Jaccard index

The Jaccard Index, also known as the Jaccard similarity coefficient, is a statistic used for gauging the similarity and diversity of sample sets. Is defined as the size of the intersection divided by the size of the union of two sets. In the context of classification problems, it's used to compare the set of predicted labels with the set of true labels. That is, the result between Number of true positives divided by the sum of the false positives and false negatives.

To apply this coefficient, the *jaccard_similarity_score* module must be imported from the *sklearn.metrics* library so the *jaccard_similarity_score* method can be used. If the entire set of predicted labels for a sample strictly match with the true set of labels, then the subset accuracy is 1.0; otherwise it is 0.0.

```
from sklearn.metrics import jaccard_score
jaccard_score(y_test, yhat)
```

```
0.24796208530805688
```

Figure 157 - Jaccard index for SVM evaluation

7.5.6 Evaluation with a confusion matrix

After the modelling is completed, there can be done an algorithm evaluation. With the *sklearn.metrics* library, the *classification_report, confusion_matrix* are imported for further use, as well as the *itertools* library. Them a confusion matrix is created with a function

The *plot_confusion_matrix* function:

1. **def plot_confusion_matrix(…) – definition:**
    a. *Cm* – The confusion matrix to be plotted.
    b. *Classes* – An array of class names.
    c. *normalize*: If set to True, the confusion matrix will be normalized.
    d. *Title* - Title of the plot.
    e. *Cmap* – Colormap used for the plot, defaulting to a blue color map.
2. **if normalize…else – normalization:**
    a. If normalize is True, each row of the confusion matrix is divided by the sum of that row (*e.g.* by the total number of instances in the true class), to show proportions instead of raw counts.
    b. The function prints the confusion matrix (either normalized or raw).
3. **Plotting:**
    a. The confusion matrix is displayed as an image where each grid cell's color intensity represents the number (or proportion, if normalized) of instances.
    b. ***plt.imshow()*** – Displays the confusion matrix as an image.
    c. ***plt.title(), plt.colorbar()*** – Adds a title and a color bar to the plot, respectively.
    d. ***plt.xticks(), plt.yticks()*** – The x-axis and y-axis are labelled with class names, and tick marks are set for each class.
4. **Loop to add text in Each Cell:**
    a. ***For i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1]))*** – Goes through each cell in the matrix to add text that shows the count (or proportion) in that cell.
    b. ***plt.text()*** **–** Adds the text to each cell. The color of the text is set to white if the background (the cell's value) is above a threshold (for better readability) and black otherwise.
5. **Final Plot Adjustments:**
    a. ***plt.tight_layout()*** – Adjusts subplot params for a nice fit.
    b. ***plt.xlabel, plt.ylabel*** – The labels for the axes are set to 'True label' and 'Predicted label', respectively.

Confusion matrix computation:

- Now that the previous function is defined, it can be called on the program to compute the confusion matrix for the evaluation of the algorithm accuracy. With the ***confusion_matrix(y_test, yhat, labels=[1,0])*** method the matrix is computed by comparing the *y_test* and *yhat* labels. Then, the floating-point numbers with a precision of 2 decimal places are displayed by the NumPy's ***np.set_printoptions*** method.

Confusion matrix plotting:

- After this process, the confusion matrix can be plotted with the *figure()* method and the previously created function ***plot_confusion_matrix***.

```
from sklearn.metrics import classification_report, confusion_matrix
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)


# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['is_canceled=1','is_canceled=0'],normalize= False,  title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[ 2616  6253]
 [ 1681 13328]]
```

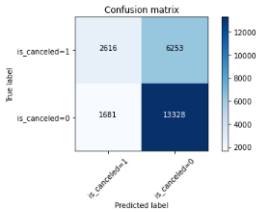


Figure 158 - SVM evaluation with a confusion matrix

7.5.7 Evaluation with precision, recall and F1-score metrics

Based on the count of each section, the precision and recall can be calculated for each label. Precision is a measure of the accuracy provided that a class label has been predicted. It is defined by the division between the true positives and the sum of the true and false positives (TP / (TP + FP)). Recall, is the true positive rate and is defined by the division between the true positives and the sum of true positives and false negatives (TP / (TP + FN))

Then, is possible to calculate the F1 scores for each label based on the precision and recall of that label. The F1score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. It is a good way to show that a classifier has a good value for both recall and precision. This way is possible to tell that the average accuracy for this classifier is the average of the f1-score for both labels.

These metrics are calculated with the ***classification_report*** method and displayed by a *print* function.

```
print (classification_report(y_test, yhat))
```

```
              precision    recall  f1-score   support

           0       0.68      0.89      0.77     15009
           1       0.61      0.29      0.40      8869

    accuracy                           0.67     23878
   macro avg       0.64      0.59      0.58     23878
weighted avg       0.65      0.67      0.63     23878
```

Figure 159 - SVM evaluation metrics

8 Python – Unsupervised learning

Python is a high level programming language, commonly used for data science due to its simplicity, flexibility and high number of libraries. These libraries include the *scikit-learn*, *TensorFlow* and *PyTorch* that offer extensive functionalities for building and training supervised unsupervised learning models.

8.1 K-Means

Despite its simplicity, the K-means is vastly used for clustering in many data science applications, especially useful if you need to quickly discover insights from unlabelled data. This algorithm can be applied in real-world applications like Customer segmentation, understanding what the visitors of a website are trying to accomplish, pattern recognition, machine learning or data compression.

8.1.1 Needed packages

Before creating or using a python program, is important to consider the needed libraries for the methods required for the script. Each of these libraries should be imported with ***import*** *library_name*, to create an alias for these libraries they can be imported as ***import*** *library_name* ***as*** *alias_name*. The libraries for the python script respective to Logistic Regression are the following:

- ***random*** – Includes functions for generating random numbers and performing random operations.
- ***numpy*** – Essential for numerical computations and handling arrays.
- ***matplotlib.pyplot*** – Used for creating a wide variety of static, animated and interactive visualizations.
- ***KMeans*** from ***sklearn.cluster*** – Used for partitioning data into clusters based on attributes.
- ***pandas*** – Ideal for data manipulation and analysis, especially with tabular data.

```
import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import pandas as pd
%matplotlib inline
```

Figure 160 – K-Means needed packages

8.1.2 Data visualization and pre-processing

After importing the needed packages, is important to verify the data to use on the algorithm. This way, the dataset is uploaded with the *read_csv* method and set in an array with *np.asarray*. Is also important to normalize the data since this method helps with mathematical-based algorithms to interpret features with different magnitudes and distributions equally. For this, the *standardScaler()* method can be used to normalize the dataset.

```
df = pd.read_csv("dataset_hotel_bookings.csv")
df.head()
```

```
C:\Users\gabri\AppData\Local\Temp\ipykernel_21844\2769375820.py:1: DtypeWarning: Columns (10) have mixed types. Specify dtype o
ption on import or set low_memory=False.
  df = pd.read_csv("dataset_hotel_bookings.csv")
```

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in
0	1	0	342	2015	7	27	1	0	
1	1	0	737	2015	7	27	1	0	
2	1	0	7	2015	7	27	1	0	
3	1	0	13	2015	7	27	1	0	
4	1	0	14	2015	7	27	1	0	

5 rows × 35 columns

Pre-processing

```
X = np.asarray(df[['hotel', 'lead_time', 'arrival_date_year', 'arrival_date_month', 'stays_in_weekend_nights', 'stays_in_week_ni
df.head()
```

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in
0	1	0	342	2015	7	27	1	0	
1	1	0	737	2015	7	27	1	0	
2	1	0	7	2015	7	27	1	0	
3	1	0	13	2015	7	27	1	0	
4	1	0	14	2015	7	27	1	0	

5 rows × 35 columns

Figure 161 – K-Means data visualization and pre-processing

```
from sklearn.preprocessing import StandardScaler
X = np.nan_to_num(X)
Clus_dataSet = StandardScaler().fit_transform(X)
Clus_dataSet
```

```
array([[ 1.40722407,  2.22705112, -1.63476794,  0.14479897, -0.92889042,
        -1.31023993],
       [ 1.40722407,  5.9233847 , -1.63476794,  0.14479897, -0.92889042,
        -1.31023993],
       [ 1.40722407, -0.90781407, -1.63476794,  0.14479897, -0.92889042,
        -0.78620716],
       ...,
       [-0.71061889, -0.65515329,  1.19219514,  0.46836014,  1.07389483,
         1.3099239 ],
       [-0.71061889,  0.0466822 ,  1.19219514,  0.46836014,  1.07389483,
         1.3099239 ],
       [-0.71061889,  0.94503163,  1.19219514,  0.46836014,  1.07389483,
         2.35798943]])
```

Figure 162 - K-Means data normalization

8.1.3 Modelling and insights

To model a K-Means algorithm, the number of clusters must be set first. Then, by calling the *KMeans* method this algorithm is initialized with a method set by the first parameter, the number of clusters set by the second parameter and the number of times that the algorithm will be run by the third parameter. Then, the model is fitted with the *fit* method, the labels are extracted with *k_means.labels_* and printed next.

```
clusterNum = 3
k_means = KMeans(init = "k-means++", n_clusters = clusterNum, n_init = 12)
k_means.fit(X)
labels = k_means.labels_
print(labels)
```

```
[2 2 1 ... 1 0 0]
```

Figure 163 - K-Means modelling

Now, the labels can be assigned to each row of the dataframe with ***df["Clus_km"] = labels***. Then, the centroid values can be checked by averaging the features in each cluster with ***df.groupby('Clus_km').mean()***.

```
df["Clus_km"] = labels
df.head(5)
```

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_
0	1	0	342	2015	7	
1	1	0	737	2015	7	
2	1	0	7	2015	7	
3	1	0	13	2015	7	
4	1	0	14	2015	7	

5 rows × 36 columns

We can easily check the centroid values by averaging the features in each cluster.

```
df.groupby('Clus_km').mean()
```

th	stays_in_weekend_nights	stays_in_week_nights	adults	...	booking_changes	
49	1.150491	3.034895	1.920898	...	0.249621	8
36	0.816191	2.168326	1.803362	...	0.211931	9
12	0.907506	2.745428	1.944808	...	0.195682	6

Figure 164 - K-Means insights

The distribution between the average days in the wating list and the if a reservation is cancelled can be calculated with ***area = np.pi * (X[:, 1])**2*** and plotted with ***plt.scatter(X[:, 0], X[:, 3], s=area, c=labels.astype(float), alpha=0.5)***.

```
area = np.pi * ( X[:, 1])**2
plt.scatter(X[:, 0], X[:, 3], s=area, c=labels.astype(float), alpha=0.5)
plt.xlabel('is_canceled', fontsize=18)
plt.ylabel('days_in_waiting_list', fontsize=16)

plt.show()
```

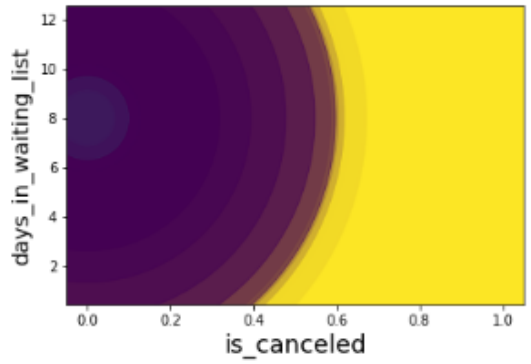


Figure 165 - K-Means distribution

Another way to represent the data distribution is with a 3D representation. For this the *Axes3D* model must be imported form *mpl_toolkits.mplot3d* library to create a figure with *plt.figure* and an axis with *Axes3D*. Then it can be plotted with *plt* and displayed with *ax.scatter*.

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(1, figsize=(8, 6))
plt.clf()
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)

plt.cla()
ax.set_xlabel('time between entrance and arrival')
ax.set_ylabel('Is cancelled')
ax.set_zlabel('Days in the waiting list')

ax.scatter(X[:, 1], X[:, 0], X[:, 3], c= labels.astype(np.float))
```

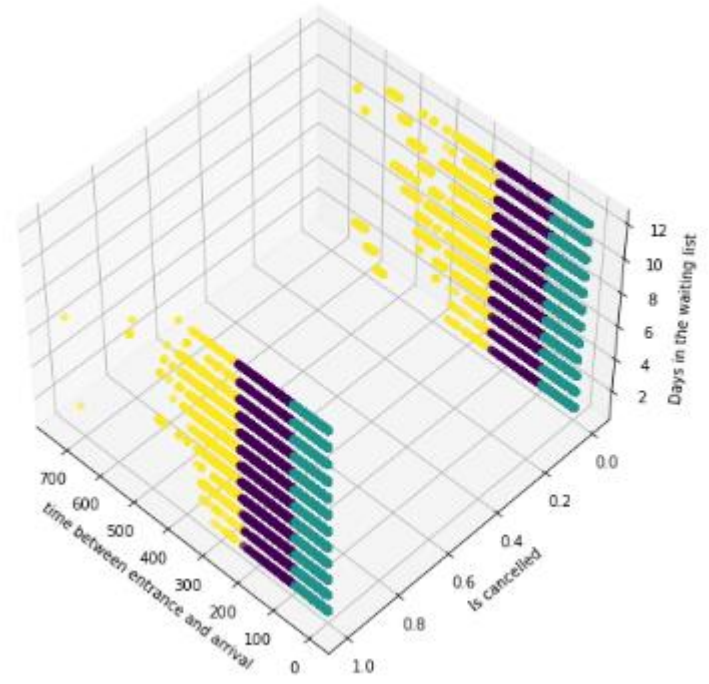


Figure 166 - K-Means 3D representation of distribution

8.2 Hierarchical Clustering

8.2.1 Needed packages

Before creating or using a python program, is important to consider the needed libraries for the methods required for the script. Each of these libraries should be imported with ***import*** *library_name*, to create an alias for these libraries they can be imported as ***import*** *library_name* ***as*** *alias_name*. The libraries for the python script respective to Logistic Regression are the following:

- ***pandas*** – Ideal for data manipulation and analysis, especially with tabular data.
- ***numpy*** – Essential for numerical computations and handling arrays.
- ***ndimage from Scipy*** – Provides a collection of functions for multi-dimensional image processing.
- ***hierarchy*** from ***scipy.cluster*** – Used for hierarchical clustering algorithms and functions to visualize cluster hierarchies.
- ***distance_matrix*** from ***scipy.spatial*** – Used to compute the distance matrix between each pair of points.
- ***matplotlib.pyplot*** – Used for creating a wide variety of static, animated and interactive visualizations.

- ***manifold*** from ***sklearn*** – Used for manifold learning techniques, useful in dimensionality reduction.
- ***AgglomerativeClustering*** from ***sklearn.cluster*** – Used to group objects based on their similarity.

```
import numpy as np
import pandas as pd
from scipy import ndimage
from scipy.cluster import hierarchy
from scipy.spatial import distance_matrix
import matplotlib.pyplot as plt
from sklearn import manifold
from sklearn.cluster import AgglomerativeClustering
%matplotlib inline
```

Figure 167 - Hierarchical Clustering needed packages

8.2.2 Feature selection and normalization

Before applying the clustering algorithm, is important to select the data to analyse and to normalize it. By storing the data in a array with the features to use, it can be normalized using the *MinMaxScaler* method from the *sklearn.preprocessing* library.

```
X = df[['lead_time',  'arrival_date_month', 'adults', 'days_in_waiting_list',
from sklearn.preprocessing import MinMaxScaler
x = X.values #returns a numpy array
min_max_scaler = MinMaxScaler()
feature_mtx = min_max_scaler.fit_transform(x)
feature_mtx [0:5]
```

```
array([[0.46404342, 0.54545455, 0.03636364, 0.        , 0.        ,
        0.        ],
       [1.        , 0.54545455, 0.03636364, 0.        , 0.        ,
        0.        ],
       [0.00949796, 0.54545455, 0.01818182, 0.        , 0.        ,
        0.        ],
       [0.01763908, 0.54545455, 0.01818182, 0.        , 0.        ,
        0.        ],
       [0.01899593, 0.54545455, 0.03636364, 0.        , 0.        ,
        0.2       ]])
```

Figure 168 - Hierarchical Clustering feature selection and normalization

8.2.3 Clustering with Scipy

There are many ways to achieve clustering of a dataset, one of them is through the *scipy* library. First, because the dataset used detains a large number of samples, is recommended to reduce this number since the memory allocated can be non-sufficient and lead to problems or even the program not compiling. So, a sample size must set and the *np.random.choice* method is applied to randomize the data used.

Them the distance matrix is computed in the smaller sample. First the number of rows in the smaller matrix is set with the *shape* method. Then, a loop is used to iterate over each pair of samples in the smaller matrix. Inside this loop, is created another loop to iterate over each column. Then with the *scipy.spatial.distance.euclidean* method, the Euclidean distance between the i-th and j-th samples in the matrix is calculated. To conclude this code section, outside of the loop, the hierarchical/agglomerative clustering is performed on the data sample in *complete* mode that represents the linkage method which computes the maximum distance between clusters before merging them.

```
import scipy
import pylab
import scipy.cluster.hierarchy
from scipy.spatial.distance import pdist
from scipy.cluster.hierarchy import linkage
# Random sampling
sample_size = 5000  # Adjust as needed
indices = np.random.choice(feature_mtx.shape[0], sample_size, replace=False)
sampled_feature_mtx = feature_mtx[indices]

# Compute the distance matrix on the smaller dataset
leng = smaller_feature_mtx.shape[0]
D = np.zeros([leng, leng])
for i in range(leng):
    for j in range(leng):
        D[i, j] = scipy.spatial.distance.euclidean(smaller_feature_mtx[i], smaller_feature_mtx[j])

condensed_dist_matrix = pdist(sampled_feature_mtx, metric='euclidean')
Z = hierarchy.linkage(condensed_dist_matrix, 'complete')
```

Figure 169 – Hierarchical clustering matrix's Euclidean distance

After calculating the Euclidean distance, a hierarchy of clusters can be built with a method of cluster analysis, by using the *fcluster* method from the *scipy.cluster.hierarchy* module. First the maximum distance is set to 3. Then, the clusters are formed with the set distance and a parameter set to *distance* that means that clusters are formed when the distance between objects is less than or equal to the second parameter. The result is held on the *clusters* variable that holds the cluster labels for each data point. The clusters can be determined directly as well.

```
from scipy.cluster.hierarchy import fcluster
max_d = 3
clusters = fcluster(Z, max_d, criterion='distance')
clusters
```

```
array([1, 1, 1, ..., 1, 1, 1], dtype=int32)
```

Also, you can determine the number of clusters directly:

```
from scipy.cluster.hierarchy import fcluster
k = 5
clusters = fcluster(Z, k, criterion='maxclust')
clusters
```

```
array([5, 2, 5, ..., 3, 2, 5], dtype=int32)
```

Figure 170 - Hierarchical clustering cluster determination

The dendrogram is the plotted with the *pylab.figure* method and with a functioned defined as *llf*. This function takes an index and returns a formatted string with the objective of formatting labels for each leaf of the dendrogram. The dendrogram is then created with the *hierarchy.dendrogram* method.

```
fig = pylab.figure(figsize=(18,50))
def llf(id):
    return '[%s %s %s]' % (df['lead_time'][id], df['arrival_date_month'][id], df['days_in_waiting_list'][id] )

dendro = hierarchy.dendrogram(Z,  leaf_label_func=llf, leaf_rotation=0, leaf_font_size =12, orientation = 'right')
```

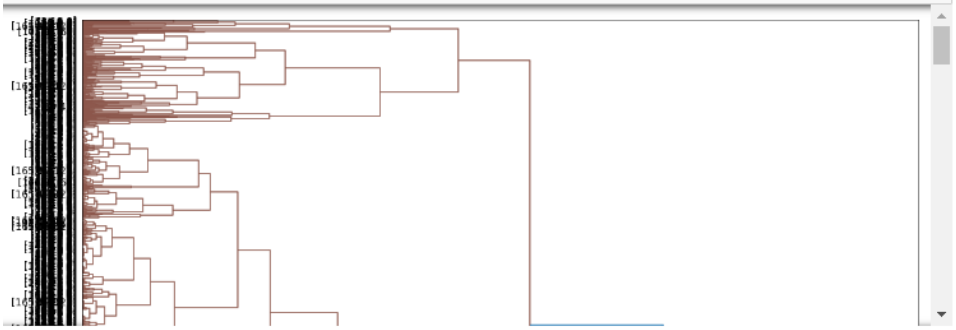


Figure 171 - Hierarchical clustering dendrogram

8.2.4 Clustering with scikit-learn

Another approach to clustering can be through the scikit-learn package. First, with the *distance_matrix* method, a matrix is computed for an array. Then, this matrix is used to perform a hierarchical clustering using a bottom up approach with the AgglomerativeClustering' function from scikit-learn library. After this, a new field is added to the dataframe to show the cluster of each row to then compute and plot the clusters.

```
dist_matrix = distance_matrix(sampled_feature_mtx,sampled_feature_mtx)
print(dist_matrix)
```

```
[[0.         0.48233673 0.20990164 ... 0.77548294 0.62534333 0.19723422]
 [0.48233673 0.         0.51359993 ... 0.46551311 0.25664604 0.56484105]
 [0.20990164 0.51359993 0.         ... 0.67939247 0.63021473 0.24816421]
 ...
 [0.77548294 0.46551311 0.67939247 ... 0.         0.42610566 0.8303487 ]
 [0.62534333 0.25664604 0.63021473 ... 0.42610566 0.         0.64981255]
 [0.19723422 0.56484105 0.24816421 ... 0.8303487  0.64981255 0.        ]]
```

Figure 172 - Hierarchical clustering matrix creation

```
agglom = AgglomerativeClustering(n_clusters = 6, linkage = 'complete')
agglom.fit(sampled_feature_mtx)
agglom.labels_
```

```
array([0, 3, 0, ..., 2, 3, 0], dtype=int64)
```

Figure 173 - Hierarchical clustering forming

```
subset_df = df.sample(n=5000)
subset_df['cluster_'] = agglom.labels_
subset_df.head()
```

	hotel	is_canceled	lead_time	arrival_date_year
87866	0	0	77	2016
95671	0	0	54	2016
92047	0	0	127	2016
531	1	0	64	2015
02140	0	0	215	2016

rows × 36 columns

Figure 174 - Hierarchical clustering added field

```
import matplotlib.cm as cm
n_clusters = max(agglom.labels_)+1
colors = cm.rainbow(np.linspace(0, 1, n_clusters))
cluster_labels = list(range(0, n_clusters))

# Create a figure of size 6 inches by 4 inches.
plt.figure(figsize=(16,14))

for color, label in zip(colors, cluster_labels):
    subset = df[df.is_canceled == label]
    for i in subset.index:
        plt.text(subset.lead_time[i], subset.adults[i], str(subset.days_in_waiting_list[i]), rotation=25)
    plt.scatter(subset.lead_time, subset.adults, s=subset.days_in_waiting_list, color=color, label='cluster'+str(label), alpha=0
plt.legend()
plt.title('lead_time')
plt.xlabel('adults')
plt.ylabel('days_in_waiting_list')
```

```
Text(0, 0.5, 'days_in_waiting_list')
```

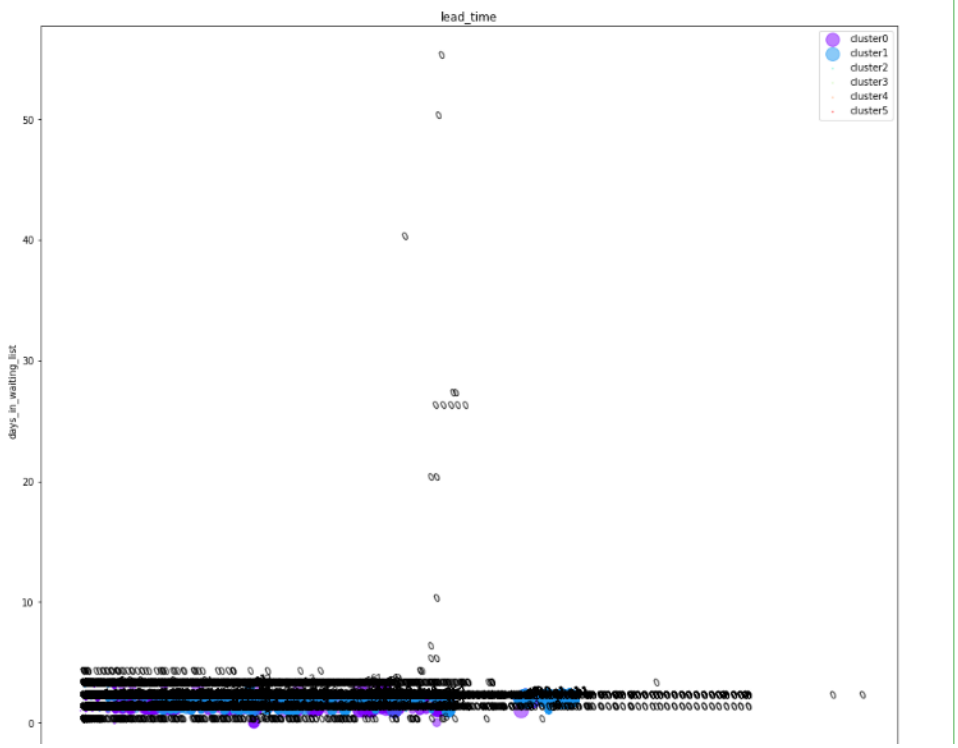


Figure 175 - Hierarchical clustering

8.3 DBSCAN

8.3.1 Needed packages

Before creating or using a python program, is important to consider the needed libraries for the methods required for the script. Each of these libraries should be imported with ***import*** *library_name*, to create an alias for these libraries they can be imported as ***import*** *library_name* ***as*** *alias_name*. The libraries for the python script respective to Logistic Regression are the following:

- ***pandas*** – Ideal for data manipulation and analysis, especially with tabular data.
- ***pylab*** – Used for plotting and visualizing data.
- ***numpy*** – Essential for numerical computations and handling arrays.

```
import csv
import pandas as pd
import numpy as np
```

Figure 176 - DBSCAN needed packages

8.3.2 Data visualization

After importing the needed packages, is important to verify the data to use on the algorithm. This way, the dataset is uploaded with the *read_csv* method and set in an array with *np.asarray*. Is also important to normalize the data since this method helps with mathematical-based algorithms to interpret features with different magnitudes and distributions equally. For this, the *standardScaler()* method can be used to normalize the dataset.

```
filename='dataset_hotel_bookings.csv'

#Read csv
pdf = pd.read_csv(filename)
pdf.head(5)
```

	hotel	is_canceled	lead_time	arrival_date_year
0	1	0	342	2015
1	1	0	737	2015
2	1	0	7	2015
3	1	0	13	2015
4	1	0	14	2015

5 rows × 35 columns

Figure 177 - DBSCAN data visualization

8.3.3 Visualization

```
# Importing necessary libraries
from mpl_toolkits.basemap import Basemap  # Basemap toolkit for rendering geographic maps
import matplotlib.pyplot as plt  # Matplotlib's pyplot for plotting
from pylab import rcParams  # PyLab's rcParams for setting figure properties
%matplotlib inline  # Magic command for Jupyter Notebook to display plots inline

# Setting the size of the figure for plots
rcParams['figure.figsize'] = (14,10)  # Set the default figure size to 14x10 inches

# Defining geographical boundaries for the map
llon = -140  # Lower longitude boundary
ulon = -50   # Upper longitude boundary
llat = 40    # Lower latitude boundary
ulat = 65    # Upper latitude boundary

# Selecting specific columns from the DataFrame for analysis
pdf = pdf[['lead_time', 'arrival_date_month', 'stays_in_weekend_nights', 'stays_in_week_nights']]

# Creating a Basemap instance
my_map = Basemap(projection='merc',  # Mercator projection
                 resolution='l',     # Low resolution
                 area_thresh=1000.0, # Minimum area threshold in square kilometers for displaying features
                 llcrnrlon=llon, llcrnrlat=llat,  # Lower-left corner longitude and latitude
                 urcrnrlon=ulon, urcrnrlat=ulat)  # Upper-right corner longitude and latitude

# Drawing map elements
my_map.drawcoastlines()  # Draw coastlines
my_map.drawcountries()   # Draw country boundaries
# my_map.drawmapboundary()  # Uncomment to draw the map boundary
my_map.fillcontinents(color='white', alpha=0.3)  # Fill continents with white color and some transparency
my_map.shadedrelief()    # Add shaded relief to the map for a 3D effect

# Projecting the longitude and latitude data to the map's coordinate system
xs, ys = my_map(np.asarray(pdf.Long), np.asarray(pdf.Lat))  # Convert longitude and latitude to map coordinates
pdf['xm'] = xs.tolist()  # Store the projected x-coordinates in the DataFrame
pdf['ym'] = ys.tolist()  # Store the projected y-coordinates in the DataFrame

# Visualization: Plotting data points on the map
for index, row in pdf.iterrows():  # Iterate through each row in the DataFrame
    my_map.plot(row.xm, row.ym, markerfacecolor=([1,0,0]), marker='o', markersize=5, alpha=0.75)  # Plot each data point

# Display the plot
plt.show()  # Show the final plot
```

Figure 178 - DBSCAN visualization

8.3.4 Clustering

```
# Importing necessary libraries for clustering
from sklearn.cluster import DBSCAN  # DBSCAN clustering algorithm
import sklearn.utils  # General utility functions from scikit-learn
from sklearn.preprocessing import StandardScaler  # StandardScaler for normalization

# Setting a random state for reproducibility
sklearn.utils.check_random_state(1000)  # Ensure a deterministic random state

# Preparing the data for clustering
Clus_dataSet = pdf[['xm', 'ym']]  # Selecting only the 'xm' and 'ym' columns for clustering
Clus_dataSet = np.nan_to_num(Clus_dataSet)  # Replacing NaN values with 0 (or another small number)
Clus_dataSet = StandardScaler().fit_transform(Clus_dataSet)  # Normalizing the data

# Compute DBSCAN
db = DBSCAN(eps=0.15, min_samples=10).fit(Clus_dataSet)  # Applying DBSCAN algorithm to the normalized data
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)  # Initializing a mask for core samples
core_samples_mask[db.core_sample_indices_] = True  # Marking core samples in the mask
labels = db.labels_  # Extracting the labels assigned by the DBSCAN algorithm
pdf["Clus_Db"] = labels  # Adding the cluster labels to the original DataFrame

# Calculating the number of clusters
realClusterNum = len(set(labels)) - (1 if -1 in labels else 0)  # The number of actual clusters, excluding noise points
clusterNum = len(set(labels))  # Total number of clusters including noise (-1 label)

# Display a sample of the clustered data
pdf[["Stn_Name", "Tx", "Tm", "Clus_Db"]].head(5)  # Displaying the first 5 rows with cluster labels
```

Figure 179 - DBSCAN clustering

8.3.5 Visualization of clusters based on an aspect

```
# Importing necessary libraries for map visualization
from mpl_toolkits.basemap import Basemap  # Basemap toolkit for creating geographic maps
import matplotlib.pyplot as plt  # Matplotlib's pyplot for plotting
from pylab import rcParams  # PyLab's rcParams for setting figure properties
%matplotlib inline  # Magic command for Jupyter Notebook to display plots inline
rcParams['figure.figsize'] = (14,10)  # Setting the default figure size

# Creating a Basemap instance for map visualization
my_map = Basemap(projection='merc',  # Mercator projection
                 resolution='l',     # Low resolution of the map
                 area_thresh=1000.0, # Minimum area threshold for displaying features
                 llcrnrlon=llon, llcrnrlat=llat,  # Lower-left corner longitude and latitude
                 urcrnrlon=ulon, urcrnrlat=ulat)  # Upper-right corner longitude and latitude

# Drawing map elements
my_map.drawcoastlines()  # Draw coastlines on the map
my_map.drawcountries()   # Draw country boundaries on the map
# my_map.drawmapboundary()  # Uncomment to draw the map boundary
my_map.fillcontinents(color='white', alpha=0.3)  # Fill continents with white color and some transparency
my_map.shadedrelief()    # Add shaded relief to the map for a 3D effect

# Creating a color map for different clusters
colors = plt.get_cmap('jet')(np.linspace(0.0, 1.0, clusterNum))  # Using 'jet' colormap for cluster colors

# Visualization of clusters on the map
for clust_number in set(labels):  # Loop through each cluster number
    # Assign gray color to noise points, otherwise use a color from the color map
    c = (([0.4, 0.4, 0.4]) if clust_number == -1 else colors[np.int(clust_number)])

    # Selecting data points that belong to the current cluster
    clust_set = pdf[pdf.Clus_Db == clust_number]

    # Plotting data points for the cluster on the map
    my_map.scatter(clust_set.xm, clust_set.ym, color=c, marker='o', s=20, alpha=0.85)

    # If not a noise cluster, calculate and display the cluster center and print average temperature
    if clust_number != -1:
        cenx = np.mean(clust_set.xm)  # Calculate mean x-coordinate of the cluster
        ceny = np.mean(clust_set.ym)  # Calculate mean y-coordinate of the cluster
        plt.text(cenx, ceny, str(clust_number), fontsize=25, color='red')  # Display cluster number at the cluster center
        print("Cluster " + str(clust_number) + ', Avg Temp: ' + str(np.mean(clust_set.Tm)))  # Print average temperature of the
```

Figure 180 - DBSCAN clusters based on an aspect

8.3.6 Clustering based on several aspects

```
# Importing necessary libraries for clustering and data preprocessing
from sklearn.cluster import DBSCAN  # DBSCAN clustering algorithm
import sklearn.utils  # General utility functions from scikit-learn
from sklearn.preprocessing import StandardScaler  # StandardScaler for normalization

# Setting a random state for reproducibility
sklearn.utils.check_random_state(1000)  # Ensure a deterministic random state

# Preparing the data for clustering
Clus_dataSet = pdf[['xm', 'ym', 'Tx', 'Tm', 'Tn']]  # Selecting specific columns for clustering
Clus_dataSet = np.nan_to_num(Clus_dataSet)  # Replacing NaN values with 0 (or another small number)
Clus_dataSet = StandardScaler().fit_transform(Clus_dataSet)  # Normalizing the data

# Compute DBSCAN
db = DBSCAN(eps=0.3, min_samples=10).fit(Clus_dataSet)  # Applying DBSCAN algorithm to the normalized data
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)  # Initializing a mask for core samples
core_samples_mask[db.core_sample_indices_] = True  # Marking core samples in the mask
labels = db.labels_  # Extracting the labels assigned by the DBSCAN algorithm
pdf["Clus_Db"] = labels  # Adding the cluster labels to the original DataFrame

# Calculating the number of clusters
realClusterNum = len(set(labels)) - (1 if -1 in labels else 0)  # The number of actual clusters, excluding noise points
clusterNum = len(set(labels))  # Total number of clusters including noise (-1 label)

# Display a sample of the clustered data
pdf[["Stn_Name", "Tx", "Tm", "Clus_Db"]].head(5)  # Displaying the first 5 rows with cluster labels
```

Figure 181 - Clustering based on several aspects

9 Conclusion

This report has provided a comprehensive overview of the essential components of data analysis within the realm of Machine Learning. It delved into descriptive data analysis, which forms the foundation of understanding a dataset by applying statistical measures and visual representations.

Exploratory Data Analysis was explored as a means to uncover underlying patterns and relationships, employing visual techniques like scatter plots and histograms. Furthermore, inferential data analysis was discussed, which enables the making of predictions about populations based on sample data.

In addition to theoretical insights, practical experimentation was conducted across various platforms. SPSS was employed for its robust capabilities in descriptive and exploratory analysis, while Jupyter Notebooks facilitated the utilization of the Python language for data manipulation and modelling. Orange (3) emerged as a valuable tool, offering a graphical approach to inferential data analysis.

Overall, this document underscores the critical role of data analysis in the Machine Learning landscape and highlights the diverse range of tools available for practitioners to effectively analyse and extract insights from their datasets.

10 References

GPL Reference Guide for IBM SPSS Statistics – downloaded from https://www.ibm.com/support/pages/ibm-spss-statistics-28-documentation - 27/09/20223

IBM SPSS Missing Values 28 – downloaded from https://www.ibm.com/support/pages/ibm-spss-statistics-28-documentation - 27/09/20223

IBM SPSS Custom Tables 28 – downloaded from https://www.ibm.com/support/pages/ibm-spss-statistics-28-documentation - 27/09/2023

IBM SPSS Advanced Statistics 28 – downloaded from https://www.ibm.com/support/pages/ibm-spss-statistics-28-documentation - 27/09/2023

IMB Documentation – Recode into Different Variables

IMB Documentation – Crosstabs

IMB Documentation – Select cases

Orange Documentation - Predictions