Report for the Data Science Course*

Enhance Surgical Time Schedule

Primary Topic: Data Mining, Secondary Topic: Data Visualization

Course: 2018-1A – Group: 92 – Submission Date: 2022-04-24

Sara-Jane Bittner
Student Number: 2876574
University of Twente
The Netherlands
s.bittner@student.utwente.nl

Xiao-Lan Bokma
Student Number: 2437236
University of Twente
The Netherlands
x.l.m.bokma@student.utwente.nl

ABSTRACT

BACKGROUND: *Operation Room planning* represents a crucial part of hospitals. However, the estimation of the surgical duration shows deficiencies resulting in un- or over-utilized operation rooms.

METHODS: The data set contained 4085 observations, while after the preprocessing is contained 1670 observations. The outcome was a new planned duration time, with a set of eleven significant features derived from patient and surgery data. For the prediction of the surgical duration time an experiment with 13 classifiers was conducted. Including five categories: Dummy classifiers, regression models, unsupervised classifiers, supervised classifiers and a self-implemented *Inverted Indexes* algorithm.

RESULTS: For the prediction of continuous surgery time, the *linear regression* model performed better than the original prediction. It showed a RMSE difference of 13.172. For the categorized data the improvement was less distinct. The *Random forest* model improved the predicted of the surgical duration time by an accuracy difference of 0.066 to the original prediction.

CONCLUSION: The current prediction system for surgical duration time of *Thorax Center Enschede* shows major deviations to the actual duration time. The developed algorithm, especially for the continuous prediction showed a lowered the error rate. Thus, it is advised to implement a prediction algorithm as such in daily *OR-planning* to decrease uncertainty and prevent costs and not utilized *OR* time. For example according to the example of the *duration generator* introduced in this paper.

KEYWORDS

surgical duration time, OR-planning, Operation Room, Prediction Algorithm, Pre-Processing, Pipeline, Duration Generator, Feature Selection

1 INTRODUCTION

Operation Rooms (OR) represent a crucial part of our health care system as around 60% of the patients, that are admitted to the hospital are treated in an *OR* [8]. However, the nowadays *OR-planning* shows many deficiencies in execution, leading to high financial loses [11, 18, 21] and decreasing efficiency and satisfaction for patients and hospital employees [5, 7, 25]. One main source of these inefficiencies is overtime of planned surgeries [17]. In context of this paper *overtime* is defined as a deviation from the predicted surgery time and the actual surgery time. It counts in both directions, under-utilization of the surgery time, where the surgery is finished earlier then predicted, or an over-utilization, where the surgery lasts longer than predicted. Further, the main source of overtime is poor prediction of the surgery time [19]. This is due to the high complexity and uncertainty of the prediction [10, 13]. This project aims to develop an algorithm which performs better in predicting the surgery time then the current hospital system of the *Thorax Center Enschede.* The purpose includes to reduce the overall overtime and thus decreasing the costs and increase patient satisfaction. In order to achieve the research goal the following research questions were established:

Research Questions:

(1) How is the current hospital system performing?
(2) What classifier shows the best performance measures?
  (a) For continuous classification?
  (b) For categorical classification?
(3) Is the algorithm performing better than the initial hospital system?

2 APPROACH

This chapter will describe the data exploration, pre-processing and the feature selection.

2.1 Data Exploration

The following section covers the exploration of the data set with the software *Tableau.* For this the data set was prepared and restructured to a star-scheme (figure 1. The full star scheme and the corresponding coding can be found in appendix A.

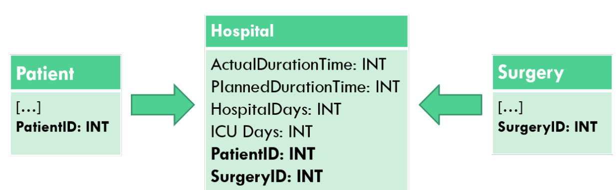

Figure 1: A simplified version of the star scheme

Data Set. The data set was provided by the *Thorax Center Enschede.* It contained in total 4085 rows and 36 columns. The data

*Adapted from the ACM SigConf Template. More information about the template can be found at https://www.acm.org/publications/proceedings-template types of the features can be divided in binary, categorical and numerical. The distribution of the columns into these data types can be derived from appendix B (directly after retyping) and appendix C (after the column modifications). The cleaned data set results in 1670 rows and 69 columns.

Prediction Time with Original System. Figure 2 shows a boxplot with the current overtime: the deviation shows how close the current prediction time is to the actual duration time. It is highlighted that the *predicted duration time* is off from the *actual duration time*, as it spreads more widely in a range of -300 to 500. This indicates that there are deficiencies in the prediction in the original hospital system. A more detailed insight can be derived from Appendix E.

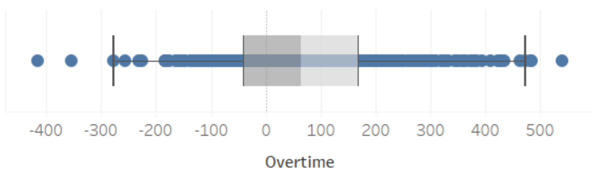

Figure 2: Boxplot of current Overtime

Table 1: The bins of the categorical features. The min and max refer to the minimum and maximum values of that column

Feature	Bins
BMI	min, 18.5, 24.9, 29.9, max
Age	min, 57, 66, 76, max
Overtime	min, -70, -25, 25, 70, 120, max
Actual Duration Time	min, 180, 240, 280, 345, max
Planned Duration Time	min, 180, 240, 280, max

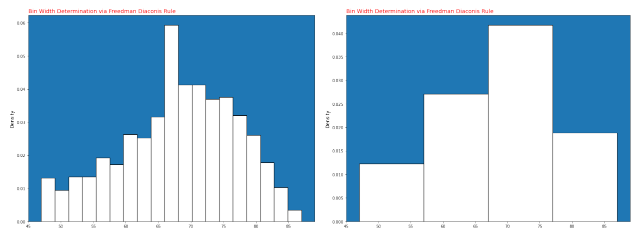

Figure 3: Histogram of 'Age' simplified in bins

2.2 Pre-Processing

The following section covers the preparation of the data in seven steps. The steps aim to transfer the data into a *pandas datatype* that can be inserted in the classification algorithm. The steps are: *re-typing of the columns, splitting the OperationType string, binning the data, splitting the OPerationType Strings, removal of outliers and columns/rows with too much missing data, addition of new columns, encoding data,* and *normalization of the data.*

2.2.1 Re-Typing of Columns. The data of the .csv files is imported as strings, thus the *pandas dataframe* makes these columns 'object' columns. Other columns are recognized as numerical columns, while they should be approached as binary: it contains two *categories* with the labels 0 and 1. These columns should be converted to 'int8' and eventually be labeled as 'category'. The real numerical values should be converted to 'foat64' in order to calculate with it or derive conclusions from. Appendix **??** shows the data types after the conversion.

2.2.2 Binning. Ordinal numerical data is converted to ordinal categorical data, such that the classifier is better interpretable. Table 1 shows the features that are converted to the defined bins, For example, the BMI ranges are according to the Dutch 'voedingscentrum' [24] and are named accordingly: underweight, normal, overweight,and obese. Other ranges are visually defined according to their histograms, such that the bins are approximately similar. Figure 3 shows the transfer of the full histogram of 'age', to the ranged 'age, *Actual Duration Time* and *Planned Duration Time* were binned the same, as the predictor will need to compare the values with each other.

2.2.3 Splitting the OperationType String. It appeared that OperationType had 360 unique values. This is due to the combined strings, containing the operation types each. Appendix D shows the data while the strings are still combined. The solution to this problem was to split the strings on the '+' sign and directly hot-encode the data.

2.2.4 Missing Values in the data set. Missing values cause problems for both the pre-processing pipeline and the final classification as they cannot be properly inputted, Therefore, missing values, will cause errors and are going to be removed from the *surgical data set.* First, features columns were deleted that contained >50% of empty cells. This threshold was chosen, because restoring more than 50% of missing values in a column would compromise the validity of the data greatly. Based on this process the features the following features were removed from the data set: *Left Ventricle Function, EuroScore 2* and *Renal Function.* Secondly, rows with any *NaN*-values were removed from the data set. This resulted in 2160 remaining data points. This which was considered as satisfactorily for the training and classifying process.

2.2.5 Outliers in the data set. Data outside the statistical norm of the data set is either removed (numerical data) or regrouped (categorical data). Outliers add to the curse of dimensionality: new phenomena appear in higher dimensional data, while this would not appear in lower dimensional data. By removing the outlier, it will remove complexity if the data. Also, outliers can heavily increase the variability in the data, which will lead in less powerful statistical tests. Lastly, removing outliers will also prevent overfitting of the algorithm.

Removal of Numerical Outliers. Numerical outliers are determined by their z-scores. Z-scores show how many deviations a certain data point below or above the mean is. The z-scores are calculated according equation 1. Figure 4 shows the data before and after the outlier cutting. These show the inter-quartile ranges (IQRs). It shows the 25th and 75th percentile of the dataset. The bottom and the top whiskers display the minimum and maximum

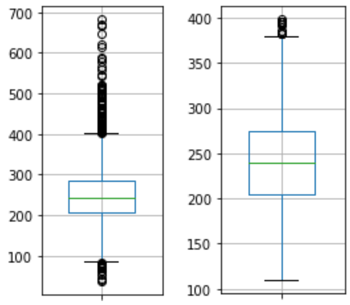

Figure 4: The boxplot for *Actual Duration Time* before and after the outlier cutting

value. Every outlier outside of these IQRs will be plotted outside the whiskers.

$$z = \frac{datapoint - mean}{standarddeviation} \tag{1}$$

Where the following z-score says the following about the data point: negative values are below average, positive are above average, zero is close to average and |3| is considered unusual. As the outlier cutting for a z-sore of 3 still showed a lot of outliers outside the IQR range in the boxplots, and cutting at 1 wold affect the wiskers, the cutting value is set to a z-score of 2.

Removal and Grouping of Categorical Outliers. Categorical outliers are values of categorical features with a low count in the data set. This causes two main problem in the prediction: First, rare values, do mostly not provide enough predicting value. Secondly, nominal data that gets encoded with *One Hot Encoding* creates a high increase in features, as every unique value is transferred to an own feature. This can lead to a tremendous amount of features, causing problems in complexity and lead to overfitting. However, rare values can not be easily deleted, as it can not be determined if the value does provide additional information, which might be needed. Therefore, we decided for two steps: First, values that made under **1%** of the total data were removed. Second, categorical values of columns that represented less than **5%** of the data after the initial removal, were summarized with the value *Other*. This way, the additional information value of less common data was not lost, but summarized.

2.2.6 Categorical Encoding. The following subsection covers the encoding of categorical data. Categorical data contains label values. Categorical data needs to be encoded into numerical data, because machine learning algorithm cannot process non-numerical values. Two main methods can be applied to achieve this transfer from the *SkitLearn-* Library, based on if the categorical feature contains nominal or ordinal values: First, for ordinal data, therefore data values that contain a ranking, the **Label Encoder** was implemented. This encoder assigns each label in a feature an unique integer. Second, for nominal data, which does not contain an ranking order, **One Hot Encoding** was implemented. This encoder creates dummy variables. That way, each label is added as an additional binary feature to the table [4, 20]. For this project the categorical feature-columns of **Surgical Case Durations**-data set, were checked for ordinality. If this requirement was fulfilled the **Label Encoder** was

Table 2: Example for One-Hot-Encoding for Nominal data

Surgeon
1
2
[...]

encoded to:

$Surgeon_1$	$Surgeon_2$	[...]
0	1	[...]
1	0	[...]
[...]	[...]	[...]

Table 3: Example for Label-Encoding for Ordinal data

BMI Range
1
0
2
[...]

meaning:

Encoding
0 = underweight
1 = normal weight
2 = overweight

applied, otherwise the **One Hot Encoding** [20, 22]. An Overview of the split of the categorical feature columns for pre-processing can be found in table B.

Detection of Multicollinearity. One major drawback of the **One Hot Encoding**-method is the dummy variables lead to the problem of multicollinearity. This is the occurrence of high intercorrelations in two or more independent variables in a multiple regression model. [9] In order to prevent the possible multicollinearity in dummy variables that were created from one feature, the **Varicance Inflation Factor (VIF)** can be checked.[23] The calculation and filtering of the data showed that no multicollinearity is given for the categorical features.

2.2.7 Normalization. The following section covers the normalization process, which transfers the numeric columns of the data set into a similar scale. Machine Learning Algorithms are sensitive to different scaled features, so that columns with higher scales will influence the model in a higher extent, compared to columns with a smaller scale. The goal of the normalization process is to transfer numerical columns into similar scales, so that the various numerical features influence the model to the same extent.In context of this project the **Min-Max Normalization** was applied for the normalization process.It performs a linear transform in which the numerical features will be transformed to a scale of 0 to 1. [12] Given the name the minimum value will be tranformed to 0, and the maximum value to 1. The formula can be seen in in 2.

$$v_i' = \frac{v_i - min_A}{max_A - min_A} \cdot (newmax_A - newmin_A) + newmin_A \tag{2}$$

Attention must be drawn to the fact that the Min-Max Normalization is prone to the influence of outliers. However, this problem was solved in context of chapter 2.2.5. [14, 20]

2.3 Feature Selection

The following section covers the feature selection process. For this features were selected based on the *Backward Elimination* algorithm.

2.3.1 Feature Selection Algorithm. A reduction of features was of importance, because the project counted a total of 65 after the pre-processing. This amount raises the complexity and training time of the algorithm highly. Moreover, not every feature might contribute to the prediction process significantly and a too high amount of features might lead to overfitting [16]. To reduce these *Wrapper*-methods were considered because they also take interrelations of features into consideration. These methods, create subsets of features and choose the best performing model based on the evaluation metric [16]. Finally, for the project, the method ***Backward Elimination*** was implemented. This selection process starts with the full model and removes iterative insignificant features from the model until the feature set only consists of significant features [22]. Aligning with the *star schema* (see 1) for this data set two *Backward Eliminations* were calculated: One for the surgical features and one for the patient features. This way, significant features for both sub-set of features should be derived, to not underestimate the influence of some in the greater context. After deriving the features for both sets individually they were merged into one feature set. Thus, resulting in 19 significant features, which can be derived in more detail from table 4. The explanations to each selected feature can be derived from appendix I

Table 4: By Backward Elimination selected Features

Category	Features
Scores related to heart conditions	CCS, NYHA
Previous Patient Conditions	Previous Heart Surgery, Presence of Hypertension, Presence of Pulmonale Hypertension, Presence of Active Endocarditis, Artrial Fibrillation
Surgery Planning	Aortic Surgery, Amount of Bypasses, Use of Cardiopulmonary Bypasses, Operation Types [9]

3 EXPERIMENT

This section describes the implementation of the classifiers, in implementation of *inverted indexes* and the *duration time generator.*

3.1 Experimental Setup

The output data for the pipeline is split into two versions: *ActualDurationTime* (continuous data) and *ActualDurationTimeRange* (categorical data), since these two could give different insights in the performance of the algorithms. To define whether a model performed *better* than the original predictions, scores and errors for the original predictions will be included as the first row in the comparison tables table 13 for the continuous data and table 14 for the ranged data (categorical).

3.1.1 Classification Exploration. First the four dummy classifiers are implemented, to compare classifiers to the performance of*simple rules.* Secondly, regression algorithms are only implemented for the continuous version of the data, as they can be represented in a linear. The regression method for categorical data would calculate with rounded values. Thus, it would most likely perform worse than with continuous data. Thirdly, three supervised and five unsuperivsed learning algorithms were implemented. As the clustering algorithms are for discrete data, it is beforehand known that the continuous data would not perform well. However, it is purposely implemented for the binned outcome data, as the bins could be considered discrete as well. Finally, another technique outside the sklearn and scikit library is implemented from the field *information retrieval*, called: *Inverted Indexes.* It is a fast method to retrieve similar documents, while doing full-text searches[15]. The expressions in features are mapped to all the index rows they appear in, Thus, their indexes are inverted. Table 5 and 6 show how this is done with a vector style for the project.

Table 5: Normal Indexing

RowID	Vector
1	OR1, SurgeonA, ...
2	OR1, SurgeonA, ...
3	OR2, SurgeonB, ...
4	OR2, SurgeonB, ...

Table 6: Inverted Indexing

Vector Values	RowIDs
OR1	1, 2, ...
OR2	4, 3, ...
SurgeonA	1. 2, ...
SurgeonB	4, ...

This method is especially effective when the data is in bins (categorical): the bins will then have more overlap with similar rows in the data set. Leading to more indicators in the same area that will point towards a specific result. Since the data in the project can also be considered 'vectors', it was considered interesting to implement it for this data set as well.

3.1.2 Classification Testing. Two main measurement methods can be distinguished: *Accuracy* [27] and *RMSE* [26]. *Accuracy* represents the degree of which the predictions are close to the true values, e.g. the *actual duration time.* The calculation can be derived from the equation 3.

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn} \tag{3}$$

Where *tp* stands for 'true positive', *fp* for 'false positive', and *fn* for 'false negative'.
As accuracy is binary, there is no room for distinguishing predictions that are just slightly of from the actual time from predictions that are greatly off. The scores will show a biased result in which the algorithm performed overly strict.

The Root Mean Squared Error (RMSE) show the result in a more metric way. The RMSE is the averaged distance between the predicted output (*Planned Duration Time*) and between the true observations (*Actual Duration Time*) [26]. Equation 4 shows how these errors are calculated.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_{pred} - y_{true})^2} \tag{4}$$

Where N stands for the sample size, y_{pred} for the predicted value, and y_{true} for the observed value.

RMSE is a more reliable method for the continuous prediction of duration time as it considers the distance between the predicted and the observed value: the higher the value, the bigger the error and the worse the algorithm performs.

4 RESULTS

The following section covers the the results of the testing of the various classifiers in section 3.1. The full result tables can be found in appendix F.

For the **continuous prediction** the *Linear Regression* performed best with a *RMSE* of 45.91 Overall, regressor algorithms and supervised classifiers showed similar and satisfying results, while unsupervised clustering algorithms performed poorly on the data set. For the **categorical prediction** the *Random Forest Classifier* performed best with an *accuracy* of 44.5%. An overview of the best perfoming classifier per category of classifiers can be derived from table 7. Additionally, the performance of the original prediction of the hospital is displayed on the first row. Here *accuracy* represents the performance of the classifiers on the categorical prediction, while *RMSE* represents the performance of them for the continuous prediction.

Table 7: RMSE for the continious *Actual Duration Time* and Accuracy for the categorical *Actual Duration Time Range.*

	Continuous Data	Categorical Data
	RMSE Error	Accuracy Score
Original Prediction	**59.082**	**0.351**
Dummy Classifiers		
Most Frequent		*0.373*
Regressor	*54.959*	
...	...	...
Regression Models		
Linear Regression	45.910	
...	...	...
Unsupervised Classifiers		
Random Forest	50.970	0.445
...	...	...
Supervised Classifiers		
Birch		0.208
...	...	...
Information Retrieval Algorithm		
Inverted Indexes	58.877	0.407

Both, the best prediction for continuous and categorical data showed better performances than the hospital prediction. The *Linear Regression*, improved prediction by 13 *RMSE*, from 59 to 45. While the *Random Forest* improved the accuracy from 35.1% around 10% more to 45.5%. However, regarding the overtime only the *continuous prediction* showed a closer distribution of overtime around no-overtime, than the original prediction. Therefore, the prediction times are closer to the *Actual Duration Time*. This can be seen in figure 5. Both the comparisons for linear regression and random forest are visible in appendix G.

It needs to be highlighted, that supervised classifiers performed worst compared to the other categories of classifiers. A full overview about all 13 tested classifiers can be derived from the heatmaps in

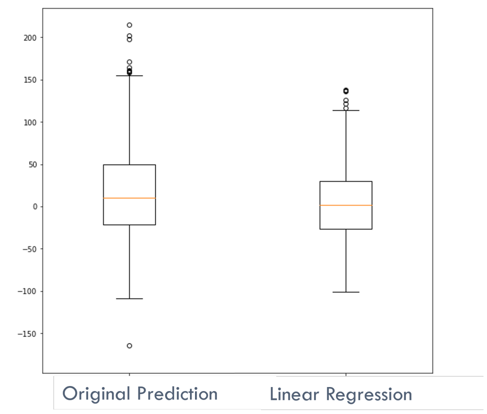

Figure 5: Boxplot comparison of Overtime for the *original prediction* and the with *Linear Regression for the continuous data*

appendix F. The full results for the continuous prediction can be seen in table 13 and in table 14 for the categorical prediction.

5 DISCUSSION

The following section contains the discussion of the results in connection to the established research question. First, the research questions are answered, followed by the ones concerning the algorithm performance.

5.1 Research Questions

(1) How is the current hospital system performing?

For the current estimation of the surgical time from the hospital system it became clear that the prediction showed deficiencies. For the continuous prediction of the *surgical duration time* the *Root Mean Square Error (RMSE)* showed a rather high value of 59.082. Moreover, the categorical predictions lacks performance with a low accuracy of 35.1%. Thus, the nowadays performance of the *Thorax Center Enschede* shows high deficiencies in the prediction of surgical duration time, influencing costs and satisfaction of hospital workers as well as patients negatively [7, 21]. Therefore, it was highlighted that to achieve a better usage of the available operation rooms and the operation time the current system has to be updated. For this a machine learning approach was suggested and developed as indicated by nowadays literature [3].

(2) What classifier shows the best performance measures?

The best performance for the continuous data is interpeted with the RMSE, which then shows us the closes fitting model [28]. However, the value itself is not interpreted with the accuracy, to interpret overfitting [1]. For continuous data, it is clear that *Linear Regression* shows the best RMSE. For categorical data the *Random Forest* performed best in terms of accuracy. Interestingly, table 14 shows that the supervised learning algorithms (clustering methods) for binned data still perform bad in both cases. From this, it can be suspected that the data is not as discrete as previously assumed. This could mean that the accuracy score as measurement is less powerful for evaluating the classifiers for this data version.

(3) Is the algorithm performing better than the initial hospital system?

The algorithm is performing better than the initial hospital system recording to the performance metrics. For categorical data a improvement of accuracy from around 10% was achieved, from 35% to 45% while the continuous prediction showed a decrease of the *Rooted Mean Squared Error* of 13 points, from 59 to 45. However, regarding to the plotting of the newly calculated *overtime* distribution only the *continuous prediction* shows a visible reduction of the distribution of overtime. Thus, showing that the predicted surgical time is closer to the actual duration time compared to the original hospital prediction. This is not the case, for the overtime boxplot of the categorical prediction. Therefore, the algorithm for continuous data would improve the performance of *OR planning* and should be implemented in the hospital environment.

5.2 Limitations

Regarding the discussed results a variety of limitations can be derived. These include the pre-processing, the classifiers, feature selection and a reflection on the outcome data versions.

5.2.1 Pre-Processing. Missing data have been removed fully in the data set. However, strategies to replace the missing data could enlarge the data set and therefore, support a better prediction [2]. For this, different approaches can be considered: First, numerical data could be replaced with the median or the mean. Further, missing values for categorical features could be replaced by the most frequent expression. These count to *parameter estimation*, but include possible limits, such as over-representing an expression or neglecting interrelations [2]. Lastly, an imputation algorithm could predict missing values, which includes interrelations of features [2]. Further work should implement imputation of missing data as a compensation strategy, as more than 15% of the data set are removed in this project [2].

5.2.2 Feature Selection. 19 features were selected by the *backward elimination* algorithm. However, this selection can be improved by *Human Verification of Feature.* It became clear that features were included were the significance should be questioned. An example for such a feature is the operation-type *shaving*, as it is a pre-procedural to the operation and therefore, does not influence the surgery time itself. Several researches have been indicating that a *human-in-the-loop* approach, where feedback is given by experts and based in research to selected features can improve the performance of the algorithm significantly [6]. Therefore, future research should include a *human-in-the loop* approach.

5.2.3 Classifier. Limitations regarding the classifiers include two parts: First, the *surgical data set* was not reshaped to the preferred input regarding the pre-selected classifiers. Therefore, future research should consider the preferred shapes and pre-process the data accordingly. Secondly, it could be distinguished between over-utilized and under-utilized predictions. Therefore, enhancing the scheduling by differentiating features that are significant to these two cases.

6 CONCLUSION

All in all, the current prediction system of the surgical duration time of the *Thorax Center Enschede* does show deficiencies. The algorithm introduced in this paper improves the prediction of the duration, by a reduction in *RMSE* of 13 from 59 from the *original prediction* to 45 using the *Linear Regression.* Therefore, it is recommended to implement the new prediction system to reduce uncertainty of *OR-planning.* With this it is aimed to prevent costs and dissatisfaction of patients. As a further step, the new system should be established in a user application for the *OP-planners.* A proposal for this is included in the following additional chapter as the *Duration Generator.* This code was fully implemented in the project code and shows the first steps of transferring the results of this paper into a viable application in the context of the hospital environment.

6.1 Duration Generator

The *Duration Generator* represents the implementation of the algorithm as a user application. The user can input the patient and surgery data and will then get a predicted time for the surgery. The target group are *OP planners.* The *Duration Generator* derives the needed entry features and their unique values from the *surgicalData* data set that is used for the prediction algorithm. Then the user is asked to input the value for each feature for the patient. An example input can be seen in figure 6.

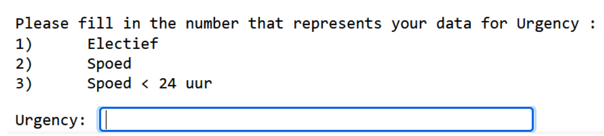

Figure 6: The input procedure of the *DurationGenerator.* The corresponding number to the value can be put in the text field.

Further, the algorithm also prevents errors due to wrongly inputted values with checking if the input is numerical or if it extends the range of provided values. In this case the user gets an error message and need to refill the entry. Based on the information the *Duration Time* of the surgery is predicted with the model trained of the *Linear Regression.* This can be seen in figure 7.

```
The assumed value for ActualDurationTime in this operation is 271
```

Figure 7: The output for the user application that contains the predicted duration of the surgery

To an further extent, the *Actual Duration Time* could be inputted in the data row after the predicted surgery. With this method new data points can be collected. After a certain amount of new data points the model can be updated. This way, it could improve it's performance based on the increasing size of the data set and the information it is containing with time.
However, future research should develop a more intuitive user interface to successfully implement this prediction algorithm in a hospital environment

A FULL STAR SCHEMA

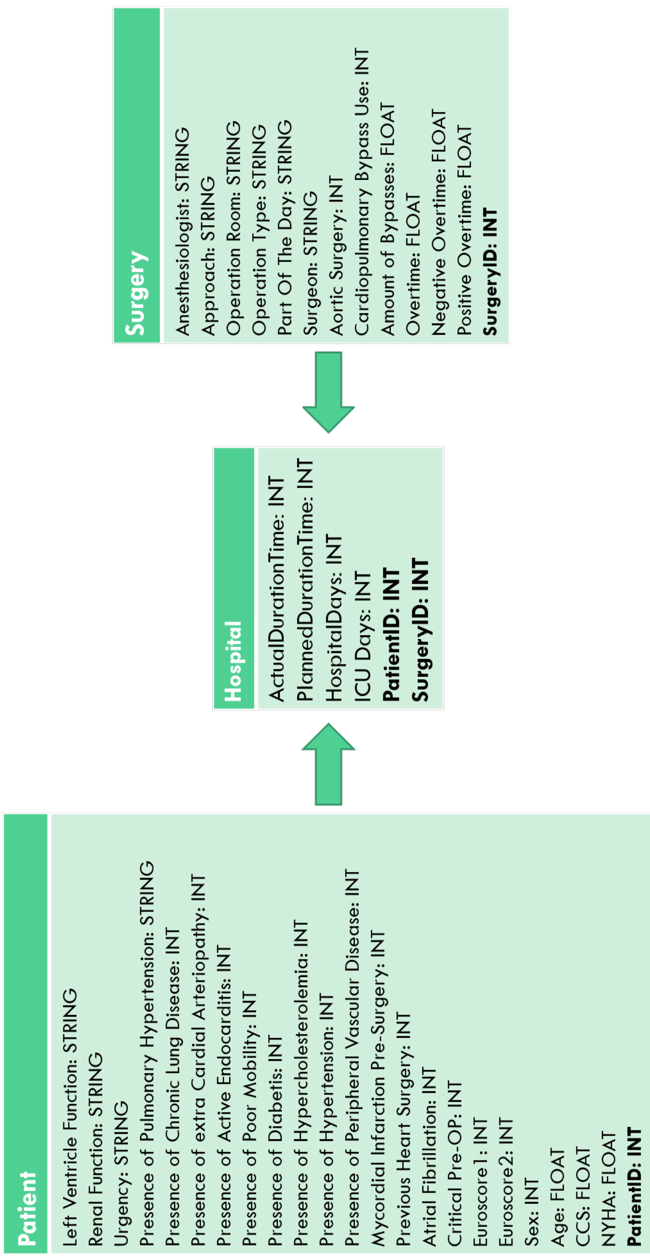

Figure 8: Star Schema

B COLUMNS AFTER THE CONVERSION

This appendix shows two tables: one table with the data types, directly after conversion (table 8) and after the column modifications (table 9), but before the encoding. The categorical ordinal and non-ordinal columns will be label- or one hot encoded respectively.

Table 8: The column types and their corresponding dType, before column modifications

Feature	Type	dType
Sex	Binary	int8
Atrial Fibrillation	Binary	int8
Presence of Chronic Lung Disease	Binary	int8
Presence of extra Cardial Arteriopathy	Binary	int8
Previous Heart Surgery	Binary	int8
Presence of active Endocarditis	Binary	int8
Critical Pre-OP	Binary	int8
Mycordial Infarction Pre Surgery	Binary	int8
Aortic Surgery	Binary	int8
Presence of Poor Mobility	Binary	int8
Presence of Diabetis	Binary	int8
Presence of Hypercholesterolemia	Binary	int8
Presence of Hypertension	Binary	int8
Presence of Peripheral Vascular Disease	Binary	int8
Cardiopulmonary Bypass Use	Binary	int8
Age	Numerical (Ordinal)	float64
BMI	Numerical (Ordinal)	float64
Actual Duration Time	Numerical (Ordinal)	float64
Planned Duration Time	Numerical (Ordinal)	float64
Amount Of Bypasses	Numerical (Non-Ordinal)	float64
Euroscore1	Numerical (Non-Ordinal)	float64
Euroscore2	Numerical (Non-Ordinal)	float64
Hospital Days	Numerical (Non-Ordinal)	float64
ICU Days	Numerical (Non-Ordinal)	float64
CCS	Categorical (Ordinal)	category
NYHA	Categorical (Ordinal)	category
Urgency	Categorical (Ordinal)	category
Left Ventricle Function	Categorical (Ordinal)	category
Renal Function	Categorical (Ordinal)	category
Presence of Pulmonary Hypertension	Categorical (Ordinal)	category
Surgeon	Categorical (Non-Ordinal)	category
Operation Room	Categorical (Non-Ordinal)	category
Operation Type	Categorical (Non-Ordinal)	category
Anesthesiologist	Categorical (Non-Ordinal)	category
Approach	Categorical (Non-Ordinal)	category
Part Of the Day	Categorical (Non-Ordinal)	category

C COLUMNS AFTER CUTTING MISSING VALUES

Table 9: The column types and their corresponding dType, before column modifications

Feature	Type	dType
Sex	Binary	int8
Atrial Fibrillation	Binary	int8
Presence of Chronic Lung Disease	Binary	int8
Presence of extra cardial Arteriopathy	Binary	int8
Previous Heart Surgery	Binary	int8
Presence of active Endocarditis	Binary	int8
Critical Pre-OP	Binary	int8
Mycordial Infarction Pre Surgery	Binary	int8
Aortic Surgery	Binary	int8
Presence of Poor Mobility	Binary	int8
Presence of Diabetes	Binary	int8
Presence of Hypercholesterolemia	Binary	int8
Presence of Hypertension	Binary	int8
Presence of Peripheral Vascular Disease	Binary	int8
Cardiopulmonary Bypass Use	Binary	int8
Age **Range**	**Categorical (Ordinal)**	**category**
BMI **Range**	**Categorical (Ordinal)**	**category**
Actual Duration Time **Range**	**Categorical (Ordinal)**	**category**
Planned Duration Time **Range**	**Categorical (Ordinal)**	**category**
Overtime Range	**Categorical (Ordinal)**	**category**
Amount Of Bypasses	Numerical (Non-Ordinal)	float64
Euroscore1	Numerical (Non-Ordinal)	float64
~~Euroscore2~~	~~Numerical (Non-Ordinal)~~	~~float64~~
Hospital Days	Numerical (Non-Ordinal)	float64
ICU Days	Numerical (Non-Ordinal)	float64
CCS	Categorical (Ordinal)	category
NYHA	Categorical (Ordinal)	category
Urgency	Categorical (Ordinal)	category
~~Left Ventricle Function~~	~~Categorical (Ordinal)~~	~~category~~
~~Renal Function~~	~~Categorical (Ordinal)~~	~~category~~
Presence of Pulmonary Hypertension	Categorical (Ordinal)	category
Surgeon	Categorical (Non-Ordinal)	category
Operation Room	Categorical (Non-Ordinal)	category
Operation Type	Categorical (Non-Ordinal)	category
Anesthesiologist	Categorical (Non-Ordinal)	category
Approach	Categorical (Non-Ordinal)	category
Part Of the Day	Categorical (Non-Ordinal)	category

D OPERATION TYPES

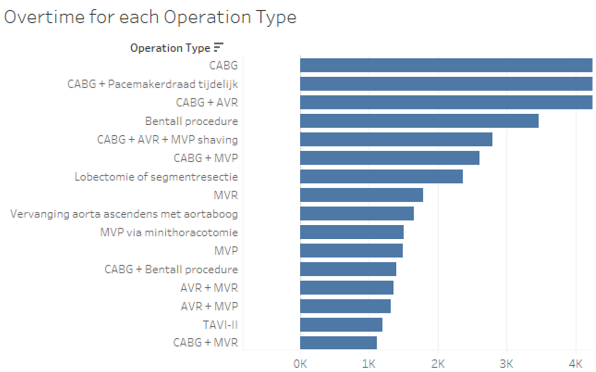

Figure 9: Operation Types that show the most overtime in the current prediction model

```
CABG
67
AVR
84
CABG + Pacemakerdraad tijdelijk
29
CABG + AVR
12
Wondtoilet
33

...
CABG + MVR + Vervanging aorta ascendens met aortaboog
1
CABG + Pacemakerdraad tijdelijk + ECMO
1
CABG + Plaatsen epicardiale electrode na openen pericard + uitwendige pacemaker + Ballonpomp, open
1
CABG + Pleurabiopsie.
1
Capsulotomie/capsulectomie met verwijderen mammaprothese na augmentatie + Thoraxwandreconstructie, Ravitch-procedure
1
Name: OperationType, Length: 360, dtype: int64
```

Figure 10: Output for: OperationTypes 'Unique Values'

The relevant operation types, that cover at least 99 percent of all operations, are: ['cabg', 'avr', 'pacemakerdraad tijdelijk', 'mvp', 'mvp shaving', 'wondtoilet', 'tvp', 'mvr'] and 'Other'.

Table 10: All the Operation types after splitting

amputatie teen
wondtoilet
arthroscopische acrominoclaviculaire reconstructie
habituele schouderluxatie
ascendensvervanging
asd
avp
avr
mvp shaving
klassieke aortabuisprothese onvertakt
tumor atrium
vervanging aortawortel
mvp
maze
tvp
mvr
ballonpomp, punctie
vervanging aorta ascendens met aortaboog
vervanging aortawortel, aorta ascendens en boog
ablatio mamma
aortareconstructie
vsd
pacemakerdraad tijdelijk
biventriculaire pacemaker
epicardiale lv-lead
lobectomie of segmentresectie
longbiopsie, open
mamma amputatie
morrow
wigresectie
aneurysma spurium
pvi
tumor mediastinum
reconstructie aortawortel
1 draads pacemaker
reven aorta ascendens
sluiten van een eenvoudig ventrikel-septum defect
plaatsen electrode(s)
totale thyreoidectomie
vervanging aorta ascendens
inbrengen lvad / bivad
vervanging aortaboog
verwijderen pacemaker of icd
avr via minithoracotomie
bentall procedure
boxlaesie
klepdragende vaatprothese
ecmo
perifere canulatie
ballonpomp, open
pvr
bilobectomie, open procedure
kwabresectie lever, open
bilobectomie, vats
vervangen pacemaker of icd
bullectomie met partiele pleurectomie
decorticatie long
bullectomie met partiële pleurectomie, vats
cabg

Table 11: (Continuation) All the Operation types after splitting

mvp shavingasd
pericardectomie (subtotaal)
ventrikelaneurysma
ballonpomp, punctiepacemakerdraad tijdelijk
coronaire anomalie
verwijderen corpus alienum
plaatsen epicardiale electrode na openen pericard
uitwendige pacemaker
pleurabiopsie.
refixatie sternum
rethoracotomie met hart-longmachine tijdens dezelfde opname
tavi-ii
cabg via minithoracotomie
cabgpacemakerdraad tijdelijk
capsulotomie/capsulectomie met verwijderen mammaprothese na augmentatie
thoraxwandreconstructie, ravitch-procedure
decorticatie long, vats
endoscopische decorticatie long
diagnostische pleurapunctie
drainage pericard
endoscopische bullectomie met partiele pleurectomie
endoscopische lobectomie of segmentresectie
endoscopische wigresectie
excisie biopsie mamma
endoscopische longbiopsie
endoscopische ok empyema thoracis
excisie aandoening thoraxwand, vats
grote en/of gecomplceerde huidtransplantatie
grote borstwandresectie in verband met een doorgegroeide maligniteit.
grote gecompliceerde transpositie gesteeld
hernia cicatricalis/ littekenbreuk
implanteren icd
inbrengen endocardiale electrode en bevestigen tweede electrode op het epicard, of bevestigen beide
inbrengen van stimulatie-electrode en aansluiten subc. geplaatste pacemaker
klassieke aortabroek prothese
mediane/ laterae clavicula resectie
klassieke aortabuisprothese met zijtak(ken)
klieven achilles peesschede (evt verlengen)
borstwandresectie
extrapleurale pneumolyse
proefthoracotomie
sleeve-resectie, open procedure.
nuss-procedure
wigresectie, vats
longbiopsie, vats
mamma ablatio
mediastinoscopie
operatieve behandeling van een empyema thoracis, open procedure.
turt en /of blaasbiopten
pvivsd
mvp via minithoracotomie
tvppvi
vats pvi
mvpventrikelaneurysma
mvr via minithoracotomie

Table 12: (Continuation) All the Operation types after splitting

nuss bar verwijderen
capsulotomie/capsulectomie met vervangen mammaprothese na augmentatie
ok empyema thoracis
open operatie van een of meerdere mediastinumtumoren, eventueel midsternaal.
openen hartzakje zonder hartingreep eventueel drainage van een pericarditis via een thoracotomie
hechten een of twee buigpezen
leverruptuur / abces / cysten
primair hechten grotere zenuw
operatie wegens een perforerende hartverwonding.
pericard drainage
operatieve behandeling van een empyema, vats.
operatieve verwijdering gezwellenravitch-procedure
partiele pericardresectie via thoracotomie.
pleurabiopsie.wigresectie
partiële pleurectomie
percardiectomie (subtotaal)
pneumonectomie, open procedure.
therapeutische pleurapunctie.
ventrikelseptumruptuur
pericard-fenestratie via vats.
reconstructie van de aorta of haar directe zijtakken zoals de arteria subclavia
pleurabiopsie, vats
pleurectomie, vats
pleurodese m.b.v. vats
pleurodese, open
pleuro-pneumonectomie, open procedure.
pneumonectomie
pneumonectomie met uitgebreide verwijdering lymfklieren, open procedure.
poging tot vats pvi
proef laparotomie
sluiten bronchusfistel via thoracotomie
ravitch-procedure
sternumfractuur
sluiten open thoraxverwonding.
staaldraden verwijderen
rethoracotomie
sleeve resectie
sleeve-resectie, vats.
sluiten bronchusfistel
sluiten fistel thoraxwand
staaldraden verwijderenwondtoilet
tavi-1
vervanging aortawortel aorta ascendens en boog
tumor ventrikel
vats boxlaesie
tvp shaving
vrije lap mond / pharynx / oesophagus

E OVERTIME

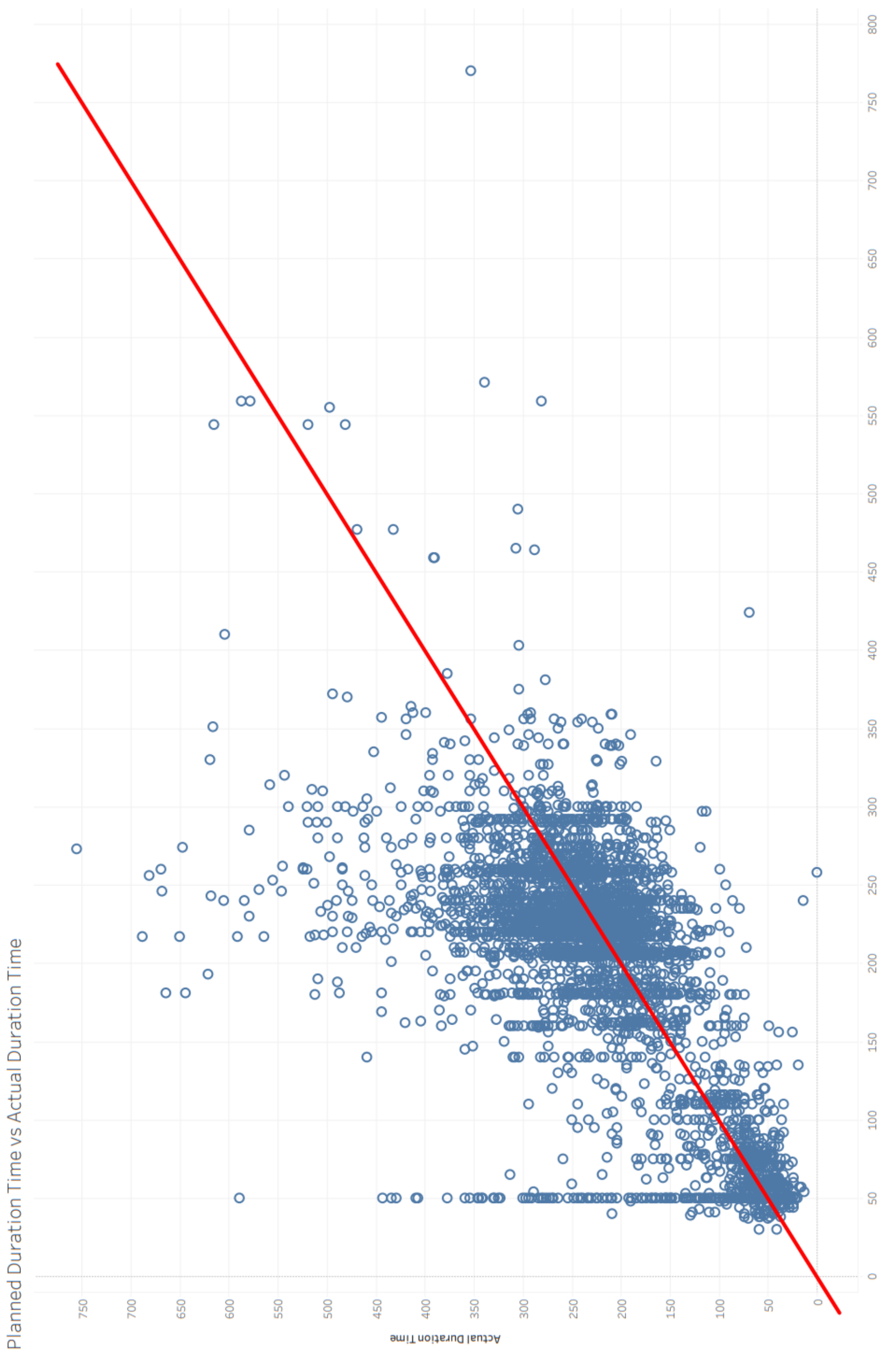

Figure 11: The Actual Duration Time Plotted against the current Planned Duration Time

F FULL RESULTS

Table 13: Scores and errors of the classifiers displayed as a heatmap for *ActualDurationTime*.

	MAE Error	RMSE Error
Original Prediction	**45.401**	**59.082**
Dummy classifiers		
Most Frequent	44.950	57.003
Prior	44.950	57.003
Stratified	61.293	76.394
Uniform	73.483	89.824
Dummy Regressor	43.616	54.959
Regression Models		
Linear Regression	35.918	45.910
Logistic Regression	41.517	53.062
SVM	42.513	54.535
Unsupervised Classifiers		
Decision Tree	41.186	51.907
Random Forest	40.062	50.970
K Neighbors	61.583	76.545
Supervised Classifiers		
Affinity Propagation	246.908	252.903
Birch	245.575	251.615
K means	245.521	251.567
Mini Batch K means	245.521	251.567
Gaussian Mixture	245.559	251.595
Information Retrieval Algorithm		
Inverted Indexes	45.752	58.877

Table 14: Scores and errors of the classifiers displayed as a heatmap for *ActualDurationTimeRange*.

	Accuracy Score	F1 Macro Score	F1 Weighted Score	MAE Error	RMSE Error
Original Prediction	**0.351**	**0.211**	**0.305**	**0.892**	**1.213**
Dummy Classifiers					
Most Frequent	0.373	0.109	0.203	0.920	1.257
Prior	0.373	0.109	0.203	0.920	1.257
Stratified	0.295	0.211	0.286	1.126	1.468
Uniform	0.182	0.164	0.202	1.505	1.849
Unsupervised Classifiers					
Decision Tree	0.417	0.199	0.307	0.784	1.101
Random Forest	0.445	0.260	0.365	0.733	1.054
K Neighbors	0.407	0.297	0.375	0.818	1.153
Supervised Classifiers					
Affinity Propagation	0.000	0.000	0.000	2.705	2.898
Birch	0.208	0.114	0.171	1.463	1.818
K means	0.208	0.110	0.174	1.445	1.793
Mini Batch K means	0.208	0.110	0.174	1.445	1.793
Gaussian Mixture	0.208	0.114	0.169	1.453	1.802
Information Retrieval Algorithm					
Inverted Indexes	0.407	0.319	0.377	0.814	1.153

G BOXPLOT COMPARISONS

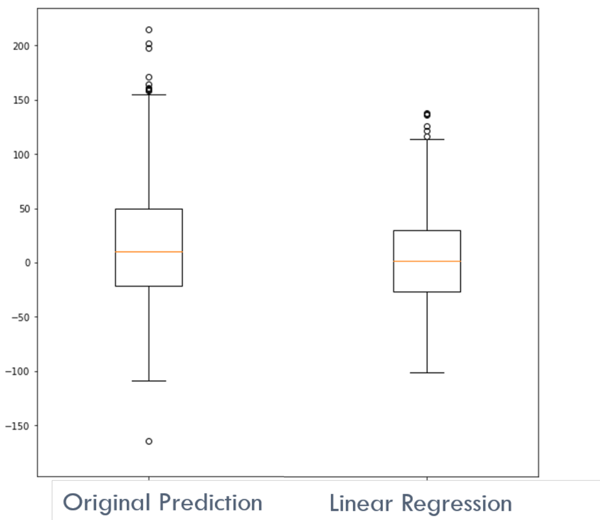

Figure 12: The comparison of deviation for *Overtime* between the original predictions and the predictions of linear regression for the continuous outcome data

Figure 13: The comparison of deviation for *Overtime* between the original predictions and the predictions of random forest for the categorical outcome data

H DURATION GENERATOR

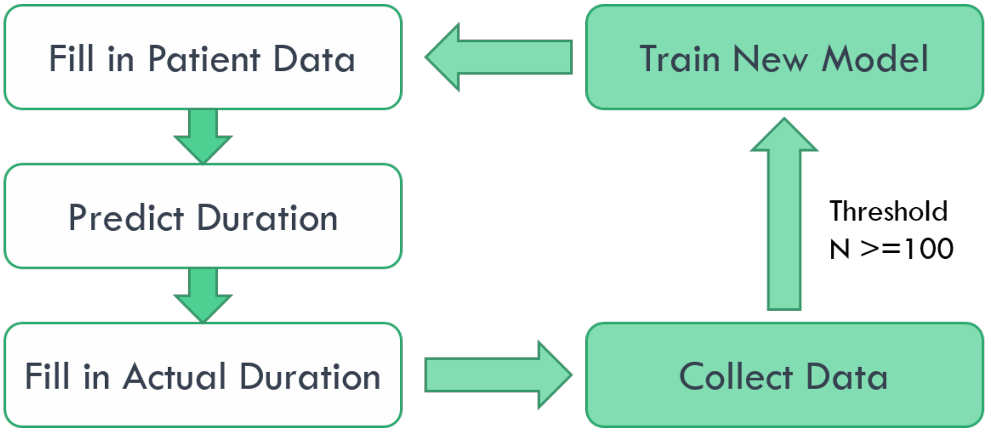

Figure 14: The structure of the *Duration Time Generator*

I BACKWARDS ELIMINATION

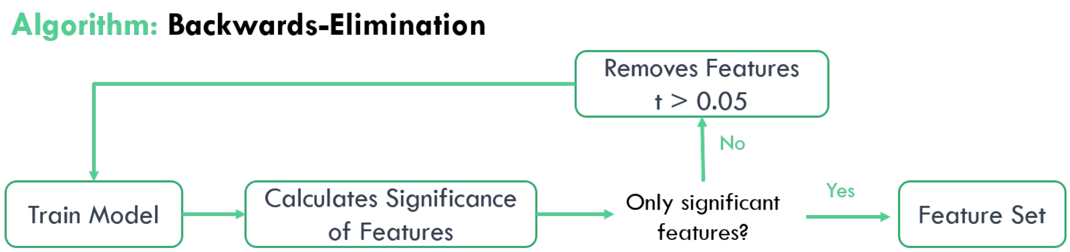

Figure 15: The algorithm structure of Bakwards Elminiation

Table 15: By Backward Elimination selected Features

Feature	Explanation
CCS	5 Point category that ranks the severeness of angina
NYHA	4-Point category that ranks the severity of symptoms
Previous Heart Surgery	Binary value, if the patient had a previous heart surgery
Aortic Surgery	Binary value, if surgery is planned on the aorta
Amount of Bypasses	The amount of bypasses planned for the surgery
Use of Cardiopulmonary Bypasses	Binary value, if a heart-lung machine will be used
Presence of Hypertension	Binary value, if deviations of blood pressure are present
Presence of Pulmonale Hypertension	categorical, if elevated blood pressure is present
Presence of Active Endocarditis	Binary, value, if patient is still on treatment for endocarditis
Artrial Fibrillation	Binary value, if the normal sinus rhythm of the heart is present
Operation Types [9]	Planned Operation Types for the Surgery. Full list: other, cabg, avr, pacemakerdraad tijdelijk, mvp, mvp shaving, wondtoilet, tvp, mvr

J THE DATA VISUALIZATION CODE

(starting on next page)

1 DPV

Group 92

Sara-Jane Bittner

Xiao-Lan Bokma

[ ]:
```
#imports

#dpv_part
from sqlalchemy import create_engine # needed for DB connection
import tqdm


import os #get the os filepath
import pandas as pd
# import matplotlib as mpb
# import sklearn as sk
import numpy as np
from numpy import mean, std , nan

from collections import Counter
import seaborn as sb
import statsmodels.api as sm
import scipy.stats as stats
# import pyreadstat
import matplotlib.pyplot as plt
from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn import metrics
from sklearn.metrics import accuracy_score, mean_squared_error,␣
↪mean_absolute_error
from sklearn.metrics import f1_score, recall_score, average_precision_score,␣
↪balanced_accuracy_score
from sklearn import linear_model, model_selection, svm
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn import tree

from nltk.probability import FreqDist
from operator import itemgetter
from pprint import pprint
```

[ ]:
```
pip install psycopg2-binary
```

```
[ ]: #Generate a path of the folder and the file
path = os.path.join('data','surgical_case_durations.csv')
#print(path)
```

```
[ ]: #Read the csv file and print the first five rows of the data
surgicalData=pd.read_csv(path,sep=';',encoding="ISO-8859-1")

#delete last row as it is empty
surgicalData.drop(surgicalData.tail(1).index,inplace=True)
#surgical.head()
```

1.0.1 1.1. Renaming the columns

From Dutch to English equivalents

```
[ ]: #Renaming the columns to english equivalents
surgicalData = surgicalData.rename(columns={"Geplande operatieduur":␣
 ↪"PlannedDurationTime",
                                    "Operatieduur": "ActualDurationTime",
                                    "Operatietype":"OperationType",
                                    "Chirurg":"Surgeon",
                                    "Anesthesioloog":"Anesthesiologist",
                                    "Benadering":"Approach",
                                    "OK":"OperationRoom",
                                    "Casustype":"Urgency",
                                    "Dagdeel":"PartOfDay",
                                    "Leeftijd":"Age",
                                    "AF":"AtrialFibrillation",
                                    "HLM":"CardiopulmonaryBypassUse",
                                    "Geslacht":"Sex",
                                    "Aantal anastomosen":"AmountOfBypasses",
                                    "Chronische longziekte":
 ↪"P_ChronicLungDisease",
                                    "Extracardiale vaatpathie":
 ↪"P_extracardialArteriopathy",
                                    "Actieve endocarditis":␣
 ↪"P_activeEndocarditis",
                                    "Hypertensie":"P_Hypertension",
                                    "Pulmonale hypertensie":
 ↪"P_PulmonaleHypertension",
                                    "Slechte mobiliteit":"P_PoorMobility",
                                    "Hypercholesterolemie":
 ↪"P_Hypercholesterolemia",
                                    "Perifeer vaatlijden":
 ↪"P_PeripherialVascularDisease",
                                    "Linker ventrikel functie":
 ↪"LeftVentricleFunction",
```

```
                                    "Nierfunctie":"RenalFunction",
                                    "DM":"P_Diabetis",
                                    "Eerdere hartchirurgie":
↪"PerviousHeartSurgery",
                                    "Kritische preoperatieve status":
↪"CriticalPre-OP",
                                    "Myocard infact <90 dagen":
↪"MycordialInfarctionPreSurgery",
                                    "Aorta chirurgie":"AorticSurgery",
                                    "Ziekenhuis ligduur":"HospitalDays",
                                    "IC ligduur":"ICUDays",

                                   })
```

[ ]:
```
patientFeatures = ['Urgency', 'Sex','Age',
       'AtrialFibrillation', 'P_ChronicLungDisease',
       'P_extracardialArteriopathy', 'PerviousHeartSurgery',
       'P_activeEndocarditis', 'CriticalPre-OP',
       'MycordialInfarctionPreSurgery',
       'P_PulmonaleHypertension', 'LeftVentricleFunction', 'Euroscore1',
       'Euroscore2', 'RenalFunction', 'P_PoorMobility', 'P_Diabetis',
       'P_Hypercholesterolemia', 'P_Hypertension',
       'P_PeripherialVascularDisease', 'CCS', 'NYHA'
       ]

surgeryFeatures = ['Surgeon', 'Anesthesiologist', 'Approach',
       'OperationRoom','PartOfDay', 'AorticSurgery', 'AmountOfBypasses',␣
↪'CardiopulmonaryBypassUse', 'OperationType']
```

1.1 Preprocessing for Visualization

[ ]:
```
def replace_with_nan(dataFrame, value):
    """
    Replace specific a value in this dataframe to a np.nan value.

    @type dataFrame: dataframe
    @param dataFrame: The dataframe with value that needs replacement.
    @type value: string
    @param value: The value that needs to be replaced by np.nan values
    @rtype: dataframe
    @returns: dataFrame with the values replaced by np.nan values.
    """
    #loop through all the columns in the data and replace the value within each␣
↪column.
    for column in dataFrame:
        dataFrame[column] = dataFrame[column].replace([value], nan)
```

```
    return dataFrame
```

```
[ ]: def get_non_dtype_columns(dataFrame, dTypeColumnNames, dType):
        """
        Get the columns of the dType list that are not yet of that dtype (yet):

        @type dataFrame: dataframe
        @param dataFrame: The dataframe to get the non_dTypeColumnNames columns from.
        @type dTypeColumnNames: list
        @param dTypeColumnNames: list of the column names that should be of the␣
     ↪specified dType.
        @type dType: string
        @param dType: the specific dType that the columns should be

        @rtype: list
        @returns: non_dTypeColumns which are not of the specific dtype (yet).
        """
        non_dTypeColumnNames = dataFrame[dTypeColumnNames].
     ↪select_dtypes(exclude=[dType]).columns
    #     print("nondtypecolumnnames", non_dTypeColumnNames)
        return non_dTypeColumnNames

    # def decimal_separator_to_period (dataFrame, non_numericalColumnNames):
    #     for columnName in non_numericalColumnNames:
    #         dataFrame[columnName].str.replace(',', '.')

    def convert_df_type(dataFrame, columns, dType):
        """
        Convert the dtype of specific columns in the dataFrame to the specified␣
     ↪dType.

        @type dataFrame: dataframe
        @param dataFrame: The dataframe with columns that need to be converted to␣
     ↪the dType.
        @type columns: list
        @param columns: list of the columns that should be the specified dType.
        @type dType: string
        @param dType: the dtype that the columns should be
        @rtype: dataframe
        @returns: dataFrame with the columns 'columns' to dtype 'dType'
        """
    #     print("dTypes before: ",dataFrame.dtypes)

        if (dType == 'float64'):
            dataFrame = replace_with_nan (dataFrame, value = 'Onbekend')
            non_dTypeColumnNames = get_non_dtype_columns(dataFrame, columns, dType)
```

```
        for columnName in non_dTypeColumnNames:
            dataFrame[columnName] = dataFrame[columnName].str.replace(',', '.').
 ↪astype(float)

    elif (dType == 'category'):
        dataFrame = pd.concat([
        dataFrame.select_dtypes([], ['object']),
        dataFrame.select_dtypes(['object']).apply(pd.Series.astype, dtype=dType)
        ], axis=1)
#     print("dTypes after: ", dataFrame.dtypes)
    return dataFrame
```

[ ]:
```
surgicalData = replace_with_nan(surgicalData, 'Onbekend')
surgicalData = convert_df_type(surgicalData, ['ICUDays', 'HospitalDays'],␣
 ↪'float64')
```

1.2 Patient Table

- Assumption: every row is a distinct patient

[ ]:
```
#Creating the patient table
patient = surgicalData[patientFeatures]

#Re-indexing patient table
patient['patientID'] = patient.reset_index().index

#Changing order in table patient
patient = patient[['patientID', 'Urgency', 'Sex','Age',
       'AtrialFibrillation', 'P_ChronicLungDisease',
       'P_extracardialArteriopathy', 'PerviousHeartSurgery',
       'P_activeEndocarditis', 'CriticalPre-OP',
       'MycordialInfarctionPreSurgery',
       'P_PulmonaleHypertension', 'LeftVentricleFunction', 'Euroscore1',
       'Euroscore2', 'RenalFunction', 'P_PoorMobility', 'P_Diabetis',
       'P_Hypercholesterolemia', 'P_Hypertension',
       'P_PeripherialVascularDisease', 'CCS', 'NYHA'
       ]]
```

1.3 Surgery Table

[ ]:
```
#Creating the surgery table
surgery = surgicalData[surgeryFeatures]

#Re-indexing surgery table
surgery['surgeryID'] = surgery.reset_index().index

#Changing order in table surgery
```

```
surgery = surgery[['surgeryID', 'Surgeon', 'Anesthesiologist', 'Approach',
       'OperationRoom','PartOfDay', 'AorticSurgery', 'AmountOfBypasses',␣
 ↪'CardiopulmonaryBypassUse', 'OperationType']]
```

2 Overtime

[ ]:
```
#Creating the Overtime Table
OvertimeDf = pd.DataFrame()

#Calculating Overtime
OvertimeDf['NumOvertime'] = surgicalData['ActualDurationTime'] -␣
 ↪surgicalData['PlannedDurationTime'][:]
```

[ ]:
```
OvertimeSpecifics = pd.DataFrame(np.nan,index=range(0,4086),␣
 ↪columns=['GeneralOvertime','PosOvertime','NegOvertime'])



#replace All Values within the range -10 and 10 discrepancy from the planned␣
 ↪surgery time with NoOvertime
OvertimeSpecifics['GeneralOvertime']=np.where(OvertimeDf['NumOvertime'].
 ↪between(-10,10), 'NoOvertime', OvertimeSpecifics['GeneralOvertime'])

#Set positive and negative Overtime
OvertimeSpecifics['PosOvertime'] = np.where(OvertimeDf['NumOvertime'] <=10,␣
 ↪'NoOvertime', OvertimeSpecifics['PosOvertime'])
OvertimeSpecifics['NegOvertime'] = np.where(OvertimeDf['NumOvertime'] >= (-10),␣
 ↪'NoOvertime', OvertimeSpecifics['NegOvertime'])



#Set positive and negative NoOvertime
OvertimeSpecifics['PosOvertime'] = np.where(OvertimeDf['NumOvertime'] >10,␣
 ↪'Overtime', OvertimeSpecifics['PosOvertime'])
OvertimeSpecifics['NegOvertime'] = np.where(OvertimeDf['NumOvertime'] < (-10),␣
 ↪'Overtime', OvertimeSpecifics['NegOvertime'])

#Set NoOvertime for Overtime
OvertimeSpecifics['GeneralOvertime'] = np.where(OvertimeDf['NumOvertime'] >10,␣
 ↪'Overtime', OvertimeSpecifics['GeneralOvertime'])
OvertimeSpecifics['GeneralOvertime'] = np.where(OvertimeDf['NumOvertime'] <␣
 ↪(-10), 'Overtime', OvertimeSpecifics['GeneralOvertime'])
```

[ ]:
```
#Concat various overtime columns
OvertimeDf =pd.concat([OvertimeSpecifics, OvertimeDf], axis=1)
OvertimeDf
```

[ ]:
```
#add overtime columns to surgery table
surgery =pd.concat([surgery, OvertimeDf], axis=1)
```

3 HospitalTable

[ ]:
```
#create hospital table
hospital =␣
↪surgicalData[['PlannedDurationTime','ActualDurationTime','ICUDays','HospitalDays']]

# Adding patient and surgery ID to to the other tables
hospital = pd.concat([hospital, surgery['surgeryID']], axis=1)
hospital = pd.concat([hospital, patient['patientID']], axis=1)
display(hospital)
```

4 DataBase

[ ]:
```
#first create link to database
# Replace username with the user name password with the password
driver='postgresql'
username='dab_ds21221b_65'
dbname=username # it is always same as username
password='LsUHCtQF77KuFAMI'
server='bronto.ewi.utwente.nl'
port='5432'
# Creating the connetcion pool for SQL
engine = create_engine(driver+'://' +username+':'+password+'@'+server+':'+port+'/
↪'+dbname)

#create tables
patient.to_sql('patient', engine,schema='assProject', if_exists='append',␣
↪index=False)
surgery.to_sql('surgery', engine,schema='assProject', if_exists='append',␣
↪index=False)
hospital.to_sql('hospital', engine,schema='assProject', if_exists='append',␣
↪index=False)
```

K THE MAIN CODE: ACTUAL DURATION TIME RANGE

(starting on next page)

1 Project: Data Science - Surgical Duration: Actual Duration Time

Group 92

Sara-Jane Bittner

Xiao-Lan Bokma

2 Chapter 1 - Setup

[1]:
```
#imports
import os #get the os filepath
import pandas as pd
from collections import Counter

import statsmodels.api as sm
from scipy.stats import zscore

from matplotlib.pyplot import boxplot
from statsmodels.stats.outliers_influence import variance_inflation_factor

from numpy import mean, std, nan, float64, abs, delete, absolute, sqrt, array

from sklearn.preprocessing import MinMaxScaler, LabelEncoder,␣
 ↪OneHotEncoder,LabelBinarizer
from sklearn.metrics import accuracy_score, mean_squared_error,␣
 ↪mean_absolute_error
from sklearn.metrics import f1_score, recall_score, average_precision_score,␣
 ↪balanced_accuracy_score
from sklearn import model_selection, svm, tree
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.dummy import DummyClassifier, DummyRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.cluster import AffinityPropagation, KMeans, MiniBatchKMeans, Birch
from sklearn.mixture import GaussianMixture
from sklearn.neighbors import KNeighborsClassifier

from nltk.probability import FreqDist
from operator import itemgetter
from pprint import pprint
```

[2]:
```
#This will allow our project to allways be executed in the same manor, providing␣
 ↪us with a stable result.
randomSeed=12345
```

3 Chapter 2 - Used functions

The following will provide us with all the functions, that are used in our pipeline in chapter 3

3.0.1 2.1 Import the data

[3]:
```
def import_df_from_file(filename: str):
    """
    Get the path of this file and look into the data folder that is located in␣
 ↪the same folder as this file,
    read the .csv file and delete the last row, as it is empty

    @type filename: str
    @param filename: The filename of the .csv spreadsheet
    @rtype: dataframe
    @returns: dataFrame appearing in the .csv file
    """

    path = os.path.join('data', filename)
    dataFrame=pd.read_csv(path,sep=';',encoding="ISO-8859-1")
    dataFrame.drop(dataFrame.tail(1).index,inplace=True)
    display(dataFrame)
    return dataFrame
```

3.1 2.2 - Preprocessing

3.1.1 2.2.1 Retype the columns

Currently, all the dataframe columns are of type "Object", since the data is in string format. All the columns that are not in this shape, will need to be retyped.

The columns that should be float64 are predefined already. The columns that are not recognized as float64 yet will undergo the modification. To retype the float64 columns, the string 'Onbekend' needs to be replaced with np.nan values.

[4]:
```
def replace_with_nan(dataFrame, value):
    """
    Replace specific a value in this dataframe to a np.nan value.

    @type dataFrame: dataframe
    @param dataFrame: The dataframe with value that needs replacement.
    @type value: string
    @param value: The value that needs to be replaced by np.nan values
    @rtype: dataframe
    @returns: dataFrame with the values replaced by np.nan values.
    """
    #loop through all the columns in the data and replace the value within each␣
 ↪column.
```

```
    for column in dataFrame:
        dataFrame[column] = dataFrame[column].replace([value], nan)
    return dataFrame

def get_non_dtype_columns(dataFrame, dTypeColumnNames, dType):
    """
    Get the columns of the dType list that are not yet of that dtype (yet):

    @type dataFrame: dataframe
    @param dataFrame: The dataframe to get the non_dTypeColumnNames columns from.
    @type dTypeColumnNames: list
    @param dTypeColumnNames: list of the column names that should be of the␣
↪specified dType.
    @type dType: string
    @param dType: the specific dType that the columns should be

    @rtype: list
    @returns: non_dTypeColumns which are not of the specific dtype (yet).
    """
    non_dTypeColumnNames = dataFrame[dTypeColumnNames].
↪select_dtypes(exclude=[dType]).columns
    return non_dTypeColumnNames

def convert_df_type(dataFrame, columns, dType):
    """
    Convert the dtype of specific columns in the dataFrame to the specified␣
↪dType.

    @type dataFrame: dataframe
    @param dataFrame: The dataframe with columns that need to be converted to␣
↪the dType.
    @type columns: list
    @param columns: list of the columns that should be the specified dType.
    @type dType: string
    @param dType: the dtype that the columns should be
    @rtype: dataframe
    @returns: dataFrame with the columns 'columns' to dtype 'dType'
    """

    if (dType == 'float64'):

        print(columns)
        non_dTypeColumnNames = get_non_dtype_columns(dataFrame, columns, dType)

        #change the columns that should be float64 but aren't yet.
        # if the float value cannot be determined due to the decimal
        # separator, commas will be resplaced by periods.
```

```
        for columnName in non_dTypeColumnNames:
            if dataFrame[columnName].dtype=="int8" or dataFrame[columnName].
↪dtype=="int64":
                dataFrame[columnName] = dataFrame[columnName].astype(float)
            else:
                dataFrame[columnName] = dataFrame[columnName].str.replace(',', '.
↪').astype(float)

    #retype all the object columns to category type
    elif (dType == 'category'):
        dataFrame = pd.concat([
        dataFrame.select_dtypes([], ['object']),
        dataFrame.select_dtypes(['object']).apply(pd.Series.astype, dtype=dType)
        ], axis=1)

    #convert the columns that are categories into intergers,
    # based on the provided list of columns
    elif (dType == 'int8'):
        for columnName in columns:
            dataFrame[columnName] = dataFrame[columnName].astype('category').cat.
↪codes
    return dataFrame
```

3.1.2 2.2.2 Remove the Rows and Columns with too much NAN

Drop columns that have more than 50% nan values. Drop the rows that will have one or more missing cells.

[5]:
```
def dropWithTooMuchNan(dataFrame, axis):
    """
    Drop the row/column that contains too many Nan-Values.

    @type dataFrame: dataframe
    @param dataFrame: The dataframe that should have Nan rows/columns removed.
    @param axis: integer
    @param axis: 0 is about the row axis, 1 is the column axis.
    @rtype: dataframe
    @returns: dataFrame with the columns 'columns' to dtype 'dType'
    """

    if axis == COLUMN_AXIS:
        allColumnNames = dataFrame.columns
        NANColumns =[]
        #drop the columns that have missing values
        for columnName in allColumnNames:
            if dataFrame[columnName].isnull().values.any():
                howManyNaN = dataFrame[columnName].isnull().sum()
```

```
                if howManyNaN > ((len(dataFrame[columnName])/100)*50):
                    NANColumns += [columnName]
                    dataFrame.drop(labels=columnName,axis=COLUMN_AXIS,␣
↪inplace= True)
            return dataFrame, NANColumns

    elif axis == ROW_AXIS:
        # drop the rows that have missing values.
        dataFrame.dropna(thresh= len(dataFrame.columns), inplace = True)
        dataFrame.isnull().values.any()
        dataFrame = dataFrame.reset_index(drop=True)

    return dataFrame
```

3.1.3 2.2.3 Creation of new columns

Creating the Overtime column based on the current planned Duration time.

[6]:
```
def createOvertimeColumn(dataFrame):
    """
    Create the Overtime column by substracting ActualDurationTime from the␣
↪PlannedDurationTime

    @type dataFrame: dataframe
    @param dataFrame: The dataframe that containt the two columns that should␣
↪become a new column.
    @rtype: dataframe
    @returns: dataFrame with the extra "Overtime" column.
    """
    dataFrame['Overtime'] = dataFrame['ActualDurationTime'] -␣
↪dataFrame['PlannedDurationTime']
    RANGE_COLUMN_NAMES.append('Overtime')
    ORDINAL_COLUMN_NAMES.append('Overtime')
    return dataFrame
```

Create the OperationType Columns

[7]:
```
#splitting the OperationType Columns-String by the '+' sign.
class OPTypeEncoder:
    relevant_op_types=["other"] #list of the n% most common operations
    relevant_op_types_Names=["OPType_other"]
    n=99

    def __init__(self, n):
        '''main'''
        self.n=n

    def splitOPTypeText(self,text):
```

```
        """
        Split the string by the '+' sign.

        @type text: string
        @param text: the string that needs to be split
        @rtype: list
        @returns: list that contains all the separate operationTypes.
        """
        thisPatientsOPs=str(text).lower().strip().split(" + ")
        return [i.strip() for i in thisPatientsOPs]

    def fit(self,dataframe, columnname):
        """
        Create a list of relevant operationtypes. 99% are included,
            the other 1% is put in the category called 'other'

        @type dataFrame: dataframe
        @param dataFrame: The dataframe that contains the column with strings␣
↪that
                            need to be split.
        @type columnName: string
        @param columnName: the name of the column that contains the strings that
                            need to be split.
        @rtype: list
        @returns: relevant names that appear outside the 1% marge.

        """

        #get a list of all operations, that have ever been done
        completeOPList=[]
        for patient in dataframe[columnname]:
            #print(patient)
            thisPatientsOPs=self.splitOPTypeText(patient)
            completeOPList+=thisPatientsOPs

        #find out how often they have been done
        to_n_uses=FreqDist(completeOPList)
        dictionary_items = to_n_uses.items()
        to_n_uses = sorted(dictionary_items, key=itemgetter(1), reverse=True )

        #get the operations, that together are used in n% percent of the cases
        threshold=(self.n*len(dataframe[columnname]))/100 # Define how many␣
↪cases should be covered by the amount of operations.
        i=0
        for ot in to_n_uses:
            #print(ot)
            if ot[0] == "nan":
```

```
                pass
            else:
                i+=ot[1]
                self.relevant_op_types.append((ot[0]))
                if i>=threshold:
                    break
        print("The relevant op types, that cover at least",str(self.n)+"% of all␣
↪operations, are:", self.relevant_op_types)
        self.relevant_op_types_Names=["OPType_"+i for i in self.
↪relevant_op_types]


    def transform(self,dataframe, columnname):
        """
        Transform the current shape of the dataframe column.
        Add J if this operation was done for this patient,
        Add N if this operation was not done for this patient
        Add nan, if the operation was unknown.



        @type dataFrame: dataframe
        @param dataFrame: The dataframe that contains the column with strings␣
↪that
                            need to be reshaped.
        @type columnName: string
        @param columnName: the name of the column that contains the strings that
                            need to be reshaped.
        @rtype: dataframe
        @returns: dataFrame with the newly shaped OperationType column
        """

        allPatientsOHE=[]

        #add J if this operation was done for this patient,
        #add N if this operation was not done for this patient
        #add nan, if the operation was unknown
        for patient in dataframe[columnname]:
            #print(patient)
            thisPatientsOPs=self.splitOPTypeText(patient)

            #add a column for each relevant operationtype
            thisPatientsOHE=["N"]*len(self.relevant_op_types)
            for op in thisPatientsOPs:
                if op =="nan":
                    thisPatientsOHE=[nan]*len(self.relevant_op_types)
                elif op in self.relevant_op_types:
                    i=self.relevant_op_types.index(op)
```

```
                thisPatientsOHE[i]="J"
            elif not(op in self.relevant_op_types):
                thisPatientsOHE[0]="J"
        allPatientsOHE.append(thisPatientsOHE)

    #turn the results into a dataframe
    allPatientsOHE_DF=pd.DataFrame(allPatientsOHE, columns= self.
↪relevant_op_types_Names)

    #remove old column from df
    dataframe = dataframe.drop(labels = columnname, axis=COLUMN_AXIS)

    #add new columns to df
    dataframe = pd.merge(dataframe, allPatientsOHE_DF, left_index=True,␣
↪right_index=True)

    #edit
    CATEGORICAL_COLUMN_NAMES.remove(columnname)
    SURGERY_FEATURES.remove(columnname)
    for column in self.relevant_op_types_Names:
        BINARY_COLUMN_NAMES.append(column)
        SURGERY_FEATURES.append(column)


    return dataframe

def fit_transform(self,dataframe, columnname):
    """
    Transform the current shape of the dataframe column.
    Add J if this operation was done for this patient,
    Add N if this operation was not done for this patient
    Add nan, if the operation was unknown.


    @type dataFrame: dataframe
    @param dataFrame: The dataframe that contains the column with strings␣
↪that
                    need to be reshaped.
    @type columnName: string
    @param columnName: the name of the column that contains the strings that
                    need to be reshaped.
    @rtype: dataframe
    @returns: dataFrame with the newly shaped OperationType column
    """
    self.fit(dataframe, columnname)
    return self.transform(dataframe, columnname)
```

3.1.4 2.3.4 Outliers

Categorical Outlier Grouping

[8]:
```python
def groupOutliers(dataFrame, columnName, treshhold):
    '''
    Group the outlier categories into one group, called 'other'.

    @type dataFrame: dataframe
    @param dataFrame: the dataframe with the categorical column
    @type columnNames: list
    @param columnNames: list of the column names that need to be outlier cut
    @type treshold: float
    @param treshold: percentage that needs to be cut off.

    @rtype dataFrame: dataframe
    @returns dataFram: dataframe with the cut off outliers based on the treshold.
    '''
    treshAmount = (len(dataFrame)//100)*treshhold
    threshCounter = 0
    countValues = dataFrame[columnName].value_counts()
    uniqueValues = dataFrame[columnName].unique()

    for uniqueValue in reversed(uniqueValues):
        if (threshCounter + countValues[uniqueValue]) <= treshAmount:
            threshCounter += countValues[uniqueValue]
            dataFrame[columnName].replace([uniqueValue],'other', inplace=True)

    dataFrame[columnName] = dataFrame[columnName].cat.remove_unused_categories()
    return dataFrame

def categorical_outlier_cutting(dataFrame, columnNames, treshold):
    '''
    Loop through the columnNames that needs to be outliercut.

    @type dataFrame: dataframe
    @param dataFrame: the dataframe with the categorical column
    @type columnNames: list
    @param columnNames: list of the column names that need to be outlier cut
    @type treshold: float
    @param treshold: percentage that needs to be cut off.

    @rtype dataFrame: dataframe
    @returns dataFram: dataframe with the cut off outliers based on the treshold.
    '''
    for column in columnNames:
        dataFrame=groupOutliers(dataFrame, column, treshold)
    return dataFrame
```

Numerical Outlier Cutting

[9]:
```
def numerical_outlier_cutting(dataFrame, numericalColumnNames):
    '''
    Define the numerical outliers, based on the z-scores. Based on the range 2,␣
↪the outliers will be cut.

    @type dataFrame: dataFrame
    @param dataFrame: the datafram that contains the columns that need to have␣
↪the values outliercut.
    @type numericalColumnNames: list
    @param numericalColumnNames: list of the columnNames with numerical data.

    @rtype dataFrame: dataframe
    @returns dataFrame: dataframe with the numerical outliers cut.
    '''
    numericalData = dataFrame[numericalColumnNames]

    z_scores = zscore(numericalData)
    abs_z_scores = abs(z_scores)
    filtered_entries = (abs_z_scores < 2).all(axis=1)
    dataFrame = dataFrame[filtered_entries]
    dataFrame = dataFrame.reset_index(drop=True)
    return dataFrame
```

3.1.5 2.2.5 Creating categorical data from numerical data ranges.

[10]:
```
def get_bins (columnData, columnName):
    """
    Get the bin values and the according names for those bins.

    @type columnData: list
    @param columnData: The list of values appearing in that column.
    @type columnName: string
    @param columnName: the name of the column that contains the strings that
                        needs to have the bins and the name of the bins.
    @rtype: list (floats), list (strings)
    @returns: the list of 'bins' (values) for the specific columnName and the
                titles that the bins should get.
    """
    minValue = columnData.min()
    maxValue = columnData.max()

    if columnName == "BMI":
        bins = [(minValue-1),18.5,24.9,29.9,maxValue]
        binNames = ['underweight', 'normal', 'overweight', 'obese']
    elif columnName == "Age":
        bins = [int(minValue-1), 57, 66,76, int(maxValue)]
```

```
        binNames = [ (str(x+1)+'-'+str(y)) for x,y in zip(bins[0::1], bins[1::
↪1]) ]
    elif columnName == "Overtime":
        bins = [(minValue-1),(-70),(-25),25,70, 120,(maxValue)]
        binNames = ['Ahead_Of_Time_Extreme','Ahead_Of_Time_Medium',␣
↪'In_Time','Overtime_Small','Overtime_Medium', 'Overtime_Extreme']
    elif columnName == "ActualDurationTime":
        bins = [int(minValue-1), 180, 240, 280, 345,int(maxValue)]
        binNames = [ (str(x+1)+'-'+str(y)) for x,y in zip(bins[0::1], bins[1::
↪1]) ]
    elif columnName == "PlannedDurationTime":
        bins = [int(minValue-1), 180, 240, 280, int(maxValue)]
        binNames = [ (str(x+1)+'-'+str(y)) for x,y in zip(bins[0::1], bins[1::
↪1]) ]
    return bins, binNames

def convert_categorical_range(dataFrame, columnName):
    """
    Convert the current columnName with float data to a columnName + 'Range'
        and the data in bins.


    @type dataFrame: dataframe
    @param dataFrame: The dataframe that contains the column with floatdata
                      that needs to be binned.
    @type columnName: string
    @param columnName: the name of the column that needs to have the data binned.
    @rtype: dataframe, list (string)
    @returns: dataFrame with the column that needs to be binned.
              list with the bin names in strings.
    """

    #get the specific binvalues and the binnames
    bins, binNames = get_bins(dataFrame[columnName], columnName)

    #put the data in bins with according binvalues and the binnames.
    #call the new column the same + 'range'.
    dataFrame[columnName + 'Range'] = pd.cut(dataFrame[columnName], bins,␣
↪labels=binNames)

    #if the column was not the outcome data, then drop it.
    #outcomedata will be kept within the dataset.
    if columnName is not 'PlannedDurationTime' and columnName is not␣
↪'ActualDurationTime':
        dataFrame.drop(labels = columnName, axis=COLUMN_AXIS, inplace = True)
```

```
    #print the values appearing in the new column, bin values, length of the␣
↪binvalues,
    #the bin names and the length of the bin names.
    print(dataFrame[columnName + 'Range'].value_counts())
    print (bins)
    print(len(bins))
    print(binNames)
    print(len(binNames))

    return dataFrame, binNames

def to_categorical_range(dataFrame):
    """
    Loop though the list of names that should be binned.

    @type dataFrame: dataFrame
    @param dataFrame: the dataFrame that will contain the columns that
                        needs to be converted from floats to ranged data.

    @rtype dataFrame, column2binNames: dataFrame, dictionary
    @return dataFrame, column2binNames: dataFrame with the specific float␣
↪columns converted to ranged data.
                                            column2binNames is a dictionary,␣
↪with columnName as key and the
                                            bin names as values.
    """
    column2binNames={}
    for columnName in RANGE_COLUMN_NAMES:
        dataFrame, binNames = convert_categorical_range(dataFrame, columnName)
        column2binNames[columnName+"Range"]=binNames
    return dataFrame, column2binNames
```

```
<>:59: SyntaxWarning: "is not" with a literal. Did you mean "!="?
<>:59: SyntaxWarning: "is not" with a literal. Did you mean "!="?
<>:59: SyntaxWarning: "is not" with a literal. Did you mean "!="?
<>:59: SyntaxWarning: "is not" with a literal. Did you mean "!="?
<ipython-input-10-8fc6ea43bdb3>:59: SyntaxWarning: "is not" with a literal. Did
you mean "!="?
  if columnName is not 'PlannedDurationTime' and columnName is not
'ActualDurationTime':
<ipython-input-10-8fc6ea43bdb3>:59: SyntaxWarning: "is not" with a literal. Did
you mean "!="?
  if columnName is not 'PlannedDurationTime' and columnName is not
'ActualDurationTime':
```

3.1.6 2.2.6 Encoding

Label Encoding

[11]:
```python
def label_encoding(dataFrame, columns):
    """
    Convert the ORDINAL categorical columns into label encoded columns.

    @type dataFrame: dataframe
    @param dataFrame: The dataframe that contains ORDINAL categorical data.
    @type columns: list
    @param columns: list of column names that contain ORDINAL categorical data.
    @rtype dataFrame, columnNameToLE: dataframe, dictionary
    @returns dataFrame, columnNameToLE: new dataframe with the ordinal␣
 ↪categorical columns label encoded.
                            dictionary with column name as key and the category␣
 ↪names as values.
    """

    columnNameToLE={}

    #loop through the columns that needs to be label encoded.
    for columnName in columns:
        le = LabelEncoder()
        le = le.fit(dataFrame[columnName])
        dataFrame[columnName] = le.transform(dataFrame[columnName])
        columnNameToLE[columnName]=le

    return dataFrame, columnNameToLE
```

One Hot Encoding

[12]:
```python
def IntTransfer(dataFrame, columnName):
    '''
    Make string type floats into int.
    When the decimal separator is ',' replace it with a '.' first.

    @type dataFrame: dataframe
    @param dataFrame:
    @type columnName: string
    @param columnName:

    @rtype dataFrame: dataframe
    @returns dataFrame:

    '''
    for i in range(0,len(dataFrame[columnName])):
        try:
            dataFrame[columnName][i]= int(float(dataFrame[columnName][i].
 ↪replace(",",".")))
        except ValueError:
```

```python
            dataFrame[columnName][i]= dataFrame[columnName][i]

    return dataFrame


def one_hot_encoding(columnNames, dataFrame):
    '''
    Onehot-encodes NOMINAL categorical columns of the dataFrame, which are␣
↪listed in columnNames

    @type dataFrame: dataframe
    @param dataFrame: dataframe that contains the data that needs to be onehot␣
↪encoded.
    @type columnNames: list
    @param columnNames: list of column names that need to be onehot encoded

    @rtype dataFrame, columnNameToOHE.keys(), columnNameToOHE: dataframe, list,␣
↪dictionary
    @returns dataFrame, columnNameToOHE.keys(), columnNameToOHE:
                    dataframe with the columndata onehot encoded.
                    the keys of the onehot encoded categories
                    the full dictionary with the old columnName as key and the␣
↪category names as values.
    '''


    columnNameToOHE={}

    #for each column, which needs to be OHEd create an OH-Encoder and the␣
↪corresponding Encoding. Insert it in the Dataframe
    for columnName in columnNames:
        print(columnName)
        dataFrame = IntTransfer(dataFrame,columnName)
        hotCodedArray=array(dataFrame[columnName][:]).reshape(-1, 1)

        #create an OH-Encoder and the corresponding Encoding

        #enc= LabelBinarizer()
        enc = OneHotEncoder(handle_unknown='ignore')
        hotCodedArray=enc.fit_transform(hotCodedArray).toarray()
        columnNameToOHE[columnName]=enc

        #Find out columnames
        hotCodedcolumns=[]
        for category in list(enc.categories_[0]):
            hotCodedcolumns+=[columnName+"_"+str(category)]
```

```
        #create a new df, using the hot coded columns as Column Names
        hotCoded = pd.DataFrame(hotCodedArray, columns = hotCodedcolumns)

        #remove old data column and add new dataframe
        dataFrame.drop(labels=columnName,axis=COLUMN_AXIS, inplace= True)
        dataFrame = pd.merge(dataFrame, hotCoded, left_index=True,␣
 ↪right_index=True)

    return dataFrame, columnNameToOHE.keys(), columnNameToOHE
```

Calculate the VIF for One Hot encoding

[13]:
```
def calculateVIF(df, featuresToTest):
    """
    Calculate the VIF values for the dataframe, from the list of features that␣
 ↪needs to be tested.

    @type df: dataframe
    @param df: dataframe with the features of the featurelist of which
                    VIF values need to be calculated from.
    @type featuresToTest: list
    @param featuresToTest: feature list of column names that have data which
                    VIF values need to be calculated from.
    @rtype vif_data: dataframe
    @returns vif_data: dataframe with the vif_data.
    """



    #Filters the whole dataset for the feature set of one OneHotEncoded-Column
    independentVariables = df.filter(featuresToTest)

    #print(independentVariables.head())
    X = independentVariables

    # VIF dataframe
    vif_data = pd.DataFrame()
    vif_data["feature"] = X.columns

    # calculating VIF for each feature
    vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in␣
 ↪range(len(X.columns))]
    return vif_data

def checkForMultiCol(df, featuresToTest):
    """
    Check for multicolinearity in the dataframe with the featuresToTest list.
```

```
    @type df: datafram
    @param df: dataframe with the features of the featurelist of which
                    multicolinearity needs ot be tested upon.
    @type featuresToTest: list
    @param featuresToTest: feature list of column names that have data which
                    multicolinearity needs ot be tested upon.
    @rtype multicols: list
    @returns multicols: list of features which have mutlicolinearity appear.
    """

    multicols =[]
    #calculate the Variance Inflation Factor and gives a table with all VIF with␣
↪the features back
    vifData = calculateVIF(df, featuresToTest)
    #Goes through the table and checks for multicollinearity
    for i in range(0,len(vifData)-1):
            # >= 5 because from 5 multicollinearity is there
            if vifData['VIF'][i] >= 5:
                print(vifData['VIF'][i])
                #saves features that are multicollinearity in list
                multicols +=[vifData['feature'][i]]
    return multicols

def multi_col_test(dataFrame):
    """
    Check for multicolinearity in the OneHtEncoded data.

    @type dataFrame: dataframe
    @param dataFrame: dataframe with the onehot encoded data.

    @rtype: boolean
    @returns: boolean value whether the multicolinearity appeared in the data or␣
↪not.
    """
    #Test if there is Multicollinearity in the OneHotEncoding
    multicols =[]
    for i in range(0,len(CATEGORICAL_COLUMN_NAMES)):
        #print(list(hotCodedNames[i]))
        multicols +=checkForMultiCol(dataFrame,␣
↪list(CATEGORICAL_COLUMN_NAMES[i]))
    if (len(multicols)==0):
        return True
    else:
        return False
```

3.2 2.2.7 Normalization

Normalize: - Numerical. - label encoded data.

Don't normalize: - Binary Data . - One Hot Encoded data.

[14]:
```
def normalize_data(dataFrame):
    """
    Normalize data in the dataframe for the columns that appear in the column␣
↪names list.

    @type dataFrame: dataframe
    @param dataFrame: dataframe that contains the not-normalized data for the␣
↪column names list.

    @rtype dataFrame: dataframe
    @returns dataFrame: the new dataframe with normalized values for the columns␣
↪in the column names list.

    """

    normalizationData = dataFrame[NORMALIZATION_COLUMN_NAMES]

    x = normalizationData.values #returns a numpy array
    min_max_scaler = MinMaxScaler()
    x_scaled = min_max_scaler.fit_transform(x)
    normalizedDf = pd.DataFrame(x_scaled, columns=NORMALIZATION_COLUMN_NAMES)
    dataFrame.drop(labels = NORMALIZATION_COLUMN_NAMES, inplace=True, axis = 1)
    dataFrame = pd.merge(dataFrame, normalizedDf, left_index=True,␣
↪right_index=True)
    return dataFrame
```

3.3 2.3 -Feature Selection

3.3.1 2.3.1 New Feature Split

[15]:
```
def get_feature_data(dataFrame, featureColumnNames):
    """
    Get the feature data form the dataframe with the  column names of those␣
↪features.

    @type dataFrame: dataframe
    @param dataFrame: the dataframe that contains the featuredata for the␣
↪featurenames in the list.

    @rtype featureData: dataframe
    @returns featureData: stripped dataFrame from the original dataframe, that␣
↪now contains only
```

```
                            the data from the feature column names.
    """

    #drop the outcome features in the featureData list.
    featureData = dataFrame.drop(labels = OUTCOME_COLUMN_NAMES, axis = COLUMN_AXIS)

    #filter out the columnNames from a specific pool of columnNames:
    # - surgery features or patient features
    featureData = featureData.filter(featureColumnNames)

    return featureData

def backward_elimination(X, y):
    """
    Use backward elimination on X and y.

    @type X: list
    @param X: x data of the model.
    @type y: list
    @param y: y data corresponding to the x data in the model.

    @rtypes featuresDataSelected, list(selectedFeaturesColumnNames): list, list
    @returns featuresDataSelected, list(selectedFeaturesColumnNames):
            The featuredata that had an significant impact on the outcomedata and
            column names.

    """

    #Adding constant column of ones, mandatory for sm.OLS model
    X_1 = sm.add_constant(X)

    model = sm.OLS(y,X_1).fit()

    #Extract columnsNames
    columnNames = list(X.columns)
    SIGNIFICANCE = 0.05

    #Set the pmax variable to 1. Will be replaced in while loop.
    pmax = 1

    #while the length of the columnNames is not empty (yet).
    #wrapper methods:
    #     Backwards elimination = single features only.
    #     Filter method is = combination of features.
    while (len(columnNames)>0):
```

```
        # initialize an empty list for all the pValues.
        p= []

        # make the statistical model and fit it to the data.
        X_1 = X[columnNames]
        X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE␣
↪AND FIX IT!
        model = sm.OLS(y,X_1).fit()

        new_arr = delete(model.pvalues.values, (len(model.pvalues.values)-1))

        #make a dataframe with the columnNames and their corresponding P values.
        p = pd.Series(new_arr,index = columnNames)

        # extract the maximum P value of the column.
        pmax = max(p)
        feature_with_p_max = p.idxmax()

        # compare the p value with the significance value.
        # if the pmax is bigger than the significance value, the specific
        # feature is insignificant and removes it from the list.
        if(pmax>SIGNIFICANCE):
            #print(feature_with_p_max)
            columnNames.remove(feature_with_p_max)

        # if there is no maximum p-value that is insignificant anymore, the␣
↪while loop will break.
        else:
            break

    selectedFeatures = columnNames

    featuresDataSelected = X.filter(selectedFeatures)
    selectedFeaturesColumnNames = featuresDataSelected.columns

    return featuresDataSelected, list(selectedFeaturesColumnNames)
```

[16]:

```
def get_target_names_string(yColumn):
    """
    Get the targetnames of the outcomedata. Turn it into one string.

    @type yColumn: list
    @param yColumn: list of outcomedata.

    @rtype targetNamesString: string
    @returns targetNamesString: one string with all the outcomedata.
    """
```

```
    #initialize which column will be the y in the model.
    #print(yColumn.unique())

    #initialize the list for the targetnames.
    targetNames = list(yColumn.unique())

    #initalize for the classification report
    targetNamesString = [str(i) for i in targetNames]

    return targetNamesString
```

3.3.2 2.3.2 Inverted Indexes Method

[17]:
```
def add_index_to_value_column(temp_dict, column, value, index):
    """
    Add the index of a row to a vector value in the dictionary.
    If the in vector value is not a key yet, it creates a new key
    and adds it.

    @type temp_dict: dictionary
    @param temp_dict: contains the vector values as keys and
                        has the row indexes as values.
    @type column: string
    @param column: name of the certain column in which the vector values appear.
    @type value: string/float
    @param value: vector value that the row(index) contains for that specific␣
 ↪column.
    @type index: integer
    @param index: the index of the row in the dataset.

    @rtype: dictionary
    @returns: the dictionary with the new value.
    """

    if value not in temp_dict[column]: temp_dict[column][value] = []

    #add the index to the specific vector value of that column
    temp_dict[column][value].extend(index)
    return temp_dict

def create_ii(X_array):
    """
    Enumerate through the while X training set.
    Make entries (keys) for all the vector values appearing in the training set.

    @type X_array: array
```

```
    @param X_array: X data of the training set.

    @rtype ii: dictionary
    @returns ii: a dictionary with the vector values as keys and
                 the rowindexes as dctionary values.
    """

    ii = None

    #create inverted index
    for index, vector in enumerate(X_array):
        #build inverted index and make slots for the columns
        if ii == None: ii = [{} for _ in vector]

        #for every column in this rowVector
        for column, value in enumerate(vector):
            ii = add_index_to_value_column(ii, column, value, [index])
    return ii

def retrieve_vectors_ii(ii, query_vector):
    """
    Counts for the query vector the amount of times a certain
    row appears for each vector value: the score about how
    similar those rows are. Then returns the index or indices
    that appear to be the most similar.

    @type ii: dictionary
    @param ii: a dictionary with the vector values as keys and
                 the rowindexes as dctionary values.
    @type query_vector: list
    @param query_vector: the list of feature values as a
                 'query vector'. This is one row of the
                 the x test data.

    @rtype max_keys: list
    @returns max_keys: a list of indices that are most similar to
                 the query vector.
    """

    scores = {}

    #for every column in the query vector
    for qcolumn, qvalue in enumerate(query_vector):

        #count the map indexes(documents) containing this word and how many␣
↪times document appears in ii.
        if qvalue in ii[qcolumn]:
```

```
            #get the indices that have this column value
            ret_indices = ii[qcolumn][qvalue]

            #add the indices to the score for each appearing indices.
            #it counts for every column the reoccuring indices.
            for i in ret_indices:
                if i not in scores:
                    scores[i] = 1
                else:
                    scores[i] += 1

    # find the highest score: the score for the indices that
    # are most similar.
    # Retrieve the indices with this score: most similar to the query vector.
    max_val = max(scores.values())
    max_keys = [key for key in scores if scores[key] == max_val]
    return max_keys

def most_frequent(List):
    """
    Returns the value that appears the most in this list.

    @type List: list
    @param List: list of which the most appearing
                 needs to be calculated.
    @rtype: value of the list
    @returns: the most frequent value of the list.
    """

    return max(set(List), key = List.count)

def retrieve_y_values(indices, y_array):
    """
    Retrieves the most appearing y_value out of the given indices.

    @type indices: list
    @param indices: pre-calculated indices of the X train
                    set that are most similar to a query index.
    @type y_array: list
    @param y_array: outcome data from the training set.

    @rtype y_value: string/float
    @returns y_value: the y_train-value corresponding to the most
                      similar index/indices to the query row.
    """
    count = 0
```

```
    #make a list with the length of the amount of retrieved indices
    y_value = [None]*len(indices)

    #count for each appearing index, how many times it appears in total.
    for index in indices:
        y_value[count] = y_array[index]
        count += 1

    return y_value


def predict_y_ii (ii, X_array, y_array):
    """
    Predict the outcome values for the X test set,
    with the inverted indexes dictionary and the
    y training data.

    @type ii: dictionary
    @param ii: a dictionary with the vector values as keys and
                the rowindexes as dctionary values.

    @type X_array: list
    @param X_array: x test data
    @type y_array: list
    @param y_array: y training data

    @rtype pred: list
    @returns pred: the predicted values for this x test set.
    """

    pred = [None]*len(X_array)

    for i, qvector in enumerate(X_array):

        #retrieve the similar indices for this query vector
        most_sim_indices = retrieve_vectors_ii(ii, qvector)

        #retrieve the according y values for the most appearing indices.
        y_values = retrieve_y_values(most_sim_indices, y_array)

        prediction = most_frequent(y_values)
        pred[i] = prediction
    return pred
```

3.4 2.4 Evaluate the models

[18]:
```
def reportPerformance(groundTruth, predictions, report_mode):
    """#reports the quality of the prediction.
    #the value that determines the best classifier, depends on the report_mode
    "continuos" --> RMSE
    "categorical" --> Accuracy

    if report_mode == "continuos" the accuracy/F1-Scores is not computed, to␣
 ↪avoid problems with regression based classifiers
    """
    if report_mode != "continuos":
        #printing the classification report of the performance
        accuracy=accuracy_score(groundTruth, predictions)
        f1macro=f1_score(groundTruth, predictions, average='macro')
        f1weighted=f1_score(groundTruth, predictions, average='weighted')
        print ('Accuracy score \t %0.3f'%(accuracy))
        print ('F1 Macro score \t %0.3f'%(f1macro))
        print ('F1 Weighted sc \t %0.3f'%(f1weighted))

    MAE=mean_absolute_error(groundTruth, predictions)
    RMSE=sqrt(mean_squared_error(groundTruth, predictions))

    #     print ('Recall \t{}'.format(recall_score(groundTruth, predictions,␣
 ↪average = 'weighted')))
    #     print ('Average prec \t{}'.format(average_precision_score(groundTruth,␣
 ↪predictions)))
    print ('MAE error \t %0.3f'%(MAE))
    print ('RMSE error \t %0.3f'%(RMSE))
    print ()
    if report_mode == "continuos":
        return -RMSE
    else:
        return accuracy
```

4 Chapter 3 - The Pipeline

4.1 3.0 Preparation

Constants

[19]:
```
FILENAME = 'surgical_case_durations.csv'        # The Filename of the csv file,␣
 ↪that we are using.
COLUMN_AXIS = 1                                 #
ROW_AXIS = 0                                    #
```

Here you can choose what you want to predict. Choose between:

'ActualDurationTime'

'ActualDurationTimeRange'

[20]:
```
# PREDICTED_COLUMN='ActualDurationTime'
PREDICTED_COLUMN=input("Please, Type in your choice 'ActualDurationTime' or␣
 ↪'ActualDurationTimeRange': ")    # The Column that we want to predict using␣
 ↪this pipeline. Can be: ActualDurationTimeRange or ActualDurationTime
```

```
Please, Type in your choice 'ActualDurationTime' or 'ActualDurationTimeRange':
ActualDurationTime
```

Lists and Dictionaries

[21]:
```
#Dictionary for the column renamings
TO_ENGLISH_COLUMNS={"Geplande operatieduur": "PlannedDurationTime",
                    "Operatieduur": "ActualDurationTime",
                    "Operatietype":"OperationType",
                    "Chirurg":"Surgeon",
                    "Anesthesioloog":"Anesthesiologist",
                    "Benadering":"Approach",
                    "OK":"OperationRoom",
                    "Casustype":"Urgency",
                    "Dagdeel":"PartOfDay",
                    "Leeftijd":"Age",
                    "AF":"AtrialFibrillation",
                    "HLM":"CardiopulmonaryBypassUse",
                    "Geslacht":"Sex",
                    "Aantal anastomosen":"AmountOfBypasses",
                    "Chronische longziekte":"P_ChronicLungDisease",
                    "Extracardiale vaatpathie":"P_extracardialArteriopathy",
                    "Actieve endocarditis": "P_activeEndocarditis",
                    "Hypertensie":"P_Hypertension",
                    "Pulmonale hypertensie":"P_PulmonaleHypertension",
                    "Slechte mobiliteit":"P_PoorMobility",
                    "Hypercholesterolemie":"P_Hypercholesterolemia",
                    "Perifeer vaatlijden":"P_PeripherialVascularDisease",
                    "Linker ventrikel functie":"LeftVentricleFunction",
                    "Nierfunctie":"RenalFunction",
                    "DM":"P_Diabetis",
                    "Eerdere hartchirurgie":"PreviousHeartSurgery",
                    "Kritische preoperatieve status":"CriticalPre-OP",
                    "Myocard infact <90 dagen":"MycordialInfarctionPreSurgery",
                    "Aorta chirurgie":"AorticSurgery",
                    "Ziekenhuis ligduur":"HospitalDays",
                    "IC ligduur":"ICUDays"
                   }

#All features, that are related to the patient
PATIENT_FEATURES= ['Urgency', 'Sex',
       'AtrialFibrillation', 'P_ChronicLungDisease',
```

```
       'P_extracardialArteriopathy', 'PreviousHeartSurgery',
       'P_activeEndocarditis', 'CriticalPre-OP',
       'MycordialInfarctionPreSurgery', 'AorticSurgery',
       'P_PulmonaleHypertension', 'LeftVentricleFunction', 'Euroscore1',
       'Euroscore2', 'RenalFunction', 'P_PoorMobility', 'P_Diabetis',
       'P_Hypercholesterolemia', 'P_Hypertension',
       'P_PeripherialVascularDisease', 'CCS', 'NYHA', 'AmountOfBypasses',
       'CardiopulmonaryBypassUse']

#Features that are related to the surgery
SURGERY_FEATURES =[ 'Surgeon', 'Anesthesiologist', 'Approach',
       'OperationRoom','PartOfDay', 'OperationType']
#________________________________________

RANGE_COLUMN_NAMES = ['Age', 'BMI', 'ActualDurationTime',
                         'PlannedDurationTime'] #and overtime

ORDINAL_COLUMN_NAMES = RANGE_COLUMN_NAMES + ['CCS', 'NYHA', 'Urgency',
                         'LeftVentricleFunction', 'RenalFunction',
                         'P_PulmonaleHypertension']

NON_ORDINAL_COLUMN_NAMES = ['AmountOfBypasses', 'Euroscore1',
                            'Euroscore2', 'HospitalDays', 'ICUDays']

NUMERICAL_COLUMN_NAMES = RANGE_COLUMN_NAMES + NON_ORDINAL_COLUMN_NAMES

CATEGORICAL_COLUMN_NAMES = ['Surgeon', 'OperationRoom', 'OperationType',
                            'Anesthesiologist', 'Approach', 'PartOfDay']
#________________________________________

BINARY_COLUMN_NAMES = ['Sex', 'AtrialFibrillation', 'P_ChronicLungDisease',
                        'P_extracardialArteriopathy', 'PreviousHeartSurgery',
                        'P_activeEndocarditis', 'CriticalPre-OP',
                        'MycordialInfarctionPreSurgery', 'AorticSurgery',
                        'P_PoorMobility', 'P_Diabetis', 'P_Hypercholesterolemia',
                        'P_Hypertension', 'P_PeripherialVascularDisease',
                        'CardiopulmonaryBypassUse']

NORMALIZATION_COLUMN_NAMES = ['CCS', 'NYHA', 'Urgency', 'BMIRange',
                       'LeftVentricleFunction', 'RenalFunction',
                       'P_PulmonaleHypertension', 'AgeRange', 'Euroscore1',
                       'Euroscore2']

OUTCOME_COLUMN_NAMES = ['HospitalDays', 'ICUDays', 'PlannedDurationTime',
                        'PlannedDurationTimeRange', 'OvertimeRange',
                        'ActualDurationTimeRange','ActualDurationTime']
```

4.2 3.1 Importing the Data

[22]:
```
# Import the data from the file: 'surgical_case_durations.csv'. Store it in a␣
↪dataframe
surgicalData = import_df_from_file(FILENAME)
```

```
                                        Operatietype            Chirurg  \
0                          Amputatie teen + Wondtoilet               6,00
1     Arthroscopische acrominoclaviculaire reconstru...  Ander specialisme
2                                  Ascendensvervanging               4,00
3                                  Ascendensvervanging               7,00
4                                  Ascendensvervanging               6,00
...                                                ...                ...
4081                                       Wondtoilet               2,00
4082                                       Wondtoilet               4,00
4083                                       Wondtoilet               4,00
4084                                       Wondtoilet              15,00
4085                                       Wondtoilet               1,00

     Anesthesioloog            Benadering     OK       Casustype  Dagdeel  \
0          Onbekend                  NaN   TOK3        Electief   Middag
1          Onbekend                  NaN  OK 10        Electief   Middag
2              8,00  Volledige sternotomie   TOK1        Electief  Ochtend
3              6,00  Volledige sternotomie   TOK2        Electief   Middag
4          Onbekend                  NaN   TOK2           Spoed    Avond
...             ...                  ...    ...             ...      ...
4081       Onbekend                  NaN   TOK1        Electief   Middag
4082       Onbekend                  NaN   TOK3        Electief   Middag
4083       Onbekend                  NaN   TOK3  Spoed < 24 uur  Ochtend
4084       Onbekend                  NaN   TOK2        Electief   Middag
4085       Onbekend                  NaN   TOK1        Electief  Ochtend

      Leeftijd Geslacht   AF  ... Hypertensie Perifeer vaatlijden  CCS NYHA  \
0         51.0      NaN  NaN  ...         NaN                 NaN  NaN  NaN
1         50.0      NaN  NaN  ...         NaN                 NaN  NaN  NaN
2         78.0        V    N  ...           N                   J  0.0  2.0
3         66.0        V    J  ...           N                   N  2.0  1.0
4         72.0      NaN  NaN  ...         NaN                 NaN  NaN  NaN
...        ...      ...  ...  ...         ...                 ...  ...  ...
4081      62.0      NaN  NaN  ...         NaN                 NaN  NaN  NaN
4082      62.0      NaN  NaN  ...         NaN                 NaN  NaN  NaN
4083      62.0      NaN  NaN  ...         NaN                 NaN  NaN  NaN
4084      57.0      NaN  NaN  ...         NaN                 NaN  NaN  NaN
4085      64.0      NaN  NaN  ...         NaN                 NaN  NaN  NaN

     Aantal anastomosen  HLM Geplande operatieduur Operatieduur  \
0                   NaN  NaN                  62.0         52.0
1                   NaN  NaN                  85.0         70.0
```

```
2                    0.0    N                     229.0        170.0
3                    0.0    N                     140.0        190.0
4                    NaN  NaN                     370.0        480.0
...                  ...  ...                       ...          ...
4081                 NaN  NaN                      78.0         65.0
4082                 NaN  NaN                      60.0         25.0
4083                 NaN  NaN                      46.0         39.0
4084                 NaN  NaN                      60.0         46.0
4085                 NaN  NaN                      53.0         49.0

     Ziekenhuis ligduur IC ligduur
0              Onbekend   Onbekend
1              Onbekend   Onbekend
2                  4,00       2,00
3                  6,00       1,00
4              Onbekend   Onbekend
...                 ...        ...
4081              44,00       0,00
4082           Onbekend   Onbekend
4083           Onbekend   Onbekend
4084           Onbekend   Onbekend
4085           Onbekend   Onbekend

[4086 rows x 36 columns]
```

4.3 3.2 Preprocessing

4.3.1 3.2.0 - Translate column-names to english

[23]:
```
# translate the columns to english
surgicalData = surgicalData.rename(columns = TO_ENGLISH_COLUMNS)
```

4.3.2 3.2.1 Add new Columns

Overtime Column Creating the Overtime Column based on the planned- and actual duration time.

[24]:
```
# create the overtime column
surgicalData = createOvertimeColumn(surgicalData)
```

OperationType The OperationType column has a string with all the operations performed on the subject. The operations, that make up a to over 99% of the total operations are included

[25]:
```
OperationTypeEncoder=OPTypeEncoder(n=99)
surgicalData=OperationTypeEncoder.fit_transform(surgicalData, 'OperationType')
```

```
The relevant op types, that cover at least 99% of all operations, are: ['other',
'cabg', 'avr', 'pacemakerdraad tijdelijk', 'mvp', 'mvp shaving', 'wondtoilet',
```

'tvp', 'mvr']

4.3.3 3.2.2 Remove NaNs

[26]:
```
#in the dataset we have some "onbekend"-values, that should be NaNs too.
surgicalData = replace_with_nan (surgicalData, value = 'Onbekend')

#drop columns and rows with too much nan
surgicalData, droppedColumns = dropWithTooMuchNan(surgicalData, axis =␣
 ↪COLUMN_AXIS)

#remove the columnnames in the lists that are removed from the dataset as well.
for removeColumnName in droppedColumns:
    if removeColumnName in RANGE_COLUMN_NAMES:
        RANGE_COLUMN_NAMES.remove(removeColumnName)
    if removeColumnName in ORDINAL_COLUMN_NAMES:
        ORDINAL_COLUMN_NAMES.remove(removeColumnName)
    if removeColumnName in NON_ORDINAL_COLUMN_NAMES:
        NON_ORDINAL_COLUMN_NAMES.remove(removeColumnName)
    if removeColumnName in NUMERICAL_COLUMN_NAMES:
        NUMERICAL_COLUMN_NAMES.remove(removeColumnName)
    if removeColumnName in CATEGORICAL_COLUMN_NAMES:
        CATEGORICAL_COLUMN_NAMES.remove(removeColumnName)
    if removeColumnName in BINARY_COLUMN_NAMES:
        BINARY_COLUMN_NAMES.remove(removeColumnName)
    if removeColumnName in NORMALIZATION_COLUMN_NAMES:
        NORMALIZATION_COLUMN_NAMES.remove(removeColumnName)

surgicalData = dropWithTooMuchNan(surgicalData, axis = ROW_AXIS)
```

4.3.4 3.2.3 Retype the data

[27]:
```
# retype Object columns to categorical columns, if possible
surgicalData = convert_df_type(surgicalData, list(surgicalData), 'category')

# retype the columns that are not float64 yet
surgicalData = convert_df_type(surgicalData, NUMERICAL_COLUMN_NAMES, 'float64')

# retype the columns that are not integers yet
surgicalData = convert_df_type(surgicalData, BINARY_COLUMN_NAMES, 'int8')
```

```
['Age', 'BMI', 'ActualDurationTime', 'PlannedDurationTime', 'AmountOfBypasses',
'Euroscore1', 'HospitalDays', 'ICUDays']
```

4.3.5 3.2.4 Outlier removal

Outlier cutting for the categorical outliers: those outliers will be grouped into "other", while the numerical outliers will be removed from the dataset.

[28]:
```
#categoricalOutlier Cutting
surgicalData = categorical_outlier_cutting(surgicalData,␣
↪CATEGORICAL_COLUMN_NAMES, 10)
# #numericalOutlier Cutting
surgicalData = numerical_outlier_cutting(surgicalData, NUMERICAL_COLUMN_NAMES)
```

[29]:
```
#Needed for the Duration Generator --> Chapter 4
priorData=surgicalData[:]
surgicalData.head()
```

[29]:
```
    Age  CCS  NYHA  AmountOfBypasses  PlannedDurationTime  ActualDurationTime  \
0  66.0  2.0   1.0               0.0                140.0               190.0
1  71.0  0.0   1.0               0.0                241.0               239.0
2  66.0  0.0   1.0               0.0                240.0               269.0
3  52.0  0.0   1.0               0.0                180.0               305.0
4  80.0  2.0   2.0               0.0                218.0               300.0

   Overtime Surgeon Anesthesiologist             Approach  ... ICUDays  \
0      50.0    7,00             6,00 Volledige sternotomie  ...     1.0
1      -2.0    4,00            10,00 Volledige sternotomie  ...     1.0
2      29.0    3,00            15,00 Volledige sternotomie  ...     0.0
3     125.0    1,00            11,00 Volledige sternotomie  ...     1.0
4      82.0    2,00             5,00 Volledige sternotomie  ...     4.0

  OPType_other OPType_cabg  OPType_avr  OPType_pacemakerdraad tijdelijk  \
0            0           1           1                                1
1            0           1           1                                1
2            0           1           1                                1
3            0           1           1                                1
4            0           1           0                                1

   OPType_mvp  OPType_mvp shaving  OPType_wondtoilet  OPType_tvp  OPType_mvr
0           1                   1                  0           1           1
1           1                   1                  0           1           1
2           1                   1                  0           1           1
3           1                   1                  0           1           1
4           1                   0                  0           1           1

[5 rows x 42 columns]
```

4.3.6 3.2.5 Categorical Data

Creating categorical data from ranged numerical data

[30]:
```
surgicalData, column2binNames = to_categorical_range(surgicalData)

#replace the current category name with the categoryname+'Range', it the lists.
```

```
for removeColumnName in RANGE_COLUMN_NAMES:
    replaceColumnName = removeColumnName+'Range'
    if removeColumnName in RANGE_COLUMN_NAMES:
        RANGE_COLUMN_NAMES = list(map(lambda x: x.replace(removeColumnName,␣
↪replaceColumnName), RANGE_COLUMN_NAMES))
    if removeColumnName in ORDINAL_COLUMN_NAMES:
        ORDINAL_COLUMN_NAMES = list(map(lambda x: x.replace(removeColumnName,␣
↪replaceColumnName), ORDINAL_COLUMN_NAMES))
    if removeColumnName in NON_ORDINAL_COLUMN_NAMES:
        NON_ORDINAL_COLUMN_NAMES = list(map(lambda x: x.
↪replace(removeColumnName, replaceColumnName), NON_ORDINAL_COLUMN_NAMES))
    if removeColumnName in NUMERICAL_COLUMN_NAMES:
        NUMERICAL_COLUMN_NAMES = list(map(lambda x: x.replace(removeColumnName,␣
↪replaceColumnName), NUMERICAL_COLUMN_NAMES))
    if removeColumnName in CATEGORICAL_COLUMN_NAMES:
        CATEGORICAL_COLUMN_NAMES = list(map(lambda x: x.
↪replace(removeColumnName, replaceColumnName), CATEGORICAL_COLUMN_NAMES))
    if removeColumnName in BINARY_COLUMN_NAMES:
        BINARY_COLUMN_NAMES = list(map(lambda x: x.replace(removeColumnName,␣
↪replaceColumnName), BINARY_COLUMN_NAMES))
    if removeColumnName in NORMALIZATION_COLUMN_NAMES:
        NORMALIZATION_COLUMN_NAMES = list(map(lambda x: x.
↪replace(removeColumnName, replaceColumnName), NORMALIZATION_COLUMN_NAMES))
```

```
67-76    694
58-66    417
77-87    315
47-57    244
Name: AgeRange, dtype: int64
[46, 57, 66, 76, 87]
5
['47-57', '58-66', '67-76', '77-87']
4
overweight     819
normal         466
obese          385
underweight      0
Name: BMIRange, dtype: int64
[17.91, 18.5, 24.9, 29.9, 36.1]
5
['underweight', 'normal', 'overweight', 'obese']
4
181-240    665
241-280    443
281-345    305
110-180    192
346-398     65
```

```
Name: ActualDurationTimeRange, dtype: int64
[109, 180, 240, 280, 345, 398]
6
['110-180', '181-240', '241-280', '281-345', '346-398']
5
181-240    1043
241-280     408
140-180     119
281-330     100
Name: PlannedDurationTimeRange, dtype: int64
[139, 180, 240, 280, 330]
5
['140-180', '181-240', '241-280', '281-330']
4
In_Time                 633
Overtime_Small          398
Ahead_Of_Time_Medium    312
Overtime_Medium         164
Ahead_Of_Time_Extreme    94
Overtime_Extreme         69
Name: OvertimeRange, dtype: int64
[-165.0, -70, -25, 25, 70, 120, 215.0]
7
['Ahead_Of_Time_Extreme', 'Ahead_Of_Time_Medium', 'In_Time', 'Overtime_Small',
'Overtime_Medium', 'Overtime_Extreme']
6
```

4.3.7 3.2.6 Encoding

Label Encoding This is done for numerical data that is ordinal.

[31]:
```
#create one list of column names that needs to be label encoded
toBeLE=ORDINAL_COLUMN_NAMES+BINARY_COLUMN_NAMES
#label encode the binary data and the ordinal categorical columns
surgicalData,columnNameToLE = label_encoding(surgicalData, toBeLE)
```

One Hot Encoding One hot encoding should be done for the features, that are nominal-scaled.

label encoding is done for categorical data, with string names, so that it can be understood by the ml-algorithms

[32]:
```
#define what column names should be onehot encoded
toBeOHE=CATEGORICAL_COLUMN_NAMES

#onehot encode the nominal categorical columns
surgicalData, codedNames, columnNameToOHE  = one_hot_encoding(toBeOHE,␣
↪surgicalData)
```

```
#remove the original column names of the features. (they got replacewd by the␣
↪values and then encoded)
intersectionSFC = set(SURGERY_FEATURES).intersection(CATEGORICAL_COLUMN_NAMES)
intersectionPFC = set(PATIENT_FEATURES).intersection(CATEGORICAL_COLUMN_NAMES)
for columnName in intersectionSFC:
    SURGERY_FEATURES.remove(columnName)
for columnName in intersectionPFC:
    PATIENT_FEATURES.remove(columnName)

#replace the old columns with the encoded ones.
CATEGORICAL_COLUMN_NAMES = list(codedNames)+OperationTypeEncoder.
↪relevant_op_types_Names


#add the encoded columns to the features.
SURGERY_FEATURES += list(codedNames)
PATIENT_FEATURES += list(codedNames)
```

```
Surgeon
OperationRoom
Anesthesiologist
Approach
PartOfDay
```

Variance Inflation Factor (VIF) The VIF values indicate how many times larger the variance would become, to create a dataset that would have been uncorrelated. If this VIF value is smaller than 5, that means no significant multicollinearity appears within the dataset. The following code will then produce the output 'True'

[33]:
```
print(multi_col_test(surgicalData))
#continue the coding if this is true
```

```
True
```

4.3.8 3.2.7 Normalizing

Normalize: - Numerical. - label encoded data.

Don't normalize: - Binary Data . - One Hot Encoded data.

[34]:
```
#normalize the data
surgicalData = normalize_data(surgicalData)
```

[35]:
```
#save the normalized data to a .csv file
surgicalData.to_csv("PreprocessedSurgicalData.csv", index=False)
```

5 3.3: Feature Selection

[36]:
```
#read the .csv file with the normalized data.
featureData=pd.read_csv("PreprocessedSurgicalData.csv")
```

[37]:
```
#get data that is our X and data that could be our Y
outcomeData = surgicalData[OUTCOME_COLUMN_NAMES]
y = outcomeData[PREDICTED_COLUMN]
#get the targetNamesString for the classification report
targetNamesString = get_target_names_string(y)


#find relevant Features based on the surgery
featureData = get_feature_data(surgicalData, SURGERY_FEATURES)
selectedSurgeryFeaturesData, selectedSurgeryFeatureColumnNames =␣
 ↪backward_elimination(featureData, y)
#print(selectedSurgeryFeatureColumnNames)
#find relevant Features based on the patient
featureData = get_feature_data(surgicalData, PATIENT_FEATURES)
selectedPatientFeaturesData, selectedPatientFeatureColumnNames =␣
 ↪backward_elimination(featureData, y)
#print(selectedPatientFeatureColumnNames)

#combine the selected features from the patient table and the surgery table.
selectedFeatureColumnNames=selectedSurgeryFeatureColumnNames+selectedPatientFeatureColumnNames
selectedFeatureNames=list(selectedFeatureColumnNames)

print("The selected Features are:",selectedFeatureNames)

#get the feature data corresponding to the column names
featureData = surgicalData[selectedFeatureNames.copy()]
```

```
The selected Features are: ['OPType_other', 'OPType_cabg', 'OPType_avr',
'OPType_pacemakerdraad tijdelijk', 'OPType_mvp', 'OPType_mvp shaving',
'OPType_wondtoilet', 'OPType_tvp', 'OPType_mvr', 'AtrialFibrillation',
'PreviousHeartSurgery', 'P_activeEndocarditis', 'AorticSurgery',
'P_PulmonaleHypertension', 'P_Hypertension', 'CCS', 'NYHA', 'AmountOfBypasses',
'CardiopulmonaryBypassUse']
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
```

[38]:
```
#save the feature data to a .csv file
featureData.to_csv("SelectedFeatureData.csv", index=False)
```

6 Chapter 3.4: Evaluation

[39]:
```
#read the feature data from the .csv file
featureData=pd.read_csv("SelectedFeatureData.csv")
```

6.1 3.4.1 train-test split

[40]:
```
X=featureData
#As we want to get the performance metrics for the original prediction as well␣
↪and need the
```

```
#same split for this, we add 'PlannedDurationTime' here before the␣
↪train-test-split
X = pd.merge(X, surgicalData['PlannedDurationTime'], left_index=True,␣
↪right_index=True)
X = pd.merge(X, surgicalData['PlannedDurationTimeRange'], left_index=True,␣
↪right_index=True)

#split the dataset into training (70%) and testing (30%) sets
X_train,X_test,y_train,y_test = model_selection.train_test_split(X,y,test_size=0.
↪3,random_state=randomSeed)


#Now that the split is done we split the 'Planned Duration Time' from feature␣
↪split
y_test_Original = pd.merge(X_test['PlannedDurationTime'][:
↪],X_test['PlannedDurationTimeRange'][:], left_index=True, right_index=True)
X_train.
↪drop(labels=['PlannedDurationTime','PlannedDurationTimeRange'],axis=COLUMN_AXIS,␣
↪inplace= True)
X_test.
↪drop(labels=['PlannedDurationTime','PlannedDurationTimeRange'],axis=COLUMN_AXIS,␣
↪inplace= True)
```

6.2 3.4.2 Choose a classifier

6.2.1 Dummy-Classifier

This is to get a Baseline: the other classifiers should perform better than these, otherwise any algorithm could potentially perform better than the current setup.

[41]:
```
#cretae list of dummy classifiers for the comparison
dummy_clfs=[DummyClassifier(strategy="most_frequent"),
            DummyClassifier(strategy="prior"),
            DummyClassifier(strategy="stratified"),
            DummyClassifier(strategy="uniform")
           ]
```

6.2.2 Regression Based Classifier

Linear Regression, LogisticRegression, Support Vector Machine

[42]:
```
#create a list of regressor classifiers for the comparison
reg_clfs=[DummyRegressor(strategy="mean"),
          LinearRegression(),
          svm.SVC(random_state=randomSeed)]
```

6.2.3 Unsupervised Classifiers

[43]:
```
#create a list of the unsupervised classifiers for the comparison
tree_clfs=[tree.DecisionTreeClassifier(max_depth = 1, random_state=randomSeed),
           RandomForestClassifier(n_estimators=100,max_depth=5,␣
 ↪random_state=randomSeed)
          ]
```

6.2.4 Supervised (clustering) Methods

[44]:
```
#create a list of the supervised classifiers fort the comparison
cluster_clfs=[
    AffinityPropagation(damping=0.9, random_state=randomSeed),
    Birch(threshold=0.01, n_clusters=2),
    KMeans(n_clusters=2, random_state=randomSeed),
    MiniBatchKMeans(n_clusters=2, random_state=randomSeed),
    GaussianMixture(n_components=2, random_state=randomSeed),
    KNeighborsClassifier()
]
```

6.3 3.4.3 Evaluate the models

[47]:
```
#define the best classifier for the final comparison.
bestClassifier=""

#Filter whether the ranged planned duration time should be used or the␣
 ↪continuous version.
if PREDICTED_COLUMN=="ActualDurationTime":
    report_mode="continuos"
    bestEvalScore=-100000
    planned_y_test = y_test_Original['PlannedDurationTime']
if PREDICTED_COLUMN=="ActualDurationTimeRange":
    report_mode="categorical"
    bestEvalScore=0
    planned_y_test = y_test_Original['PlannedDurationTimeRange']

#Loop thorugh the dummy classifiers and report their performance
print("dummy-classifier\n")
strategy=["most_frequent","prior","stratified","uniform"]
for c,clf in enumerate(dummy_clfs):
    print(clf.__class__.__name__+"-Strategy:"+strategy[c])
    clf.fit(X_train, y_train)
    predicted=clf.predict(X_test)
    EvalScore=reportPerformance(y_test, predicted,report_mode)
    if EvalScore>bestEvalScore:
        bestEvalScore=EvalScore
        bestClassifier=clf
```

```
#only useful if we look at continuos data, execute the dummy regressor
if PREDICTED_COLUMN=="ActualDurationTime":
    print("regression based\n")
    for clf in reg_clfs:
        print(clf.__class__.__name__)
        clf.fit(X_train, y_train)
        predicted=clf.predict(X_test)
        EvalScore=reportPerformance(y_test, predicted, report_mode)
        if EvalScore>bestEvalScore:
            bestEvalScore=EvalScore
            bestClassifier=clf

#Loop thorugh the unsupervised classifiers and report their performance
print("tree-based\n")
for clf in tree_clfs:
    print(clf.__class__.__name__)
    clf.fit(X_train, y_train)
    predicted=clf.predict(X_test)
    EvalScore=reportPerformance(y_test, predicted,report_mode)
    if EvalScore>bestEvalScore:
        bestEvalScore=EvalScore
        bestClassifier=clf

#Loop thorugh the supervised classifiers and report their performance
print("clustering based\n")
for clf in cluster_clfs:
    print(clf.__class__.__name__)
    clf.fit(X_train, y_train)
    predicted=clf.predict(X_test)
    EvalScore=reportPerformance(y_test, predicted,report_mode)
    if EvalScore>bestEvalScore:
        bestEvalScore=EvalScore
        bestClassifier=clf

# execute the inverted indexes method and report its performance
print("InverseIndexes")
ii = create_ii(X_train.to_numpy())
predicted = predict_y_ii(ii, X_test.to_numpy(), y_train.to_numpy())
EvalScore=reportPerformance(y_test, predicted,report_mode)
if EvalScore>bestEvalScore:
    bestEvalScore=EvalScore
    bestClassifier=clf

#print the performance of the current prediction
print("Original Prediction")
predicted = planned_y_test
```

```
EvalScore=reportPerformance(y_test, predicted,report_mode)
if EvalScore>bestEvalScore:
    bestEvalScore=EvalScore
    bestClassifier=clf

#report for the continuous data the classifier with the lowest RMSE
if report_mode=="continuos":
    print("The classifier with the lowest RMSE (",str(-bestEvalScore),") is",␣
 ↪bestClassifier.__class__.__name__)

#report for the categorical dat the classifier with the highest accuracy score
if report_mode=="categorical":
    print("The classifier with the highest acccuracy (",bestEvalScore,") is",␣
 ↪bestClassifier.__class__.__name__)
```

```
dummy-classifier

DummyClassifier-Strategy:most_frequent
MAE error        44.950
RMSE error       57.003

DummyClassifier-Strategy:prior
MAE error        44.950
RMSE error       57.003

DummyClassifier-Strategy:stratified
MAE error        62.128
RMSE error       79.076

DummyClassifier-Strategy:uniform
MAE error        74.820
RMSE error       92.659

regression based

DummyRegressor
MAE error        43.616
RMSE error       54.959

LinearRegression
MAE error        35.918
RMSE error       45.910

SVC
MAE error        42.513
RMSE error       54.535

tree-based
```

```
DecisionTreeClassifier
MAE error        41.186
RMSE error       51.907

RandomForestClassifier
MAE error        40.062
RMSE error       50.970

clustering based

AffinityPropagation

C:\Users\xlbok\AppData\Roaming\Python\Python38\site-
packages\sklearn\cluster\_affinity_propagation.py:250: ConvergenceWarning:
Affinity propagation did not converge, this model will not have any cluster
centers.
  warnings.warn(
C:\Users\xlbok\AppData\Roaming\Python\Python38\site-
packages\sklearn\cluster\_affinity_propagation.py:528: ConvergenceWarning: This
model does not have any cluster centers because affinity propagation did not
converge. Labeling every sample as '-1'.
  warnings.warn(
C:\Users\xlbok\AppData\Roaming\Python\Python38\site-
packages\sklearn\base.py:441: UserWarning: X does not have valid feature names,
but Birch was fitted with feature names
  warnings.warn(

MAE error        246.908
RMSE error       252.903

Birch
MAE error        245.575
RMSE error       251.615

KMeans
MAE error        245.521
RMSE error       251.567

MiniBatchKMeans
MAE error        245.521
RMSE error       251.567

GaussianMixture
MAE error        245.559
RMSE error       251.595

KNeighborsClassifier
MAE error        61.583
```

```
RMSE error       76.545

InverseIndexes
C:\Users\xlbok\AppData\Roaming\Python\Python38\site-
packages\sklearn\base.py:441: UserWarning: X does not have valid feature names,
but KNeighborsClassifier was fitted with feature names
  warnings.warn(
MAE error        45.752
RMSE error       58.877

Original Prediction
MAE error        45.401
RMSE error       59.082

The classifier with the lowest RMSE ( 45.91019239849399 ) is LinearRegression
```

7 ______________________

8 Chapter 4: Duration Time Generator

- Duration Time Generator

[48]:
```
#for hot encoding
hotColumnNames = ['Surgeon', 'OperationRoom', 'Anesthesiologist', 'Approach',
                  'PartOfDay']  #TODO Ähnlich zu CATEGORICAL_COLUMN_NAMES

#for label encoding
catOrdinalColumnNames = ['CCS', 'NYHA', 'Urgency','BMIRange',
                         'LeftVentricleFunction', 'RenalFunction',
                         'P_PulmonaleHypertension',␣
 ↪'AgeRange','OvertimeRange','ActualTimeRange']

#for binary labeling
binaryColumnNames=['Sex', 'AtrialFibrillation', 'P_ChronicLungDisease',
                  'P_extracardialArteriopathy', 'PreviousHeartSurgery',
                  'P_activeEndocarditis', 'CriticalPre-OP',
                  'MycordialInfarctionPreSurgery', 'AorticSurgery',
                  'P_PoorMobility', 'P_Diabetis', 'P_Hypercholesterolemia',
                  'P_Hypertension', 'P_PeripherialVascularDisease',␣
 ↪'CardiopulmonaryBypassUse']
```

4.1 Varibales Needed: - please fill the surgery's Data in

Information about the code below: It consists of multiple lists of characteristics. These lists are seperated into three types:

- Patient Data: All the information about the patient

- Surgery Data: All the information about the surgery
- Outcome Data: All the information about the outcome of the surgery. This is asked after the operation, to retrain the model, when enough people have commited the data to us.

[49]:
```
#Entries that consider data that is clear after the Surgery
outcomeData =['ActualDurationTime',
              'ICUDays',
              'HospitalDays']

#Prior_Entries, e.g. the data, that is entered before the operation #only ask␣
↪for necessary data
priorEntries= list(set(SURGERY_FEATURES + PATIENT_FEATURES)&set(priorData.
↪columns))
```

Out of the above we define if we want to use categorical or numerical questioning for our data.

- categoryPriorEntries -> Categorical Question
- numericalPriorEntries -> numerical Question

[50]:
```
#choice category entries, e.g. entries that are prior entries and categorical
categoryPriorEntries =␣
↪list(set(CATEGORICAL_COLUMN_NAMES+catOrdinalColumnNames+binaryColumnNames)&set(priorEntries))

#numerical Entries, e.g. entries that are prior entries and numerical
numericalPriorEntries = [Entry for Entry in priorEntries if Entry not in␣
↪categoryPriorEntries]
```

Lets see, how many entries we have per category:

[51]:
```
print('Len prior Entries', len(priorEntries))
print('Len cat Entries', len(categoryPriorEntries))
print('Len num Entries', len(numericalPriorEntries))
```

```
Len prior Entries 35
Len cat Entries 33
Len num Entries 2
```

8.1 4.2 Collect needed Data

[52]:
```
#generate a dictionary, that maps from the column-name to the answering options
categoryPriorEntries=list(set(priorData.columns)&set(categoryPriorEntries))
optionsForEntry={}
for c,entry in enumerate(categoryPriorEntries):
    data=set(str(i) for i in priorData[entry].unique())
    optionsForEntry[entry] = data
```

[53]:
```
def askUserForCategory(name: str, options: set) ->str:
    """
    asks the user for the correct entry for their patient.
```

```
    Input:
        name: the name of the entry which is currently in question.
        options: is a set with all the options for the entry.


    returns:
        selected: is a string with the choosen option
    """
    #sort the options alphabetically or numerical
    options= sorted(options[name])

    print('Please fill in the number that represents your data for',name,':')

    #print all the options with a representative
    for index,optionName in enumerate(options):
        print(str(index+1) + ')\t' + optionName)

    #check for valid input
    lenListOptions = len(options)
    while True:
        inputRaw = input(name + ': ')
        try:
            inputNo = int(inputRaw) - 1
            if inputNo > -1 and inputNo < lenListOptions:
                selected = list(options)[inputNo]
                print('Selected ' +  name + ': ' + selected)
                return selected
            else:
                print('Please select a valid ' + name + ' number')
        except ValueError:
            print('Please fill in the index of your choice, not the name.')
```

The following code will ask the user for the characteristic that is relevant for the entry

[54]:
```
def get_new_patient_data():
    """
    Get the patient data, the entries prior to the operation.

    Input:

    returns:
        newDataDF: dictionary with all the data entries for the patient data
    """
    # create a dict, that saves all the Entries prior to the operation
    dictDataEntries= dict.fromkeys(priorEntries)
```

```
    # Go through all catgeorical Data points and ask for entry
    for i in range(0, len(categoryPriorEntries)):
        print()
        entry = categoryPriorEntries[i]
        newDataPoint = askUserForCategory(entry,optionsForEntry)
        dictDataEntries[entry]= [newDataPoint]
        #categoryPriorEntries[i][1]=  newDataPoint

    # Go through all numerical Data points and ask for entry
    for i in range(0, len(numericalPriorEntries)):
        entry = numericalPriorEntries[i]
        print()
        newDataPoint =input('Fill in the numerical value for '+ entry + ': ')
        dictDataEntries[entry]= [newDataPoint]

    newDataDF=pd.DataFrame.from_dict(dictDataEntries)

    return newDataDF
```

[55]:
```
#create new patient data by asking it in the therminal
newDataDF=get_new_patient_data()
#save the data from the user to a .csv file.
newDataDF.to_csv("new_data.csv", index= False)
```

```
Please fill in the number that represents your data for OPType_other :
1)      0
2)      1
OPType_other: 1
Selected OPType_other: 0

Please fill in the number that represents your data for P_Hypertension :
1)      0
2)      1
P_Hypertension: 1
Selected P_Hypertension: 0

Please fill in the number that represents your data for OPType_avr :
1)      0
2)      1
OPType_avr: 1
Selected OPType_avr: 0

Please fill in the number that represents your data for PreviousHeartSurgery :
1)      0
2)      1
PreviousHeartSurgery: 1
Selected PreviousHeartSurgery: 0
```

```
Please fill in the number that represents your data for OPType_tvp :
1)      0
2)      1
OPType_tvp: 1
Selected OPType_tvp: 0

Please fill in the number that represents your data for PartOfDay :
1)      Middag
2)      Ochtend
3)      other
PartOfDay: 1
Selected PartOfDay: Middag

Please fill in the number that represents your data for Sex :
1)      0
2)      1
Sex: 1
Selected Sex: 0

Please fill in the number that represents your data for Urgency :
1)      Electief
2)      Spoed
3)      Spoed < 24 uur
Urgency: 1
Selected Urgency: Electief

Please fill in the number that represents your data for AtrialFibrillation :
1)      0
2)      1
AtrialFibrillation: 1
Selected AtrialFibrillation: 0

Please fill in the number that represents your data for OperationRoom :
1)      TOK1
2)      TOK2
3)      TOK3
4)      other
OperationRoom: 1
Selected OperationRoom: TOK1

Please fill in the number that represents your data for CCS :
1)      0.0
2)      1.0
3)      2.0
4)      3.0
5)      4.0
CCS: 1
```

```
Selected CCS: 0.0

Please fill in the number that represents your data for OPType_wondtoilet :
1)      0
OPType_wondtoilet: 1
Selected OPType_wondtoilet: 0

Please fill in the number that represents your data for NYHA :
1)      1.0
2)      2.0
3)      3.0
4)      4.0
NYHA: 1
Selected NYHA: 1.0

Please fill in the number that represents your data for
MycordialInfarctionPreSurgery :
1)      0
2)      1
MycordialInfarctionPreSurgery: 1
Selected MycordialInfarctionPreSurgery: 0

Please fill in the number that represents your data for OPType_mvp shaving :
1)      0
2)      1
OPType_mvp shaving: 1
Selected OPType_mvp shaving: 0

Please fill in the number that represents your data for Surgeon :
1)      1,00
2)      2,00
3)      3,00
4)      4,00
5)      5,00
6)      6,00
7)      7,00
8)      other
Surgeon: 1
Selected Surgeon: 1,00

Please fill in the number that represents your data for CriticalPre-OP :
1)      0
2)      1
CriticalPre-OP: 1
Selected CriticalPre-OP: 0

Please fill in the number that represents your data for P_ChronicLungDisease :
1)      0
```

```
2)      1
P_ChronicLungDisease: 1
Selected P_ChronicLungDisease: 0

Please fill in the number that represents your data for P_PulmonaleHypertension
:
1)      Ernstig
2)      Matig
3)      Normaal
P_PulmonaleHypertension: 1
Selected P_PulmonaleHypertension: Ernstig

Please fill in the number that represents your data for P_Diabetis :
1)      0
2)      1
P_Diabetis: 1
Selected P_Diabetis: 0

Please fill in the number that represents your data for P_PoorMobility :
1)      0
2)      1
P_PoorMobility: 1
Selected P_PoorMobility: 0

Please fill in the number that represents your data for OPType_pacemakerdraad
tijdelijk :
1)      0
2)      1
OPType_pacemakerdraad tijdelijk: 1
Selected OPType_pacemakerdraad tijdelijk: 0

Please fill in the number that represents your data for Anesthesiologist :
1)      10,00
2)      11,00
3)      12,00
4)      13,00
5)      14,00
6)      15,00
7)      18,00
8)      5,00
9)      6,00
10)     7,00
11)     8,00
12)     9,00
13)     other
Anesthesiologist: 1
Selected Anesthesiologist: 10,00
```

```
Please fill in the number that represents your data for OPType_mvp :
1)      0
2)      1
OPType_mvp: 1
Selected OPType_mvp: 0

Please fill in the number that represents your data for P_Hypercholesterolemia :
1)      0
2)      1
P_Hypercholesterolemia: 1
Selected P_Hypercholesterolemia: 0

Please fill in the number that represents your data for P_activeEndocarditis :
1)      0
2)      1
P_activeEndocarditis: 1
Selected P_activeEndocarditis: 0

Please fill in the number that represents your data for
P_PeripherialVascularDisease :
1)      0
2)      1
P_PeripherialVascularDisease: 1
Selected P_PeripherialVascularDisease: 0

Please fill in the number that represents your data for
P_extracardialArteriopathy :
1)      0
2)      1
P_extracardialArteriopathy: 1
Selected P_extracardialArteriopathy: 0

Please fill in the number that represents your data for Approach :
1)      Volledige sternotomie
2)      other
Approach: 1
Selected Approach: Volledige sternotomie

Please fill in the number that represents your data for OPType_cabg :
1)      0
2)      1
OPType_cabg: 1
Selected OPType_cabg: 0

Please fill in the number that represents your data for AorticSurgery :
1)      0
2)      1
AorticSurgery: 1
```

```
Selected AorticSurgery: 0

Please fill in the number that represents your data for OPType_mvr :
1)      0
2)      1
OPType_mvr: 1
Selected OPType_mvr: 0

Please fill in the number that represents your data for CardiopulmonaryBypassUse
:
1)      0
2)      1
CardiopulmonaryBypassUse: 1
Selected CardiopulmonaryBypassUse: 0

Fill in the numerical value for Euroscore1: 1

Fill in the numerical value for AmountOfBypasses: 1
```

9 5.3 Transform Data According to Preprocessing

[56]:
```python
newDataDF=pd.read_csv("new_data.csv")
newDataDF.head()
```

[56]:
```
   OPType_other  P_Hypertension  OPType_avr  PreviousHeartSurgery  OPType_tvp  \
0             0               0           0                     0           0

  PartOfDay  Sex   Urgency  AtrialFibrillation OperationRoom  ...  OPType_mvp  \
0    Middag    0  Electief                  0          TOK1  ...           0

   P_Hypercholesterolemia  P_activeEndocarditis  P_PeripherialVascularDisease  \
0                       0                     0                             0

   P_extracardialArteriopathy               Approach  OPType_cabg  \
0                           0  Volledige sternotomie            0

  AorticSurgery  OPType_mvr  CardiopulmonaryBypassUse
0             0           0                         0

[1 rows x 35 columns]
```

Encode Labels /One hot encode

[57]:
```python
#Loop through the new data and label encode/onehot encode the data
# based on in what list the columnName appears.
for columnName in newDataDF.columns:
    if columnName in columnNameToLE.keys():
```

```
        print("LE",columnName)
        newDataDF[columnName] = columnNameToLE[columnName].
↪transform(newDataDF[columnName])
    elif columnName in columnNameToOHE.keys():
        print("OHE",columnName)
        #create OHE
        enc=columnNameToOHE[columnName]
        data=newDataDF[columnName]
        hotCodedArray=enc.transform([list(data)]).toarray()
        #Find out columnames
        hotCodedcolumns=[]
        for category in list(enc.categories_[0]):
            hotCodedcolumns+=[columnName+"_"+str(category)]

        #create a new df, using the hot coded columns as Column Names
        hotCoded = pd.DataFrame(hotCodedArray, columns = hotCodedcolumns)

        #remove old data column and add new dataframe
        newDataDF.drop(labels=columnName,axis=COLUMN_AXIS, inplace= True)
        newDataDF = pd.merge(newDataDF, hotCoded, left_index=True,␣
↪right_index=True)
```

```
LE OPType_other
LE P_Hypertension
LE OPType_avr
LE PreviousHeartSurgery
LE OPType_tvp
OHE PartOfDay
LE Sex
LE Urgency
LE AtrialFibrillation
OHE OperationRoom
LE CCS
LE OPType_wondtoilet
LE NYHA
LE MycordialInfarctionPreSurgery
LE OPType_mvp shaving
OHE Surgeon
LE CriticalPre-OP
LE P_ChronicLungDisease
LE P_PulmonaleHypertension
LE P_Diabetis
LE P_PoorMobility
LE OPType_pacemakerdraad tijdelijk
OHE Anesthesiologist
LE OPType_mvp
LE P_Hypercholesterolemia
LE P_activeEndocarditis
```

```
LE P_PeripherialVascularDisease
LE P_extracardialArteriopathy
OHE Approach
LE OPType_cabg
LE AorticSurgery
LE OPType_mvr
LE CardiopulmonaryBypassUse
```

[58]:

```python
#Output of the encoded data.
newDataDF.head()
```

[58]:

```
   OPType_other  P_Hypertension  OPType_avr  PreviousHeartSurgery  OPType_tvp  \
0             0               0           0                     0           0

   Sex  Urgency  AtrialFibrillation  CCS  OPType_wondtoilet  ...  \
0    0        0                   0    0                  0  ...

   Anesthesiologist_15,00  Anesthesiologist_18,00  Anesthesiologist_5,00  \
0                     0.0                     0.0                    0.0

   Anesthesiologist_6,00  Anesthesiologist_7,00  Anesthesiologist_8,00  \
0                    0.0                    0.0                    0.0

   Anesthesiologist_9,00  Anesthesiologist_other  \
0                    0.0                     0.0

   Approach_Volledige sternotomie  Approach_other
0                             1.0             0.0

[1 rows x 60 columns]
```

remove features, that have not been selected

[59]:

```python
#Print the previously filtered features.
print("The selected Features are:", selectedFeatureNames)

#get the feature data
featureData= surgicalData[selectedFeatureNames].copy()
```

```
The selected Features are: ['OPType_other', 'OPType_cabg', 'OPType_avr',
'OPType_pacemakerdraad tijdelijk', 'OPType_mvp', 'OPType_mvp shaving',
'OPType_wondtoilet', 'OPType_tvp', 'OPType_mvr', 'AtrialFibrillation',
'PreviousHeartSurgery', 'P_activeEndocarditis', 'AorticSurgery',
'P_PulmonaleHypertension', 'P_Hypertension', 'CCS', 'NYHA', 'AmountOfBypasses',
'CardiopulmonaryBypassUse']
```

9.1 5.4 Predict new Data

[60]:
```python
# load best model
patient_data=featureData
prediction=bestClassifier.predict(featureData)

#check if it has a bin or not and predict the new duration time.
if PREDICTED_COLUMN in column2binNames.keys():
    print("The assumed value for",PREDICTED_COLUMN,"in this operation is",␣
↪column2binNames[PREDICTED_COLUMN][prediction[0]])
else:
    print("The assumed value for",PREDICTED_COLUMN,"in this operation is",␣
↪prediction[0])
```

```
The assumed value for ActualDurationTime in this operation is 271.7657319295055
```

10

L THE MAIN CODE: ACTUAL DURATION TIME RANGE

(starting on next page)

1 Project: Data Science - Surgical Duration: Actual Duration Time Range

Group 92

Sara-Jane Bittner

Xiao-Lan Bokma

2 Chapter 1 - Setup

[1]:
```
#imports
import os #get the os filepath
import pandas as pd
from collections import Counter

import statsmodels.api as sm
from scipy.stats import zscore

from matplotlib.pyplot import boxplot
from statsmodels.stats.outliers_influence import variance_inflation_factor

from numpy import mean, std, nan, float64, abs, delete, absolute, sqrt, array

from sklearn.preprocessing import MinMaxScaler, LabelEncoder,␣
 ↪OneHotEncoder,LabelBinarizer
from sklearn.metrics import accuracy_score, mean_squared_error,␣
 ↪mean_absolute_error
from sklearn.metrics import f1_score, recall_score, average_precision_score,␣
 ↪balanced_accuracy_score
from sklearn import model_selection, svm, tree
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.dummy import DummyClassifier, DummyRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.cluster import AffinityPropagation, KMeans, MiniBatchKMeans, Birch
from sklearn.mixture import GaussianMixture
from sklearn.neighbors import KNeighborsClassifier

from nltk.probability import FreqDist
from operator import itemgetter
from pprint import pprint
```

[2]:
```
#This will allow our project to allways be executed in the same manor, providing␣
 ↪us with a stable result.
randomSeed=12345
```

3 Chapter 2 - Used functions

The following will provide us with all the functions, that are used in our pipeline in chapter 3

3.0.1 2.1 Import the data

[3]:
```
def import_df_from_file(filename: str):
    """
    Get the path of this file and look into the data folder that is located in␣
 ↪the same folder as this file,
    read the .csv file and delete the last row, as it is empty

    @type filename: str
    @param filename: The filename of the .csv spreadsheet
    @rtype: dataframe
    @returns: dataFrame appearing in the .csv file
    """

    path = os.path.join('data', filename)
    dataFrame=pd.read_csv(path,sep=';',encoding="ISO-8859-1")
    dataFrame.drop(dataFrame.tail(1).index,inplace=True)
    display(dataFrame)
    return dataFrame
```

3.1 2.2 - Preprocessing

3.1.1 2.2.1 Retype the columns

Currently, all the dataframe columns are of type "Object", since the data is in string format. All the columns that are not in this shape, will need to be retyped.

The columns that should be float64 are predefined already. The columns that are not recognized as float64 yet will undergo the modification. To retype the float64 columns, the string 'Onbekend' needs to be replaced with np.nan values.

[4]:
```
def replace_with_nan(dataFrame, value):
    """
    Replace specific a value in this dataframe to a np.nan value.

    @type dataFrame: dataframe
    @param dataFrame: The dataframe with value that needs replacement.
    @type value: string
    @param value: The value that needs to be replaced by np.nan values
    @rtype: dataframe
    @returns: dataFrame with the values replaced by np.nan values.
    """
    #loop through all the columns in the data and replace the value within each␣
 ↪column.
```

```
    for column in dataFrame:
        dataFrame[column] = dataFrame[column].replace([value], nan)
    return dataFrame

def get_non_dtype_columns(dataFrame, dTypeColumnNames, dType):
    """
    Get the columns of the dType list that are not yet of that dtype (yet):

    @type dataFrame: dataframe
    @param dataFrame: The dataframe to get the non_dTypeColumnNames columns from.
    @type dTypeColumnNames: list
    @param dTypeColumnNames: list of the column names that should be of the␣
↪specified dType.
    @type dType: string
    @param dType: the specific dType that the columns should be

    @rtype: list
    @returns: non_dTypeColumns which are not of the specific dtype (yet).
    """
    non_dTypeColumnNames = dataFrame[dTypeColumnNames].
↪select_dtypes(exclude=[dType]).columns
    return non_dTypeColumnNames

def convert_df_type(dataFrame, columns, dType):
    """
    Convert the dtype of specific columns in the dataFrame to the specified␣
↪dType.

    @type dataFrame: dataframe
    @param dataFrame: The dataframe with columns that need to be converted to␣
↪the dType.
    @type columns: list
    @param columns: list of the columns that should be the specified dType.
    @type dType: string
    @param dType: the dtype that the columns should be
    @rtype: dataframe
    @returns: dataFrame with the columns 'columns' to dtype 'dType'
    """

    if (dType == 'float64'):

        print(columns)
        non_dTypeColumnNames = get_non_dtype_columns(dataFrame, columns, dType)

        #change the columns that should be float64 but aren't yet.
        # if the float value cannot be determined due to the decimal
        # separator, commas will be resplaced by periods.
```

```
        for columnName in non_dTypeColumnNames:
            if dataFrame[columnName].dtype=="int8" or dataFrame[columnName].
↪dtype=="int64":
                dataFrame[columnName] = dataFrame[columnName].astype(float)
            else:
                dataFrame[columnName] = dataFrame[columnName].str.replace(',', '.
↪').astype(float)

    #retype all the object columns to category type
    elif (dType == 'category'):
        dataFrame = pd.concat([
        dataFrame.select_dtypes([], ['object']),
        dataFrame.select_dtypes(['object']).apply(pd.Series.astype, dtype=dType)
        ], axis=1)

    #convert the columns that are categories into intergers,
    # based on the provided list of columns
    elif (dType == 'int8'):
        for columnName in columns:
            dataFrame[columnName] = dataFrame[columnName].astype('category').cat.
↪codes
    return dataFrame
```

3.1.2 2.2.2 Remove the Rows and Columns with too much NAN

Drop columns that have more than 50% nan values. Drop the rows that will have one or more missing cells.

[5]:
```
def dropWithTooMuchNan(dataFrame, axis):
    """
    Drop the row/column that contains too many Nan-Values.

    @type dataFrame: dataframe
    @param dataFrame: The dataframe that should have Nan rows/columns removed.
    @param axis: integer
    @param axis: 0 is about the row axis, 1 is the column axis.
    @rtype: dataframe
    @returns: dataFrame with the columns 'columns' to dtype 'dType'
    """

    if axis == COLUMN_AXIS:
        allColumnNames = dataFrame.columns
        NANColumns =[]
        #drop the columns that have missing values
        for columnName in allColumnNames:
            if dataFrame[columnName].isnull().values.any():
                howManyNaN = dataFrame[columnName].isnull().sum()
```

```
                if howManyNaN > ((len(dataFrame[columnName])/100)*50):
                    NANColumns += [columnName]
                    dataFrame.drop(labels=columnName,axis=COLUMN_AXIS,␣
↪inplace= True)
            return dataFrame, NANColumns

    elif axis == ROW_AXIS:
        # drop the rows that have missing values.
        dataFrame.dropna(thresh= len(dataFrame.columns), inplace = True)
        dataFrame.isnull().values.any()
        dataFrame = dataFrame.reset_index(drop=True)

    return dataFrame
```

3.1.3 2.2.3 Creation of new columns

Creating the Overtime column based on the current planned Duration time.

[6]:
```
def createOvertimeColumn(dataFrame):
    """
    Create the Overtime column by substracting ActualDurationTime from the␣
↪PlannedDurationTime

    @type dataFrame: dataframe
    @param dataFrame: The dataframe that containt the two columns that should␣
↪become a new column.
    @rtype: dataframe
    @returns: dataFrame with the extra "Overtime" column.
    """
    dataFrame['Overtime'] = dataFrame['ActualDurationTime'] -␣
↪dataFrame['PlannedDurationTime']
    RANGE_COLUMN_NAMES.append('Overtime')
    ORDINAL_COLUMN_NAMES.append('Overtime')
    return dataFrame
```

Create the OperationType Columns

[7]:
```
#splitting the OperationType Columns-String by the '+' sign.
class OPTypeEncoder:
    relevant_op_types=["other"] #list of the n% most common operations
    relevant_op_types_Names=["OPType_other"]
    n=99

    def __init__(self, n):
        '''main'''
        self.n=n

    def splitOPTypeText(self,text):
```

```
        """
        Split the string by the '+' sign.

        @type text: string
        @param text: the string that needs to be split
        @rtype: list
        @returns: list that contains all the separate operationTypes.
        """
        thisPatientsOPs=str(text).lower().strip().split(" + ")
        return [i.strip() for i in thisPatientsOPs]

    def fit(self,dataframe, columnname):
        """
        Create a list of relevant operationtypes. 99% are included,
            the other 1% is put in the category called 'other'

        @type dataFrame: dataframe
        @param dataFrame: The dataframe that contains the column with strings␣
↪that
                            need to be split.
        @type columnName: string
        @param columnName: the name of the column that contains the strings that
                            need to be split.
        @rtype: list
        @returns: relevant names that appear outside the 1% marge.

        """

        #get a list of all operations, that have ever been done
        completeOPList=[]
        for patient in dataframe[columnname]:
            #print(patient)
            thisPatientsOPs=self.splitOPTypeText(patient)
            completeOPList+=thisPatientsOPs

        #find out how often they have been done
        to_n_uses=FreqDist(completeOPList)
        dictionary_items = to_n_uses.items()
        to_n_uses = sorted(dictionary_items, key=itemgetter(1), reverse=True )

        #get the operations, that together are used in n% percent of the cases
        threshold=(self.n*len(dataframe[columnname]))/100 # Define how many␣
↪cases should be covered by the amount of operations.
        i=0
        for ot in to_n_uses:
            #print(ot)
            if ot[0] == "nan":
```

```
                pass
            else:
                i+=ot[1]
                self.relevant_op_types.append((ot[0]))
                if i>=threshold:
                    break
        print("The relevant op types, that cover at least",str(self.n)+"% of all␣
↪operations, are:", self.relevant_op_types)
        self.relevant_op_types_Names=["OPType_"+i for i in self.
↪relevant_op_types]


    def transform(self,dataframe, columnname):
        """
        Transform the current shape of the dataframe column.
        Add J if this operation was done for this patient,
        Add N if this operation was not done for this patient
        Add nan, if the operation was unknown.


        @type dataFrame: dataframe
        @param dataFrame: The dataframe that contains the column with strings␣
↪that
                            need to be reshaped.
        @type columnName: string
        @param columnName: the name of the column that contains the strings that
                            need to be reshaped.
        @rtype: dataframe
        @returns: dataFrame with the newly shaped OperationType column
        """

        allPatientsOHE=[]

        #add J if this operation was done for this patient,
        #add N if this operation was not done for this patient
        #add nan, if the operation was unknown
        for patient in dataframe[columnname]:
            #print(patient)
            thisPatientsOPs=self.splitOPTypeText(patient)

            #add a column for each relevant operationtype
            thisPatientsOHE=["N"]*len(self.relevant_op_types)
            for op in thisPatientsOPs:
                if op =="nan":
                    thisPatientsOHE=[nan]*len(self.relevant_op_types)
                elif op in self.relevant_op_types:
                    i=self.relevant_op_types.index(op)
```

```
                thisPatientsOHE[i]="J"
            elif not(op in self.relevant_op_types):
                thisPatientsOHE[0]="J"
        allPatientsOHE.append(thisPatientsOHE)

    #turn the results into a dataframe
    allPatientsOHE_DF=pd.DataFrame(allPatientsOHE, columns= self.
↪relevant_op_types_Names)

    #remove old column from df
    dataframe = dataframe.drop(labels = columnname, axis=COLUMN_AXIS)

    #add new columns to df
    dataframe = pd.merge(dataframe, allPatientsOHE_DF, left_index=True,␣
↪right_index=True)

    #edit
    CATEGORICAL_COLUMN_NAMES.remove(columnname)
    SURGERY_FEATURES.remove(columnname)
    for column in self.relevant_op_types_Names:
        BINARY_COLUMN_NAMES.append(column)
        SURGERY_FEATURES.append(column)


    return dataframe

def fit_transform(self,dataframe, columnname):
    """
    Transform the current shape of the dataframe column.
    Add J if this operation was done for this patient,
    Add N if this operation was not done for this patient
    Add nan, if the operation was unknown.


    @type dataFrame: dataframe
    @param dataFrame: The dataframe that contains the column with strings␣
↪that
                    need to be reshaped.
    @type columnName: string
    @param columnName: the name of the column that contains the strings that
                    need to be reshaped.
    @rtype: dataframe
    @returns: dataFrame with the newly shaped OperationType column
    """
    self.fit(dataframe, columnname)
    return self.transform(dataframe, columnname)
```

3.1.4 2.3.4 Outliers

Categorical Outlier Grouping

[8]:
```
def groupOutliers(dataFrame, columnName, treshhold):
    '''
    Group the outlier categories into one group, called 'other'.

    @type dataFrame: dataframe
    @param dataFrame: the dataframe with the categorical column
    @type columnNames: list
    @param columnNames: list of the column names that need to be outlier cut
    @type treshold: float
    @param treshold: percentage that needs to be cut off.

    @rtype dataFrame: dataframe
    @returns dataFram: dataframe with the cut off outliers based on the treshold.
    '''
    treshAmount = (len(dataFrame)//100)*treshhold
    threshCounter = 0
    countValues = dataFrame[columnName].value_counts()
    uniqueValues = dataFrame[columnName].unique()

    for uniqueValue in reversed(uniqueValues):
        if (threshCounter + countValues[uniqueValue]) <= treshAmount:
            threshCounter += countValues[uniqueValue]
            dataFrame[columnName].replace([uniqueValue],'other', inplace=True)

    dataFrame[columnName] = dataFrame[columnName].cat.remove_unused_categories()
    return dataFrame

def categorical_outlier_cutting(dataFrame, columnNames, treshold):
    '''
    Loop through the columnNames that needs to be outliercut.

    @type dataFrame: dataframe
    @param dataFrame: the dataframe with the categorical column
    @type columnNames: list
    @param columnNames: list of the column names that need to be outlier cut
    @type treshold: float
    @param treshold: percentage that needs to be cut off.

    @rtype dataFrame: dataframe
    @returns dataFram: dataframe with the cut off outliers based on the treshold.
    '''
    for column in columnNames:
        dataFrame=groupOutliers(dataFrame, column, treshold)
    return dataFrame
```

Numerical Outlier Cutting

[9]:
```
def numerical_outlier_cutting(dataFrame, numericalColumnNames):
    '''
    Define the numerical outliers, based on the z-scores. Based on the range 2,␣
 ↪the outliers will be cut.

    @type dataFrame: dataFrame
    @param dataFrame: the datafram that contains the columns that need to have␣
 ↪the values outliercut.
    @type numericalColumnNames: list
    @param numericalColumnNames: list of the columnNames with numerical data.

    @rtype dataFrame: dataframe
    @returns dataFrame: dataframe with the numerical outliers cut.
    '''
    numericalData = dataFrame[numericalColumnNames]

    z_scores = zscore(numericalData)
    abs_z_scores = abs(z_scores)
    filtered_entries = (abs_z_scores < 2).all(axis=1)
    dataFrame = dataFrame[filtered_entries]
    dataFrame = dataFrame.reset_index(drop=True)
    return dataFrame
```

3.1.5 2.2.5 Creating categorical data from numerical data ranges.

[10]:
```
def get_bins (columnData, columnName):
    """
    Get the bin values and the according names for those bins.

    @type columnData: list
    @param columnData: The list of values appearing in that column.
    @type columnName: string
    @param columnName: the name of the column that contains the strings that
                        needs to have the bins and the name of the bins.
    @rtype: list (floats), list (strings)
    @returns: the list of 'bins' (values) for the specific columnName and the
                titles that the bins should get.
    """
    minValue = columnData.min()
    maxValue = columnData.max()

    if columnName == "BMI":
        bins = [(minValue-1),18.5,24.9,29.9,maxValue]
        binNames = ['underweight', 'normal', 'overweight', 'obese']
    elif columnName == "Age":
        bins = [int(minValue-1), 57, 66,76, int(maxValue)]
```

```
        binNames = [ (str(x+1)+'-'+str(y)) for x,y in zip(bins[0::1], bins[1::
↪1]) ]
    elif columnName == "Overtime":
        bins = [(minValue-1),(-70),(-25),25,70, 120,(maxValue)]
        binNames = ['Ahead_Of_Time_Extreme','Ahead_Of_Time_Medium',␣
↪'In_Time','Overtime_Small','Overtime_Medium', 'Overtime_Extreme']
    elif columnName == "ActualDurationTime":
        bins = [int(minValue-1), 180, 240, 280, 345,int(maxValue)]
        binNames = [ (str(x+1)+'-'+str(y)) for x,y in zip(bins[0::1], bins[1::
↪1]) ]
    elif columnName == "PlannedDurationTime":
        bins = [int(minValue-1), 180, 240, 280, int(maxValue)]
        binNames = [ (str(x+1)+'-'+str(y)) for x,y in zip(bins[0::1], bins[1::
↪1]) ]
    return bins, binNames

def convert_categorical_range(dataFrame, columnName):
    """
    Convert the current columnName with float data to a columnName + 'Range'
        and the data in bins.


    @type dataFrame: dataframe
    @param dataFrame: The dataframe that contains the column with floatdata
                        that needs to be binned.
    @type columnName: string
    @param columnName: the name of the column that needs to have the data binned.
    @rtype: dataframe, list (string)
    @returns: dataFrame with the column that needs to be binned.
                list with the bin names in strings.
    """

    #get the specific binvalues and the binnames
    bins, binNames = get_bins(dataFrame[columnName], columnName)

    #put the data in bins with according binvalues and the binnames.
    #call the new column the same + 'range'.
    dataFrame[columnName + 'Range'] = pd.cut(dataFrame[columnName], bins,␣
↪labels=binNames)

    #if the column was not the outcome data, then drop it.
    #outcomedata will be kept within the dataset.
    if columnName is not 'PlannedDurationTime' and columnName is not␣
↪'ActualDurationTime':
        dataFrame.drop(labels = columnName, axis=COLUMN_AXIS, inplace = True)
```

```
    #print the values appearing in the new column, bin values, length of the␣
↪binvalues,
    #the bin names and the length of the bin names.
    print(dataFrame[columnName + 'Range'].value_counts())
    print (bins)
    print(len(bins))
    print(binNames)
    print(len(binNames))

    return dataFrame, binNames

def to_categorical_range(dataFrame):
    """
    Loop though the list of names that should be binned.

    @type dataFrame: dataFrame
    @param dataFrame: the dataFrame that will contain the columns that
                       needs to be converted from floats to ranged data.

    @rtype dataFrame, column2binNames: dataFrame, dictionary
    @return dataFrame, column2binNames: dataFrame with the specific float␣
↪columns converted to ranged data.
                                        column2binNames is a dictionary,␣
↪with columnName as key and the
                                        bin names as values.
    """
    column2binNames={}
    for columnName in RANGE_COLUMN_NAMES:
        dataFrame, binNames = convert_categorical_range(dataFrame, columnName)
        column2binNames[columnName+"Range"]=binNames
    return dataFrame, column2binNames
```

```
<>:59: SyntaxWarning: "is not" with a literal. Did you mean "!="?
<>:59: SyntaxWarning: "is not" with a literal. Did you mean "!="?
<>:59: SyntaxWarning: "is not" with a literal. Did you mean "!="?
<>:59: SyntaxWarning: "is not" with a literal. Did you mean "!="?
<ipython-input-10-8fc6ea43bdb3>:59: SyntaxWarning: "is not" with a literal. Did
you mean "!="?
  if columnName is not 'PlannedDurationTime' and columnName is not
'ActualDurationTime':
<ipython-input-10-8fc6ea43bdb3>:59: SyntaxWarning: "is not" with a literal. Did
you mean "!="?
  if columnName is not 'PlannedDurationTime' and columnName is not
'ActualDurationTime':
```

3.1.6 2.2.6 Encoding

Label Encoding

[11]:
```python
def label_encoding(dataFrame, columns):
    """
    Convert the ORDINAL categorical columns into label encoded columns.

    @type dataFrame: dataframe
    @param dataFrame: The dataframe that contains ORDINAL categorical data.
    @type columns: list
    @param columns: list of column names that contain ORDINAL categorical data.
    @rtype dataFrame, columnNameToLE: dataframe, dictionary
    @returns dataFrame, columnNameToLE: new dataframe with the ordinal␣
↪categorical columns label encoded.
                              dictionary with column name as key and the category␣
↪names as values.
    """

    columnNameToLE={}

    #loop through the columns that needs to be label encoded.
    for columnName in columns:
        le = LabelEncoder()
        le = le.fit(dataFrame[columnName])
        dataFrame[columnName] = le.transform(dataFrame[columnName])
        columnNameToLE[columnName]=le

    return dataFrame, columnNameToLE
```

One Hot Encoding

[12]:
```python
def IntTransfer(dataFrame, columnName):
    '''
    Make string type floats into int.
    When the decimal separator is ',' replace it with a '.' first.

    @type dataFrame: dataframe
    @param dataFrame:
    @type columnName: string
    @param columnName:

    @rtype dataFrame: dataframe
    @returns dataFrame:

    '''
    for i in range(0,len(dataFrame[columnName])):
        try:
            dataFrame[columnName][i]= int(float(dataFrame[columnName][i].
↪replace(",",".")))
        except ValueError:
```

```
            dataFrame[columnName][i]= dataFrame[columnName][i]

    return dataFrame


def one_hot_encoding(columnNames, dataFrame):
    '''
    Onehot-encodes NOMINAL categorical columns of the dataFrame, which are␣
 ↪listed in columnNames

    @type dataFrame: dataframe
    @param dataFrame: dataframe that contains the data that needs to be onehot␣
 ↪encoded.
    @type columnNames: list
    @param columnNames: list of column names that need to be onehot encoded

    @rtype dataFrame, columnNameToOHE.keys(), columnNameToOHE: dataframe, list,␣
 ↪dictionary
    @returns dataFrame, columnNameToOHE.keys(), columnNameToOHE:
                   dataframe with the columndata onehot encoded.
                   the keys of the onehot encoded categories
                   the full dictionary with the old columnName as key and the␣
 ↪category names as values.
    '''


    columnNameToOHE={}

    #for each column, which needs to be OHEd create an OH-Encoder and the␣
 ↪corresponding Encoding. Insert it in the Dataframe
    for columnName in columnNames:
        print(columnName)
        dataFrame = IntTransfer(dataFrame,columnName)
        hotCodedArray=array(dataFrame[columnName][:]).reshape(-1, 1)

        #create an OH-Encoder and the corresponding Encoding

        #enc= LabelBinarizer()
        enc = OneHotEncoder(handle_unknown='ignore')
        hotCodedArray=enc.fit_transform(hotCodedArray).toarray()
        columnNameToOHE[columnName]=enc

        #Find out columnames
        hotCodedcolumns=[]
        for category in list(enc.categories_[0]):
            hotCodedcolumns+=[columnName+"_"+str(category)]
```

```
        #create a new df, using the hot coded columns as Column Names
        hotCoded = pd.DataFrame(hotCodedArray, columns = hotCodedcolumns)

        #remove old data column and add new dataframe
        dataFrame.drop(labels=columnName,axis=COLUMN_AXIS, inplace= True)
        dataFrame = pd.merge(dataFrame, hotCoded, left_index=True,␣
 ↪right_index=True)

    return dataFrame, columnNameToOHE.keys(), columnNameToOHE
```

Calculate the VIF for One Hot encoding

[13]:
```
def calculateVIF(df, featuresToTest):
    """
    Calculate the VIF values for the dataframe, from the list of features that␣
 ↪needs to be tested.

    @type df: dataframe
    @param df: dataframe with the features of the featurelist of which
                    VIF values need to be calculated from.
    @type featuresToTest: list
    @param featuresToTest: feature list of column names that have data which
                    VIF values need to be calculated from.
    @rtype vif_data: dataframe
    @returns vif_data: dataframe with the vif_data.
    """



    #Filters the whole dataset for the feature set of one OneHotEncoded-Column
    independentVariables = df.filter(featuresToTest)

    #print(independentVariables.head())
    X = independentVariables

    # VIF dataframe
    vif_data = pd.DataFrame()
    vif_data["feature"] = X.columns

    # calculating VIF for each feature
    vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in␣
 ↪range(len(X.columns))]
    return vif_data

def checkForMultiCol(df, featuresToTest):
    """
    Check for multicolinearity in the dataframe with the featuresToTest list.
```

```
    @type df: datafram
    @param df: dataframe with the features of the featurelist of which
                    multicolinearity needs ot be tested upon.
    @type featuresToTest: list
    @param featuresToTest: feature list of column names that have data which
                    multicolinearity needs ot be tested upon.
    @rtype multicols: list
    @returns multicols: list of features which have mutlicolinearity appear.
    """

    multicols =[]
    #calculate the Variance Inflation Factor and gives a table with all VIF with␣
 ↪the features back
    vifData = calculateVIF(df, featuresToTest)
    #Goes through the table and checks for multicollinearity
    for i in range(0,len(vifData)-1):
            # >= 5 because from 5 multicollinearity is there
            if vifData['VIF'][i] >= 5:
                print(vifData['VIF'][i])
                #saves features that are multicollinearity in list
                multicols +=[vifData['feature'][i]]
    return multicols

def multi_col_test(dataFrame):
    """
    Check for multicolinearity in the OneHtEncoded data.

    @type dataFrame: dataframe
    @param dataFrame: dataframe with the onehot encoded data.

    @rtype: boolean
    @returns: boolean value whether the multicolinearity appeared in the data or␣
 ↪not.
    """
    #Test if there is Multicollinearity in the OneHotEncoding
    multicols =[]
    for i in range(0,len(CATEGORICAL_COLUMN_NAMES)):
        #print(list(hotCodedNames[i]))
        multicols +=checkForMultiCol(dataFrame,␣
 ↪list(CATEGORICAL_COLUMN_NAMES[i]))
    if (len(multicols)==0):
        return True
    else:
        return False
```

3.2 2.2.7 Normalization

Normalize: - Numerical. - label encoded data.

Don't normalize: - Binary Data . - One Hot Encoded data.

[14]:
```
def normalize_data(dataFrame):
    """
    Normalize data in the dataframe for the columns that appear in the column␣
↪names list.

    @type dataFrame: dataframe
    @param dataFrame: dataframe that contains the not-normalized data for the␣
↪column names list.

    @rtype dataFrame: dataframe
    @returns dataFrame: the new dataframe with normalized values for the columns␣
↪in the column names list.

    """

    normalizationData = dataFrame[NORMALIZATION_COLUMN_NAMES]

    x = normalizationData.values #returns a numpy array
    min_max_scaler = MinMaxScaler()
    x_scaled = min_max_scaler.fit_transform(x)
    normalizedDf = pd.DataFrame(x_scaled, columns=NORMALIZATION_COLUMN_NAMES)
    dataFrame.drop(labels = NORMALIZATION_COLUMN_NAMES, inplace=True, axis = 1)
    dataFrame = pd.merge(dataFrame, normalizedDf, left_index=True,␣
↪right_index=True)
    return dataFrame
```

3.3 2.3 -Feature Selection

3.3.1 2.3.1 New Feature Split

[15]:
```
def get_feature_data(dataFrame, featureColumnNames):
    """
    Get the feature data form the dataframe with the  column names of those␣
↪features.

    @type dataFrame: dataframe
    @param dataFrame: the dataframe that contains the featuredata for the␣
↪featurenames in the list.

    @rtype featureData: dataframe
    @returns featureData: stripped dataFrame from the original dataframe, that␣
↪now contains only
```

```
                            the data from the feature column names.
    """

    #drop the outcome features in the featureData list.
    featureData = dataFrame.drop(labels = OUTCOME_COLUMN_NAMES, axis =␣
 ↪COLUMN_AXIS)

    #filter out the columnNames from a specific pool of columnNames:
    # - surgery features or patient features
    featureData = featureData.filter(featureColumnNames)

    return featureData

def backward_elimination(X, y):
    """
    Use backward elimination on X and y.

    @type X: list
    @param X: x data of the model.
    @type y: list
    @param y: y data corresponding to the x data in the model.

    @rtypes featuresDataSelected, list(selectedFeaturesColumnNames): list, list
    @returns featuresDataSelected, list(selectedFeaturesColumnNames):
            The featuredata that had an significant impact on the outcomedata and
            column names.

    """

    #Adding constant column of ones, mandatory for sm.OLS model
    X_1 = sm.add_constant(X)

    model = sm.OLS(y,X_1).fit()

    #Extract columnsNames
    columnNames = list(X.columns)
    SIGNIFICANCE = 0.05

    #Set the pmax variable to 1. Will be replaced in while loop.
    pmax = 1

    #while the length of the columnNames is not empty (yet).
    #wrapper methods:
    #     Backwards elimination = single features only.
    #     Filter method is = combination of features.
    while (len(columnNames)>0):
```

```
        # initialize an empty list for all the pValues.
        p= []

        # make the statistical model and fit it to the data.
        X_1 = X[columnNames]
        X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE␣
↪AND FIX IT!
        model = sm.OLS(y,X_1).fit()

        new_arr = delete(model.pvalues.values, (len(model.pvalues.values)-1))

        #make a dataframe with the columnNames and their corresponding P values.
        p = pd.Series(new_arr,index = columnNames)

        # extract the maximum P value of the column.
        pmax = max(p)
        feature_with_p_max = p.idxmax()

        # compare the p value with the significance value.
        # if the pmax is bigger than the significance value, the specific
        # feature is insignificant and removes it from the list.
        if(pmax>SIGNIFICANCE):
            #print(feature_with_p_max)
            columnNames.remove(feature_with_p_max)

        # if there is no maximum p-value that is insignificant anymore, the␣
↪while loop will break.
        else:
            break

    selectedFeatures = columnNames

    featuresDataSelected = X.filter(selectedFeatures)
    selectedFeaturesColumnNames = featuresDataSelected.columns

    return featuresDataSelected, list(selectedFeaturesColumnNames)
```

[16]:
```
def get_target_names_string(yColumn):
    """
    Get the targetnames of the outcomedata. Turn it into one string.

    @type yColumn: list
    @param yColumn: list of outcomedata.

    @rtype targetNamesString: string
    @returns targetNamesString: one string with all the outcomedata.
    """
```

```
    #initialize which column will be the y in the model.
    #print(yColumn.unique())

    #initialize the list for the targetnames.
    targetNames = list(yColumn.unique())

    #initalize for the classification report
    targetNamesString = [str(i) for i in targetNames]

    return targetNamesString
```

3.3.2 2.3.2 Inverted Indexes Method

[17]:
```
def add_index_to_value_column(temp_dict, column, value, index):
    """
    Add the index of a row to a vector value in the dictionary.
    If the in vector value is not a key yet, it creates a new key
    and adds it.

    @type temp_dict: dictionary
    @param temp_dict: contains the vector values as keys and
                        has the row indexes as values.
    @type column: string
    @param column: name of the certain column in which the vector values appear.
    @type value: string/float
    @param value: vector value that the row(index) contains for that specific␣
 ↪column.
    @type index: integer
    @param index: the index of the row in the dataset.

    @rtype: dictionary
    @returns: the dictionary with the new value.
    """

    if value not in temp_dict[column]: temp_dict[column][value] = []

    #add the index to the specific vector value of that column
    temp_dict[column][value].extend(index)
    return temp_dict

def create_ii(X_array):
    """
    Enumerate through the while X training set.
    Make entries (keys) for all the vector values appearing in the training set.

    @type X_array: array
```

```
    @param X_array: X data of the training set.

    @rtype ii: dictionary
    @returns ii: a dictionary with the vector values as keys and
                the rowindexes as dctionary values.
    """

    ii = None

    #create inverted index
    for index, vector in enumerate(X_array):
        #build inverted index and make slots for the columns
        if ii == None: ii = [{} for _ in vector]

        #for every column in this rowVector
        for column, value in enumerate(vector):
            ii = add_index_to_value_column(ii, column, value, [index])
    return ii

def retrieve_vectors_ii(ii, query_vector):
    """
    Counts for the query vector the amount of times a certain
    row appears for each vector value: the score about how
    similar those rows are. Then returns the index or indices
    that appear to be the most similar.

    @type ii: dictionary
    @param ii: a dictionary with the vector values as keys and
                the rowindexes as dctionary values.
    @type query_vector: list
    @param query_vector: the list of feature values as a
                'query vector'. This is one row of the
                the x test data.

    @rtype max_keys: list
    @returns max_keys: a list of indices that are most similar to
                the query vector.
    """

    scores = {}

    #for every column in the query vector
    for qcolumn, qvalue in enumerate(query_vector):

        #count the map indexes(documents) containing this word and how many␣
 ↪times document appears in ii.
        if qvalue in ii[qcolumn]:
```

```python
            #get the indices that have this column value
            ret_indices = ii[qcolumn][qvalue]

            #add the indices to the score for each appearing indices.
            #it counts for every column the reoccuring indices.
            for i in ret_indices:
                if i not in scores:
                    scores[i] = 1
                else:
                    scores[i] += 1

    # find the highest score: the score for the indices that
    # are most similar.
    # Retrieve the indices with this score: most similar to the query vector.
    max_val = max(scores.values())
    max_keys = [key for key in scores if scores[key] == max_val]
    return max_keys

def most_frequent(List):
    """
    Returns the value that appears the most in this list.

    @type List: list
    @param List: list of which the most appearing
                needs to be calculated.
    @rtype: value of the list
    @returns: the most frequent value of the list.
    """

    return max(set(List), key = List.count)

def retrieve_y_values(indices, y_array):
    """
    Retrieves the most appearing y_value out of the given indices.

    @type indices: list
    @param indices: pre-calculated indices of the X train
                    set that are most similar to a query index.
    @type y_array: list
    @param y_array: outcome data from the training set.

    @rtype y_value: string/float
    @returns y_value: the y_train-value corresponding to the most
                    similar index/indices to the query row.
    """
    count = 0
```

```
    #make a list with the length of the amount of retrieved indices
    y_value = [None]*len(indices)

    #count for each appearing index, how many times it appears in total.
    for index in indices:
        y_value[count] = y_array[index]
        count += 1

    return y_value


def predict_y_ii (ii, X_array, y_array):
    """
    Predict the outcome values for the X test set,
    with the inverted indexes dictionary and the
    y training data.

    @type ii: dictionary
    @param ii: a dictionary with the vector values as keys and
                the rowindexes as dctionary values.

    @type X_array: list
    @param X_array: x test data
    @type y_array: list
    @param y_array: y training data

    @rtype pred: list
    @returns pred: the predicted values for this x test set.
    """

    pred = [None]*len(X_array)

    for i, qvector in enumerate(X_array):

        #retrieve the similar indices for this query vector
        most_sim_indices = retrieve_vectors_ii(ii, qvector)

        #retrieve the according y values for the most appearing indices.
        y_values = retrieve_y_values(most_sim_indices, y_array)

        prediction = most_frequent(y_values)
        pred[i] = prediction
    return pred
```

3.4 2.4 Evaluate the models

[18]:
```
def reportPerformance(groundTruth, predictions, report_mode):
    """#reports the quality of the prediction.
    #the value that determines the best classifier, depends on the report_mode
    "continuos" --> RMSE
    "categorical" --> Accuracy

    if report_mode == "continuos" the accuracy/F1-Scores is not computed, to␣
↪avoid problems with regression based classifiers
    """
    if report_mode != "continuos":
        #printing the classification report of the performance
        accuracy=accuracy_score(groundTruth, predictions)
        f1macro=f1_score(groundTruth, predictions, average='macro')
        f1weighted=f1_score(groundTruth, predictions, average='weighted')
        print ('Accuracy score \t %0.3f'%(accuracy))
        print ('F1 Macro score \t %0.3f'%(f1macro))
        print ('F1 Weighted sc \t %0.3f'%(f1weighted))

    MAE=mean_absolute_error(groundTruth, predictions)
    RMSE=sqrt(mean_squared_error(groundTruth, predictions))

    #     print ('Recall \t{}'.format(recall_score(groundTruth, predictions,␣
↪average = 'weighted')))
    #     print ('Average prec \t{}'.format(average_precision_score(groundTruth,␣
↪predictions)))
    print ('MAE error \t %0.3f'%(MAE))
    print ('RMSE error \t %0.3f'%(RMSE))
    print ()
    if report_mode == "continuos":
        return -RMSE
    else:
        return accuracy
```

4 Chapter 3 - The Pipeline

4.1 3.0 Preparation

Constants

[19]:
```
FILENAME = 'surgical_case_durations.csv'        # The Filename of the csv file,␣
↪that we are using.
COLUMN_AXIS = 1                                 #
ROW_AXIS = 0                                    #
```

Here you can choose what you want to predict. Choose between:

'ActualDurationTime'

'ActualDurationTimeRange'

[20]:
```
# PREDICTED_COLUMN='ActualDurationTime'
PREDICTED_COLUMN=input("Please, Type in your choice 'ActualDurationTime' or␣
↪'ActualDurationTimeRange': ")    # The Column that we want to predict using␣
↪this pipeline. Can be: ActualDurationTimeRange or ActualDurationTime
```

```
Please, Type in your choice 'ActualDurationTime' or 'ActualDurationTimeRange':
ActualDurationTimeRange
```

Lists and Dictionaries

[21]:
```
#Dictionary for the column renamings
TO_ENGLISH_COLUMNS={"Geplande operatieduur": "PlannedDurationTime",
                    "Operatieduur": "ActualDurationTime",
                    "Operatietype":"OperationType",
                    "Chirurg":"Surgeon",
                    "Anesthesioloog":"Anesthesiologist",
                    "Benadering":"Approach",
                    "OK":"OperationRoom",
                    "Casustype":"Urgency",
                    "Dagdeel":"PartOfDay",
                    "Leeftijd":"Age",
                    "AF":"AtrialFibrillation",
                    "HLM":"CardiopulmonaryBypassUse",
                    "Geslacht":"Sex",
                    "Aantal anastomosen":"AmountOfBypasses",
                    "Chronische longziekte":"P_ChronicLungDisease",
                    "Extracardiale vaatpathie":"P_extracardialArteriopathy",
                    "Actieve endocarditis": "P_activeEndocarditis",
                    "Hypertensie":"P_Hypertension",
                    "Pulmonale hypertensie":"P_PulmonaleHypertension",
                    "Slechte mobiliteit":"P_PoorMobility",
                    "Hypercholesterolemie":"P_Hypercholesterolemia",
                    "Perifeer vaatlijden":"P_PeripherialVascularDisease",
                    "Linker ventrikel functie":"LeftVentricleFunction",
                    "Nierfunctie":"RenalFunction",
                    "DM":"P_Diabetis",
                    "Eerdere hartchirurgie":"PreviousHeartSurgery",
                    "Kritische preoperatieve status":"CriticalPre-OP",
                    "Myocard infact <90 dagen":"MycordialInfarctionPreSurgery",
                    "Aorta chirurgie":"AorticSurgery",
                    "Ziekenhuis ligduur":"HospitalDays",
                    "IC ligduur":"ICUDays"
                   }

#All features, that are related to the patient
PATIENT_FEATURES= ['Urgency', 'Sex',
       'AtrialFibrillation', 'P_ChronicLungDisease',
```

```
        'P_extracardialArteriopathy', 'PreviousHeartSurgery',
        'P_activeEndocarditis', 'CriticalPre-OP',
        'MycordialInfarctionPreSurgery', 'AorticSurgery',
        'P_PulmonaleHypertension', 'LeftVentricleFunction', 'Euroscore1',
        'Euroscore2', 'RenalFunction', 'P_PoorMobility', 'P_Diabetis',
        'P_Hypercholesterolemia', 'P_Hypertension',
        'P_PeripherialVascularDisease', 'CCS', 'NYHA', 'AmountOfBypasses',
        'CardiopulmonaryBypassUse']

#Features that are related to the surgery
SURGERY_FEATURES =[ 'Surgeon', 'Anesthesiologist', 'Approach',
        'OperationRoom','PartOfDay', 'OperationType']
#________________________________________

RANGE_COLUMN_NAMES = ['Age', 'BMI', 'ActualDurationTime',
                        'PlannedDurationTime'] #and overtime

ORDINAL_COLUMN_NAMES = RANGE_COLUMN_NAMES + ['CCS', 'NYHA', 'Urgency',
                        'LeftVentricleFunction', 'RenalFunction',
                        'P_PulmonaleHypertension']

NON_ORDINAL_COLUMN_NAMES = ['AmountOfBypasses', 'Euroscore1',
                            'Euroscore2', 'HospitalDays', 'ICUDays']

NUMERICAL_COLUMN_NAMES = RANGE_COLUMN_NAMES + NON_ORDINAL_COLUMN_NAMES

CATEGORICAL_COLUMN_NAMES = ['Surgeon', 'OperationRoom', 'OperationType',
                            'Anesthesiologist', 'Approach', 'PartOfDay']
#________________________________________

BINARY_COLUMN_NAMES = ['Sex', 'AtrialFibrillation', 'P_ChronicLungDisease',
                        'P_extracardialArteriopathy', 'PreviousHeartSurgery',
                        'P_activeEndocarditis', 'CriticalPre-OP',
                        'MycordialInfarctionPreSurgery', 'AorticSurgery',
                        'P_PoorMobility', 'P_Diabetis', 'P_Hypercholesterolemia',
                        'P_Hypertension', 'P_PeripherialVascularDisease',
                        'CardiopulmonaryBypassUse']

NORMALIZATION_COLUMN_NAMES = ['CCS', 'NYHA', 'Urgency', 'BMIRange',
                       'LeftVentricleFunction', 'RenalFunction',
                       'P_PulmonaleHypertension', 'AgeRange', 'Euroscore1',
                       'Euroscore2']

OUTCOME_COLUMN_NAMES = ['HospitalDays', 'ICUDays', 'PlannedDurationTime',
                        'PlannedDurationTimeRange', 'OvertimeRange',
                        'ActualDurationTimeRange','ActualDurationTime']
```

4.2 3.1 Importing the Data

[22]:
```
# Import the data from the file: 'surgical_case_durations.csv'. Store it in a␣
↪dataframe
surgicalData = import_df_from_file(FILENAME)
```

```
                                         Operatietype            Chirurg  \
0                            Amputatie teen + Wondtoilet               6,00
1     Arthroscopische acrominoclaviculaire reconstru...  Ander specialisme
2                                   Ascendensvervanging               4,00
3                                   Ascendensvervanging               7,00
4                                   Ascendensvervanging               6,00
...                                                 ...                ...
4081                                        Wondtoilet               2,00
4082                                        Wondtoilet               4,00
4083                                        Wondtoilet               4,00
4084                                        Wondtoilet              15,00
4085                                        Wondtoilet               1,00

     Anesthesioloog             Benadering     OK       Casustype  Dagdeel  \
0          Onbekend                    NaN   TOK3        Electief   Middag
1          Onbekend                    NaN  OK 10        Electief   Middag
2              8,00  Volledige sternotomie   TOK1        Electief  Ochtend
3              6,00  Volledige sternotomie   TOK2        Electief   Middag
4          Onbekend                    NaN   TOK2           Spoed    Avond
...             ...                    ...    ...             ...      ...
4081       Onbekend                    NaN   TOK1        Electief   Middag
4082       Onbekend                    NaN   TOK3        Electief   Middag
4083       Onbekend                    NaN   TOK3  Spoed < 24 uur  Ochtend
4084       Onbekend                    NaN   TOK2        Electief   Middag
4085       Onbekend                    NaN   TOK1        Electief  Ochtend

      Leeftijd Geslacht   AF  ... Hypertensie Perifeer vaatlijden  CCS NYHA  \
0         51.0      NaN  NaN  ...         NaN                 NaN  NaN  NaN
1         50.0      NaN  NaN  ...         NaN                 NaN  NaN  NaN
2         78.0        V    N  ...           N                   J  0.0  2.0
3         66.0        V    J  ...           N                   N  2.0  1.0
4         72.0      NaN  NaN  ...         NaN                 NaN  NaN  NaN
...        ...      ...  ...  ...         ...                 ...  ...  ...
4081      62.0      NaN  NaN  ...         NaN                 NaN  NaN  NaN
4082      62.0      NaN  NaN  ...         NaN                 NaN  NaN  NaN
4083      62.0      NaN  NaN  ...         NaN                 NaN  NaN  NaN
4084      57.0      NaN  NaN  ...         NaN                 NaN  NaN  NaN
4085      64.0      NaN  NaN  ...         NaN                 NaN  NaN  NaN

     Aantal anastomosen  HLM Geplande operatieduur Operatieduur  \
0                   NaN  NaN                  62.0         52.0
1                   NaN  NaN                  85.0         70.0
```

```
2                    0.0    N                 229.0        170.0
3                    0.0    N                 140.0        190.0
4                    NaN  NaN                 370.0        480.0
...                  ...  ...                   ...          ...
4081                 NaN  NaN                  78.0         65.0
4082                 NaN  NaN                  60.0         25.0
4083                 NaN  NaN                  46.0         39.0
4084                 NaN  NaN                  60.0         46.0
4085                 NaN  NaN                  53.0         49.0

     Ziekenhuis ligduur IC ligduur
0              Onbekend   Onbekend
1              Onbekend   Onbekend
2                  4,00       2,00
3                  6,00       1,00
4              Onbekend   Onbekend
...                 ...        ...
4081              44,00       0,00
4082           Onbekend   Onbekend
4083           Onbekend   Onbekend
4084           Onbekend   Onbekend
4085           Onbekend   Onbekend

[4086 rows x 36 columns]
```

4.3 3.2 Preprocessing

4.3.1 3.2.0 - Translate column-names to english

[23]:
```
# translate the columns to english
surgicalData = surgicalData.rename(columns = TO_ENGLISH_COLUMNS)
```

4.3.2 3.2.1 Add new Columns

Overtime Column Creating the Overtime Column based on the planned- and actual duration time.

[24]:
```
# create the overtime column
surgicalData = createOvertimeColumn(surgicalData)
```

OperationType The OperationType column has a string with all the operations performed on the subject. The operations, that make up a to over 99% of the total operations are included

[25]:
```
OperationTypeEncoder=OPTypeEncoder(n=99)
surgicalData=OperationTypeEncoder.fit_transform(surgicalData, 'OperationType')
```

```
The relevant op types, that cover at least 99% of all operations, are: ['other',
'cabg', 'avr', 'pacemakerdraad tijdelijk', 'mvp', 'mvp shaving', 'wondtoilet',
```

'tvp', 'mvr']

4.3.3 3.2.2 Remove NaNs

[26]:
```
#in the dataset we have some "onbekend"-values, that should be NaNs too.
surgicalData = replace_with_nan (surgicalData, value = 'Onbekend')

#drop columns and rows with too much nan
surgicalData, droppedColumns = dropWithTooMuchNan(surgicalData, axis =␣
↪COLUMN_AXIS)

#remove the columnnames in the lists that are removed from the dataset as well.
for removeColumnName in droppedColumns:
    if removeColumnName in RANGE_COLUMN_NAMES:
        RANGE_COLUMN_NAMES.remove(removeColumnName)
    if removeColumnName in ORDINAL_COLUMN_NAMES:
        ORDINAL_COLUMN_NAMES.remove(removeColumnName)
    if removeColumnName in NON_ORDINAL_COLUMN_NAMES:
        NON_ORDINAL_COLUMN_NAMES.remove(removeColumnName)
    if removeColumnName in NUMERICAL_COLUMN_NAMES:
        NUMERICAL_COLUMN_NAMES.remove(removeColumnName)
    if removeColumnName in CATEGORICAL_COLUMN_NAMES:
        CATEGORICAL_COLUMN_NAMES.remove(removeColumnName)
    if removeColumnName in BINARY_COLUMN_NAMES:
        BINARY_COLUMN_NAMES.remove(removeColumnName)
    if removeColumnName in NORMALIZATION_COLUMN_NAMES:
        NORMALIZATION_COLUMN_NAMES.remove(removeColumnName)

surgicalData = dropWithTooMuchNan(surgicalData, axis = ROW_AXIS)
```

4.3.4 3.2.3 Retype the data

[27]:
```
# retype Object columns to categorical columns, if possible
surgicalData = convert_df_type(surgicalData, list(surgicalData), 'category')

# retype the columns that are not float64 yet
surgicalData = convert_df_type(surgicalData, NUMERICAL_COLUMN_NAMES, 'float64')

# retype the columns that are not integers yet
surgicalData = convert_df_type(surgicalData, BINARY_COLUMN_NAMES, 'int8')
```

```
['Age', 'BMI', 'ActualDurationTime', 'PlannedDurationTime', 'AmountOfBypasses',
'Euroscore1', 'HospitalDays', 'ICUDays']
```

4.3.5 3.2.4 Outlier removal

Outlier cutting for the categorical outliers: those outliers will be grouped into "other", while the numerical outliers will be removed from the dataset.

[28]:
```
#categoricalOutlier Cutting
surgicalData = categorical_outlier_cutting(surgicalData,␣
 ↪CATEGORICAL_COLUMN_NAMES, 10)
# #numericalOutlier Cutting
surgicalData = numerical_outlier_cutting(surgicalData, NUMERICAL_COLUMN_NAMES)
```

[29]:
```
#Needed for the Duration Generator --> Chapter 4
priorData=surgicalData[:]
surgicalData.head()
```

[29]:
```
    Age  CCS  NYHA  AmountOfBypasses  PlannedDurationTime  ActualDurationTime  \
0  66.0  2.0   1.0               0.0                140.0               190.0
1  71.0  0.0   1.0               0.0                241.0               239.0
2  66.0  0.0   1.0               0.0                240.0               269.0
3  52.0  0.0   1.0               0.0                180.0               305.0
4  80.0  2.0   2.0               0.0                218.0               300.0

   Overtime Surgeon Anesthesiologist               Approach  ... ICUDays  \
0      50.0    7,00             6,00 Volledige sternotomie  ...     1.0
1      -2.0    4,00            10,00 Volledige sternotomie  ...     1.0
2      29.0    3,00            15,00 Volledige sternotomie  ...     0.0
3     125.0    1,00            11,00 Volledige sternotomie  ...     1.0
4      82.0    2,00             5,00 Volledige sternotomie  ...     4.0

  OPType_other OPType_cabg  OPType_avr  OPType_pacemakerdraad tijdelijk  \
0            0           1           1                                1
1            0           1           1                                1
2            0           1           1                                1
3            0           1           1                                1
4            0           1           0                                1

   OPType_mvp  OPType_mvp shaving  OPType_wondtoilet  OPType_tvp  OPType_mvr
0           1                   1                  0           1           1
1           1                   1                  0           1           1
2           1                   1                  0           1           1
3           1                   1                  0           1           1
4           1                   0                  0           1           1

[5 rows x 42 columns]
```

4.3.6 3.2.5 Categorical Data

Creating categorical data from ranged numerical data

[30]:
```
surgicalData, column2binNames = to_categorical_range(surgicalData)

#replace the current category name with the categoryname+'Range', it the lists.
```

```
for removeColumnName in RANGE_COLUMN_NAMES:
    replaceColumnName = removeColumnName+'Range'
    if removeColumnName in RANGE_COLUMN_NAMES:
        RANGE_COLUMN_NAMES = list(map(lambda x: x.replace(removeColumnName,␣
↪replaceColumnName), RANGE_COLUMN_NAMES))
    if removeColumnName in ORDINAL_COLUMN_NAMES:
        ORDINAL_COLUMN_NAMES = list(map(lambda x: x.replace(removeColumnName,␣
↪replaceColumnName), ORDINAL_COLUMN_NAMES))
    if removeColumnName in NON_ORDINAL_COLUMN_NAMES:
        NON_ORDINAL_COLUMN_NAMES = list(map(lambda x: x.
↪replace(removeColumnName, replaceColumnName), NON_ORDINAL_COLUMN_NAMES))
    if removeColumnName in NUMERICAL_COLUMN_NAMES:
        NUMERICAL_COLUMN_NAMES = list(map(lambda x: x.replace(removeColumnName,␣
↪replaceColumnName), NUMERICAL_COLUMN_NAMES))
    if removeColumnName in CATEGORICAL_COLUMN_NAMES:
        CATEGORICAL_COLUMN_NAMES = list(map(lambda x: x.
↪replace(removeColumnName, replaceColumnName), CATEGORICAL_COLUMN_NAMES))
    if removeColumnName in BINARY_COLUMN_NAMES:
        BINARY_COLUMN_NAMES = list(map(lambda x: x.replace(removeColumnName,␣
↪replaceColumnName), BINARY_COLUMN_NAMES))
    if removeColumnName in NORMALIZATION_COLUMN_NAMES:
        NORMALIZATION_COLUMN_NAMES = list(map(lambda x: x.
↪replace(removeColumnName, replaceColumnName), NORMALIZATION_COLUMN_NAMES))
```

```
67-76    694
58-66    417
77-87    315
47-57    244
Name: AgeRange, dtype: int64
[46, 57, 66, 76, 87]
5
['47-57', '58-66', '67-76', '77-87']
4
overweight     819
normal         466
obese          385
underweight      0
Name: BMIRange, dtype: int64
[17.91, 18.5, 24.9, 29.9, 36.1]
5
['underweight', 'normal', 'overweight', 'obese']
4
181-240    665
241-280    443
281-345    305
110-180    192
346-398     65
```

```
Name: ActualDurationTimeRange, dtype: int64
[109, 180, 240, 280, 345, 398]
6
['110-180', '181-240', '241-280', '281-345', '346-398']
5
181-240    1043
241-280     408
140-180     119
281-330     100
Name: PlannedDurationTimeRange, dtype: int64
[139, 180, 240, 280, 330]
5
['140-180', '181-240', '241-280', '281-330']
4
In_Time                  633
Overtime_Small           398
Ahead_Of_Time_Medium     312
Overtime_Medium          164
Ahead_Of_Time_Extreme     94
Overtime_Extreme          69
Name: OvertimeRange, dtype: int64
[-165.0, -70, -25, 25, 70, 120, 215.0]
7
['Ahead_Of_Time_Extreme', 'Ahead_Of_Time_Medium', 'In_Time', 'Overtime_Small',
'Overtime_Medium', 'Overtime_Extreme']
6
```

4.3.7 3.2.6 Encoding

Label Encoding This is done for numerical data that is ordinal.

[31]:
```
#create one list of column names that needs to be label encoded
toBeLE=ORDINAL_COLUMN_NAMES+BINARY_COLUMN_NAMES
#label encode the binary data and the ordinal categorical columns
surgicalData,columnNameToLE = label_encoding(surgicalData, toBeLE)
```

One Hot Encoding One hot encoding should be done for the features, that are nominal-scaled.

label encoding is done for categorical data, with string names, so that it can be understood by the ml-algorithms

[32]:
```
#define what column names should be onehot encoded
toBeOHE=CATEGORICAL_COLUMN_NAMES

#onehot encode the nominal categorical columns
surgicalData, codedNames, columnNameToOHE  = one_hot_encoding(toBeOHE,␣
↪surgicalData)
```

```
#remove the original column names of the features. (they got replacewd by the␣
↪values and then encoded)
intersectionSFC = set(SURGERY_FEATURES).intersection(CATEGORICAL_COLUMN_NAMES)
intersectionPFC = set(PATIENT_FEATURES).intersection(CATEGORICAL_COLUMN_NAMES)
for columnName in intersectionSFC:
    SURGERY_FEATURES.remove(columnName)
for columnName in intersectionPFC:
    PATIENT_FEATURES.remove(columnName)

#replace the old columns with the encoded ones.
CATEGORICAL_COLUMN_NAMES = list(codedNames)+OperationTypeEncoder.
↪relevant_op_types_Names


#add the encoded columns to the features.
SURGERY_FEATURES += list(codedNames)
PATIENT_FEATURES += list(codedNames)
```

```
Surgeon
OperationRoom
Anesthesiologist
Approach
PartOfDay
```

Variance Inflation Factor (VIF) The VIF values indicate how many times larger the variance would become, to create a dataset that would have been uncorrelated. If this VIF value is smaller than 5, that means no significant multicollinearity appears within the dataset. The following code will then produce the output 'True'

[33]:
```
print(multi_col_test(surgicalData))
#continue the coding if this is true
```

```
True
```

4.3.8 3.2.7 Normalizing

Normalize: - Numerical. - label encoded data.

Don't normalize: - Binary Data . - One Hot Encoded data.

[34]:
```
#normalize the data
surgicalData = normalize_data(surgicalData)
```

[35]:
```
#save the normalized data to a .csv file
surgicalData.to_csv("PreprocessedSurgicalData.csv", index=False)
```

5 3.3: Feature Selection

[36]:
```
#read the .csv file with the normalized data.
featureData=pd.read_csv("PreprocessedSurgicalData.csv")
```

[37]:
```
#get data that is our X and data that could be our Y
outcomeData = surgicalData[OUTCOME_COLUMN_NAMES]
y = outcomeData[PREDICTED_COLUMN]
#get the targetNamesString for the classification report
targetNamesString = get_target_names_string(y)


#find relevant Features based on the surgery
featureData = get_feature_data(surgicalData, SURGERY_FEATURES)
selectedSurgeryFeaturesData, selectedSurgeryFeatureColumnNames =␣
 ↪backward_elimination(featureData, y)
#print(selectedSurgeryFeatureColumnNames)
#find relevant Features based on the patient
featureData = get_feature_data(surgicalData, PATIENT_FEATURES)
selectedPatientFeaturesData, selectedPatientFeatureColumnNames =␣
 ↪backward_elimination(featureData, y)
#print(selectedPatientFeatureColumnNames)

#combine the selected features from the patient table and the surgery table.
selectedFeatureColumnNames=selectedSurgeryFeatureColumnNames+selectedPatientFeatureColumnNames
selectedFeatureNames=list(selectedFeatureColumnNames)

print("The selected Features are:",selectedFeatureNames)

#get the feature data corresponding to the column names
featureData = surgicalData[selectedFeatureNames.copy()]
```

```
The selected Features are: ['OPType_other', 'OPType_cabg', 'OPType_avr',
'OPType_pacemakerdraad tijdelijk', 'OPType_mvp', 'OPType_mvp shaving',
'OPType_wondtoilet', 'OPType_tvp', 'OPType_mvr', 'AtrialFibrillation',
'PreviousHeartSurgery', 'AorticSurgery', 'P_Hypertension', 'CCS', 'NYHA',
'AmountOfBypasses', 'CardiopulmonaryBypassUse']
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
<ipython-input-15-d6bb81231a50>:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_1['const'] = 1 #TODO FIND OUT WHY THE PROGRAMM SPITS OUT WARNINGS HERE AND
FIX IT!
```

[38]:
```
#save the feature data to a .csv file
featureData.to_csv("SelectedFeatureData.csv", index=False)
```

6 Chapter 3.4: Evaluation

[39]:
```
#read the feature data from the .csv file
featureData=pd.read_csv("SelectedFeatureData.csv")
```

6.1 3.4.1 train-test split

[40]:
```
X=featureData
#As we want to get the performance metrics for the original prediction as well␣
↪and need the
#same split for this, we add 'PlannedDurationTime' here before the␣
↪train-test-split
X = pd.merge(X, surgicalData['PlannedDurationTime'], left_index=True,␣
↪right_index=True)
X = pd.merge(X, surgicalData['PlannedDurationTimeRange'], left_index=True,␣
↪right_index=True)

#split the dataset into training (70%) and testing (30%) sets
X_train,X_test,y_train,y_test = model_selection.train_test_split(X,y,test_size=0.
↪3,random_state=randomSeed)


#Now that the split is done we split the 'Planned Duration Time' from feature␣
↪split
y_test_Original = pd.merge(X_test['PlannedDurationTime'][:
↪],X_test['PlannedDurationTimeRange'][:], left_index=True, right_index=True)
X_train.
↪drop(labels=['PlannedDurationTime','PlannedDurationTimeRange'],axis=COLUMN_AXIS,␣
↪inplace= True)
X_test.
↪drop(labels=['PlannedDurationTime','PlannedDurationTimeRange'],axis=COLUMN_AXIS,␣
↪inplace= True)
```

6.2 3.4.2 Choose a classifier

6.2.1 Dummy-Classifier

This is to get a Baseline: the other classifiers should perform better than these, otherwise any algorithm could potentially perform better than the current setup.

[41]:
```
#cretae list of dummy classifiers for the comparison
dummy_clfs=[DummyClassifier(strategy="most_frequent"),
            DummyClassifier(strategy="prior"),
```

```
             DummyClassifier(strategy="stratified"),
             DummyClassifier(strategy="uniform")
            ]
```

6.2.2 Regression Based Classifier

Linear Regression, LogisticRegression, Support Vector Machine

[42]:
```
#create a list of regressor classifiers for the comparison
reg_clfs=[DummyRegressor(strategy="mean"),
          LinearRegression(),
          svm.SVC(random_state=randomSeed)]
```

6.2.3 Unsupervised Classifiers

[43]:
```
#create a list of the unsupervised classifiers for the comparison
tree_clfs=[tree.DecisionTreeClassifier(max_depth = 1, random_state=randomSeed),
           RandomForestClassifier(n_estimators=100,max_depth=5,␣
 ↪random_state=randomSeed)
          ]
```

6.2.4 Supervised (clustering) Methods

[44]:
```
#create a list of the supervised classifiers fort the comparison
cluster_clfs=[
    AffinityPropagation(damping=0.9, random_state=randomSeed),
    Birch(threshold=0.01, n_clusters=2),
    KMeans(n_clusters=2, random_state=randomSeed),
    MiniBatchKMeans(n_clusters=2, random_state=randomSeed),
    GaussianMixture(n_components=2, random_state=randomSeed),
    KNeighborsClassifier()
]
```

6.3 3.4.3 Evaluate the models

[45]:
```
#define the best classifier for the final comparison.
bestClassifier=""

#Filter whether the ranged planned duration time should be used or the␣
 ↪continuous version.
if PREDICTED_COLUMN=="ActualDurationTime":
    report_mode="continuos"
    bestEvalScore=-100000
    planned_y_test = y_test_Original['PlannedDurationTime']
if PREDICTED_COLUMN=="ActualDurationTimeRange":
    report_mode="categorical"
    bestEvalScore=0
```

```
    planned_y_test = y_test_Original['PlannedDurationTimeRange']

#Loop thorugh the dummy classifiers and report their performance
print("dummy-classifier\n")
strategy=["most_frequent","prior","stratified","uniform"]
for c,clf in enumerate(dummy_clfs):
    print(clf.__class__.__name__+"-Strategy:"+strategy[c])
    clf.fit(X_train, y_train)
    predicted=clf.predict(X_test)
    EvalScore=reportPerformance(y_test, predicted,report_mode)
    if EvalScore>bestEvalScore:
        bestEvalScore=EvalScore
        bestClassifier=clf

#only useful if we look at continuos data, execute the dummy regressor
if PREDICTED_COLUMN=="ActualDurationTime":
    print("regression based\n")
    for clf in reg_clfs:
        print(clf.__class__.__name__)
        clf.fit(X_train, y_train)
        predicted=clf.predict(X_test)
        EvalScore=reportPerformance(y_test, predicted, report_mode)
        if EvalScore>bestEvalScore:
            bestEvalScore=EvalScore
            bestClassifier=clf

#Loop thorugh the unsupervised classifiers and report their performance
print("tree-based\n")
for clf in tree_clfs:
    print(clf.__class__.__name__)
    clf.fit(X_train, y_train)
    predicted=clf.predict(X_test)
    EvalScore=reportPerformance(y_test, predicted,report_mode)
    if EvalScore>bestEvalScore:
        bestEvalScore=EvalScore
        bestClassifier=clf

#Loop thorugh the supervised classifiers and report their performance
print("clustering based\n")
for clf in cluster_clfs:
    print(clf.__class__.__name__)
    clf.fit(X_train, y_train)
    predicted=clf.predict(X_test)
    EvalScore=reportPerformance(y_test, predicted,report_mode)
    if EvalScore>bestEvalScore:
        bestEvalScore=EvalScore
        bestClassifier=clf
```

```
# execute the inverted indexes method and report its performance
print("InverseIndexes")
ii = create_ii(X_train.to_numpy())
predicted = predict_y_ii(ii, X_test.to_numpy(), y_train.to_numpy())
EvalScore=reportPerformance(y_test, predicted,report_mode)
if EvalScore>bestEvalScore:
    bestEvalScore=EvalScore
    bestClassifier=clf

#print the performance of the current prediction
print("Original Prediction")
predicted = planned_y_test
EvalScore=reportPerformance(y_test, predicted,report_mode)
if EvalScore>bestEvalScore:
    bestEvalScore=EvalScore
    bestClassifier=clf

#report for the continuous data the classifier with the lowest RMSE
if report_mode=="continuos":
    print("The classifier with the lowest RMSE (",str(-bestEvalScore),") is",␣
 ↪bestClassifier.__class__.__name__)

#report for the categorical dat the classifier with the highest accuracy score
if report_mode=="categorical":
    print("The classifier with the highest acccuracy (",bestEvalScore,") is",␣
 ↪bestClassifier.__class__.__name__)
```

```
dummy-classifier

DummyClassifier-Strategy:most_frequent
Accuracy score   0.373
F1 Macro score   0.109
F1 Weighted sc   0.203
MAE error        0.920
RMSE error       1.257

DummyClassifier-Strategy:prior
Accuracy score   0.373
F1 Macro score   0.109
F1 Weighted sc   0.203
MAE error        0.920
RMSE error       1.257

DummyClassifier-Strategy:stratified
Accuracy score   0.299
F1 Macro score   0.235
F1 Weighted sc   0.299
```

```
MAE error        1.140
RMSE error       1.509

DummyClassifier-Strategy:uniform
Accuracy score   0.196
F1 Macro score   0.176
F1 Weighted sc   0.218
MAE error        1.473
RMSE error       1.821

tree-based

DecisionTreeClassifier
Accuracy score   0.417
F1 Macro score   0.199
F1 Weighted sc   0.307
MAE error        0.784
RMSE error       1.101

RandomForestClassifier
Accuracy score   0.445
F1 Macro score   0.260
F1 Weighted sc   0.365
MAE error        0.733
RMSE error       1.054

clustering based

AffinityPropagation

C:\Users\xlbok\AppData\Roaming\Python\Python38\site-
packages\sklearn\cluster\_affinity_propagation.py:250: ConvergenceWarning:
Affinity propagation did not converge, this model will not have any cluster
centers.
  warnings.warn(
C:\Users\xlbok\AppData\Roaming\Python\Python38\site-
packages\sklearn\cluster\_affinity_propagation.py:528: ConvergenceWarning: This
model does not have any cluster centers because affinity propagation did not
converge. Labeling every sample as '-1'.
  warnings.warn(
C:\Users\xlbok\AppData\Roaming\Python\Python38\site-
packages\sklearn\base.py:441: UserWarning: X does not have valid feature names,
but Birch was fitted with feature names
  warnings.warn(

Accuracy score   0.000
F1 Macro score   0.000
F1 Weighted sc   0.000
MAE error        2.705
```

```
RMSE error       2.898

Birch
Accuracy score   0.208
F1 Macro score   0.114
F1 Weighted sc   0.171
MAE error        1.463
RMSE error       1.818

KMeans
Accuracy score   0.208
F1 Macro score   0.110
F1 Weighted sc   0.174
MAE error        1.445
RMSE error       1.793

MiniBatchKMeans
Accuracy score   0.208
F1 Macro score   0.110
F1 Weighted sc   0.174
MAE error        1.445
RMSE error       1.793

GaussianMixture
Accuracy score   0.208
F1 Macro score   0.114
F1 Weighted sc   0.169
MAE error        1.453
RMSE error       1.802

KNeighborsClassifier
Accuracy score   0.407
F1 Macro score   0.297
F1 Weighted sc   0.375
MAE error        0.818
RMSE error       1.153

InverseIndexes

C:\Users\xlbok\AppData\Roaming\Python\Python38\site-
packages\sklearn\base.py:441: UserWarning: X does not have valid feature names,
but KNeighborsClassifier was fitted with feature names
  warnings.warn(

Accuracy score   0.407
F1 Macro score   0.319
F1 Weighted sc   0.377
MAE error        0.814
RMSE error       1.153
```

```
Original Prediction
Accuracy score   0.351
F1 Macro score   0.211
F1 Weighted sc   0.305
MAE error        0.892
RMSE error       1.213

The classifier with the highest acccuracy ( 0.44510978043912175 ) is
RandomForestClassifier
```

7 ______________________

8 Chapter 4: Duration Time Generator

- Duration Time Generator

[46]:
```
#for hot encoding
hotColumnNames = ['Surgeon', 'OperationRoom', 'Anesthesiologist', 'Approach',
                  'PartOfDay']  #TODO Ähnlich zu CATEGORICAL_COLUMN_NAMES

#for label encoding
catOrdinalColumnNames = ['CCS', 'NYHA', 'Urgency','BMIRange',
                         'LeftVentricleFunction', 'RenalFunction',
                         'P_PulmonaleHypertension',␣
 ↪'AgeRange','OvertimeRange','ActualTimeRange']

#for binary labeling
binaryColumnNames=['Sex', 'AtrialFibrillation', 'P_ChronicLungDisease',
                   'P_extracardialArteriopathy', 'PreviousHeartSurgery',
                   'P_activeEndocarditis', 'CriticalPre-OP',
                   'MycordialInfarctionPreSurgery', 'AorticSurgery',
                   'P_PoorMobility', 'P_Diabetis', 'P_Hypercholesterolemia',
                   'P_Hypertension', 'P_PeripherialVascularDisease',␣
 ↪'CardiopulmonaryBypassUse']
```

4.1 Varibales Needed: - please fill the surgery's Data in

Information about the code below: It consists of multiple lists of characteristics. These lists are seperated into three types:

- Patient Data: All the information about the patient
- Surgery Data: All the information about the surgery
- Outcome Data: All the information about the outcome of the surgery. This is asked after the operation, to retrain the model, when enough people have commited the data to us.

[47]:
```
#Entries that consider data that is clear after the Surgery
outcomeData =['ActualDurationTime',
```

```
              'ICUDays',
              'HospitalDays']

#Prior_Entries, e.g. the data, that is entered before the operation #only ask␣
 ↪for necessary data
priorEntries= list(set(SURGERY_FEATURES + PATIENT_FEATURES)&set(priorData.
 ↪columns))
```

Out of the above we define if we want to use categorical or numerical questioning for our data.

- categoryPriorEntries -> Categorical Question
- numericalPriorEntries -> numerical Question

[48]:
```
#choice category entries, e.g. entries that are prior entries and categorical
categoryPriorEntries =␣
 ↪list(set(CATEGORICAL_COLUMN_NAMES+catOrdinalColumnNames+binaryColumnNames)&set(priorEntries))

#numerical Entries, e.g. entries that are prior entries and numerical
numericalPriorEntries = [Entry for Entry in priorEntries if Entry not in␣
 ↪categoryPriorEntries]
```

Lets see, how many entries we have per category:

[49]:
```
print('Len prior Entries', len(priorEntries))
print('Len cat Entries', len(categoryPriorEntries))
print('Len num Entries', len(numericalPriorEntries))
```

```
Len prior Entries 35
Len cat Entries 33
Len num Entries 2
```

8.1 4.2 Collect needed Data

[50]:
```
#generate a dictionary, that maps from the column-name to the answering options
categoryPriorEntries=list(set(priorData.columns)&set(categoryPriorEntries))
optionsForEntry={}
for c,entry in enumerate(categoryPriorEntries):
    data=set(str(i) for i in priorData[entry].unique())
    optionsForEntry[entry] = data
```

[51]:
```
def askUserForCategory(name: str, options: set) ->str:
    """
    asks the user for the correct entry for their patient.

    Input:
        name: the name of the entry which is currently in question.
        options: is a set with all the options for the entry.
```

```
    returns:
        selected: is a string with the choosen option
    """
    #sort the options alphabetically or numerical
    options= sorted(options[name])

    print('Please fill in the number that represents your data for',name,':')

    #print all the options with a representative
    for index,optionName in enumerate(options):
        print(str(index+1) + ')\t' + optionName)

    #check for valid input
    lenListOptions = len(options)
    while True:
        inputRaw = input(name + ': ')
        try:
            inputNo = int(inputRaw) - 1
            if inputNo > -1 and inputNo < lenListOptions:
                selected = list(options)[inputNo]
                print('Selected ' +  name + ': ' + selected)
                return selected
            else:
                print('Please select a valid ' + name + ' number')
        except ValueError:
            print('Please fill in the index of your choice, not the name.')
```

The following code will ask the user for the characteristic that is relevant for the entry

[52]:
```
def get_new_patient_data():
    """
    Get the patient data, the entries prior to the operation.

    Input:

    returns:
        newDataDF: dictionary with all the data entries for the patient data
    """
    # create a dict, that saves all the Entries prior to the operation
    dictDataEntries= dict.fromkeys(priorEntries)

    # Go through all catgeorical Data points and ask for entry
    for i in range(0, len(categoryPriorEntries)):
        print()
        entry = categoryPriorEntries[i]
        newDataPoint = askUserForCategory(entry,optionsForEntry)
        dictDataEntries[entry]= [newDataPoint]
```

```
        #categoryPriorEntries[i][1]=  newDataPoint

    # Go through all numerical Data points and ask for entry
    for i in range(0, len(numericalPriorEntries)):
        entry = numericalPriorEntries[i]
        print()
        newDataPoint =input('Fill in the numerical value for '+ entry + ': ')
        dictDataEntries[entry]= [newDataPoint]

    newDataDF=pd.DataFrame.from_dict(dictDataEntries)

    return newDataDF
```

[53]:
```
#create new patient data by asking it in the therminal
newDataDF=get_new_patient_data()
#save the data from the user to a .csv file.
newDataDF.to_csv("new_data.csv", index= False)
```

```
Please fill in the number that represents your data for OPType_pacemakerdraad
tijdelijk :
1)      0
2)      1
OPType_pacemakerdraad tijdelijk: 1
Selected OPType_pacemakerdraad tijdelijk: 0

Please fill in the number that represents your data for Surgeon :
1)      1,00
2)      2,00
3)      3,00
4)      4,00
5)      5,00
6)      6,00
7)      7,00
8)      other
Surgeon: 1
Selected Surgeon: 1,00

Please fill in the number that represents your data for CCS :
1)      0.0
2)      1.0
3)      2.0
4)      3.0
5)      4.0
CCS: 1
Selected CCS: 0.0

Please fill in the number that represents your data for Sex :
```

```
1)      0
2)      1
Sex: 1
Selected Sex: 0

Please fill in the number that represents your data for Anesthesiologist :
1)      10,00
2)      11,00
3)      12,00
4)      13,00
5)      14,00
6)      15,00
7)      18,00
8)      5,00
9)      6,00
10)     7,00
11)     8,00
12)     9,00
13)     other
Anesthesiologist: 1
Selected Anesthesiologist: 10,00

Please fill in the number that represents your data for CardiopulmonaryBypassUse
:
1)      0
2)      1
CardiopulmonaryBypassUse: 1
Selected CardiopulmonaryBypassUse: 0

Please fill in the number that represents your data for
P_PeripherialVascularDisease :
1)      0
2)      1
P_PeripherialVascularDisease: 1
Selected P_PeripherialVascularDisease: 0

Please fill in the number that represents your data for OPType_mvr :
1)      0
2)      1
OPType_mvr: 1
Selected OPType_mvr: 0

Please fill in the number that represents your data for OPType_mvp shaving :
1)      0
2)      1
OPType_mvp shaving: 1
Selected OPType_mvp shaving: 0
```

```
Please fill in the number that represents your data for NYHA :
1)      1.0
2)      2.0
3)      3.0
4)      4.0
NYHA: 1
Selected NYHA: 1.0

Please fill in the number that represents your data for P_Hypertension :
1)      0
2)      1
P_Hypertension: 1
Selected P_Hypertension: 0

Please fill in the number that represents your data for OPType_avr :
1)      0
2)      1
OPType_avr: 1
Selected OPType_avr: 0

Please fill in the number that represents your data for P_PulmonaleHypertension
:
1)      Ernstig
2)      Matig
3)      Normaal
P_PulmonaleHypertension: 1
Selected P_PulmonaleHypertension: Ernstig

Please fill in the number that represents your data for OperationRoom :
1)      TOK1
2)      TOK2
3)      TOK3
4)      other
OperationRoom: 1
Selected OperationRoom: TOK1

Please fill in the number that represents your data for P_ChronicLungDisease :
1)      0
2)      1
P_ChronicLungDisease: 1
Selected P_ChronicLungDisease: 0

Please fill in the number that represents your data for OPType_mvp :
1)      0
2)      1
OPType_mvp: 1
Selected OPType_mvp: 0
```

```
Please fill in the number that represents your data for OPType_cabg :
1)      0
2)      1
OPType_cabg: 1
Selected OPType_cabg: 0

Please fill in the number that represents your data for AtrialFibrillation :
1)      0
2)      1
AtrialFibrillation: 1
Selected AtrialFibrillation: 0

Please fill in the number that represents your data for PreviousHeartSurgery :
1)      0
2)      1
PreviousHeartSurgery: 1
Selected PreviousHeartSurgery: 0

Please fill in the number that represents your data for P_Diabetis :
1)      0
2)      1
P_Diabetis: 1
Selected P_Diabetis: 0

Please fill in the number that represents your data for OPType_wondtoilet :
1)      0
OPType_wondtoilet: 1
Selected OPType_wondtoilet: 0

Please fill in the number that represents your data for P_activeEndocarditis :
1)      0
2)      1
P_activeEndocarditis: 1
Selected P_activeEndocarditis: 0

Please fill in the number that represents your data for
P_extracardialArteriopathy :
1)      0
2)      1
P_extracardialArteriopathy: 1
Selected P_extracardialArteriopathy: 0

Please fill in the number that represents your data for AorticSurgery :
1)      0
2)      1
AorticSurgery: 1
Selected AorticSurgery: 0
```

```
Please fill in the number that represents your data for P_Hypercholesterolemia :
1)      0
2)      1
P_Hypercholesterolemia: 1
Selected P_Hypercholesterolemia: 0

Please fill in the number that represents your data for
MycordialInfarctionPreSurgery :
1)      0
2)      1
MycordialInfarctionPreSurgery: 1
Selected MycordialInfarctionPreSurgery: 0

Please fill in the number that represents your data for Urgency :
1)      Electief
2)      Spoed
3)      Spoed < 24 uur
Urgency: 1
Selected Urgency: Electief

Please fill in the number that represents your data for CriticalPre-OP :
1)      0
2)      1
CriticalPre-OP: 1
Selected CriticalPre-OP: 0

Please fill in the number that represents your data for PartOfDay :
1)      Middag
2)      Ochtend
3)      other
PartOfDay: 1
Selected PartOfDay: Middag

Please fill in the number that represents your data for Approach :
1)      Volledige sternotomie
2)      other
Approach: 1
Selected Approach: Volledige sternotomie

Please fill in the number that represents your data for OPType_tvp :
1)      0
2)      1
OPType_tvp: 1
Selected OPType_tvp: 0

Please fill in the number that represents your data for P_PoorMobility :
1)      0
2)      1
```

```
P_PoorMobility: 1
Selected P_PoorMobility: 0

Please fill in the number that represents your data for OPType_other :
1)      0
2)      1
OPType_other: 1
Selected OPType_other: 0

Fill in the numerical value for AmountOfBypasses: 1

Fill in the numerical value for Euroscore1: 1
```

9 5.3 Transform Data According to Preprocessing

[54]:
```
newDataDF=pd.read_csv("new_data.csv")
newDataDF.head()
```

[54]:
```
   OPType_pacemakerdraad tijdelijk Surgeon  CCS  Sex Anesthesiologist  \
0                                0    1,00  0.0    0            10,00

   CardiopulmonaryBypassUse  P_PeripherialVascularDisease  OPType_mvr  \
0                         0                             0           0

   OPType_mvp shaving  NYHA  ...  Euroscore1  P_Hypercholesterolemia  \
0                   0   1.0  ...           1                       0

  MycordialInfarctionPreSurgery   Urgency  CriticalPre-OP  PartOfDay  \
0                             0  Electief               0     Middag

                Approach  OPType_tvp  P_PoorMobility  OPType_other
0  Volledige sternotomie           0               0             0

[1 rows x 35 columns]
```

Encode Labels /One hot encode

[55]:
```
#Loop through the new data and label encode/onehot encode the data
# based on in what list the columnName appears.
for columnName in newDataDF.columns:
    if columnName in columnNameToLE.keys():
        print("LE",columnName)
        newDataDF[columnName] = columnNameToLE[columnName].
 ↪transform(newDataDF[columnName])
    elif columnName in columnNameToOHE.keys():
        print("OHE",columnName)
        #create OHE
```

```
        enc=columnNameToOHE[columnName]
        data=newDataDF[columnName]
        hotCodedArray=enc.transform([list(data)]).toarray()
        #Find out columnames
        hotCodedcolumns=[]
        for category in list(enc.categories_[0]):
            hotCodedcolumns+=[columnName+"_"+str(category)]

        #create a new df, using the hot coded columns as Column Names
        hotCoded = pd.DataFrame(hotCodedArray, columns = hotCodedcolumns)

        #remove old data column and add new dataframe
        newDataDF.drop(labels=columnName,axis=COLUMN_AXIS, inplace= True)
        newDataDF = pd.merge(newDataDF, hotCoded, left_index=True,␣
↪right_index=True)
```

```
LE OPType_pacemakerdraad tijdelijk
OHE Surgeon
LE CCS
LE Sex
OHE Anesthesiologist
LE CardiopulmonaryBypassUse
LE P_PeripherialVascularDisease
LE OPType_mvr
LE OPType_mvp shaving
LE NYHA
LE P_Hypertension
LE OPType_avr
LE P_PulmonaleHypertension
OHE OperationRoom
LE P_ChronicLungDisease
LE OPType_mvp
LE OPType_cabg
LE AtrialFibrillation
LE PreviousHeartSurgery
LE P_Diabetis
LE OPType_wondtoilet
LE P_activeEndocarditis
LE P_extracardialArteriopathy
LE AorticSurgery
LE P_Hypercholesterolemia
LE MycordialInfarctionPreSurgery
LE Urgency
LE CriticalPre-OP
OHE PartOfDay
OHE Approach
LE OPType_tvp
LE P_PoorMobility
```

LE OPType_other

[56]:
```
#Output of the encoded data.
newDataDF.head()
```

[56]:
```
   OPType_pacemakerdraad tijdelijk  CCS  Sex  CardiopulmonaryBypassUse  \
0                               0    0    0                         0

   P_PeripherialVascularDisease  OPType_mvr  OPType_mvp shaving  NYHA  \
0                             0           0                   0     0

   P_Hypertension  OPType_avr  ...  Anesthesiologist_other  \
0               0           0  ...                     0.0

   OperationRoom_TOK1  OperationRoom_TOK2  OperationRoom_TOK3  \
0                 1.0                 0.0                 0.0

   OperationRoom_other  PartOfDay_Middag  PartOfDay_Ochtend  PartOfDay_other  \
0                  0.0               1.0                0.0              0.0

   Approach_Volledige sternotomie  Approach_other
0                             1.0             0.0

[1 rows x 60 columns]
```

remove features, that have not been selected

[57]:
```
#Print the previously filtered features.
print("The selected Features are:", selectedFeatureNames)

#get the feature data
featureData= surgicalData[selectedFeatureNames].copy()
```

```
The selected Features are: ['OPType_other', 'OPType_cabg', 'OPType_avr',
'OPType_pacemakerdraad tijdelijk', 'OPType_mvp', 'OPType_mvp shaving',
'OPType_wondtoilet', 'OPType_tvp', 'OPType_mvr', 'AtrialFibrillation',
'PreviousHeartSurgery', 'AorticSurgery', 'P_Hypertension', 'CCS', 'NYHA',
'AmountOfBypasses', 'CardiopulmonaryBypassUse']
```

9.1 5.4 Predict new Data

[58]:
```
# load best model
patient_data=featureData
prediction=bestClassifier.predict(featureData)

#check if it has a bin or not and predict the new duration time.
if PREDICTED_COLUMN in column2binNames.keys():
    print("The assumed value for",PREDICTED_COLUMN,"in this operation is",␣
 ↪column2binNames[PREDICTED_COLUMN][prediction[0]])
```

```
else:
    print("The assumed value for",PREDICTED_COLUMN,"in this operation is",␣
 ↪prediction[0])
```

```
The assumed value for ActualDurationTimeRange in this operation is 281-345
```

10

REFERENCES

[1] 2021. Overfitting. https://corporatefinanceinstitute.com/resources/knowledge/other/overfitting/

[2] Edgar Acuna and Caroline Rodriguez. 2004. The treatment of missing values and its effect on classifier accuracy. In *Classification, clustering, and data mining applications*. Springer, 639–647.

[3] Matthew A Bartek, Rajeev C Saxena, Stuart Solomon, Christine T Fong, Lakshmana D Behara, Ravitheja Venigandla, Kalyani Velagapudi, John D Lang, and Bala G Nair. 2019. Improving operating room efficiency: machine learning approach to predict case-time duration. *Journal of the American College of Surgeons* 229, 4 (2019), 346–354.

[4] Jason Brownlee. 2017. How to One Hot Encode Sequence Data in Python. https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/

[5] Brecht Cardoen, Erik Demeulemeester, and Jeroen Beliën. 2010. Operating room planning and scheduling: A literature review. *European journal of operational research* 201, 3 (2010), 921–932.

[6] Alvaro HC Correia and Freddy Lecue. 2019. Human-in-the-loop feature selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 2438–2445.

[7] Franklin Dexter, Richard H Epstein, Rodney D Traub, Yan Xiao, and David C Warltier. 2004. Making management decisions on the day of surgery based on operating room efficiency and patient waiting times. *The Journal of the American Society of Anesthesiologists* 101, 6 (2004), 1444–1453.

[8] Marinus JC Eijkemans, Mark Van Houdenhoven, Tien Nguyen, Eric Boersma, Ewout W Steyerberg, and Geert Kazemier. 2010. Predicting the unpredictable: a new prediction model for operating room times using individual characteristics and the surgeon’s estimate. *The Journal of the American Society of Anesthesiologists* 112, 1 (2010), 41–49.

[9] Donald E Farrar and Robert R Glauber. 1967. Multicollinearity in regression analysis: the problem revisited. *The Review of Economic and Statistics* (1967), 92–107.

[10] Nickolas K Freeman, Sharif H Melouk, and John Mittenthal. 2016. A scenario-based approach for operating theater scheduling under uncertainty. *Manufacturing & Service Operations Management* 18, 2 (2016), 245–261.

[11] Francesca Guerriero and Rosita Guido. 2011. Operational research in the management of the operating theatre: a survey. *Health care management science* 14, 1 (2011), 89–114.

[12] Jiawei Han, Jian Pei, and Micheline Kamber. 2011. *Data mining: concepts and techniques*. Elsevier.

[13] Shuangchi He, Melvyn Sim, and Meilin Zhang. 2019. Data-driven patient scheduling in emergency departments: A hybrid robust-stochastic approach. *Management Science* 65, 9 (2019), 4123–4140.

[14] T Jayalakshmi and A Santhakumaran. 2011. Statistical normalization and back propagation for classification. *International Journal of Computer Theory and Engineering* 3, 1 (2011), 1793–8201.

[15] Marcel Jirina, MJ Jirina, and K Funatsu. 2011. Classifiers based on inverted distances. *New fundamental technologies in data mining* 1 (2011), 369–387.

[16] Max Kuhn, Kjell Johnson, et al. 2013. *Applied predictive modeling*. Vol. 26. Springer.

[17] Alex Macario. 2006. Are your hospital operating rooms “efficient”? A scoring system with eight performance indicators. *The Journal of the American Society of Anesthesiologists* 105, 2 (2006), 237–240.

[18] Jerrold H May, William E Spangler, David P Strum, and Luis G Vargas. 2011. The surgical scheduling problem: Current research and future opportunities. *Production and Operations Management* 20, 3 (2011), 392–405.

[19] JJ Pandit and A Carey. 2006. Estimating the duration of common elective operations: implications for operating list management. *Anaesthesia* 61, 8 (2006), 768–776.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

[21] Adam Powell, Sergei Savin, and Nicos Savva. 2012. Physician workload and hospital reimbursement: Overworked physicians generate less revenue per patient. *Manufacturing & Service Operations Management* 14, 4 (2012), 512–528.

[22] Alakh Sethi. [n. d.]. One-Hot Encoding vs. Label Encoding using Scikit-Learn, year = 2020, url = https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/, urldate = 2022-01-22.

[23] Christopher Glen Thompson, Rae Seon Kim, Ariel M Aloe, and Betsy Jane Becker. 2017. Extracting the variance inflation factor and other multicollinearity diagnostics from typical regression results. *Basic and Applied Social Psychology* 39, 2 (2017), 81–90.

[24] Voedingscentrum. [n. d.]. BMI Berekenen. https://www.voedingscentrum.nl/nl/bmi-meter.aspx

[25] Yu Wang, Jiafu Tang, and Richard YK Fung. 2014. A column-generation-based heuristic algorithm for solving operating theater planning problem under stochastic demand and surgery cancellation risk. *International Journal of Production Economics* 158 (2014), 28–36.

[26] Cort J Willmott and Kenji Matsuura. 2005. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate research* 30, 1 (2005), 79–82.

[27] Stewart W Wilson. 1995. Classifier fitness based on accuracy. *Evolutionary computation* 3, 2 (1995), 149–175.

[28] Zach. 2021. How to interpret root mean square error (RMSE). https://www.statology.org/how-to-interpret-rmse/