Geriatric Frailty: A Data Analysis Project

The project deals with the analysis of data derived from the **FrailSafe** program (https://frailsafe-project.eu/ - I do not own the data used - All rights reserved). Funded by the European Union's Horizon 2020 research and innovation program, FrailSafe aims to understand geriatric frailty through modern data mining and analysis techniques (https://www.researchgate.net/publication/340627315_FrailSafe_An_ICT_Platform_for_Unobtrusive_Sensing_of_Multi-Domain_Frailty_for_Personalized_Interventions).

Within the frameworks of this project, we are dealing with two different datasets containing a very small percentage of the data collected by the Frailsafe research program. The first dataset (**beacons_dataset.csv**) contains daily recordings from smart beacon devices, detailing the movement of elderly individuals within their home environment. The dataset features are the following:

- **part_id**: The user ID, represented as a 4-digit number.
- **ts_date**: The date of the recording, formatted as "YYYYMMDD".
- **ts_time**: The time of the recording, formatted as "hh:mm".
- **room**: The room where the person was present at the specified date and time. It is assumed that the person stayed in the room until the next recording on the same day.

The second dataset (**clinical_dataset.csv**) includes clinical measurements obtained from the examination of elderly individuals by specialized medical personnel. These measurements describe the clinical state of the elderly individuals. A list of the recorded clinical parameters and their description is shown in the table below:

parameter	Description
fried	Categorization by Fried
hospitalization_one_year	Number of hospitalizations in the last year
hospitalization_three_years	Number of hospitalizations in the last three years
ortho_hypotension	Orthostatic hypotension detection
vision	Vision
audition	Audition
weight_loss	Unintentional weight loss
exhaustion_score	Self-reported exhaustion
raise_chair_time	Lower limb strength
balance_single	Single foot station (Balance)

gait_get_up	Timed Get Up And Go Test
gait_speed_4m	Speed for 4 meters' straight walk
gait_optional_binary	Gait optional evaluation
gait_speed_slower	Slowed walking speed
grip_strength_abnormal	Grip strength outside the norms
low_physical_activity	Low physical activity
falls_one_year	Number of falls in the last year
fractures_three_years	Number of fractures during the last 3 years
fried_clinician	Fried's categorization according to clinician's estimation
bmi_score	Body Mass Index
bmi_body_fat	Body Fat (%)
waist	Waist circumference
lean_body_mass	Lean Body Mass
screening_score	Mini Nutritional Assessment (MNA) screening score
cognitive_total_score	Montreal Cognitive Assessment (MoCA) test score
memory_complain	Memory complain
mmse_total_score	Folstein Mini-Mental State Exam score
sleep	Reported sleeping problems
depression_total_score	15-item Geriatric Depression Scale (GDS-15)
anxiety_perception	Anxiety auto-evaluation
living_alone	Living Conditions
leisure_out	Leisure activities
leisure_club	Membership of a club
social_visits	Number of visits and social interactions per week
social_calls	Number of telephone calls exchanged per week
social_phone	Approximate time spent on phone per week
social_skype	Approximate time spent on videoconference per week
social_text	Number of written messages sent by the participant per week
house_suitable_participant	Subjective suitability of the housing environment according to participant's evaluation
house_suitable_professional	Subjective suitability of the housing environment according to investigator's evaluation

stairs_number	Number of steps to access house
life_quality	Quality of life self-rating
health_rate	Self-rated health status
health_rate_comparison	Self-assessed change since last year
pain_perception	Self-rated pain
activity_regular	Regular physical activity
smoking	Smoking
alcohol_units	Alcohol Use
katz_index	Katz Index of ADL
iadl_grade	Instrumental Activities of Daily Living
comorbidities_count	Number of comorbidities
comorbidities_significant_count	Number of comorbidities which affect significantly the person's functional status
medication_count	Number of medication

We divide the analysis into four different stages:

- **Part A**: Preprocessing and 'cleaning' of the clinical_dataset.csv dataset.

- **Part B**: Using the processed clinical_dataset.csv dataset to train and compare machine learning models that will be used to predict the frailty level of the elderly individuals (Non-Frail, Pre-Frail, Frail).

- **Part C**: Preprocessing and cleaning of the beacons_dataset.csv dataset.

- **Part D**: Merging the two processed datasets into a single dataset (merged_dataset.csv) and implementing clustering using suitable algorithms.

A summary of the project is described below. The step-by-step detailed report of each project part can be found on the corresponding jupyter notebook as a set of markdown statements. The code for Part A can be found in the Jupyter notebook named **project.ipynb**, for Part B in the notebook **project2.ipynb**, and for the remaining two parts in the notebook **beacons.ipynb**.

The dataset clinical_dataset.csv contains both categorical and numerical features. There are columns of both types that have missing and erroneous values that need to be addressed. Preprocessing starts with the ‘cleaning’ of categorical variables. Erroneous values appearing are of the form ‘test_non_realizable’, ‘test_non_adequate’. Both these and the existing missing categorical values are temporarily replaced with the string ‘None’ to indicate that they are missing values.

We observe that the following features – columns of the dataset are those that contain the missing values. For each of these, the corresponding count of missing values is listed.

weight_loss --> 2

balance_single --> 50

gait_speed_slower --> 4

memory_complain --> 39

sleep -->3

living_alone --> 1

leisure_club --> 1

house_suitable_participant -->98

house_suitable_professional -->98

health_rate --> 1

health_rate_comparison --> 1

activity_regular --> 2

smoking --> 1

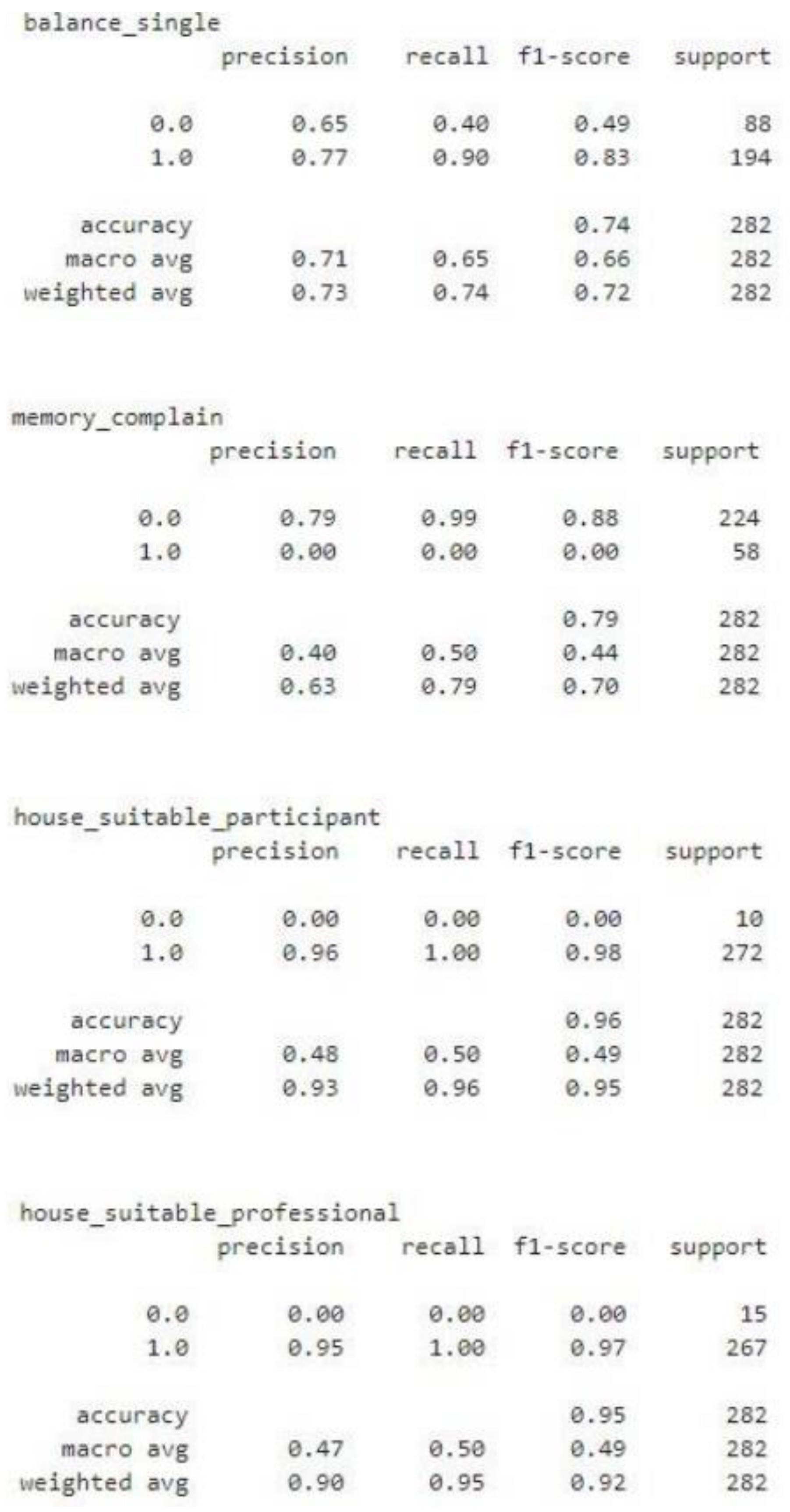
There are three different approaches we can take to handle these missing values. If the number of missing values in a column is small, the corresponding rows are removed. Conversely, if most of the values in a column are missing, the entire column is removed. When we are in an intermediate situation, as with the features 'balance_single', 'memory_complain', 'house_suitable_professional', 'house_suitable_participant' where the number of missing values is not small enough to justify removing rows without significantly reducing the size of the already limited dataset, and not large enough to justify removing the entire column, we use the method of predicting the missing values. All features with missing values, except those mentioned earlier, are addressed using the first method. The remaining four will be handled with the third method. To predict the missing values using a classifier, we first need to 'clean' the numerical columns.

Some numerical features contain erroneous values such as ‘999’, as well as missing values. After replacing erroneous values with missing values, we calculate the most representative value for each numerical column and use it to fill the missing cells. The mean is used for columns without a significant presence of outliers, while the median is used for columns where outliers are more prominent. The process for determining whether a column has a significant number of outliers is implemented using the **Z-score** statistical metric. The Z-score of a value indicates how far this value is from the mean of a distribution. Specifically, for each value in each column, a Z-score is calculated. If this Z-score exceeds a certain threshold, the corresponding value in the column is considered an outlier. Columns with a number of outliers greater than a specified value are handled using the median. For the remaining columns, the mean is used to replace the missing values. The values for the two thresholds are selected heuristically, as there is no golden rule for choosing them. The ideal process would be to test a range of pairs of values and select the one that produces the dataset leading to the best performance (regarding our final goal, which is the prediction of the frailty level) of the classifier trained on this dataset.

Having preprocessed the numerical features, except for feature scaling — which is not yet required as the classifier we will use for predicting missing categorical values (**Random Forest Classifier**) is unaffected by scale differences— we can proceed with the prediction. The model used is a random forest classifier, which is an ensemble of decision trees. This algorithm, like many others, does not work with categorical variables. Therefore, we must first encode the categorical variables into numerical ones. We distinguish between two cases: ordinal categorical features and binary non-ordinal categorical features, which are encoded using an **Ordinal Encoder** that considers the order of the categories. Non-binary, non-ordinal categorical features are encoded using the **One Hot Encoding** method (The notebook ‘project’ describes the encoding process in more detail and explains why the Ordinal Encoder is also used for non-ordinal binary features). Missing values, represented by the placeholder string ‘None’, are also encoded. We know in advance what numbers the 'None' values are encoded as in each column because they fall into the categories encoded by the Ordinal Encoder, where we manually define the encoding. Finally, these encoded values are replaced with actual missing (Null) values.

We are at the point where the clinical dataset is 'clean', except for some missing values in the columns 'balance_single', 'memory_complain', 'house_suitable_professional', and 'house_suitable_participant'. From this dataset, we retain only the rows that do not have any missing values (the classifier cannot be trained on missing values). Subsequently, for the resulting dataset (let’s call it df), we create two datasets for each of the previous features as follows. For the i-th feature fi (where fi in [memory_complain, balance_single, house_suitable_participant, house_suitable_professional]), we remove the column fi from df, resulting in the Xi dataset. The column fi is placed in a new variable yi. The Xi dataset contains the training data for the classifier that will predict the feature fi, while yi contains the corresponding labels.

We realize that since the Xi datasets differ (slightly), there will be a total of four random forest classifiers, denoted as RFCi. Each RFCi will be trained on the data of Xi and will predict the values of yi. To better estimate the performance of the models, each Xi is divided into training and test sets. We train the models on the training sets and use cross-validation to assess their performance on the training sets. The following results are obtained.



```
balance_single
              precision    recall  f1-score   support

         0.0       0.65      0.40      0.49        88
         1.0       0.77      0.90      0.83       194

    accuracy                           0.74       282
   macro avg       0.71      0.65      0.66       282
weighted avg       0.73      0.74      0.72       282
```

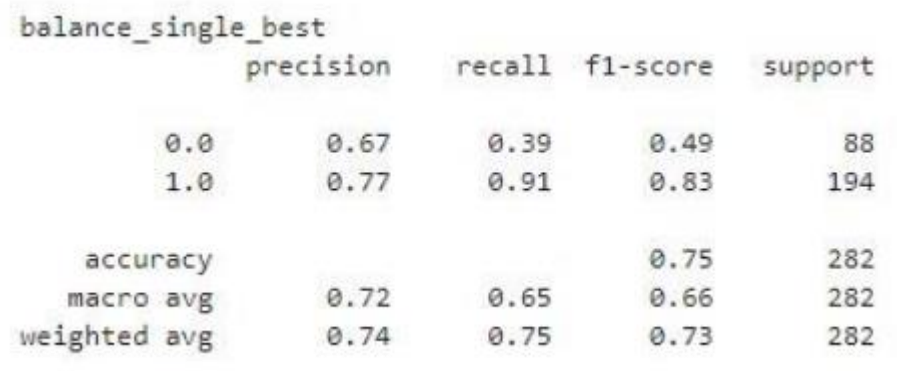
```
memory_complain
              precision    recall  f1-score   support

         0.0       0.79      0.99      0.88       224
         1.0       0.00      0.00      0.00        58

    accuracy                           0.79       282
   macro avg       0.40      0.50      0.44       282
weighted avg       0.63      0.79      0.70       282
```

```
house_suitable_participant
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00        10
         1.0       0.96      1.00      0.98       272

    accuracy                           0.96       282
   macro avg       0.48      0.50      0.49       282
weighted avg       0.93      0.96      0.95       282
```

```
house_suitable_professional
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00        15
         1.0       0.95      1.00      0.97       267

    accuracy                           0.95       282
   macro avg       0.47      0.50      0.49       282
weighted avg       0.90      0.95      0.92       282
```

Figure 1 Random Forest Classifier models. Cross validation results.

We observe that the performance of the model predicting the feature ‘balance_single’ is quite good for class 1, but there is room for improvement for class 0. This is likely due to the class imbalance of the two classes of the categorical variable balance_single (88 instances of class 0 versus 194 instances of class 1). The overall performance is characterized by an F1 score of 72% (considering the weighted average due to class imbalance). The other three models exhibit a similar behavior. They fail to correctly classify all instances belonging to the minority class in each case. The significant class imbalance in all three cases is likely the main cause. We could say that these models always predict the most frequently occurring class, something we could easily achieve manually, making their use unnecessary. The missing values in the features ‘memory_complain’, ‘balance_single’, ‘house_suitable_participant’, and ‘house_suitable_professional’ will be filled with the most frequently occurring value in each column. However, the model predicting the values of the feature ‘balance_single’ will be used, after we first implement a procedure to tune its hyperparameters (more details in the corresponding notebook). The tuned model shows a slightly better performance compared to the initial one and will therefore be the one we use.



```
balance_single_best
              precision    recall  f1-score   support

         0.0       0.67      0.39      0.49        88
         1.0       0.77      0.91      0.83       194

    accuracy                           0.75       282
   macro avg       0.72      0.65      0.66       282
weighted avg       0.74      0.75      0.73       282
```

Figure 2 Cross validation results for the Tuned Random Forest Classifier to be used for predicting missing values in the balance_single feature.

Before proceeding with the prediction of missing values for `balance_single`, we evaluate the tuned model on the test set of `balance_single` to see how well it generalizes to data it has not been trained on (until now, evaluation was performed on the training sets). The results are presented below.

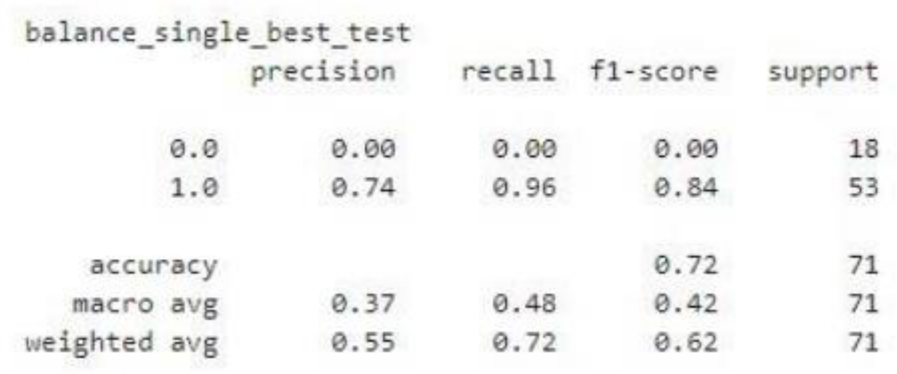
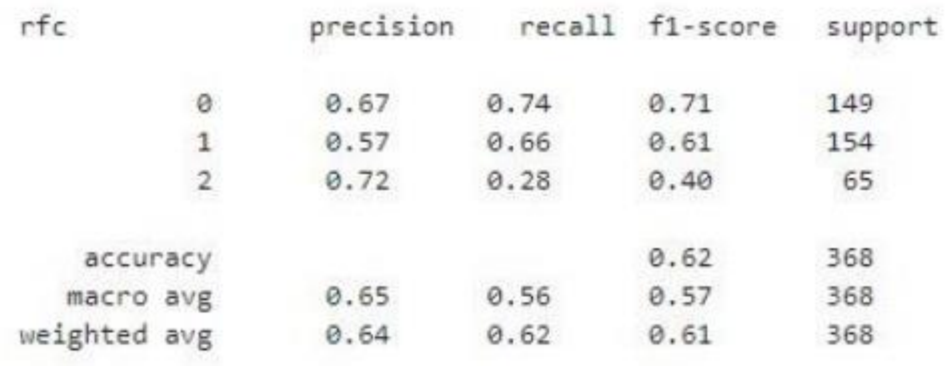

```
balance_single_best_test
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00        18
         1.0       0.74      0.96      0.84        53

    accuracy                           0.72        71
   macro avg       0.37      0.48      0.42        71
weighted avg       0.55      0.72      0.62        71
```

Figure 3 Evaluating the Tuned Random Forest Classifier to be used for predicting missing values in the balance_single feature, on the corresponding test set.

We observe that it fails to correctly classify the data belonging to class 0. We already know from the evaluation on the training set that its ability to classify class 0 is limited (it correctly identifies only 39% of the cases with 67% accuracy), but we did not expect such an absolute failure. This could be due to the model overfitting on the training set or because the test set contains a very small number of cases from this class (specifically 18), resulting in results that are not representative of reality. However, given that the model's behavior regarding class 1 is similar in both the training and test sets, and considering that the evaluation on the training set was carried out using cross-validation, the likelihood of model overfitting is small. Therefore, we consider the results from the performance measurement on the test set to be unrepresentative. The model will be used for predicting the missing values. This prediction completes the preprocessing of the clinical dataset.

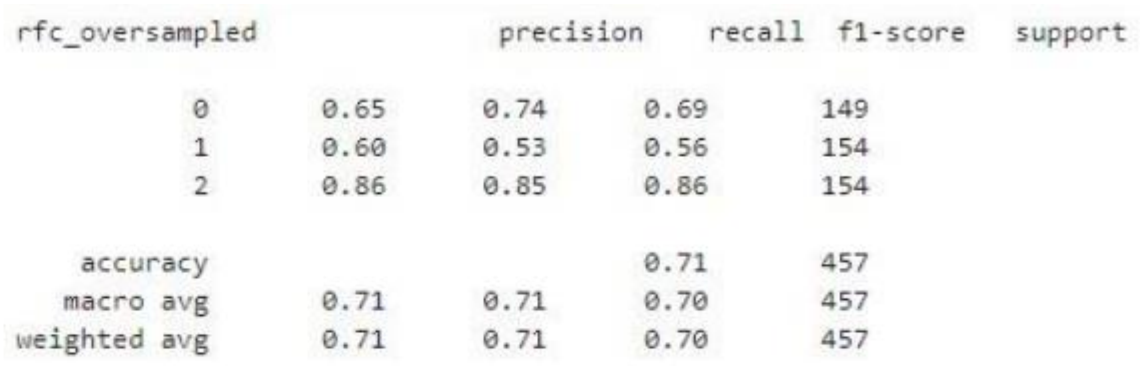
In the second part of the analysis, we use the now 'clean' and preprocessed clinical dataset to train machine learning models with the aim of predicting the value of the parameter ‘fried’. This value represents the frailty level of an individual and can take the values “Non Frail”, “Pre Frail”, or “Frail” (or 0, 1, 2 respectively for the encoded values). The classifiers to be tested are of two different types. The first is a **random forest classifier** (RFC), while the second is a **Support Vector Classifier** (SVC). Before training the two models, we scale the features. Specifically, we separate the dataset into two parts X and y, where X contains the data and y contains the labels (the 'fried' column). Then, we split these two datasets into training and test sets and normalize (standard scaling) X_train and X_test. Scaling is not required for the labels (y_train, y_test). It is important that the fit of the Standard Scaler is done on X_train and not on X, which includes the test set data, to ensure that there is no data leakage from the test set into the training process. The training of the RFC and its evaluation on the training set follows the usual process.



```
rfc                precision    recall  f1-score   support

           0       0.67      0.74      0.71       149
           1       0.57      0.66      0.61       154
           2       0.72      0.28      0.40        65

    accuracy                           0.62       368
   macro avg       0.65      0.56      0.57       368
weighted avg       0.64      0.62      0.61       368
```

Figure 4 Frailty prediction. The cross validation evaluation results of the Random Forest Classifier.

We are not satisfied with the above results, as the model exhibits very low recall (28%) for class 2 or ‘Frail’, which means it fails to recognize 72% of vulnerable individuals. Given the class imbalance (only 18.2% of the cases belong to the 'Frail' class), this result is not surprising. Its overall performance is also relatively low (weighted F1 score = 61%). To address the class imbalance and, consequently, improve performance, we will oversample the minority class 2 using the **SMOTE** algorithm (details about SMOTE can be found in the corresponding notebook), generating new synthetic data for class 2. The RFC trained on the oversampled dataset exhibits the following behavior:

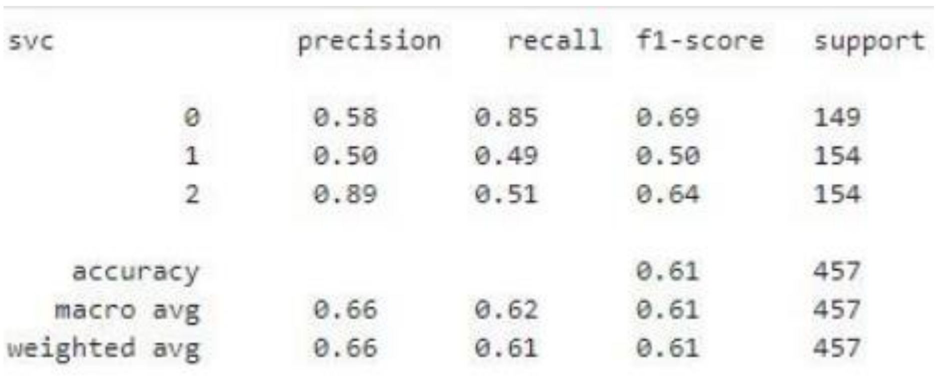


```
rfc_oversampled              precision    recall  f1-score    support

              0       0.65      0.74      0.69       149
              1       0.60      0.53      0.56       154
              2       0.86      0.85      0.86       154

       accuracy                           0.71       457
      macro avg       0.71      0.71      0.70       457
   weighted avg       0.71      0.71      0.70       457
```

Figure 5 Frailty prediction. The cross validation evaluation results of the Random Forest Classifier on the oversampled dataset.

We observe a significant increase in the model's performance regarding class 2. Precision increased by 14% and recall by 57%. Metrics for class 0 remain essentially unaffected (slight drop in precision), while for class 1 there is a small drop in precision by 3% and recall by 13%. Class 2 is considered more important than the others as it pertains to vulnerable elderly individuals. Any misclassification of these individuals could result in failing to recognize early the increased risk of health issues they face. Therefore, we accept the decreased performance in class 1 because we significantly improved performance in class 2. The overall performance of the model after the oversampling process is also improved. Thus, we have an additional reason to support this process and prefer the second model.

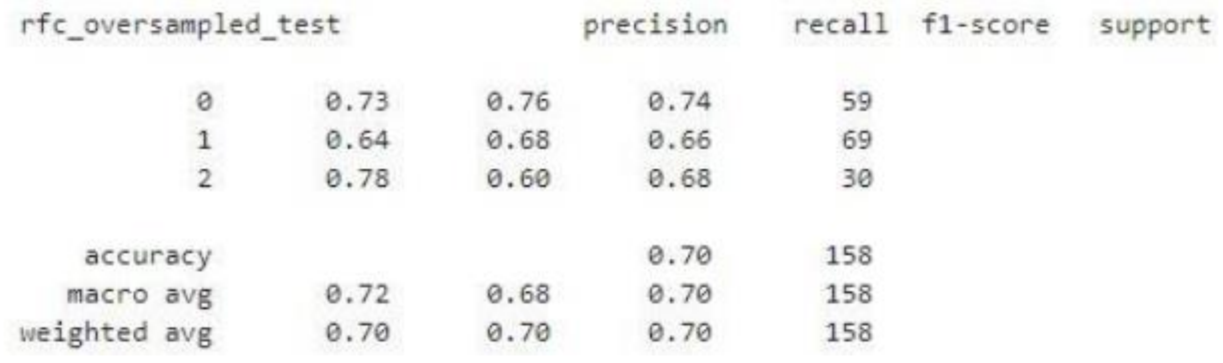
In the oversampled training dataset, we also train the second model, a support vector classifier using a polynomial kernel. The cross-validation process produces the following results.



```
svc            precision    recall  f1-score   support

           0       0.58      0.85      0.69       149
           1       0.50      0.49      0.50       154
           2       0.89      0.51      0.64       154

    accuracy                           0.61       457
   macro avg       0.66      0.62      0.61       457
weighted avg       0.66      0.61      0.61       457
```

Figure 6 Frailty prediction. The cross validation evaluation results of the Support Vector Classifier on the oversampled dataset.

Comparing it with the RFC, we observe that it performs worse. Therefore, the model we retain is the RFC. The final stage is to examine its behavior on the test set. The results are presented below.



```
rfc_oversampled_test              precision    recall  f1-score   support

           0       0.73      0.76      0.74        59
           1       0.64      0.68      0.66        69
           2       0.78      0.60      0.68        30

    accuracy                           0.70       158
   macro avg       0.72      0.68      0.70       158
weighted avg       0.70      0.70      0.70       158
```

Figure 7 Frailty prediction. The evaluation results of the Random Forest Classifier on the test set.

We observe that the metrics on the test set are better for classes 0 and 1 compared to those on the train set. However, for class 2, both precision (down by 8%) and recall (down by 25%) have significantly decreased. This could be due to the model overfitting class 2 or it might be because of the small number of instances belonging to class 2 in the test set. The overall performance of the model is similar to that in the train set, so the model is not overfitted in general. In conclusion, although there is room for improvement, the model's performance is satisfactory. The model could be used as an additional tool to assist medical staff during the clinical evaluation process of elderly individuals.

The third part of the analysis concerns the preprocessing of the beacons_dataset.csv. First, we retain only the rows with a valid part_id (the index that uniquely identifies each user). Valid part_ids are only four-digit integers. This results in a reduction of the dataset by 11,851 rows. Next, we observe that in the 'rooms' column, the same rooms are described in various ways. We correct this inconsistency in values so that each room is described in a unique way. During the analysis of the dataset, we identify a small number of rows with missing values in the 'room' column and remove them. Then, we discover that some rows appear multiple times; we remove all such duplicate rows, keeping only the first instance of each unique row (e.g., if row X appears 10 times, we retain only the first occurrence - the first copy - and remove the remaining 9 copies). Additionally, there are 54 cases where a user is recorded as being in two or more different rooms at the same time. This is obviously impossible; the user happens to be within the range of two or more beacons simultaneously, resulting in their presence being recorded in two or more different rooms. Since we cannot know the actual room the user was in and considering the number of such cases is negligible compared to the total size of the dataset, we remove all these cases.

Our goal is to create a new dataset that contains one row per user, and the features will describe the percentage of time each user has spent in rooms 'Kitchen', 'Living Room', 'Bedroom', 'Bathroom'. The original dataset contains multiple rows for each user, as each user is described by a time series of data - multiple records at regular time intervals. To separate the different users, we sort the dataset based on the 'part_id' feature and then sort based on the date and time of recording, so that each user's records appear in the correct chronological order. Since we assume that each user stayed in a room until the next recording of the same day, we can calculate the time spent in a room by computing the time difference between two consecutive recordings. The method by which this process is implemented and the details to be careful with are described in the Jupyter notebook beacons.ipynb. Having calculated the time each user has spent in the corresponding room for each record, we can easily calculate the total time spent in each room. We are only interested in the rooms Kitchen, Living Room, Bedroom, Bathroom, and thus we retain only the rows which have these specific values in the 'room' column. By summing, we find the total time recorded for each user and dividing this by the times spent in each individual room, we find the corresponding percentages, completing the processing of the beacons_dataset.

In the final stage, we merge the processed clinical dataset with the processed beacons dataset and perform clustering with the **KMeans** algorithm. Clustering is executed with a different number of centroids each time and two metrics are calculated, the **Silhouette score** and the **Davies-Bouldin index** (DBI). Based on the values of these metrics, we can compare the different clustering results and decide which number of centroids results in the best clustering. A higher silhouette score implies better clustering. Conversely, a lower value of the Davies-Bouldin index implies better clustering. From the following graphs, we observe that the maximum silhouette score occurs for k=3, and for k=3 we also see the minimum value of the Davies-Bouldin index. According to both metrics, the best clustering is achieved when the number of centroids is three.

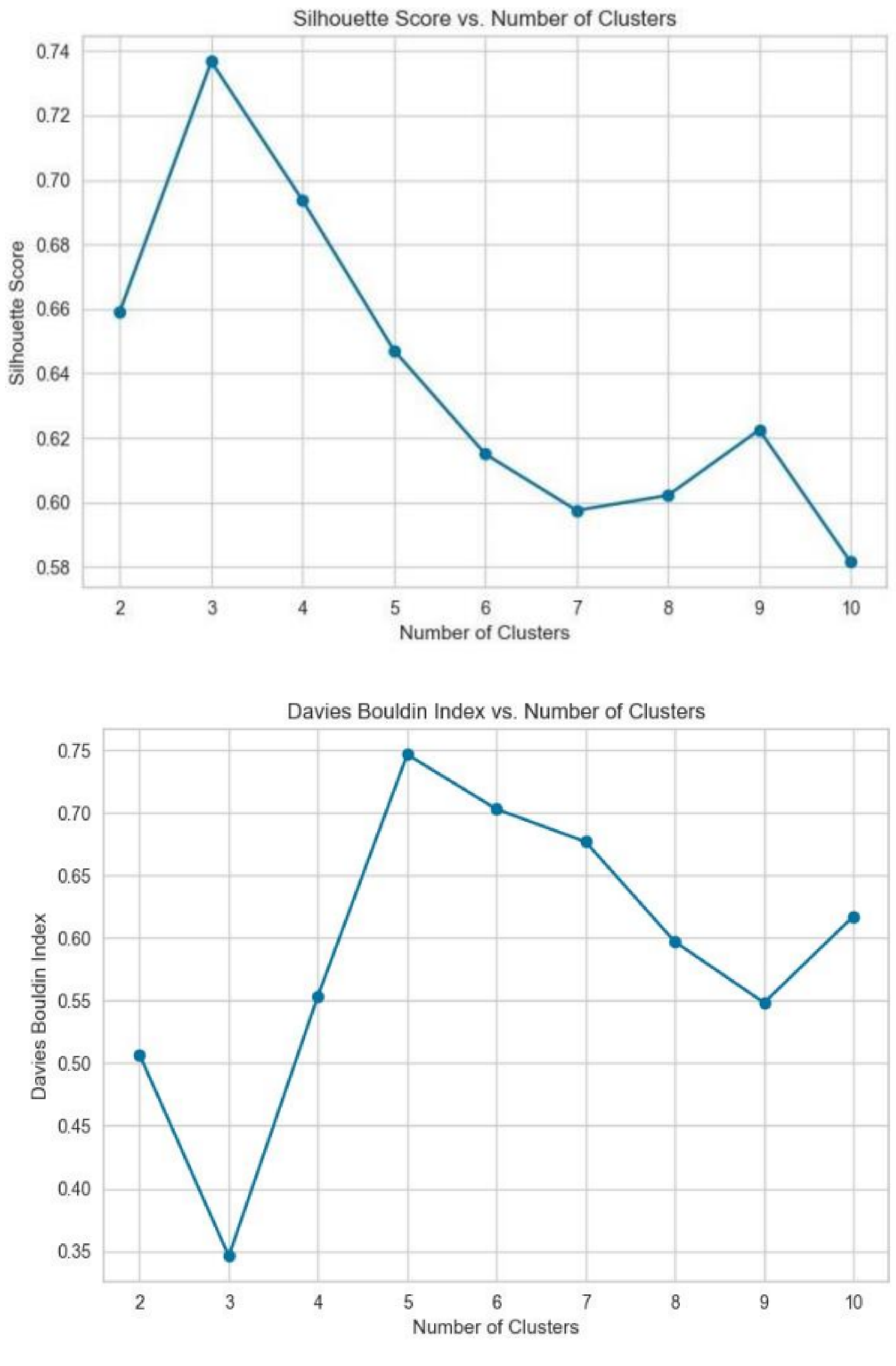


Figure 8 Silhouette and DBI vs Number of Centroids graphs.

Often, applying a dimensionality reduction algorithm before clustering leads to better results. We will examine this case by reducing the dimensions of our data using the PCA algorithm. Specifically, we use PCA to compress our data while retaining 95% of the original variance. We then perform the process described in the previous paragraph. The results obtained are as follows.

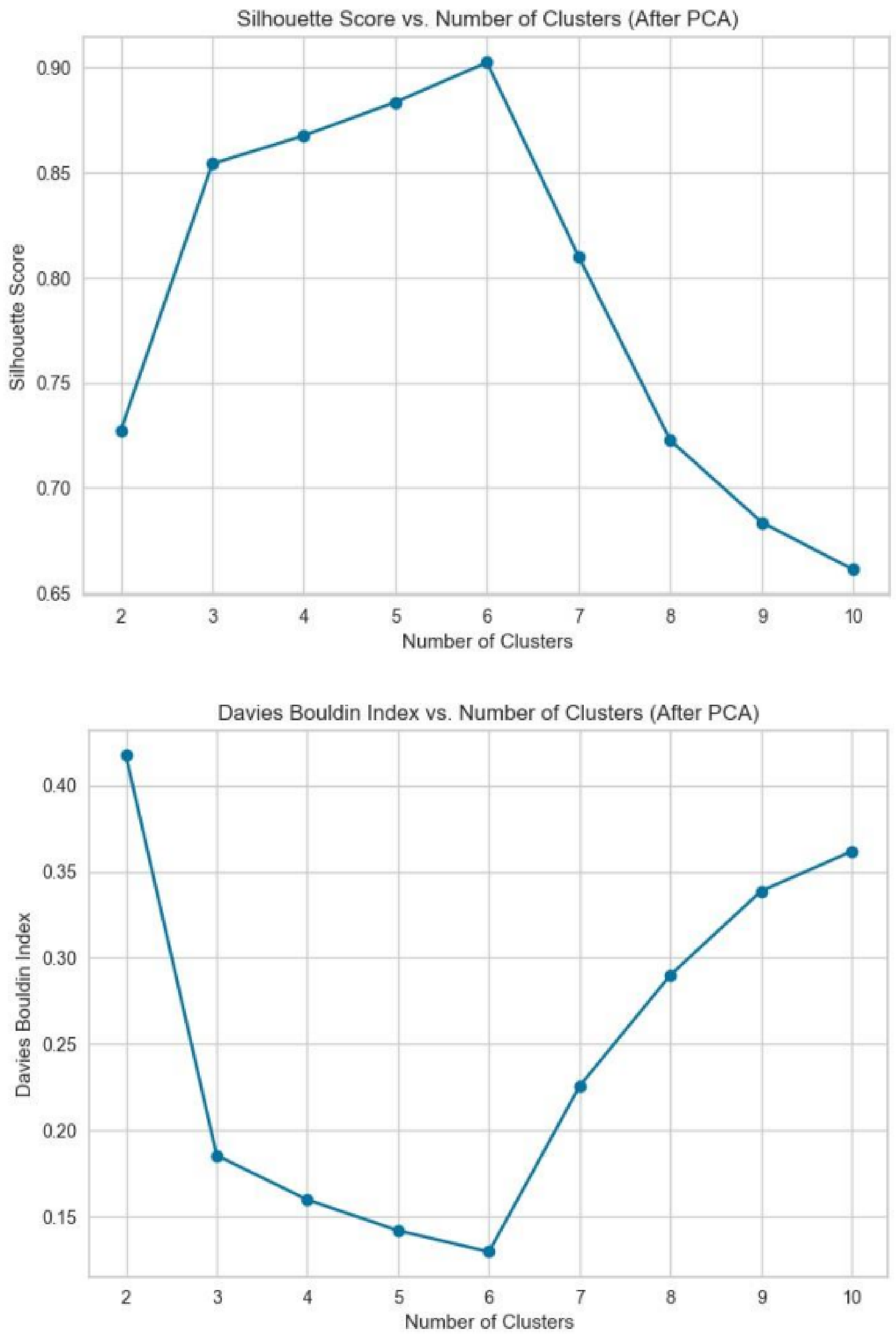
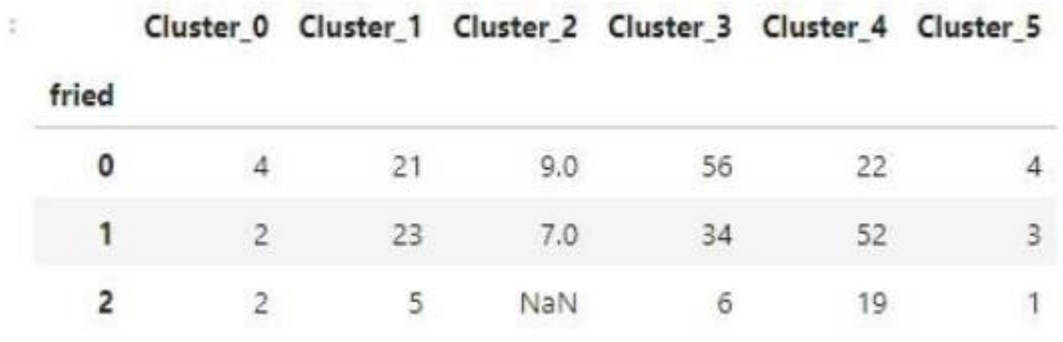


Figure 9 Silhouette and DBI vs Number of Centroids graphs after PCA.

In this case, the maximum silhouette score and the minimum Davies-Bouldin index are observed for k=6 rather than k=3. The values are approximately 0.90 and 0.15, respectively. In the case where we do not apply the PCA algorithm, these values are approximately 0.74 and 0.35. That is, the maximum silhouette score in the second case is 16% higher than the maximum silhouette score in the first case, while the minimum Davies-Bouldin index is 20% lower when performing clustering after applying PCA. We conclude that dimensionality reduction results in better clustering, with the optimal number of clusters being six instead of three. We confirmed that using a dimensionality reduction algorithm before clustering is a good strategy. High-dimensional data are often characterized by the curse of dimensionality. Many algorithms struggle to detect patterns in high-dimensional data, especially when these patterns are found in lower-dimensional subspaces.

The table below shows the distribution of data based on the clinical frailty parameter 'fried' across the six clusters.



fried	Cluster_0	Cluster_1	Cluster_2	Cluster_3	Cluster_4	Cluster_5
0	4	21	9.0	56	22	4
1	2	23	7.0	34	52	3
2	2	5	NaN	6	19	1

Figure 10 Distribution of data based on the clinical frailty parameter 'fried' across the six clusters.

The first observation is that the largest proportion of cases are found in clusters 1, 3, and 4. Clusters 0, 2, and 5 contain very few cases. One possible explanation is that these cases are various outliers grouped together. We also note that the largest proportion of cases belonging to classes 1, 2, or Pre-Frail, Frail are in cluster 4. Therefore, we should pay particular attention to users assigned to cluster 4 by the KMEANS algorithm, as there is a high likelihood that they belong to one of these two classes and, consequently, are elderly individuals at increased risk of developing serious health issues.