University of California, Berkeley

IEOR 242
Fall 2021 Project

Predicting Osteoarthritis with Machine Learning

Aishwarya Acharya, Antoine Reinaud, Oriane Cavrois
Vijitha Cheekala, William Mpondo

Contents

1 Introduction

Osteoarthritis (OA) is the most common form of arthritis of the knee that affects individuals of age 50+. OA causes the cartilage in the knee joint to gradually wears away, resulting in pain to carry out every day activities like walking. Osteoarthritis is one of the most quickly expanding chronic conditions, thanks to global ageing and the growing obesity pandemic [1]. OA, which most commonly affects knees and hips, can increase the risk of cardiovascular disease by limiting activity due to significant joint pain. It also has a detrimental impact on quality of life [2].
The damage is irreversible and treatments include physiotherapy, weight loss, painkillers, anti-inflammatory steroids, and knee replacement surgery. While risk factors for both frequent knee pain and knee OA may include old age, and overweight/obesity, not all people with these characteristics develop the condition.
Therefore early identification is critical to control the progression of the wear, which can be done by identifying the significant risk factors in data.

2 Objective

The aim of our model is to **predict if a patient will contract osteoarthritis in the next 5 years**, given physiological observations (age, body mass, knee pain, etc.) and diverse information on the patient's medical history (previous surgery, etc.).

3 Data Processing and Feature Engineering

We get data sets on the NIMH data archive website. This association conducted a ten-year long study, called the OA initiative, with almost 5,000 patients, diagnosed with OA or not. The patient's condition was observed and analyzed along the study. The aim was to better understand and prevent knee arthritis.
We had several data sets with information on each patient: general information, biomarker evolutions, serum analysis, medication information and X-ray images. For each type of information, there was one data set per year.

The data pre-processing part required several steps: Building of the outcome: At the beginning, we had one data set per year with one outcome in each data set, which was if the patient had been diagnosed with knee OA in the past year or not. Our goal was to predict, for the patients who were not sick at the beginning of the study, whether or not he/she has been diagnosed within 5 years after the beginning of the study. As this outcome was not directly available, we needed to build it with the following steps:

- Extraction of the outcome for each data set
- Creation of a new column (our final outcome) to combine the outcomes:
  - If the patient has been diagnosed at some point during the past 5 years, the final outcome is equal to 1
  - Otherwise, it is equal to 0
- Removal of the rows (patients) already sick at the beginning of the study

The next step was to see the large number of features, analyse them and figure out which ones will be the most useful. The following steps were taken before the data was set to be used in the models :

- The file had to be first converted from a `.txt` file to a `.csv` file
- The intial dimensions of the data frame was 4796 rows × 1187 columns which means there are 1187 featured. We divided the 1187 features amongst the 5 of us to manually find out which features make more sense for the analysis.
- We came down to 139 features after the manual analysis and amongst these 139 features we removed the features which had more than 25% of missing data.
- Once these are removed we came down to 121 columns and then we replaced the missing data with the most common value of the particular feature.
- Once this is done the data frame was good to go for being used in the models.
- At the end of building models, we also assigned feature importance score to all the features to understand the best 25 features and built models using just the 25 most important features to get more robust models.

4 Model Analysis

4.1 Baseline Model

Before fitting any model to the data, it is necessary to know the prediction of the baseline model. The metric used is accuracy, as the objective is to know the ratio of correctly predicted outcomes. The performance of the baseline model will be used as a reference value to compare with the other models.

The most prevalent outcome in the training set is 0 (i.e., the patient will not have OA). The performance of this prediction on the test set are as follows:

Baseline Model		
Accuracy	**TPR**	**FPR**
0.5554	0.0000	0.8000

4.2 Logistic Regression

Predicting if a patient will get OA or not is a classification problem. Therefore, one of the first models to apply is a logistic regression. To do so, we use dummy encoding. The best Logistic Regression model that we obtained is summarized in the following table.

Logistic Regression		
Accuracy	**TPR**	**FPR**
0.5743	0.5695	0.4250

Note: At first, we used a logistic regression from `statsmodels` and we developed an automated algorithm to remove one-by-one the highest p-values, calculate between each removal the new accuracy, and eventually return the best one. However, at some point, we face an '`Singular matrix`' error from `statsmodels` that we were unable to solve. Therefore, we chose to switch to `sklearn` library.

4.3 Clustering with Logistic Regression

We wanted to see if we could improve the performance of our model by using clustering. As the observations are patients, it makes sense to try to make clusters of patients with similar characteristics (age, sex, etc.) which could influence the likelihood to have osteoarthritis or not.

Pre-processing: We first clean the data and transform some features into dummy variables. We also need to normalize the features, so each feature has the same weight when computing the clusters. Then, we split the data set into a training set and a testing set.
Clustering: Then, we computed clusters on the training set. To determine the number of clusters, we performed a cross-validation, looking for minimizing the WCSS. We selected $k = 8$ clusters, as it corresponds to a knee in the WCSS curve.
Training: We split the observations of the training set according to their cluster, to have one data frame for the observations of each cluster. We chose a Logistic Regression model. We used the `sklearn` library, as our matrix was not invertible with `statsmodels`, and `sklearn` was able to deal with them anyway. We trained separately one model on one cluster's data set. At the end, we had 8 different logistic regression models, one for each cluster.
Testing: We determined to which cluster belongs each observation of the testing data set. To do so, for each observation, we computed its distance to each cluster centroid, and we chose the cluster with the smallest distance (we used a classical distance measuring). Then, we created 8 new data sets, with data set i containing the observations of cluster i. We finally assessed our models, by computing the accuracy.
Features selection: As we have a big difference between the accuracy on the training and the testing data, we are going to perform features selection to reduce the overfitting of the training data. For each model, we removed the features with a low coefficient and a high p-value (we adapted the threshold to the cluster). We trained and tested our models again, but the performance only slightly improved from the models without features selection.
Results: We computed for each model the accuracy, TPR, and FPR. The performance metrics did not improve much overall, compared to the other models, with an accuracy between 0.5549 (cluster 4) and 0.7465 (cluster 7).

Clustering w/ Logistic Regression							
Accuracy for cluster #							
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0.6218	0.6331	0.5793	0.6241	0.5549	0.5872	0.6571	0.7465

4.4 Random Forest

A Random Forest Classifier model can be used for our prediction problem. Using the accuracy as the performance metric, we perform a Random Forest with and without cross-validation. The results are summarized in the following table.

The 5 most important features are:

1. Body Mass Index 2. Daily Cholesterol Nutrients 3. Physical Activity Scale for the Elderly (PASE) 4. Daily Calcium Nutrients 5. Daily Vitamin D.

Random Forest Classifier					
RF w/o CV			**RF w/ CV**		
Accuracy	**TPR**	**FPR**	**Accuracy**	**TPR**	**FPR**
0.6070	0.302	0.149	0.6140	0.276	0.109

4.5 Gradient Boosting

We also used a boosting model, to obtain better results. Our final, and most accurate version of our Gradient Boosting model was obtained by only keeping the 25 most important features, as given by the `feature_importance` function of the Gradient Boosting model from `sklearn`.

The 5 most important features are:

1. Daily Cholesterol Nutrients 2. Daily Vitamin D 3. Daily Calcium Nutrients 4. Physical Activity Scale for the Elderly (PASE) 5. Body Mass Index

Gradient Boosting Classifier		
Accuracy	**TPR**	**FPR**
0.6337	0.4163	0.3163

4.6 Ridge

We use the `RidgeClassifier` from the `sklearn` library to try and predict the cumulative outcome. The value for the regularization parameter was found through a cross validation and the one standard error rule adapted to an accuracy scoring(see code in appendix), and allows us to improve on a classical logistic regression.

Ridge Classifier		
Accuracy	**TPR**	**FPR**
0.58596	0.35	0.18

5 Interpretation

As we can see, the models that we have were moderately performant, with accuracies ranging from 0.6 in the worst case to 0.74 in the best case (cluster and predict on cluster n°7). We can draw multiple conclusions from this:

- The ML models remain better than the baseline, but not by a great margin.
- The fact that observations (such as knee pain) are evaluated qualitatively, than encoded may lead to a loss of information that translates into poor model performance
- The fact that models only slightly improve when we select important features may be explained by the fact that some key information necessary to OA diagnosis may be missing
- The performance may also be explained by the great number of missing data, that doesn't allow us to exploit all information we may have had (90% of the features were removed in the data processing)

To improve on our prediction, we could use other models, such as neural networks, that may be able to figure out more complex patterns that the usual models we applied can not find.

Our models still allow us to infer that nutrition factors - namely calcium, cholesterol and vitamin D levels - as well as mass index and activity play a pivotal role in contracting OA.

Nevertheless, we must keep in mind that our model is destined to assist doctors in making their diagnosis, and raise awareness on a patient's chances to contract OA or not, in order to prescribe preventive treatment. The models should not be substituted to a doctor's diagnosis, but rather ran prior to one, to alert the doctor on the eventuality of OA for a patient.

6 Conclusion

This project led us to an effective applications discovered in the class. In spite of extensive data processing, feature engineering, and the implementation of multiple different models, we managed to get at best a 74% accuracy, which shows how difficult it is to do predictions in healthcare.

7 Appendix

7.1 References

1. Nelson, A. E. (2018). *Osteoarthritis year in review 2017: Clinical Osteoarthritis & Cartilage*, 26 (3), 319–325. https://doi.org/10.1016/j.joca.2017.11.014

2. Barbour, K. E., Helmick, C. G., Boring, M., Brady, T. J. (2017). *Vital signs: Prevalence of doctor-diagnosed arthritis and arthritis-attributable activity limitation*

7.2 Code

7.3 Data Cleaning

In [ ]:
```
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')

raw_data = pd.read_csv('/content/drive/MyDrive/Osteoarthritis dataset/All Cli

raw_data.head()
```

```
Mounted at /content/drive
```

Out[ ]:

	ID	VERSION	V00BLDCOLL	V00BLDHRS1	V00BLDHRS2	V00BLDRAW1	V00BLDRA
0	9000099	0.2.3	1: Sample at 1st collection only	35400.0	NaN	1: Yes	.: Miss Form/Incompl Workb
1	9000296	0.2.3	1: Sample at 1st collection only	30780.0	NaN	1: Yes	.: Miss Form/Incompl Workb
2	9000622	0.2.3	1: Sample at 1st collection only	33900.0	NaN	1: Yes	.: Miss Form/Incompl Workb
3	9000798	0.2.3	1: Sample at 1st collection only	26400.0	NaN	1: Yes	.: Miss Form/Incompl Workb
4	9001104	0.2.3	1: Sample at 1st collection only	35100.0	NaN	1: Yes	.: Miss Form/Incompl Workb

5 rows × 1187 columns

In [ ]:
```
raw_data.shape
```

Out[ ]:
```
(4796, 1187)
```

In [ ]:
```
# Initial dictionnary with following format: {name: index}
initial_dict = {}
for i in range(len(raw_data.columns)):
  initial_dict[raw_data.columns[i]] = i
```

In [ ]:
```
# Lists of kept features

list_features_Aish = ['ID','P01KPNREV','P01KPNLEV','P01KPACT30','P01HPNR12','
list_features_Antoine = ['P01BL12SXL', 'P01BL12SXR', 'P01KPACDCV', 'P01LKP30C
list_features_Oriane = ['P01OADEGCV','P01OAHIPCV','P01GOUTCV','P01OTARTCV','P
list_features_Vijitha = ['V00DISCOMF', 'V00REASW12', 'V00REASW10', 'V00REASW9
list_features_William = ['V00RPWKTYP','V00LPWKTYP','V00NPN400W', 'P01BMI','V0

list_features = list_features_Aish + list_features_Antoine + list_features_Or
```

In [ ]:
```
## Create a dictionnary with all kept features, to track original index and n
## {original index: new index} for every feature kept

dictionary_track = {}

for i in range(len(list_features)):
  original_index = initial_dict[list_features[i]]
  new_index = i
  dictionary_track[new_index] = original_index
```

In [ ]:
```
selected_data = pd.DataFrame()

for i in list_features:
  selected_data[i] = raw_data[i]

selected_data.head()
```

Out[ ]:

	ID	P01KPNREV	P01KPNLEV	P01KPACT30	P01HPNR12	P01HPNL12	V00SF2	V00SF
0	9000099	1: Yes	1: Yes	1: Yes	0: No	0: No	3: Not limited at all	2: Ye limited litt
1	9000296	1: Yes	0: No	0: No	1: Yes	0: No	3: Not limited at all	3: N limite at a
2	9000622	1: Yes	0: No	1: Yes	1: Yes	0: No	2: Yes, limited a little	2: Ye limited litt
3	9000798	0: No	1: Yes	0: No	0: No	0: No	3: Not limited at all	2: Ye limited litt
4	9001104	1: Yes	0: No	1: Yes	1: Yes	0: No	3: Not limited at all	3: N limite at a

5 rows × 139 columns

In [ ]:

```
## Check amount of missing data in every column

# Create dictionnary
missing_count = {}

# Loop over every column
for i in selected_data.columns:
  if '.: Missing Form/Incomplete Workbook' in selected_data[i].value_counts()
    missing_count[i] = round(selected_data[i].value_counts()['.: Missing Form
  else:
    missing_count[i] = 0

missing_count
```

Out[ ]:

```
{'ID': 0,
 'P01ARTDOC': 0.002,
 'P01ARTDRCV': 0.002,
 'P01ARTHOTH': 0.018,
 'P01BL12SXL': 0.008,
 'P01BL12SXR': 0.012,
 'P01BMI': 0,
 'P01FAMKR': 0.013,
 'P01GOUTCV': 0.02,
 'P01HPL12CV': 0.004,
 'P01HPNL12': 0.003,
 'P01HPNR12': 0.003,
 'P01HPR12CV': 0.003,
 'P01HRSLFXO': 0.998,
 'P01HRSRFXO': 0.997,
```

```
 'P01INJR': 0.011,
 'P01KPA30CV': 0.006,
 'P01KPACDCV': 0,
 'P01KPACT30': 0.002,
 'P01KPACTCV': 0.006,
 'P01KPL30CV': 0.004,
 'P01KPMED': 0.001,
 'P01KPMEDCV': 0.002,
 'P01KPNLEV': 0.003,
 'P01KPNREV': 0.005,
 'P01KPR30CV': 0.005,
 'P01KSURGL': 0.002,
 'P01KSURGR': 0.002,
 'P01LKP30CV': 0.002,
 'P01OADEGCV': 0.049,
 'P01OAHIPCV': 0.036,
 'P01OTARTCV': 0.03,
 'P01PMLKRCV': 0.003,
 'P01PMRKRCV': 0.003,
 'P01RAIA': 0.022,
 'P01RAJS1HR': 0.916,
 'P01RAKN6L': 0.917,
 'P01RAKN6R': 0.916,
 'P01RASTASV': 0,
 'P01RAWE6L': 0.916,
 'P01RAWE6R': 0.916,
 'P01RKP30CV': 0.002,
 'P02CNCOTH': 0.993,
 'P02KPMED': 0.001,
 'P02KPMEDCV': 0.004,
 'P02KPNLCV': 0.005,
 'P02KPNRCV': 0.006,
 'P02KSURG': 0.001,
 'P02KSURGCV': 0.001,
 'P02WTGA': 0,
 'V00ACUSCV': 0.002,
 'V00ACUTCV': 0.001,
 'V00BONEFX': 0.012,
 'V00BPDIAS': 0,
 'V00BPSYS': 0,
 'V00BRACCV': 0.001,
 'V00CALCMCV': 0.028,
 'V00CAM12': 0.001,
 'V00CAPSNCV': 0.004,
 'V00CHIRCV': 0.001,
 'V00COMORB': 0,
 'V00DIAB': 0.026,
 'V00DIETCV': 0.001,
 'V00DILKN1': 0.002,
 'V00DILKN14': 0.001,
 'V00DILKN2': 0.002,
 'V00DIRKN1': 0.002,
 'V00DIRKN14': 0.001,
 'V00DIRKN2': 0.002,
 'V00DISCOMF': 0.048,
 'V00DTCALC': 0,
 'V00DTCHOL': 0,
```

```
 'V00DTVITD': 0,
 'V00EKRSL': 0.111,
 'V00EKRSR': 0.108,
 'V00FALLCV': 0.019,
 'V00FFQYR86': 0.729,
 'V00HERBCV': 0.001,
 'V00HOMECV': 0.002,
 'V00HRTAT': 0.024,
 'V00HT25MM': 0,
 'V00HYINJCV': 0.001,
 'V00KOOSFX5': 0.123,
 'V00KSXLKN1': 0.013,
 'V00KSXRKN1': 0.013,
 'V00LFXCOMP': 0.104,
 'V00LFXPN': 0.124,
 'V00LKEFFB': 0.028,
 'V00LKEFFPT': 0.027,
 'V00LKRFXPN': 0.028,
 'V00LLWGT': 0,
 'V00LPWKTYP': 0.857,
 'V00MASSCV': 0.001,
 'V00NPN400W': 0.294,
 'V00OTH400W': 0.937,
 'V00OTHCAM': 0.002,
 'V00OTHCAMC': 0.002,
 'V00P7LKFR': 0.005,
 'V00P7LKRCV': 0.006,
 'V00P7RKFR': 0.005,
 'V00P7RKRCV': 0.005,
 'V00PASE': 0,
 'V00PN400W': 0.928,
 'V00POLYRH': 0.097,
 'V00RA': 0.175,
 'V00RAMEDS': 0.986,
 'V00REASW10': 0.997,
 'V00REASW12': 0.997,
 'V00REASW9': 0.996,
 'V00RELACV': 0.001,
 'V00REXPN': 0.122,
 'V00RFXCOMP': 0.103,
 'V00RKEFFB': 0.033,
 'V00RKEFFPT': 0.031,
 'V00RKRFXPN': 0.032,
 'V00RLWGT': 0,
 'V00RPWKTYP': 0.848,
 'V00RUBCV': 0.002,
 'V00SF2': 0.008,
 'V00SF3': 0.008,
 'V00SF8': 0.011,
 'V00SMOKE': 0.013,
 'V00SPIRCV': 0.001,
 'V00STINJCV': 0.002,
 'V00STROKE': 0.019,
 'V00VIT1': 0.037,
 'V00VIT9': 0.038,
 'V00VITDCV': 0.029,
 'V00VITMCV': 0.002,
```

```
'V00WOMTSL': 0,
'V00WOMTSR': 0,
'V00WPLKN1': 0,
'V00WPLKN2': 0.002,
'V00WPRKN1': 0.0,
'V00WPRKN2': 0.004,
'V00WT25KG': 0,
'V00WTMAXKG': 0,
'V00WTMINKG': 0,
'V00YOGACV': 0.0}
```

In [ ]:

```
## Remove columns with too many missing data
## Arbitrarily set the threshold value at 25%

threshold = 0.25

features_to_remove = []

for i in missing_count:
  if missing_count[i] > threshold:
    features_to_remove.append(i)

## Create new data frame containing only the columns with less than the thres

df = pd.DataFrame()

for i in selected_data.columns:
  if i not in features_to_remove:
    df[i] = selected_data[i]

print('Removed features (too much missing data): ', features_to_remove)
df.head()
```

```
Removed features (too much missing data):  ['P01RAJS1HR', 'P01RAKN6R', 'P01RAK
N6L', 'V00RAMEDS', 'P02CNCOTH', 'P01RAWE6L', 'P01HRSRFXO', 'P01HRSLFXO', 'P01R
AWE6R', 'V00REASW12', 'V00REASW10', 'V00REASW9', 'V00FFQYR86', 'V00RPWKTYP', '
V00LPWKTYP', 'V00NPN400W', 'V00PN400W', 'V00OTH400W']
```

Out[ ]:

	ID	P01KPNREV	P01KPNLEV	P01KPACT30	P01HPNR12	P01HPNL12	V00SF2	V00SF
0	9000099	1: Yes	1: Yes	1: Yes	0: No	0: No	3: Not limited at all	2: Ye limited litt
1	9000296	1: Yes	0: No	0: No	1: Yes	0: No	3: Not limited at all	3: N limite at a
2	9000622	1: Yes	0: No	1: Yes	1: Yes	0: No	2: Yes, limited a little	2: Ye limited litt
3	9000798	0: No	1: Yes	0: No	0: No	0: No	3: Not limited at all	2: Ye limited litt
4	9001104	1: Yes	0: No	1: Yes	1: Yes	0: No	3: Not limited at all	3: N limite at a

5 rows × 121 columns

In [ ]:

```
## Remove begining of text when it is of the form: 'number: value' --> 'value

for i in df.columns:
  for j in range(len(df[i])):
    if type(df[i][j]) == str: # if it is string, check if there is a need to
      content = [char for char in df[i][j]]
      content_isdigit = [char.isdigit() for char in content]

      if content_isdigit[0] == True and content[1] == ':' and content[2] == '
        kept_value = ''.join(content[3:])
        df[i][j] = kept_value

    else: # if the element is not a string, do nothing
      pass

df.head()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  # This is added back by InteractiveShellApp.init_path()
```

Out[ ]:

	ID	P01KPNREV	P01KPNLEV	P01KPACT30	P01HPNR12	P01HPNL12	V00SF2	V00SF
0	9000099	Yes	Yes	Yes	No	No	Not limited at all	Ye limited litt
1	9000296	Yes	No	No	Yes	No	Not limited at all	N limite at a
2	9000622	Yes	No	Yes	Yes	No	Yes, limited a little	Ye limited litt
3	9000798	No	Yes	No	No	No	Not limited at all	Ye limited litt
4	9001104	Yes	No	Yes	Yes	No	Not limited at all	N limite at a

5 rows × 121 columns

In [ ]:

```
## For each column, replace missing data by most common observation in the co

for col in df.columns:

  if df[col].dtype == 'int64' or df[col].dtype == 'float64':
    most_common_observation = df[col].describe()['mean']
  else:
    most_common_observation = df[col].describe()['top']

  for index in range(len(df[col])):
    if df[col][index] == '.: Missing Form/Incomplete Workbook':
      df[col][index] = most_common_observation

df.head()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
```

```
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
```

```
pyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning: 
```

```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
```

```
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
```

Out[ ]:

	ID	P01KPNREV	P01KPNLEV	P01KPACT30	P01HPNR12	P01HPNL12	V00SF2	V00SF
0	9000099	Yes	Yes	Yes	No	No	Not limited at all	Ye limited litt
1	9000296	Yes	No	No	Yes	No	Not limited at all	N limite at a
2	9000622	Yes	No	Yes	Yes	No	Yes, limited a little	Ye limited litt
3	9000798	No	Yes	No	No	No	Not limited at all	Ye limited litt
4	9001104	Yes	No	Yes	Yes	No	Not limited at all	N limite at a

5 rows × 121 columns

In [ ]:

```
#We upload the data sets with the information every year

raw_data_1years = pd.read_csv('/content/drive/MyDrive/Osteoarthritis dataset/
raw_data_2years = pd.read_csv('/content/drive/MyDrive/Osteoarthritis dataset/
raw_data_3years = pd.read_csv('/content/drive/MyDrive/Osteoarthritis dataset/
raw_data_4years = pd.read_csv('/content/drive/MyDrive/Osteoarthritis dataset/
raw_data_5years = pd.read_csv('/content/drive/MyDrive/Osteoarthritis dataset/

#The outcome of the initial data set is the feature P01OADEGCV
#The outcome that we consider for each data set is the features V01ARTH12, V0
```

In [ ]:
```
#Creation of a data frame with only the outcomes of the 5 years
#The aim is to add a column where the value is equal to one if the patient ha
#Hence, if for a patient, the value of the outcome is 1 in one of the 5 outco

outcome_1years = raw_data_1years['V01ARTH12']
outcome_2years = raw_data_2years['V03ARTH12']
outcome_3years = raw_data_3years['V05ARTH12']
outcome_4years = raw_data_4years['V06ARTH12']
outcome_5years = raw_data_5years['V07ARTH12']

outcomes = pd.concat([outcome_1years, outcome_2years, outcome_3years, outcome
outcomes.columns = ['Year1', 'Year2', 'Year3', 'Year4', 'Year5']
outcomes.head(10)
```

Out[ ]:

	Year1	Year2	Year3	Year4	Year5
0	1: Yes	0.0	0.0	0.0	0.0
1	0: No	0.0	0.0	0.0	0.0
2	1: Yes	NaN	NaN	NaN	NaN
3	0: No	0.0	0.0	0.0	0.0
4	0: No	1.0	NaN	NaN	NaN
5	0: No	0.0	0.0	0.0	0.0
6	1: Yes	1.0	NaN	1.0	0.0
7	0: No	0.0	1.0	0.0	0.0
8	1: Yes	0.0	0.0	0.0	0.0
9	0: No	0.0	1.0	1.0	1.0

In [ ]:
```
#Clean the first column of the data set ouctomes: we will only keep 0 or 1
for j in range(len(outcomes['Year1'])):
  content = [char for char in outcomes['Year1'][j]]

  if content[0] == '0':
    outcomes['Year1'][j] = 0.0

  if content[0] == '1':
    outcomes['Year1'][j] = 1.0

outcomes.head(20)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[ ]:

		Year1	Year2	Year3	Year4	Year5
0		1	0.0	0.0	0.0	0.0
1		0	0.0	0.0	0.0	0.0
2		1	NaN	NaN	NaN	NaN
3		0	0.0	0.0	0.0	0.0
4		0	1.0	NaN	NaN	NaN
5		0	0.0	0.0	0.0	0.0
6		1	1.0	NaN	1.0	0.0
7		0	0.0	1.0	0.0	0.0
8		1	0.0	0.0	0.0	0.0
9		0	0.0	1.0	1.0	1.0
10		0	0.0	0.0	0.0	NaN
11		0	0.0	1.0	0.0	1.0
12	.: Missing Form/Incomplete Workbook	NaN	NaN	NaN	NaN	
13		0	0.0	0.0	0.0	0.0
14		0	0.0	0.0	0.0	1.0
15		1	1.0	0.0	0.0	1.0
16		0	0.0	0.0	0.0	0.0
17		1	0.0	1.0	0.0	1.0
18		1	0.0	0.0	0.0	1.0
19		0	1.0	0.0	0.0	0.0

In [ ]:

```
#Add a new column which will be equal to 1 if one of the ouctome over the pas
cumulative_outcome = pd.DataFrame(index = range(len(outcomes)), columns = ['c
for j in range(len(outcomes)):
  if type(outcomes['Year1'][j]) == str and type(outcomes['Year2'][j]) == str
    cumulative_outcome['cumulative_outcome'][j] = 'Missing Form/Incomplete Wo
  if outcomes['Year1'][j] == 1 or outcomes['Year2'][j] == 1.0 or outcomes['Ye
    cumulative_outcome['cumulative_outcome'][j] = 'Yes'
    pass
  else:
    cumulative_outcome['cumulative_outcome'][j] = 'No'

df2 = pd.concat([df, cumulative_outcome], axis = 1)
df2.head(20)
```

Out[ ]:

	ID	P01KPNREV	P01KPNLEV	P01KPACT30	P01HPNR12	P01HPNL12	V00SF2	V00S
0	9000099	Yes	Yes	Yes	No	No	Not limited at all	Y limite lit
1	9000296	Yes	No	No	Yes	No	Not limited at all	N limit at
2	9000622	Yes	No	Yes	Yes	No	Yes, limited a little	Y limite lit
3	9000798	No	Yes	No	No	No	Not limited at all	Y limite lit
4	9001104	Yes	No	Yes	Yes	No	Not limited at all	N limit at
5	9001400	No	No	No	Yes	No	Not limited at all	N limit at
6	9001695	Yes	No	No	No	No	Not limited at all	N limit at
7	9001897	No	No	No	No	No	Not limited at all	N limit at

8	9002116	No	Yes	Yes	Yes	No	Yes, limited a little	Y limite lit
9	9002316	Yes	Yes	No	No	No	Not limited at all	N limit at
10	9002411	Yes	No	No	No	No	Yes, limited a little	Y limite lit
11	9002430	No	Yes	No	No	No	Not limited at all	N limit at
12	9002663	No	Yes	No	No	No	Not limited at all	N limit at
13	9002817	No	No	No	No	No	Not limited at all	N limit at
14	9003126	No	No	No	Yes	Yes	Yes, limited a little	Y limite
15	9003175	No	No	Yes	Yes	No	Not limited at all	N limit at
16	9003316	Yes	Yes	No	No	No	Not limited at all	N limit at
17	9003380	Yes	Yes	Yes	Yes	Yes	Yes, limited a little	Y limite lit
18	9003406	No	Yes	Yes	No	No	Yes, limited a lot	Y limite
19	9003430	No	No	No	Yes	Yes	Yes, limited	N limit

							a little	at

20 rows × 122 columns

In [ ]:
```
#Drop the rows with P01OADEGCV = Yes, because it means the patient already ha
#Drop the rows where we have missing forms (there are some missing forms for
index_yes = df2[df2['P01OADEGCV'] == 'Yes'].index
print(len(index_yes))
index_missing = df2[df2['cumulative_outcome'] == 'Missing Form/Incomplete Wor
print(len(index_missing))
df2.drop(index_yes, inplace = True)
df2.drop(index_missing, inplace = True)
df2.head()
```

```
1094
0
```

Out[ ]:

	ID	P01KPNREV	P01KPNLEV	P01KPACT30	P01HPNR12	P01HPNL12	V00SF2	V00S
1	9000296	Yes	No	No	Yes	No	Not limited at all	N limit at
2	9000622	Yes	No	Yes	Yes	No	Yes, limited a little	Y limite lit
6	9001695	Yes	No	No	No	No	Not limited at all	N limit at
7	9001897	No	No	No	No	No	Not limited at all	N limit at
10	9002411	Yes	No	No	No	No	Yes, limited a little	Y limite lit

5 rows × 122 columns

In [ ]:
```
df2.to_csv('cleaned_data.csv', index=False)
```

7.4 Models

Project_242_Random_Forest_Model (2)

December 17, 2021

[1]:
```
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from google.colab import drive

drive.mount('/content/drive')

df = pd.read_csv('/content/drive/MyDrive/Osteoarthritis dataset/All Clinical/
 ↪cleaned_data.csv')

df.head()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm

Mounted at /content/drive
```

[1]:

	ID	P01KPNREV	P01KPNLEV	…	V00LLWGT	V00RLWGT	cumulative_outcome
0	9000296	Yes	No	…	12.0	18.0	No
1	9000622	Yes	No	…	14.0	14.0	Yes
2	9001695	Yes	No	…	13.0	13.0	Yes
3	9001897	No	No	…	16.0	17.0	Yes
4	9002411	Yes	No	…	18.0	22.0	No

[5 rows x 122 columns]

[2]:
```
## Replace 'No' and 'Yes' by 0 and 1 in the outcome column

df['cumulative_outcome'] = df['cumulative_outcome'].apply(lambda x: 1 if x ==␣
 ↪'Yes' else 0)
df.head()
```

[2]:
```
         ID P01KPNREV P01KPNLEV  … V00LLWGT V00RLWGT cumulative_outcome
0   9000296       Yes        No  …     12.0     18.0                 0
1   9000622       Yes        No  …     14.0     14.0                 1
2   9001695       Yes        No  …     13.0     13.0                 1
3   9001897        No        No  …     16.0     17.0                 1
4   9002411       Yes        No  …     18.0     22.0                 0

[5 rows x 122 columns]
```

[2]:
```
```

[3]:
```
## Drop first column (ID of the patient)

df = df.drop('ID', axis='columns')


## Check columns to convert to dummies

columns_dummies = []

for col in df.columns:
  if df[col].dtype != 'float64' and df[col].dtype != 'int64':
    columns_dummies.append(col)

print(f'Number of columns to dummy encode = {len(columns_dummies)} out of␣
 ↪{len(df.columns)}')


df_enc = pd.get_dummies(df, columns = columns_dummies, drop_first = True)
```

```
Number of columns to dummy encode = 103 out of 121
```

[4]:
```
df_enc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3702 entries, 0 to 3701
Columns: 228 entries, V00WOMTSL to V00LKRFXPN_Yes
dtypes: float64(17), int64(1), uint8(210)
memory usage: 1.2 MB
```

[5]:
```
df_enc.head()
```

[5]:
```
   V00WOMTSL  V00WOMTSR  …  V00RKRFXPN_Yes  V00LKRFXPN_Yes
0        0.0        0.0  …               0               0
1        0.0       20.9  …               1               0
2        0.0        NaN  …               0               0
3        0.0        0.0  …               0               0
```

```
4        1.1        2.1  …                0                0

[5 rows x 228 columns]
```

[6]:
```
columns_with_nan_values = []
for col in df_enc.columns:
  if df_enc[col].isnull().values.any() == True:
    columns_with_nan_values.append(col)

print(columns_with_nan_values)
```

```
['V00WOMTSL', 'V00WOMTSR', 'P01KPACDCV', 'V00COMORB', 'V00HT25MM', 'V00WT25KG',
'V00WTMAXKG', 'V00WTMINKG', 'V00BPDIAS', 'V00BPSYS', 'V00DTVITD', 'V00DTCALC',
'V00DTCHOL', 'P01BMI', 'V00PASE', 'V00LLWGT', 'V00RLWGT']
```

[7]:
```
for col in columns_with_nan_values:
  col_mean = df[col].describe()['mean']
  df_enc[col] = df[col].fillna(col_mean)

df_enc.head()
```

[7]:

	V00WOMTSL	V00WOMTSR	…	V00RKRFXPN_Yes	V00LKRFXPN_Yes
0	0.0	0.000000	…	0	0
1	0.0	20.900000	…	1	0
2	0.0	10.076808	…	0	0
3	0.0	0.000000	…	0	0
4	1.1	2.100000	…	0	0

```
[5 rows x 228 columns]
```

[8]:
```
## Check if there are still NaN values in data frame
df_enc.isnull().values.any()
```

[8]:
```
False
```

[9]:
```
## Rename columns to have format usable with the Stats Model Formula input

print('Before = ', df_enc.columns)

# Replace all spaces and symbols by underscores

df_enc.columns = df_enc.columns.str.replace(' ', '_')
df_enc.columns = df_enc.columns.str.replace(',', '_')
df_enc.columns = df_enc.columns.str.replace(':', '_')
df_enc.columns = df_enc.columns.str.replace('/', '_')
df_enc.columns = df_enc.columns.str.replace('-', '_')
```

```
print('After = ', df_enc.columns)
```

```
Before =  Index(['V00WOMTSL', 'V00WOMTSR', 'P01KPACDCV', 'V00COMORB',
'V00HT25MM',
       'V00WT25KG', 'V00WTMAXKG', 'V00WTMINKG', 'V00BPDIAS', 'V00BPSYS',
       …
       'P01FAMKR_Yes', 'P02WTGA_Yes', 'V00RKEFFB_Yes',
       'V00RKEFFPT_Too tender to examine', 'V00RKEFFPT_Yes', 'V00LKEFFB_Yes',
       'V00LKEFFPT_Too tender to examine', 'V00LKEFFPT_Yes', 'V00RKRFXPN_Yes',
       'V00LKRFXPN_Yes'],
      dtype='object', length=228)
After =  Index(['V00WOMTSL', 'V00WOMTSR', 'P01KPACDCV', 'V00COMORB',
'V00HT25MM',
       'V00WT25KG', 'V00WTMAXKG', 'V00WTMINKG', 'V00BPDIAS', 'V00BPSYS',
       …
       'P01FAMKR_Yes', 'P02WTGA_Yes', 'V00RKEFFB_Yes',
       'V00RKEFFPT_Too_tender_to_examine', 'V00RKEFFPT_Yes', 'V00LKEFFB_Yes',
       'V00LKEFFPT_Too_tender_to_examine', 'V00LKEFFPT_Yes', 'V00RKRFXPN_Yes',
       'V00LKRFXPN_Yes'],
      dtype='object', length=228)
```

[10]:
```
## Divide data set in training and testing set

df_train, df_test = train_test_split(df_enc, test_size=0.3, random_state=88)
df_train.shape, df_test.shape
```

[10]:
```
((2591, 228), (1111, 228))
```

0.1 Random Forest Classifier

[11]:
```
#Random forest classifier
from sklearn.ensemble import RandomForestClassifier
```

[12]:
```
#Creating a model
y_train = df_train['cumulative_outcome']
x_train = df_train.drop(columns = ['cumulative_outcome'], axis = 1)
y_test = df_test['cumulative_outcome']
x_test = df_test.drop(columns = ['cumulative_outcome'], axis = 1)
```

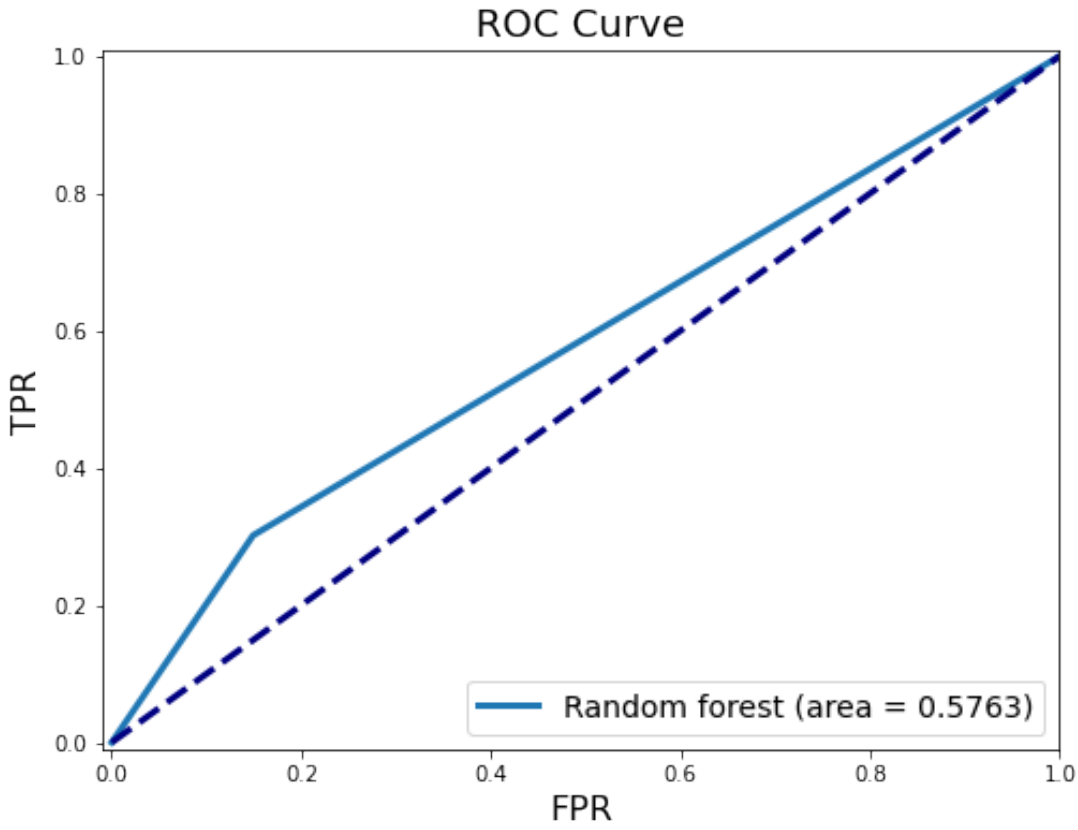
[21]:
```
# Train the random forest
rf = RandomForestClassifier(n_estimators = 200, criterion = 'entropy',␣
 ↪random_state = 0)
rf.fit(x_train, y_train)
```

[21]:
```
RandomForestClassifier(criterion='entropy', n_estimators=200, random_state=0)
```

[22]:
```
## Making predictions on the testing set
y_pred_rf = rf.predict(x_test)
## Obtaining the confusion matrix
cm_rf = confusion_matrix(y_test, y_pred_rf)
print ("Confusion Matrix : \n", cm_rf, "\n")
# Accuracy
print("Accuracy : \n", (cm_rf.ravel()[0]+cm_rf.ravel()[3])/sum(cm_rf.
 ↪ravel()),"\n")
#True positive rate
tpr = (cm_rf.ravel()[3])/(cm_rf.ravel()[2]+cm_rf.ravel()[3])
print("TPR : \n", tpr,"\n")

#False positive rate
fpr = (cm_rf.ravel()[1])/(cm_rf.ravel()[0]+cm_rf.ravel()[1])
print("FPR : \n", fpr,"\n")
```

```
Confusion Matrix :
 [[525  92]
 [345 149]]

Accuracy :
 0.6066606660666066

TPR :
 0.3016194331983806

FPR :
 0.14910858995137763
```

[15]:
```
## ROC Curve
from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y_test, y_pred_rf)
roc_auc_rf = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.title('ROC Curve', fontsize=18)
plt.xlabel('FPR', fontsize=16)
plt.ylabel('TPR', fontsize=16)
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr, tpr, lw=3, label='Random forest (area = {:0.4f})'.
 ↪format(roc_auc_rf))
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.legend(loc='lower right', fontsize=14)
plt.show()
```

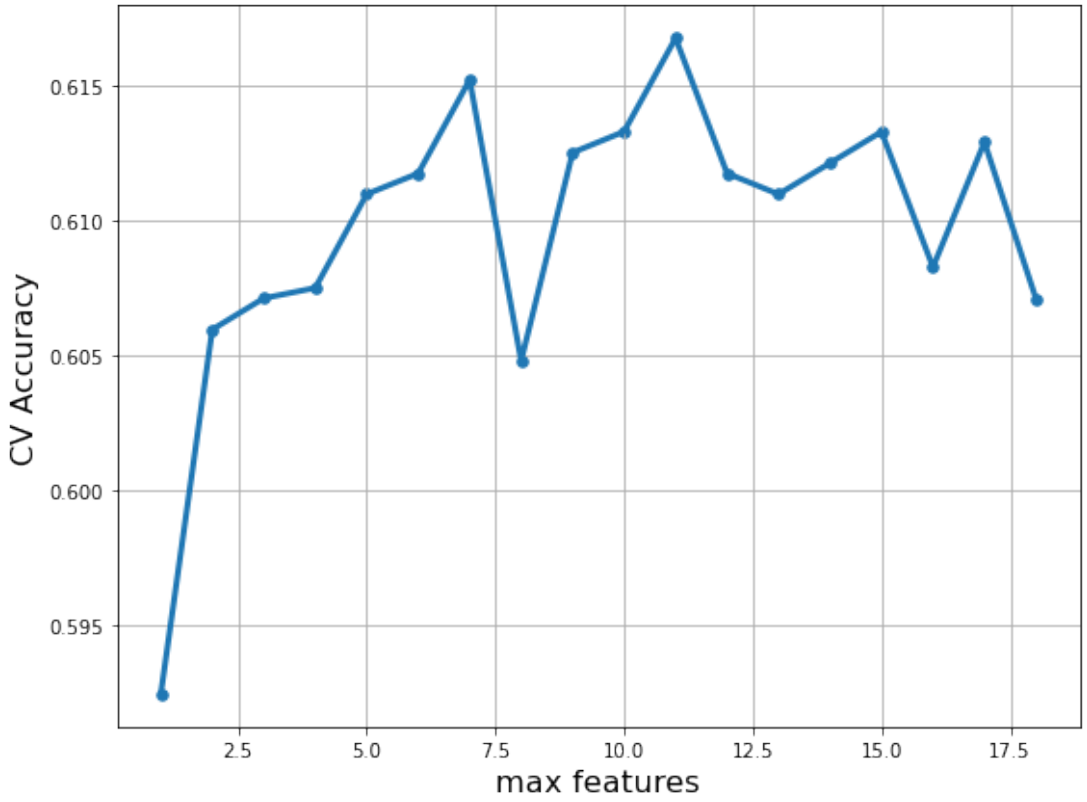


0.1.1 Random Forest with Cross Validation

[16]:
```
import time
from sklearn.model_selection import GridSearchCV

grid_values = {'max_features': np.linspace(1,18,18, dtype='int32'),
               'min_samples_leaf': [5],
               'n_estimators': [500],
               'random_state': [88]}

tic = time.time()

rf = RandomForestClassifier()
rf_cv = GridSearchCV(rf, param_grid=grid_values, cv=5)
rf_cv.fit(x_train, y_train)

toc = time.time()
print('time:', round(toc-tic, 2),'s')
```

```
time: 183.52 s
```

[ ]:
```
max_features = rf_cv.cv_results_['param_max_features'].data
accuracy_score = rf_cv.cv_results_['mean_test_score']

plt.figure(figsize=(8, 6))
plt.xlabel('max features', fontsize=16)
plt.ylabel('CV Accuracy', fontsize=16)
plt.scatter(max_features, accuracy_score, s=30)
plt.plot(max_features, accuracy_score, linewidth=3)
plt.grid(True, which='both')

plt.tight_layout()
plt.show()

print('Best parameters', rf_cv.best_params_)
```



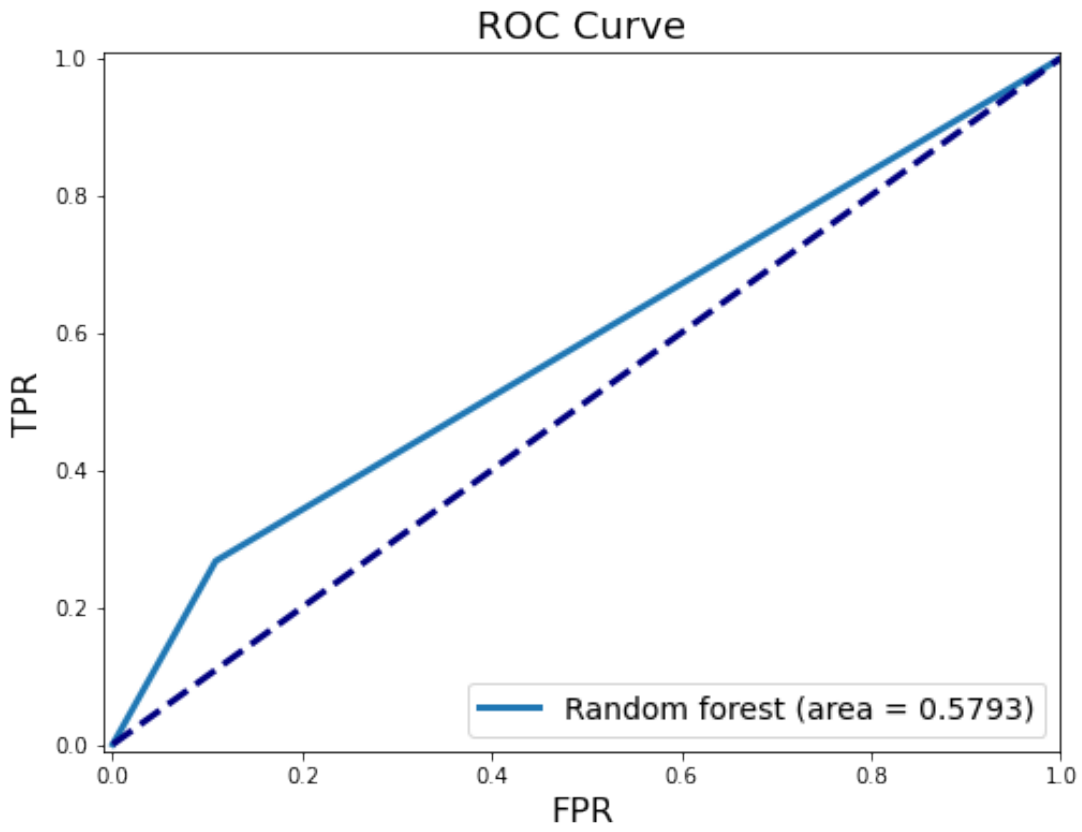

```
Best parameters {'max_features': 11, 'min_samples_leaf': 5, 'n_estimators': 500,
'random_state': 88}
```

[23]:
```
## Making predictions on the testing set
y_pred_rf_cv = rf_cv.predict(x_test)
## Obtaining the confusion matrix
```

```
cm_rf_cv = confusion_matrix(y_test, y_pred_rf_cv)
print ("Confusion Matrix : \n", cm_rf_cv, "\n")
# Accuracy
print("Accuracy : \n", (cm_rf_cv.ravel()[0]+cm_rf_cv.ravel()[3])/sum(cm_rf_cv.
 ↪ravel()),"\n")
#True positive rate
tpr = (cm_rf_cv.ravel()[3])/(cm_rf_cv.ravel()[2]+cm_rf_cv.ravel()[3])
print("TPR : \n", tpr,"\n")

#False positive rate
fpr = (cm_rf_cv.ravel()[1])/(cm_rf_cv.ravel()[0]+cm_rf_cv.ravel()[1])
print("FPR : \n", fpr,"\n")
```

```
Confusion Matrix :
 [[550  67]
 [362 132]]

Accuracy :
 0.6138613861386139

TPR :
 0.26720647773279355

FPR :
 0.1085899513776337
```

[ ]:
```
## ROC Curve
from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y_test, y_pred_rf_cv)
roc_auc_rf_cv = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.title('ROC Curve', fontsize=18)
plt.xlabel('FPR', fontsize=16)
plt.ylabel('TPR', fontsize=16)
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr, tpr, lw=3, label='Random forest (area = {:0.4f})'.
 ↪format(roc_auc_rf_cv))
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.legend(loc='lower right', fontsize=14)
plt.show()
```



[ ]:
```
pd.DataFrame({'Feature' : x_train.columns,
              'Importance score': 100*rf_cv.best_estimator_.
 ↪feature_importances_}).round(1)
```

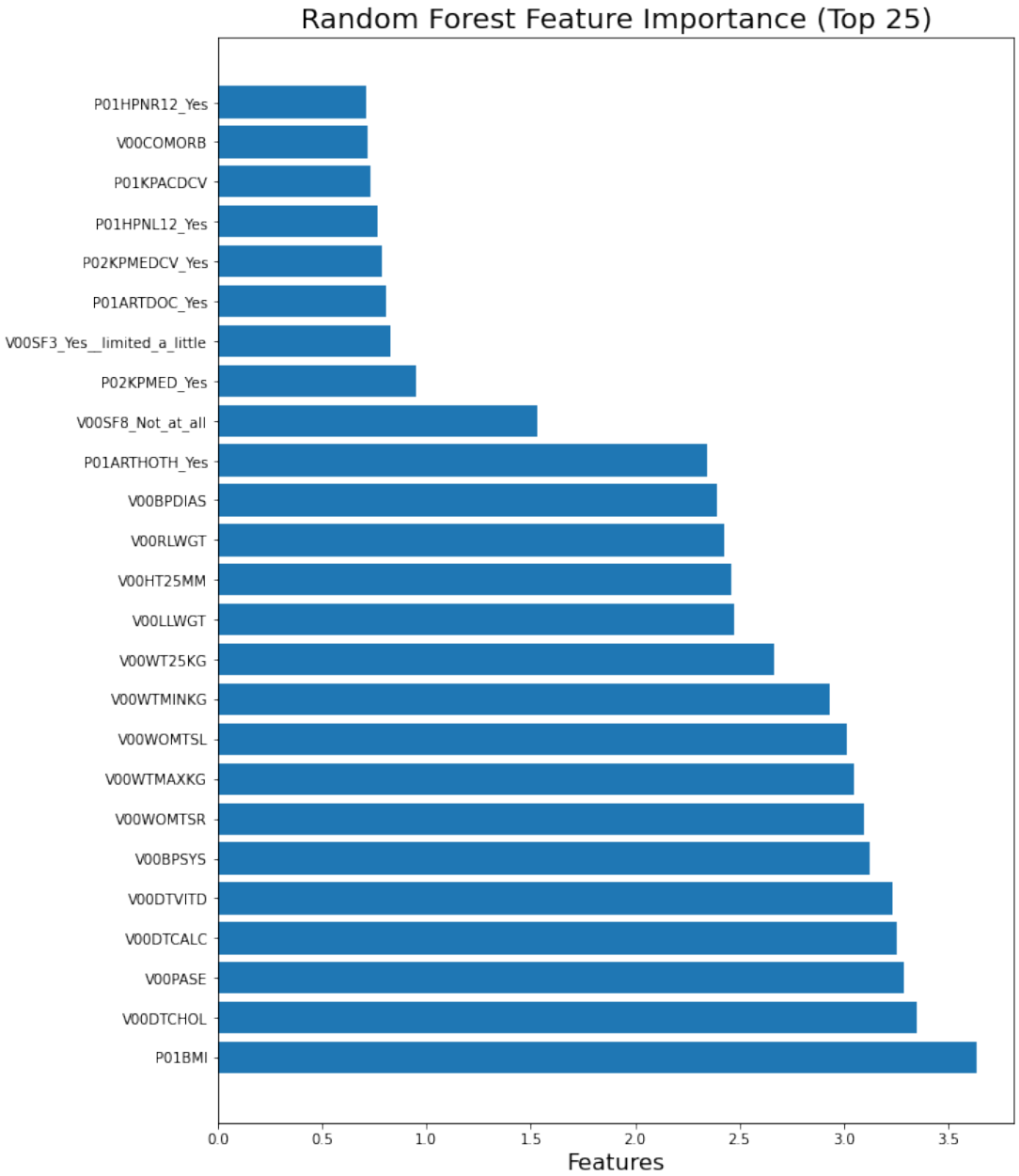
[ ]:
```
                              Feature  Importance score
0                           V00WOMTSL               3.0
1                           V00WOMTSR               3.1
2                          P01KPACDCV               0.7
3                           V00COMORB               0.7
4                           V00HT25MM               2.5
..                                ...               ...
222                    V00LKEFFB_Yes               0.2
223  V00LKEFFPT_Too_tender_to_examine               0.0
224                    V00LKEFFPT_Yes               0.1
225                    V00RKRFXPN_Yes               0.2
226                    V00LKRFXPN_Yes               0.3

[227 rows x 2 columns]
```

[ ]:
```
pd.DataFrame({'Feature' : x_train.columns,
              'Importance score': 100*rf_cv.best_estimator_.
 ↪feature_importances_}).round(1)

feature_importances = list(zip(x_train.columns,100*rf_cv.best_estimator_.
 ↪feature_importances_))
# Then sort the feature importances by most important first
feature_importances_ranked = sorted(feature_importances, key = lambda x: x[1],␣
 ↪reverse = True)
# Print out the feature and importances
[print('Feature: {:35} Importance: {}'.format(*pair)) for pair in␣
 ↪feature_importances_ranked[:25]];


# Plot the top 25 feature importance
feature_names_25 = [i[0] for i in feature_importances_ranked[:25]]
y_ticks = np.arange(0, len(feature_names_25))
x_axis = [i[1] for i in feature_importances_ranked[:25]]
plt.figure(figsize = (10, 14))
plt.barh(feature_names_25, x_axis)   #horizontal barplot
plt.title('Random Forest Feature Importance (Top 25)',
          fontdict= {'fontname':'Comic Sans MS','fontsize' : 20})
plt.xlabel('Features',fontdict= {'fontsize' : 16})
plt.show()
```

```
findfont: Font family ['Comic Sans MS'] not found. Falling back to DejaVu Sans.
```

```
Feature: P01BMI                                 Importance: 3.6340376366947424
Feature: V00DTCHOL                              Importance: 3.3487055811528417
Feature: V00PASE                                Importance: 3.28620479100094
Feature: V00DTCALC                              Importance: 3.2522299695558297
Feature: V00DTVITD                              Importance: 3.2355900217230475
Feature: V00BPSYS                               Importance: 3.1232582392101818
Feature: V00WOMTSR                              Importance: 3.0997647124565537
Feature: V00WTMAXKG                             Importance: 3.0475610693603543
Feature: V00WOMTSL                              Importance: 3.013200040499116
Feature: V00WTMINKG                             Importance: 2.9297953542139368
Feature: V00WT25KG                              Importance: 2.663661372140834
Feature: V00LLWGT                               Importance: 2.475550041302802
Feature: V00HT25MM                              Importance: 2.4584745614348344
Feature: V00RLWGT                               Importance: 2.4262824263170817
Feature: V00BPDIAS                              Importance: 2.3929988812074554
Feature: P01ARTHOTH_Yes                         Importance: 2.348317205573995
Feature: V00SF8_Not_at_all                      Importance: 1.5296990964421289
Feature: P02KPMED_Yes                           Importance: 0.9527963378641816
Feature: V00SF3_Yes__limited_a_little           Importance: 0.8251502255880654
Feature: P01ARTDOC_Yes                          Importance: 0.8045141916326934
Feature: P02KPMEDCV_Yes                         Importance: 0.7888874881939981
```

```
Feature: P01HPNL12_Yes          Importance: 0.7662990725810771
Feature: P01KPACDCV             Importance: 0.7340932775860353
Feature: V00COMORB              Importance: 0.7159320772418082
Feature: P01HPNR12_Yes          Importance: 0.7140243321671955
```



In [51]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [52]:
```
import statsmodels.formula.api as smf
```

In [53]:
```
data = pd.read_csv('/Users/william/Downloads/cleaned_data.csv')
```

In [54]:
```
data.head()
```

Out[54]:

	ID	P01KPNREV	P01KPNLEV	P01KPACT30	P01HPNR12	P01HPNL12	V00SF2	V00SF
0	9000296	Yes	No	No	Yes	No	Not limited at all	N limite at a
1	9000622	Yes	No	Yes	Yes	No	Yes, limited a little	Ye limited litt
2	9001695	Yes	No	No	No	No	Not limited at all	N limite at a
3	9001897	No	No	No	No	No	Not limited at all	N limite at a
4	9002411	Yes	No	No	No	No	Yes, limited a little	Ye limited litt

5 rows × 122 columns

In [55]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [56]:
```
from sklearn.model_selection import train_test_split

data_train, data_test = train_test_split(data, test_size=0.3, random_state=95
```

In [57]:
```
data_train
```

Out[57]:

	ID	P01KPNREV	P01KPNLEV	P01KPACT30	P01HPNR12	P01HPNL12	V00SF2	V0
1489	9428490	Yes	Yes	Yes	No	No	Yes, limited a little	lim
364	9113414	No	No	No	No	Yes	Not limited at all	I
2939	9794849	No	Yes	No	No	No	Not limited at all	I
1342	9391984	Yes	No	Yes	No	No	Not limited at all	lim
1425	9413319	Yes	No	No	Yes	Yes	Not limited at all	I
...	...	...	...	...	...	...	...	
1982	9547677	Yes	No	No	No	No	Not limited at all	I
2166	9594608	No	Yes	No	Yes	Yes	Not limited at all	I
3603	9962232	Yes	Yes	No	No	No	Not limited at all	lim
3369	9904241	Yes	Yes	No	Yes	Yes	Not limited at all	I
3478	9928863	Yes	Yes	No	Yes	No	Not limited at all	I

2591 rows × 122 columns

In [58]:
```
y_train = data_train['cumulative_outcome']
y_train = [1 if x=='Yes' else 0 for x in y_train]
x_train = data_train[data_train.columns[1:]]
x_train = x_train.drop(columns = ['cumulative_outcome'],axis=1)
```

In [59]:
```
x_train
```

Out[59]:

	P01KPNREV	P01KPNLEV	P01KPACT30	P01HPNR12	P01HPNL12	V00SF2	V00SF3	V
1489	Yes	Yes	Yes	No	No	Yes, limited a little	Yes, limited a little	Mo
364	No	No	No	No	Yes	Not limited at all	Not limited at all	N
2939	No	Yes	No	No	No	Not limited at all	Not limited at all	N
1342	Yes	No	Yes	No	No	Not limited at all	Yes, limited a little	A
1425	Yes	No	No	Yes	Yes	Not limited at all	Not limited at all	A
...	...	...	...	...	...	...	...	
1982	Yes	No	No	No	No	Not limited at all	Not limited at all	N
2166	No	Yes	No	Yes	Yes	Not limited at all	Not limited at all	A
3603	Yes	Yes	No	No	No	Not limited at all	Yes, limited a little	A
3369	Yes	Yes	No	Yes	Yes	Not limited at all	Not limited at all	N
3478	Yes	Yes	No	Yes	No	Not limited at all	Not limited at all	N

2591 rows × 120 columns

In [60]:
```
xd = pd.get_dummies(x_train)
```

In [61]:
```
xd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2591 entries, 1489 to 3478
Columns: 328 entries, V00WOMTSL to V00LKRFXPN_Yes
dtypes: float64(17), uint8(311)
memory usage: 1.1 MB
```

In [62]:
```
xd.head()
```

Out[62]:

	V00WOMTSL	V00WOMTSR	P01KPACDCV	V00COMORB	V00HT25MM	V00WT25KG	V(
1489	20.8	25.2	16.0	1.0	1600.2	68.2	
364	0.0	0.0	0.0	0.0	1676.4	72.7	
2939	1.0	0.0	0.0	0.0	1651.0	54.5	
1342	0.0	5.0	3.0	0.0	1651.0	65.9	
1425	0.0	4.0	0.0	0.0	1549.4	59.1	

5 rows × 328 columns

In [65]:
```
#We replace the missing data in the columns with the most common observation
for column in xd.columns:
    xd[column]=xd[column].fillna(xd[column].mode()[0])
```

In [66]:
```
#First model: classic gradient boosted classifier
gbc = GradientBoostingClassifier(n_estimators=3300, verbose=1,
                                 max_leaf_nodes=10)
gbc.fit(xd, y_train)
```

```
      Iter       Train Loss   Remaining Time
         1           1.3437            3.94m
         2           1.3313            3.33m
         3           1.3209            3.21m
         4           1.3110            3.12m
         5           1.3026            2.88m
         6           1.2956            2.79m
         7           1.2883            2.63m
         8           1.2821            2.47m
         9           1.2756            2.33m
        10           1.2699            2.27m
        20           1.2254            1.80m
        30           1.1945            1.79m
        40           1.1676            1.97m
        50           1.1439            2.26m
        60           1.1210            2.28m
        70           1.0982            2.22m
        80           1.0755            2.26m
        90           1.0573            2.26m
       100           1.0376            2.22m
       200           0.8876            1.94m
       300           0.7665            1.73m
       400           0.6616            1.59m
       500           0.5750            1.55m
       600           0.5027            1.50m
       700           0.4394            1.42m
       800           0.3882            1.32m
       900           0.3411            1.28m
      1000           0.3033            1.26m
      2000           0.0986           39.39s
      3000           0.0335            9.26s
```

Out[66]:
```
GradientBoostingClassifier(max_leaf_nodes=10, n_estimators=3300, verbose=1)
```

In [67]:
```
y_test = data_test['cumulative_outcome']
y_test = [1 if x=='Yes' else 0 for x in y_test]
x_test = data_test[data_test.columns[1:]]
x_test = x_test.drop(columns = ['cumulative_outcome'],axis=1)
xd_test = pd.get_dummies(x_test)
```

In [68]:
```
#treatment of missing data: we put the most common observation in place of th
for column in xd_test.columns:
    xd_test[column]=xd_test[column].fillna(xd_test[column].mode()[0])
```

In [71]:
```
from sklearn.metrics import accuracy_score


print('Accuracy:', round(accuracy_score(y_test, gbc.predict(xd_test)), 5))
```

```
Accuracy: 0.57156
```

In [72]:
```
from sklearn.metrics import roc_curve, auc

fpr, tpr, _ = roc_curve(y_test, gbc.predict(xd_test))
```

In [73]:
```
fpr
```

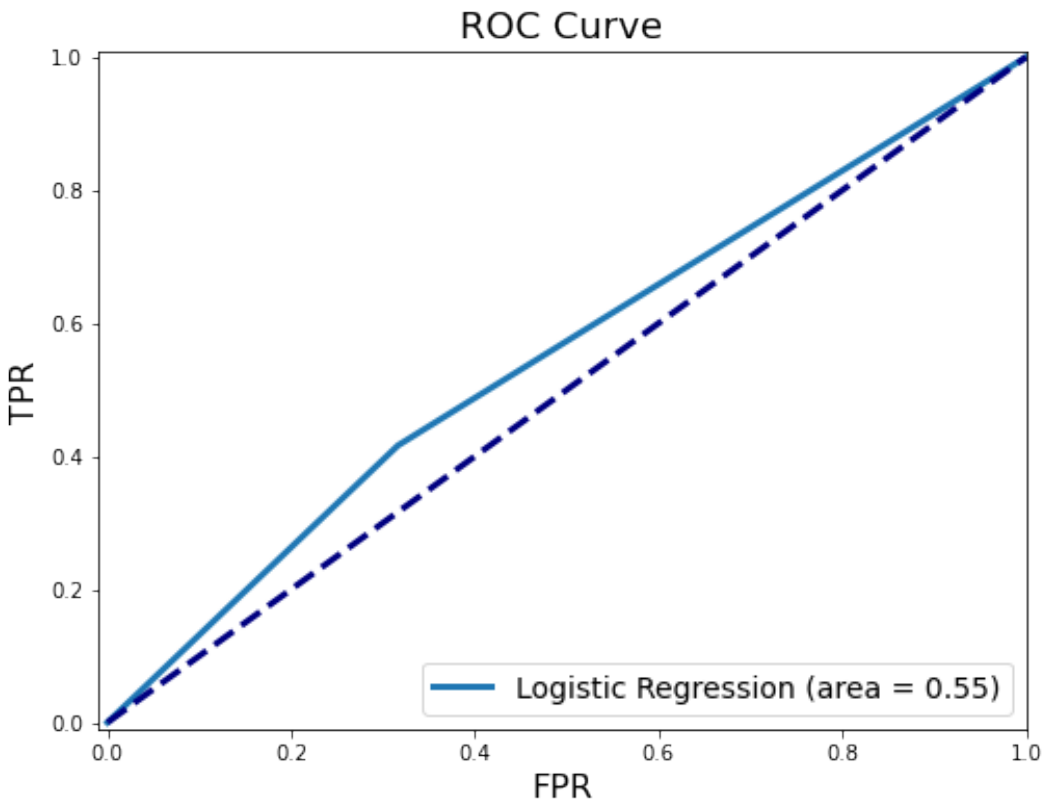
Out[73]:
```
array([0.        , 0.31627907, 1.        ])
```

In [74]:
```
tpr
```

Out[74]:
```
array([0.        , 0.41630901, 1.        ])
```

First model : accuracy 0.57, fpr = 0.32, tpr = 0.41, auc = 0.55

In [75]:
```
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(8, 6))
plt.title('ROC Curve', fontsize=18)
plt.xlabel('FPR', fontsize=16)
plt.ylabel('TPR', fontsize=16)
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr, tpr, lw=3, label='Logistic Regression (area = {:0.2f})'.format(
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.legend(loc='lower right', fontsize=14)
plt.show()
```



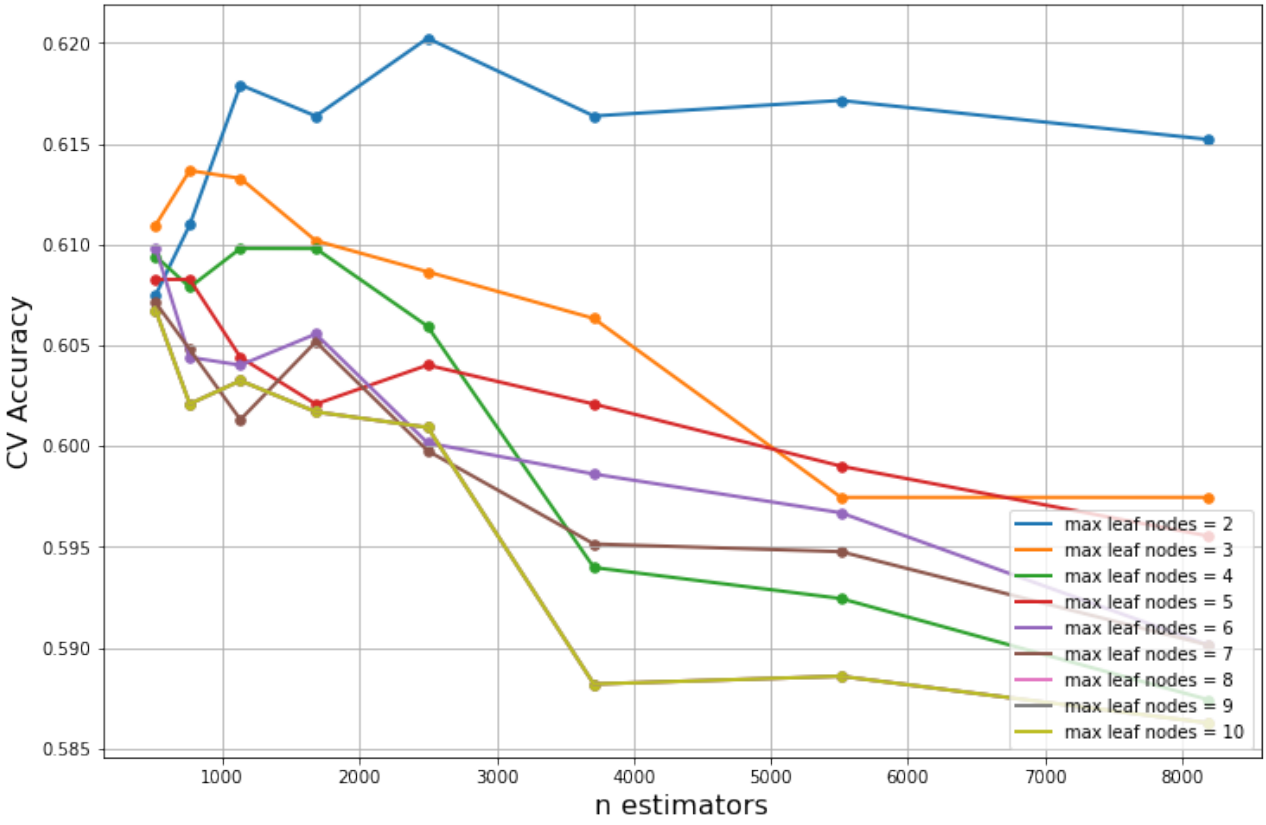
In [82]:

```
#Model 2: Gradient Boosted classifier with cross-validation
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model_selection import GridSearchCV



import time
grid_values = {'max_leaf_nodes': np.linspace(2, 10, 9, dtype='int32'),
               'min_samples_leaf': [10],
               'n_estimators': np.logspace(9, 13, num=8, base=2, dtype='int32
               'learning_rate': [0.01],
               'random_state': [88]}

tic = time.time()

gbc = GradientBoostingClassifier()
gbc_cv = GridSearchCV(gbc, param_grid=grid_values, scoring='accuracy', cv=5, 
gbc_cv.fit(xd, y_train)

toc = time.time()
print('time:', round(toc-tic, 2),'s')
```

```
  File "<ipython-input-82-e45516098f97>", line 10
    'min_samples_leaf': [10],
    ^
IndentationError: unexpected indent
```

In [83]:

```
n_estimators = gbc_cv.cv_results_['param_n_estimators'].data
accuracy_scores = gbc_cv.cv_results_['mean_test_score']

plt.figure(figsize=(12, 8))
plt.xlabel('n estimators', fontsize=16)
plt.ylabel('CV Accuracy', fontsize=16)
plt.grid(True, which='both')

N = len(grid_values['max_leaf_nodes'])
M = len(grid_values['n_estimators'])
for i in range(N):
    plt.scatter(n_estimators[(M*i):(M*i)+M], accuracy_scores[(M*i):(M*i)+M], 
    plt.plot(n_estimators[(M*i):(M*i)+M], accuracy_scores[(M*i):(M*i)+M], lin
             label='max leaf nodes = '+str(grid_values['max_leaf_nodes'][i]))
plt.legend(loc='lower right')
plt.show()
```



In [84]:

```
print(gbc_cv.best_params_)
```

```
{'learning_rate': 0.01, 'max_leaf_nodes': 2, 'min_samples_leaf': 10, 'n_estima
tors': 2496, 'random_state': 88}
```

In [85]:

```
gbc_opt = GradientBoostingClassifier(n_estimators=2496, verbose=3,min_samples
                                     max_leaf_nodes=2,learning_rate = 0.01, random
gbc_opt.fit(xd,y_train)
```

```
      Iter       Train Loss   Remaining Time
         1           1.3576            3.43m
         2           1.3569            2.56m
         3           1.3563            2.26m
         4           1.3556            2.02m
         5           1.3550            1.92m
         6           1.3544            1.93m
         7           1.3538            1.99m
         8           1.3532            2.17m
         9           1.3526            2.18m
        10           1.3521            2.19m
        11           1.3515            2.28m
        12           1.3510            2.43m
        13           1.3505            2.40m
        14           1.3499            2.32m
        15           1.3494            2.26m
        16           1.3489            2.20m
        17           1.3484            2.17m
        18           1.3479            2.26m
        19           1.3474            2.33m
        20           1.3469            2.31m
        21           1.3464            2.32m
        22           1.3459            2.36m
        23           1.3454            2.35m
        24           1.3450            2.31m
        25           1.3445            2.27m
        26           1.3440            2.25m
        27           1.3436            2.26m
        28           1.3431            2.31m
        29           1.3427            2.30m
        30           1.3423            2.29m
        31           1.3418            2.30m
        32           1.3414            2.32m
        33           1.3410            2.30m
        34           1.3406            2.28m
        35           1.3401            2.25m
        36           1.3397            2.22m
        37           1.3393            2.25m
        38           1.3389            2.28m
        39           1.3385            2.34m
        40           1.3381            2.51m
        41           1.3378            2.61m
        42           1.3374            2.63m
        43           1.3370            2.62m
        44           1.3366            2.59m
        45           1.3363            2.56m
        46           1.3359            2.55m
        47           1.3355            2.56m
        48           1.3352            2.55m
```

49	1.3348	2.54m
50	1.3345	2.53m
51	1.3341	2.52m
52	1.3338	2.51m
53	1.3335	2.49m
54	1.3331	2.47m
55	1.3328	2.46m
56	1.3325	2.48m
57	1.3322	2.49m
58	1.3318	2.49m
59	1.3315	2.48m
60	1.3312	2.48m
61	1.3309	2.48m
62	1.3306	2.48m
63	1.3303	2.50m
64	1.3300	2.49m
65	1.3297	2.49m
66	1.3294	2.49m
67	1.3291	2.47m
68	1.3288	2.48m
69	1.3285	2.48m
70	1.3282	2.48m
71	1.3279	2.49m
72	1.3276	2.48m
73	1.3274	2.46m
74	1.3271	2.45m
75	1.3268	2.44m
76	1.3265	2.43m
77	1.3263	2.46m
78	1.3260	2.45m
79	1.3257	2.44m
80	1.3255	2.43m
81	1.3252	2.42m
82	1.3250	2.40m
83	1.3247	2.39m
84	1.3244	2.39m
85	1.3242	2.38m
86	1.3240	2.39m
87	1.3237	2.40m
88	1.3235	2.40m
89	1.3232	2.39m
90	1.3230	2.38m
91	1.3227	2.37m
92	1.3225	2.36m
93	1.3223	2.35m
94	1.3220	2.34m
95	1.3218	2.34m
96	1.3216	2.34m
97	1.3214	2.33m
98	1.3211	2.31m
99	1.3209	2.30m
100	1.3207	2.31m
101	1.3205	2.29m
102	1.3203	2.30m
103	1.3201	2.31m
104	1.3198	2.31m
105	1.3196	2.31m

106	1.3194	2.30m
107	1.3192	2.29m
108	1.3190	2.29m
109	1.3188	2.29m
110	1.3186	2.30m
111	1.3184	2.31m
112	1.3182	2.34m
113	1.3180	2.34m
114	1.3178	2.34m
115	1.3176	2.32m
116	1.3174	2.32m
117	1.3173	2.31m
118	1.3171	2.30m
119	1.3169	2.29m
120	1.3167	2.29m
121	1.3165	2.30m
122	1.3163	2.31m
123	1.3161	2.33m
124	1.3160	2.33m
125	1.3158	2.33m
126	1.3156	2.33m
127	1.3154	2.33m
128	1.3153	2.34m
129	1.3151	2.35m
130	1.3149	2.35m
131	1.3147	2.34m
132	1.3146	2.33m
133	1.3144	2.32m
134	1.3142	2.32m
135	1.3140	2.32m
136	1.3139	2.33m
137	1.3137	2.33m
138	1.3135	2.33m
139	1.3134	2.33m
140	1.3132	2.32m
141	1.3130	2.31m
142	1.3129	2.34m
143	1.3127	2.34m
144	1.3126	2.34m
145	1.3124	2.33m
146	1.3122	2.33m
147	1.3121	2.32m
148	1.3119	2.32m
149	1.3118	2.33m
150	1.3116	2.33m
151	1.3114	2.32m
152	1.3113	2.32m
153	1.3111	2.32m
154	1.3110	2.31m
155	1.3108	2.30m
156	1.3107	2.30m
157	1.3105	2.30m
158	1.3104	2.30m
159	1.3102	2.29m
160	1.3101	2.29m
161	1.3099	2.28m
162	1.3098	2.28m

163	1.3096	2.27m
164	1.3095	2.27m
165	1.3093	2.26m
166	1.3092	2.26m
167	1.3090	2.25m
168	1.3089	2.25m
169	1.3088	2.25m
170	1.3086	2.24m
171	1.3085	2.23m
172	1.3083	2.23m
173	1.3082	2.22m
174	1.3081	2.22m
175	1.3079	2.21m
176	1.3078	2.21m
177	1.3076	2.20m
178	1.3075	2.20m
179	1.3074	2.20m
180	1.3072	2.19m
181	1.3071	2.19m
182	1.3070	2.18m
183	1.3068	2.17m
184	1.3067	2.17m
185	1.3066	2.17m
186	1.3064	2.17m
187	1.3063	2.17m
188	1.3062	2.16m
189	1.3061	2.16m
190	1.3059	2.15m
191	1.3058	2.14m
192	1.3057	2.14m
193	1.3056	2.13m
194	1.3054	2.13m
195	1.3053	2.13m
196	1.3052	2.13m
197	1.3051	2.12m
198	1.3049	2.12m
199	1.3048	2.12m
200	1.3047	2.12m
201	1.3046	2.12m
202	1.3044	2.11m
203	1.3043	2.11m
204	1.3042	2.11m
205	1.3041	2.10m
206	1.3040	2.10m
207	1.3039	2.09m
208	1.3037	2.09m
209	1.3036	2.08m
210	1.3035	2.09m
211	1.3034	2.10m
212	1.3033	2.10m
213	1.3032	2.09m
214	1.3031	2.09m
215	1.3029	2.08m
216	1.3028	2.08m
217	1.3027	2.07m
218	1.3026	2.07m
219	1.3025	2.06m

220	1.3024	2.06m
221	1.3023	2.05m
222	1.3022	2.05m
223	1.3021	2.04m
224	1.3019	2.04m
225	1.3018	2.04m
226	1.3017	2.03m
227	1.3016	2.03m
228	1.3015	2.02m
229	1.3014	2.02m
230	1.3013	2.02m
231	1.3012	2.01m
232	1.3011	2.01m
233	1.3010	2.00m
234	1.3009	2.00m
235	1.3008	1.99m
236	1.3007	1.99m
237	1.3006	1.99m
238	1.3005	1.99m
239	1.3004	1.99m
240	1.3003	1.99m
241	1.3002	1.99m
242	1.3001	1.99m
243	1.3000	1.98m
244	1.2999	1.98m
245	1.2998	1.97m
246	1.2997	1.97m
247	1.2996	1.97m
248	1.2995	1.96m
249	1.2994	1.96m
250	1.2993	1.95m
251	1.2992	1.95m
252	1.2991	1.95m
253	1.2990	1.94m
254	1.2989	1.94m
255	1.2988	1.94m
256	1.2987	1.93m
257	1.2986	1.93m
258	1.2985	1.93m
259	1.2984	1.92m
260	1.2983	1.92m
261	1.2982	1.92m
262	1.2982	1.91m
263	1.2981	1.91m
264	1.2980	1.91m
265	1.2979	1.90m
266	1.2978	1.90m
267	1.2977	1.90m
268	1.2976	1.90m
269	1.2975	1.90m
270	1.2974	1.89m
271	1.2973	1.89m
272	1.2972	1.89m
273	1.2972	1.88m
274	1.2971	1.88m
275	1.2970	1.88m
276	1.2969	1.88m

277	1.2968	1.87m
278	1.2967	1.87m
279	1.2966	1.87m
280	1.2965	1.86m
281	1.2964	1.86m
282	1.2964	1.86m
283	1.2963	1.85m
284	1.2962	1.85m
285	1.2961	1.85m
286	1.2960	1.85m
287	1.2959	1.84m
288	1.2958	1.84m
289	1.2958	1.84m
290	1.2957	1.84m
291	1.2956	1.83m
292	1.2955	1.83m
293	1.2954	1.83m
294	1.2953	1.82m
295	1.2953	1.82m
296	1.2952	1.82m
297	1.2951	1.82m
298	1.2950	1.81m
299	1.2949	1.81m
300	1.2948	1.81m
301	1.2948	1.80m
302	1.2947	1.80m
303	1.2946	1.80m
304	1.2945	1.80m
305	1.2944	1.79m
306	1.2943	1.79m
307	1.2943	1.79m
308	1.2942	1.79m
309	1.2941	1.78m
310	1.2940	1.78m
311	1.2940	1.78m
312	1.2939	1.77m
313	1.2938	1.77m
314	1.2937	1.77m
315	1.2936	1.77m
316	1.2936	1.76m
317	1.2935	1.76m
318	1.2934	1.76m
319	1.2933	1.76m
320	1.2932	1.76m
321	1.2932	1.75m
322	1.2931	1.75m
323	1.2930	1.75m
324	1.2929	1.75m
325	1.2929	1.74m
326	1.2928	1.74m
327	1.2927	1.74m
328	1.2926	1.74m
329	1.2926	1.73m
330	1.2925	1.73m
331	1.2924	1.73m
332	1.2923	1.73m
333	1.2923	1.73m

334	1.2922	1.73m
335	1.2921	1.73m
336	1.2920	1.72m
337	1.2920	1.72m
338	1.2919	1.72m
339	1.2918	1.72m
340	1.2917	1.71m
341	1.2917	1.71m
342	1.2916	1.71m
343	1.2915	1.71m
344	1.2915	1.70m
345	1.2914	1.70m
346	1.2913	1.70m
347	1.2912	1.70m
348	1.2912	1.70m
349	1.2911	1.69m
350	1.2910	1.69m
351	1.2910	1.69m
352	1.2909	1.69m
353	1.2908	1.68m
354	1.2907	1.68m
355	1.2907	1.68m
356	1.2906	1.68m
357	1.2905	1.68m
358	1.2905	1.67m
359	1.2904	1.67m
360	1.2903	1.67m
361	1.2903	1.67m
362	1.2902	1.67m
363	1.2901	1.66m
364	1.2901	1.66m
365	1.2900	1.67m
366	1.2899	1.67m
367	1.2899	1.67m
368	1.2898	1.66m
369	1.2897	1.66m
370	1.2897	1.66m
371	1.2896	1.66m
372	1.2895	1.65m
373	1.2895	1.65m
374	1.2894	1.65m
375	1.2893	1.65m
376	1.2893	1.65m
377	1.2892	1.64m
378	1.2891	1.64m
379	1.2891	1.64m
380	1.2890	1.64m
381	1.2889	1.64m
382	1.2889	1.63m
383	1.2888	1.63m
384	1.2887	1.63m
385	1.2887	1.63m
386	1.2886	1.63m
387	1.2885	1.63m
388	1.2885	1.62m
389	1.2884	1.62m
390	1.2883	1.62m

391	1.2883	1.62m
392	1.2882	1.62m
393	1.2882	1.62m
394	1.2881	1.61m
395	1.2880	1.61m
396	1.2880	1.61m
397	1.2879	1.61m
398	1.2878	1.61m
399	1.2878	1.61m
400	1.2877	1.61m
401	1.2876	1.60m
402	1.2876	1.60m
403	1.2875	1.60m
404	1.2875	1.60m
405	1.2874	1.60m
406	1.2873	1.60m
407	1.2873	1.59m
408	1.2872	1.59m
409	1.2872	1.59m
410	1.2871	1.59m
411	1.2870	1.59m
412	1.2870	1.58m
413	1.2869	1.58m
414	1.2868	1.58m
415	1.2868	1.58m
416	1.2867	1.58m
417	1.2867	1.58m
418	1.2866	1.58m
419	1.2865	1.58m
420	1.2865	1.58m
421	1.2864	1.58m
422	1.2864	1.58m
423	1.2863	1.58m
424	1.2862	1.58m
425	1.2862	1.58m
426	1.2861	1.58m
427	1.2861	1.58m
428	1.2860	1.58m
429	1.2859	1.59m
430	1.2859	1.59m
431	1.2858	1.58m
432	1.2858	1.58m
433	1.2857	1.58m
434	1.2857	1.58m
435	1.2856	1.58m
436	1.2855	1.58m
437	1.2855	1.58m
438	1.2854	1.58m
439	1.2854	1.58m
440	1.2853	1.58m
441	1.2852	1.58m
442	1.2852	1.58m
443	1.2851	1.58m
444	1.2851	1.58m
445	1.2850	1.58m
446	1.2850	1.58m
447	1.2849	1.57m

448	1.2848	1.57m
449	1.2848	1.57m
450	1.2847	1.57m
451	1.2847	1.57m
452	1.2846	1.57m
453	1.2846	1.57m
454	1.2845	1.57m
455	1.2844	1.58m
456	1.2844	1.58m
457	1.2843	1.58m
458	1.2843	1.58m
459	1.2842	1.57m
460	1.2842	1.57m
461	1.2841	1.57m
462	1.2841	1.57m
463	1.2840	1.57m
464	1.2839	1.57m
465	1.2839	1.57m
466	1.2838	1.57m
467	1.2838	1.57m
468	1.2837	1.57m
469	1.2837	1.57m
470	1.2836	1.57m
471	1.2836	1.57m
472	1.2835	1.56m
473	1.2835	1.56m
474	1.2834	1.56m
475	1.2833	1.56m
476	1.2833	1.56m
477	1.2832	1.56m
478	1.2832	1.56m
479	1.2831	1.56m
480	1.2831	1.56m
481	1.2830	1.56m
482	1.2830	1.56m
483	1.2829	1.56m
484	1.2829	1.56m
485	1.2828	1.56m
486	1.2828	1.56m
487	1.2827	1.56m
488	1.2827	1.56m
489	1.2826	1.56m
490	1.2825	1.56m
491	1.2825	1.56m
492	1.2824	1.56m
493	1.2824	1.56m
494	1.2823	1.56m
495	1.2823	1.56m
496	1.2822	1.56m
497	1.2822	1.56m
498	1.2821	1.56m
499	1.2821	1.56m
500	1.2820	1.56m
501	1.2820	1.56m
502	1.2819	1.55m
503	1.2819	1.55m
504	1.2818	1.55m

505	1.2818	1.55m
506	1.2817	1.55m
507	1.2817	1.55m
508	1.2816	1.55m
509	1.2816	1.54m
510	1.2815	1.54m
511	1.2815	1.54m
512	1.2814	1.54m
513	1.2814	1.54m
514	1.2813	1.54m
515	1.2813	1.54m
516	1.2812	1.54m
517	1.2812	1.53m
518	1.2811	1.53m
519	1.2811	1.53m
520	1.2810	1.53m
521	1.2810	1.53m
522	1.2809	1.53m
523	1.2809	1.52m
524	1.2808	1.52m
525	1.2808	1.52m
526	1.2807	1.52m
527	1.2807	1.52m
528	1.2806	1.52m
529	1.2806	1.52m
530	1.2805	1.51m
531	1.2805	1.51m
532	1.2804	1.51m
533	1.2804	1.51m
534	1.2803	1.51m
535	1.2803	1.51m
536	1.2802	1.51m
537	1.2802	1.51m
538	1.2801	1.50m
539	1.2801	1.50m
540	1.2800	1.50m
541	1.2800	1.50m
542	1.2799	1.50m
543	1.2799	1.50m
544	1.2798	1.50m
545	1.2798	1.49m
546	1.2797	1.49m
547	1.2797	1.49m
548	1.2796	1.49m
549	1.2796	1.49m
550	1.2795	1.49m
551	1.2795	1.48m
552	1.2794	1.48m
553	1.2794	1.48m
554	1.2794	1.48m
555	1.2793	1.48m
556	1.2793	1.48m
557	1.2792	1.48m
558	1.2792	1.48m
559	1.2791	1.48m
560	1.2791	1.48m
561	1.2790	1.48m

562	1.2790	1.48m
563	1.2789	1.48m
564	1.2789	1.47m
565	1.2788	1.47m
566	1.2788	1.47m
567	1.2787	1.47m
568	1.2787	1.47m
569	1.2787	1.47m
570	1.2786	1.47m
571	1.2786	1.47m
572	1.2785	1.47m
573	1.2785	1.47m
574	1.2784	1.47m
575	1.2784	1.47m
576	1.2783	1.47m
577	1.2783	1.47m
578	1.2782	1.46m
579	1.2782	1.46m
580	1.2781	1.47m
581	1.2781	1.47m
582	1.2780	1.46m
583	1.2780	1.46m
584	1.2780	1.46m
585	1.2779	1.46m
586	1.2779	1.46m
587	1.2778	1.46m
588	1.2778	1.46m
589	1.2777	1.46m
590	1.2777	1.46m
591	1.2776	1.46m
592	1.2776	1.46m
593	1.2776	1.46m
594	1.2775	1.46m
595	1.2775	1.46m
596	1.2774	1.46m
597	1.2774	1.45m
598	1.2773	1.45m
599	1.2773	1.45m
600	1.2772	1.45m
601	1.2772	1.45m
602	1.2772	1.45m
603	1.2771	1.45m
604	1.2771	1.45m
605	1.2770	1.45m
606	1.2770	1.45m
607	1.2769	1.44m
608	1.2769	1.44m
609	1.2768	1.44m
610	1.2768	1.44m
611	1.2768	1.44m
612	1.2767	1.44m
613	1.2767	1.44m
614	1.2766	1.43m
615	1.2766	1.43m
616	1.2765	1.43m
617	1.2765	1.43m
618	1.2765	1.43m

619	1.2764	1.43m
620	1.2764	1.43m
621	1.2763	1.43m
622	1.2763	1.43m
623	1.2762	1.43m
624	1.2762	1.43m
625	1.2762	1.42m
626	1.2761	1.42m
627	1.2761	1.42m
628	1.2760	1.42m
629	1.2760	1.42m
630	1.2759	1.42m
631	1.2759	1.42m
632	1.2759	1.42m
633	1.2758	1.42m
634	1.2758	1.42m
635	1.2757	1.42m
636	1.2757	1.42m
637	1.2756	1.42m
638	1.2756	1.42m
639	1.2756	1.41m
640	1.2755	1.41m
641	1.2755	1.41m
642	1.2754	1.41m
643	1.2754	1.41m
644	1.2754	1.41m
645	1.2753	1.41m
646	1.2753	1.41m
647	1.2752	1.41m
648	1.2752	1.41m
649	1.2751	1.41m
650	1.2751	1.41m
651	1.2751	1.41m
652	1.2750	1.41m
653	1.2750	1.40m
654	1.2749	1.40m
655	1.2749	1.40m
656	1.2749	1.40m
657	1.2748	1.40m
658	1.2748	1.40m
659	1.2747	1.40m
660	1.2747	1.40m
661	1.2746	1.40m
662	1.2746	1.40m
663	1.2746	1.40m
664	1.2745	1.40m
665	1.2745	1.40m
666	1.2744	1.40m
667	1.2744	1.40m
668	1.2744	1.40m
669	1.2743	1.40m
670	1.2743	1.40m
671	1.2742	1.40m
672	1.2742	1.40m
673	1.2742	1.40m
674	1.2741	1.40m
675	1.2741	1.40m

676	1.2740	1.39m
677	1.2740	1.39m
678	1.2740	1.39m
679	1.2739	1.39m
680	1.2739	1.39m
681	1.2738	1.39m
682	1.2738	1.39m
683	1.2738	1.39m
684	1.2737	1.38m
685	1.2737	1.38m
686	1.2736	1.38m
687	1.2736	1.38m
688	1.2736	1.38m
689	1.2735	1.38m
690	1.2735	1.38m
691	1.2734	1.38m
692	1.2734	1.38m
693	1.2734	1.38m
694	1.2733	1.37m
695	1.2733	1.37m
696	1.2732	1.37m
697	1.2732	1.37m
698	1.2732	1.37m
699	1.2731	1.37m
700	1.2731	1.37m
701	1.2730	1.37m
702	1.2730	1.36m
703	1.2730	1.36m
704	1.2729	1.36m
705	1.2729	1.36m
706	1.2729	1.36m
707	1.2728	1.36m
708	1.2728	1.36m
709	1.2727	1.36m
710	1.2727	1.35m
711	1.2727	1.35m
712	1.2726	1.35m
713	1.2726	1.35m
714	1.2725	1.35m
715	1.2725	1.35m
716	1.2725	1.35m
717	1.2724	1.35m
718	1.2724	1.34m
719	1.2724	1.34m
720	1.2723	1.34m
721	1.2723	1.34m
722	1.2722	1.34m
723	1.2722	1.34m
724	1.2722	1.34m
725	1.2721	1.34m
726	1.2721	1.33m
727	1.2720	1.33m
728	1.2720	1.33m
729	1.2720	1.33m
730	1.2719	1.33m
731	1.2719	1.33m
732	1.2719	1.33m

```
733             1.2718              1.33m
734             1.2718              1.33m
735             1.2717              1.33m
736             1.2717              1.33m
737             1.2717              1.33m
738             1.2716              1.32m
739             1.2716              1.32m
740             1.2716              1.33m
741             1.2715              1.33m
742             1.2715              1.33m
743             1.2714              1.32m
744             1.2714              1.32m
745             1.2714              1.32m
746             1.2713              1.32m
747             1.2713              1.32m
748             1.2713              1.32m
749             1.2712              1.32m
750             1.2712              1.32m
751             1.2711              1.32m
752             1.2711              1.32m
753             1.2711              1.32m
754             1.2710              1.32m
755             1.2710              1.32m
756             1.2710              1.32m
757             1.2709              1.32m
758             1.2709              1.32m
759             1.2709              1.32m
760             1.2708              1.31m
761             1.2708              1.31m
762             1.2707              1.31m
763             1.2707              1.31m
764             1.2707              1.31m
765             1.2706              1.31m
766             1.2706              1.31m
767             1.2706              1.31m
768             1.2705              1.31m
769             1.2705              1.30m
770             1.2704              1.30m
771             1.2704              1.30m
772             1.2704              1.30m
773             1.2703              1.30m
774             1.2703              1.30m
775             1.2703              1.30m
776             1.2702              1.30m
777             1.2702              1.30m
778             1.2702              1.30m
779             1.2701              1.30m
780             1.2701              1.30m
781             1.2701              1.30m
782             1.2700              1.30m
783             1.2700              1.30m
784             1.2699              1.30m
785             1.2699              1.30m
786             1.2699              1.30m
787             1.2698              1.29m
788             1.2698              1.29m
789             1.2698              1.29m
```

790	1.2697	1.29m
791	1.2697	1.29m
792	1.2697	1.29m
793	1.2696	1.29m
794	1.2696	1.29m
795	1.2695	1.29m
796	1.2695	1.29m
797	1.2695	1.29m
798	1.2694	1.29m
799	1.2694	1.29m
800	1.2694	1.29m
801	1.2693	1.29m
802	1.2693	1.29m
803	1.2693	1.28m
804	1.2692	1.28m
805	1.2692	1.28m
806	1.2692	1.28m
807	1.2691	1.28m
808	1.2691	1.28m
809	1.2691	1.28m
810	1.2690	1.28m
811	1.2690	1.28m
812	1.2690	1.27m
813	1.2689	1.27m
814	1.2689	1.27m
815	1.2689	1.27m
816	1.2688	1.27m
817	1.2688	1.27m
818	1.2687	1.27m
819	1.2687	1.27m
820	1.2687	1.27m
821	1.2686	1.26m
822	1.2686	1.26m
823	1.2686	1.26m
824	1.2685	1.26m
825	1.2685	1.26m
826	1.2685	1.26m
827	1.2684	1.26m
828	1.2684	1.26m
829	1.2684	1.26m
830	1.2683	1.25m
831	1.2683	1.25m
832	1.2683	1.25m
833	1.2682	1.25m
834	1.2682	1.25m
835	1.2682	1.25m
836	1.2681	1.25m
837	1.2681	1.25m
838	1.2681	1.25m
839	1.2680	1.25m
840	1.2680	1.24m
841	1.2680	1.24m
842	1.2679	1.24m
843	1.2679	1.24m
844	1.2679	1.24m
845	1.2678	1.24m
846	1.2678	1.24m

847	1.2678	1.24m
848	1.2677	1.24m
849	1.2677	1.23m
850	1.2677	1.23m
851	1.2676	1.23m
852	1.2676	1.23m
853	1.2676	1.23m
854	1.2675	1.23m
855	1.2675	1.23m
856	1.2675	1.23m
857	1.2674	1.23m
858	1.2674	1.23m
859	1.2674	1.22m
860	1.2673	1.22m
861	1.2673	1.22m
862	1.2673	1.22m
863	1.2672	1.22m
864	1.2672	1.22m
865	1.2672	1.22m
866	1.2671	1.22m
867	1.2671	1.22m
868	1.2671	1.22m
869	1.2670	1.22m
870	1.2670	1.22m
871	1.2670	1.21m
872	1.2669	1.21m
873	1.2669	1.21m
874	1.2669	1.21m
875	1.2668	1.21m
876	1.2668	1.21m
877	1.2668	1.21m
878	1.2667	1.21m
879	1.2667	1.21m
880	1.2667	1.21m
881	1.2666	1.21m
882	1.2666	1.21m
883	1.2666	1.20m
884	1.2665	1.20m
885	1.2665	1.20m
886	1.2665	1.20m
887	1.2664	1.20m
888	1.2664	1.20m
889	1.2664	1.20m
890	1.2663	1.20m
891	1.2663	1.20m
892	1.2663	1.20m
893	1.2662	1.20m
894	1.2662	1.19m
895	1.2662	1.19m
896	1.2661	1.19m
897	1.2661	1.19m
898	1.2661	1.19m
899	1.2660	1.19m
900	1.2660	1.19m
901	1.2660	1.19m
902	1.2660	1.19m
903	1.2659	1.19m

904	1.2659	1.19m
905	1.2659	1.19m
906	1.2658	1.19m
907	1.2658	1.19m
908	1.2658	1.19m
909	1.2657	1.19m
910	1.2657	1.18m
911	1.2657	1.18m
912	1.2656	1.18m
913	1.2656	1.18m
914	1.2656	1.18m
915	1.2655	1.18m
916	1.2655	1.18m
917	1.2655	1.18m
918	1.2654	1.18m
919	1.2654	1.18m
920	1.2654	1.18m
921	1.2653	1.18m
922	1.2653	1.18m
923	1.2653	1.18m
924	1.2653	1.18m
925	1.2652	1.18m
926	1.2652	1.18m
927	1.2652	1.19m
928	1.2651	1.19m
929	1.2651	1.18m
930	1.2651	1.19m
931	1.2650	1.19m
932	1.2650	1.19m
933	1.2650	1.19m
934	1.2649	1.18m
935	1.2649	1.18m
936	1.2649	1.18m
937	1.2648	1.18m
938	1.2648	1.18m
939	1.2648	1.18m
940	1.2648	1.18m
941	1.2647	1.18m
942	1.2647	1.18m
943	1.2647	1.18m
944	1.2646	1.17m
945	1.2646	1.17m
946	1.2646	1.17m
947	1.2645	1.17m
948	1.2645	1.17m
949	1.2645	1.17m
950	1.2644	1.17m
951	1.2644	1.17m
952	1.2644	1.17m
953	1.2644	1.17m
954	1.2643	1.17m
955	1.2643	1.16m
956	1.2643	1.16m
957	1.2642	1.16m
958	1.2642	1.16m
959	1.2642	1.16m
960	1.2641	1.16m

961	1.2641	1.16m
962	1.2641	1.16m
963	1.2640	1.16m
964	1.2640	1.16m
965	1.2640	1.16m
966	1.2640	1.15m
967	1.2639	1.15m
968	1.2639	1.15m
969	1.2639	1.15m
970	1.2638	1.15m
971	1.2638	1.15m
972	1.2638	1.15m
973	1.2637	1.15m
974	1.2637	1.15m
975	1.2637	1.15m
976	1.2637	1.15m
977	1.2636	1.15m
978	1.2636	1.15m
979	1.2636	1.15m
980	1.2635	1.14m
981	1.2635	1.14m
982	1.2635	1.14m
983	1.2634	1.14m
984	1.2634	1.14m
985	1.2634	1.14m
986	1.2634	1.14m
987	1.2633	1.14m
988	1.2633	1.14m
989	1.2633	1.14m
990	1.2632	1.13m
991	1.2632	1.13m
992	1.2632	1.13m
993	1.2632	1.13m
994	1.2631	1.13m
995	1.2631	1.13m
996	1.2631	1.13m
997	1.2630	1.13m
998	1.2630	1.13m
999	1.2630	1.12m
1000	1.2629	1.12m
1001	1.2629	1.12m
1002	1.2629	1.12m
1003	1.2629	1.12m
1004	1.2628	1.12m
1005	1.2628	1.12m
1006	1.2628	1.12m
1007	1.2627	1.12m
1008	1.2627	1.12m
1009	1.2627	1.12m
1010	1.2627	1.12m
1011	1.2626	1.12m
1012	1.2626	1.12m
1013	1.2626	1.12m
1014	1.2625	1.12m
1015	1.2625	1.11m
1016	1.2625	1.11m
1017	1.2624	1.11m

1018	1.2624	1.11m
1019	1.2624	1.11m
1020	1.2624	1.11m
1021	1.2623	1.11m
1022	1.2623	1.11m
1023	1.2623	1.11m
1024	1.2622	1.11m
1025	1.2622	1.11m
1026	1.2622	1.11m
1027	1.2622	1.11m
1028	1.2621	1.11m
1029	1.2621	1.10m
1030	1.2621	1.10m
1031	1.2620	1.10m
1032	1.2620	1.10m
1033	1.2620	1.10m
1034	1.2620	1.10m
1035	1.2619	1.10m
1036	1.2619	1.10m
1037	1.2619	1.10m
1038	1.2618	1.10m
1039	1.2618	1.10m
1040	1.2618	1.10m
1041	1.2618	1.09m
1042	1.2617	1.09m
1043	1.2617	1.09m
1044	1.2617	1.09m
1045	1.2616	1.09m
1046	1.2616	1.09m
1047	1.2616	1.09m
1048	1.2616	1.09m
1049	1.2615	1.09m
1050	1.2615	1.09m
1051	1.2615	1.08m
1052	1.2615	1.08m
1053	1.2614	1.08m
1054	1.2614	1.08m
1055	1.2614	1.08m
1056	1.2613	1.08m
1057	1.2613	1.08m
1058	1.2613	1.08m
1059	1.2613	1.08m
1060	1.2612	1.08m
1061	1.2612	1.08m
1062	1.2612	1.07m
1063	1.2611	1.07m
1064	1.2611	1.07m
1065	1.2611	1.07m
1066	1.2611	1.07m
1067	1.2610	1.07m
1068	1.2610	1.07m
1069	1.2610	1.07m
1070	1.2610	1.07m
1071	1.2609	1.07m
1072	1.2609	1.06m
1073	1.2609	1.06m
1074	1.2608	1.06m

1075	1.2608	1.06m
1076	1.2608	1.06m
1077	1.2608	1.06m
1078	1.2607	1.06m
1079	1.2607	1.06m
1080	1.2607	1.06m
1081	1.2606	1.06m
1082	1.2606	1.06m
1083	1.2606	1.06m
1084	1.2606	1.05m
1085	1.2605	1.05m
1086	1.2605	1.05m
1087	1.2605	1.05m
1088	1.2605	1.05m
1089	1.2604	1.05m
1090	1.2604	1.05m
1091	1.2604	1.05m
1092	1.2603	1.05m
1093	1.2603	1.05m
1094	1.2603	1.04m
1095	1.2603	1.04m
1096	1.2602	1.04m
1097	1.2602	1.04m
1098	1.2602	1.04m
1099	1.2602	1.04m
1100	1.2601	1.04m
1101	1.2601	1.04m
1102	1.2601	1.04m
1103	1.2600	1.04m
1104	1.2600	1.03m
1105	1.2600	1.03m
1106	1.2600	1.03m
1107	1.2599	1.03m
1108	1.2599	1.03m
1109	1.2599	1.03m
1110	1.2599	1.03m
1111	1.2598	1.03m
1112	1.2598	1.03m
1113	1.2598	1.03m
1114	1.2597	1.03m
1115	1.2597	1.02m
1116	1.2597	1.02m
1117	1.2597	1.02m
1118	1.2596	1.02m
1119	1.2596	1.02m
1120	1.2596	1.02m
1121	1.2596	1.02m
1122	1.2595	1.02m
1123	1.2595	1.02m
1124	1.2595	1.02m
1125	1.2595	1.02m
1126	1.2594	1.02m
1127	1.2594	1.02m
1128	1.2594	1.02m
1129	1.2593	1.02m
1130	1.2593	1.01m
1131	1.2593	1.01m

1132	1.2593	1.01m
1133	1.2592	1.01m
1134	1.2592	1.01m
1135	1.2592	1.01m
1136	1.2592	1.01m
1137	1.2591	1.01m
1138	1.2591	1.01m
1139	1.2591	1.01m
1140	1.2591	1.01m
1141	1.2590	1.01m
1142	1.2590	1.01m
1143	1.2590	1.01m
1144	1.2590	1.00m
1145	1.2589	1.00m
1146	1.2589	1.00m
1147	1.2589	1.00m
1148	1.2588	1.00m
1149	1.2588	1.00m
1150	1.2588	1.00m
1151	1.2588	1.00m
1152	1.2587	1.00m
1153	1.2587	1.00m
1154	1.2587	1.00m
1155	1.2587	1.00m
1156	1.2586	1.00m
1157	1.2586	1.00m
1158	1.2586	59.99s
1159	1.2586	59.93s
1160	1.2585	59.88s
1161	1.2585	59.83s
1162	1.2585	59.77s
1163	1.2585	59.72s
1164	1.2584	59.67s
1165	1.2584	59.64s
1166	1.2584	59.59s
1167	1.2584	59.53s
1168	1.2583	59.48s
1169	1.2583	59.42s
1170	1.2583	59.36s
1171	1.2583	59.31s
1172	1.2582	59.26s
1173	1.2582	59.20s
1174	1.2582	59.14s
1175	1.2582	59.09s
1176	1.2581	59.03s
1177	1.2581	58.98s
1178	1.2581	58.93s
1179	1.2580	58.87s
1180	1.2580	58.81s
1181	1.2580	58.76s
1182	1.2580	58.70s
1183	1.2579	58.64s
1184	1.2579	58.61s
1185	1.2579	58.56s
1186	1.2579	58.57s
1187	1.2578	58.57s
1188	1.2578	58.57s

1189	1.2578	58.56s
1190	1.2578	58.57s
1191	1.2577	58.55s
1192	1.2577	58.53s
1193	1.2577	58.50s
1194	1.2577	58.48s
1195	1.2576	58.44s
1196	1.2576	58.40s
1197	1.2576	58.34s
1198	1.2576	58.29s
1199	1.2575	58.24s
1200	1.2575	58.20s
1201	1.2575	58.19s
1202	1.2575	58.15s
1203	1.2574	58.11s
1204	1.2574	58.07s
1205	1.2574	58.04s
1206	1.2574	58.00s
1207	1.2573	57.97s
1208	1.2573	57.93s
1209	1.2573	57.88s
1210	1.2573	57.84s
1211	1.2572	57.81s
1212	1.2572	57.77s
1213	1.2572	57.72s
1214	1.2572	57.66s
1215	1.2571	57.60s
1216	1.2571	57.55s
1217	1.2571	57.49s
1218	1.2571	57.44s
1219	1.2570	57.45s
1220	1.2570	57.39s
1221	1.2570	57.34s
1222	1.2570	57.28s
1223	1.2569	57.22s
1224	1.2569	57.18s
1225	1.2569	57.15s
1226	1.2569	57.19s
1227	1.2568	57.15s
1228	1.2568	57.11s
1229	1.2568	57.06s
1230	1.2568	57.00s
1231	1.2567	56.98s
1232	1.2567	56.96s
1233	1.2567	56.92s
1234	1.2567	56.88s
1235	1.2566	56.85s
1236	1.2566	56.82s
1237	1.2566	56.78s
1238	1.2566	56.74s
1239	1.2565	56.72s
1240	1.2565	56.68s
1241	1.2565	56.62s
1242	1.2565	56.57s
1243	1.2565	56.51s
1244	1.2564	56.45s
1245	1.2564	56.39s

```
1246            1.2564          56.34s
1247            1.2564          56.28s
1248            1.2563          56.22s
1249            1.2563          56.16s
1250            1.2563          56.10s
1251            1.2563          56.04s
1252            1.2562          55.99s
1253            1.2562          55.93s
1254            1.2562          55.87s
1255            1.2562          55.81s
1256            1.2561          55.77s
1257            1.2561          55.73s
1258            1.2561          55.79s
1259            1.2561          55.91s
1260            1.2560          55.90s
1261            1.2560          55.85s
1262            1.2560          55.81s
1263            1.2560          55.76s
1264            1.2559          55.75s
1265            1.2559          55.70s
1266            1.2559          55.65s
1267            1.2559          55.59s
1268            1.2558          55.53s
1269            1.2558          55.50s
1270            1.2558          55.45s
1271            1.2558          55.41s
1272            1.2558          55.37s
1273            1.2557          55.32s
1274            1.2557          55.26s
1275            1.2557          55.21s
1276            1.2557          55.18s
1277            1.2556          55.15s
1278            1.2556          55.09s
1279            1.2556          55.03s
1280            1.2556          54.98s
1281            1.2555          54.92s
1282            1.2555          54.88s
1283            1.2555          54.84s
1284            1.2555          54.79s
1285            1.2554          54.80s
1286            1.2554          54.78s
1287            1.2554          54.75s
1288            1.2554          54.72s
1289            1.2553          54.67s
1290            1.2553          54.66s
1291            1.2553          54.70s
1292            1.2553          54.70s
1293            1.2553          54.65s
1294            1.2552          54.61s
1295            1.2552          54.56s
1296            1.2552          54.52s
1297            1.2552          54.50s
1298            1.2551          54.48s
1299            1.2551          54.45s
1300            1.2551          54.42s
1301            1.2551          54.38s
1302            1.2550          54.33s
```

```
1303            1.2550          54.29s
1304            1.2550          54.25s
1305            1.2550          54.22s
1306            1.2550          54.21s
1307            1.2549          54.17s
1308            1.2549          54.13s
1309            1.2549          54.07s
1310            1.2549          54.01s
1311            1.2548          53.95s
1312            1.2548          53.90s
1313            1.2548          53.85s
1314            1.2548          53.80s
1315            1.2547          53.76s
1316            1.2547          53.72s
1317            1.2547          53.69s
1318            1.2547          53.64s
1319            1.2546          53.58s
1320            1.2546          53.52s
1321            1.2546          53.46s
1322            1.2546          53.40s
1323            1.2546          53.35s
1324            1.2545          53.33s
1325            1.2545          53.30s
1326            1.2545          53.25s
1327            1.2545          53.19s
1328            1.2544          53.13s
1329            1.2544          53.07s
1330            1.2544          53.03s
1331            1.2544          52.97s
1332            1.2544          52.91s
1333            1.2543          52.85s
1334            1.2543          52.79s
1335            1.2543          52.74s
1336            1.2543          52.68s
1337            1.2542          52.62s
1338            1.2542          52.58s
1339            1.2542          52.54s
1340            1.2542          52.53s
1341            1.2541          52.51s
1342            1.2541          52.50s
1343            1.2541          52.48s
1344            1.2541          52.44s
1345            1.2541          52.41s
1346            1.2540          52.42s
1347            1.2540          52.40s
1348            1.2540          52.42s
1349            1.2540          52.41s
1350            1.2539          52.38s
1351            1.2539          52.33s
1352            1.2539          52.28s
1353            1.2539          52.23s
1354            1.2539          52.17s
1355            1.2538          52.12s
1356            1.2538          52.06s
1357            1.2538          52.01s
1358            1.2538          51.95s
1359            1.2537          51.90s
```

1360	1.2537	51.84s
1361	1.2537	51.79s
1362	1.2537	51.73s
1363	1.2536	51.68s
1364	1.2536	51.62s
1365	1.2536	51.57s
1366	1.2536	51.52s
1367	1.2536	51.50s
1368	1.2535	51.46s
1369	1.2535	51.41s
1370	1.2535	51.35s
1371	1.2535	51.30s
1372	1.2534	51.24s
1373	1.2534	51.18s
1374	1.2534	51.13s
1375	1.2534	51.07s
1376	1.2534	51.02s
1377	1.2533	50.96s
1378	1.2533	50.91s
1379	1.2533	50.85s
1380	1.2533	50.80s
1381	1.2532	50.74s
1382	1.2532	50.69s
1383	1.2532	50.63s
1384	1.2532	50.57s
1385	1.2532	50.52s
1386	1.2531	50.46s
1387	1.2531	50.41s
1388	1.2531	50.36s
1389	1.2531	50.30s
1390	1.2530	50.24s
1391	1.2530	50.19s
1392	1.2530	50.13s
1393	1.2530	50.07s
1394	1.2530	50.02s
1395	1.2529	49.97s
1396	1.2529	49.92s
1397	1.2529	49.86s
1398	1.2529	49.81s
1399	1.2528	49.75s
1400	1.2528	49.69s
1401	1.2528	49.65s
1402	1.2528	49.60s
1403	1.2528	49.54s
1404	1.2527	49.48s
1405	1.2527	49.43s
1406	1.2527	49.41s
1407	1.2527	49.39s
1408	1.2526	49.34s
1409	1.2526	49.29s
1410	1.2526	49.25s
1411	1.2526	49.21s
1412	1.2526	49.20s
1413	1.2525	49.20s
1414	1.2525	49.19s
1415	1.2525	49.16s
1416	1.2525	49.12s

```
1417            1.2525          49.09s
1418            1.2524          49.09s
1419            1.2524          49.05s
1420            1.2524          49.01s
1421            1.2524          48.97s
1422            1.2523          48.92s
1423            1.2523          48.87s
1424            1.2523          48.82s
1425            1.2523          48.77s
1426            1.2523          48.74s
1427            1.2522          48.69s
1428            1.2522          48.64s
1429            1.2522          48.58s
1430            1.2522          48.54s
1431            1.2521          48.50s
1432            1.2521          48.45s
1433            1.2521          48.40s
1434            1.2521          48.37s
1435            1.2521          48.34s
1436            1.2520          48.29s
1437            1.2520          48.24s
1438            1.2520          48.19s
1439            1.2520          48.14s
1440            1.2520          48.10s
1441            1.2519          48.05s
1442            1.2519          48.00s
1443            1.2519          47.95s
1444            1.2519          47.89s
1445            1.2518          47.84s
1446            1.2518          47.80s
1447            1.2518          47.75s
1448            1.2518          47.70s
1449            1.2518          47.66s
1450            1.2517          47.62s
1451            1.2517          47.57s
1452            1.2517          47.51s
1453            1.2517          47.46s
1454            1.2516          47.41s
1455            1.2516          47.35s
1456            1.2516          47.30s
1457            1.2516          47.25s
1458            1.2516          47.20s
1459            1.2515          47.15s
1460            1.2515          47.11s
1461            1.2515          47.07s
1462            1.2515          47.07s
1463            1.2515          47.04s
1464            1.2514          47.02s
1465            1.2514          46.98s
1466            1.2514          46.94s
1467            1.2514          46.90s
1468            1.2514          46.86s
1469            1.2513          46.82s
1470            1.2513          46.77s
1471            1.2513          46.72s
1472            1.2513          46.67s
1473            1.2512          46.62s
```

1474	1.2512	46.59s
1475	1.2512	46.56s
1476	1.2512	46.53s
1477	1.2512	46.49s
1478	1.2511	46.45s
1479	1.2511	46.40s
1480	1.2511	46.35s
1481	1.2511	46.29s
1482	1.2511	46.24s
1483	1.2510	46.19s
1484	1.2510	46.14s
1485	1.2510	46.08s
1486	1.2510	46.04s
1487	1.2509	46.01s
1488	1.2509	45.98s
1489	1.2509	45.94s
1490	1.2509	45.88s
1491	1.2509	45.83s
1492	1.2508	45.77s
1493	1.2508	45.73s
1494	1.2508	45.69s
1495	1.2508	45.64s
1496	1.2508	45.60s
1497	1.2507	45.61s
1498	1.2507	45.59s
1499	1.2507	45.54s
1500	1.2507	45.50s
1501	1.2507	45.46s
1502	1.2506	45.42s
1503	1.2506	45.38s
1504	1.2506	45.34s
1505	1.2506	45.30s
1506	1.2505	45.25s
1507	1.2505	45.20s
1508	1.2505	45.14s
1509	1.2505	45.09s
1510	1.2505	45.04s
1511	1.2504	44.99s
1512	1.2504	44.93s
1513	1.2504	44.88s
1514	1.2504	44.82s
1515	1.2504	44.77s
1516	1.2503	44.72s
1517	1.2503	44.66s
1518	1.2503	44.62s
1519	1.2503	44.56s
1520	1.2503	44.51s
1521	1.2502	44.46s
1522	1.2502	44.42s
1523	1.2502	44.37s
1524	1.2502	44.31s
1525	1.2502	44.26s
1526	1.2501	44.20s
1527	1.2501	44.15s
1528	1.2501	44.10s
1529	1.2501	44.06s
1530	1.2500	44.00s

```
1531            1.2500          43.95s
1532            1.2500          43.89s
1533            1.2500          43.84s
1534            1.2500          43.79s
1535            1.2499          43.75s
1536            1.2499          43.70s
1537            1.2499          43.64s
1538            1.2499          43.59s
1539            1.2499          43.53s
1540            1.2498          43.48s
1541            1.2498          43.43s
1542            1.2498          43.38s
1543            1.2498          43.34s
1544            1.2498          43.33s
1545            1.2497          43.30s
1546            1.2497          43.27s
1547            1.2497          43.23s
1548            1.2497          43.18s
1549            1.2497          43.14s
1550            1.2496          43.10s
1551            1.2496          43.06s
1552            1.2496          43.02s
1553            1.2496          42.98s
1554            1.2496          42.94s
1555            1.2495          42.91s
1556            1.2495          42.86s
1557            1.2495          42.81s
1558            1.2495          42.76s
1559            1.2495          42.71s
1560            1.2494          42.66s
1561            1.2494          42.60s
1562            1.2494          42.57s
1563            1.2494          42.54s
1564            1.2493          42.49s
1565            1.2493          42.44s
1566            1.2493          42.38s
1567            1.2493          42.33s
1568            1.2493          42.29s
1569            1.2492          42.28s
1570            1.2492          42.24s
1571            1.2492          42.19s
1572            1.2492          42.14s
1573            1.2492          42.09s
1574            1.2491          42.03s
1575            1.2491          42.01s
1576            1.2491          41.96s
1577            1.2491          41.93s
1578            1.2491          41.89s
1579            1.2490          41.84s
1580            1.2490          41.79s
1581            1.2490          41.74s
1582            1.2490          41.69s
1583            1.2490          41.65s
1584            1.2489          41.59s
1585            1.2489          41.57s
1586            1.2489          41.52s
1587            1.2489          41.46s
```

1588	1.2489	41.41s
1589	1.2488	41.36s
1590	1.2488	41.31s
1591	1.2488	41.26s
1592	1.2488	41.21s
1593	1.2488	41.17s
1594	1.2487	41.12s
1595	1.2487	41.08s
1596	1.2487	41.04s
1597	1.2487	40.99s
1598	1.2487	40.94s
1599	1.2486	40.89s
1600	1.2486	40.84s
1601	1.2486	40.79s
1602	1.2486	40.74s
1603	1.2486	40.69s
1604	1.2485	40.63s
1605	1.2485	40.58s
1606	1.2485	40.53s
1607	1.2485	40.48s
1608	1.2485	40.43s
1609	1.2484	40.39s
1610	1.2484	40.34s
1611	1.2484	40.29s
1612	1.2484	40.24s
1613	1.2484	40.19s
1614	1.2483	40.14s
1615	1.2483	40.09s
1616	1.2483	40.04s
1617	1.2483	39.99s
1618	1.2483	39.93s
1619	1.2482	39.88s
1620	1.2482	39.83s
1621	1.2482	39.78s
1622	1.2482	39.73s
1623	1.2482	39.68s
1624	1.2481	39.63s
1625	1.2481	39.58s
1626	1.2481	39.53s
1627	1.2481	39.48s
1628	1.2481	39.43s
1629	1.2480	39.38s
1630	1.2480	39.32s
1631	1.2480	39.27s
1632	1.2480	39.22s
1633	1.2480	39.18s
1634	1.2479	39.14s
1635	1.2479	39.09s
1636	1.2479	39.04s
1637	1.2479	38.99s
1638	1.2479	38.96s
1639	1.2478	39.00s
1640	1.2478	38.96s
1641	1.2478	38.92s
1642	1.2478	38.88s
1643	1.2478	38.84s
1644	1.2478	38.80s

```
1645            1.2477          38.76s
1646            1.2477          38.72s
1647            1.2477          38.67s
1648            1.2477          38.63s
1649            1.2477          38.58s
1650            1.2476          38.53s
1651            1.2476          38.48s
1652            1.2476          38.43s
1653            1.2476          38.38s
1654            1.2476          38.32s
1655            1.2475          38.27s
1656            1.2475          38.22s
1657            1.2475          38.17s
1658            1.2475          38.12s
1659            1.2475          38.07s
1660            1.2474          38.03s
1661            1.2474          37.98s
1662            1.2474          37.93s
1663            1.2474          37.88s
1664            1.2474          37.83s
1665            1.2473          37.78s
1666            1.2473          37.72s
1667            1.2473          37.67s
1668            1.2473          37.62s
1669            1.2473          37.57s
1670            1.2472          37.53s
1671            1.2472          37.49s
1672            1.2472          37.45s
1673            1.2472          37.41s
1674            1.2472          37.36s
1675            1.2471          37.32s
1676            1.2471          37.28s
1677            1.2471          37.26s
1678            1.2471          37.22s
1679            1.2471          37.18s
1680            1.2470          37.15s
1681            1.2470          37.15s
1682            1.2470          37.14s
1683            1.2470          37.12s
1684            1.2470          37.08s
1685            1.2470          37.03s
1686            1.2469          36.99s
1687            1.2469          36.94s
1688            1.2469          36.89s
1689            1.2469          36.84s
1690            1.2469          36.80s
1691            1.2468          36.75s
1692            1.2468          36.70s
1693            1.2468          36.65s
1694            1.2468          36.59s
1695            1.2468          36.54s
1696            1.2467          36.49s
1697            1.2467          36.44s
1698            1.2467          36.39s
1699            1.2467          36.34s
1700            1.2467          36.28s
1701            1.2466          36.23s
```

```
1702            1.2466          36.18s
1703            1.2466          36.13s
1704            1.2466          36.08s
1705            1.2466          36.03s
1706            1.2466          35.98s
1707            1.2465          35.93s
1708            1.2465          35.88s
1709            1.2465          35.83s
1710            1.2465          35.78s
1711            1.2465          35.72s
1712            1.2464          35.67s
1713            1.2464          35.62s
1714            1.2464          35.57s
1715            1.2464          35.52s
1716            1.2464          35.47s
1717            1.2463          35.41s
1718            1.2463          35.36s
1719            1.2463          35.31s
1720            1.2463          35.26s
1721            1.2463          35.21s
1722            1.2462          35.16s
1723            1.2462          35.11s
1724            1.2462          35.06s
1725            1.2462          35.00s
1726            1.2462          34.96s
1727            1.2462          34.91s
1728            1.2461          34.86s
1729            1.2461          34.82s
1730            1.2461          34.77s
1731            1.2461          34.72s
1732            1.2461          34.67s
1733            1.2460          34.62s
1734            1.2460          34.58s
1735            1.2460          34.53s
1736            1.2460          34.48s
1737            1.2460          34.43s
1738            1.2459          34.38s
1739            1.2459          34.33s
1740            1.2459          34.28s
1741            1.2459          34.25s
1742            1.2459          34.20s
1743            1.2459          34.16s
1744            1.2458          34.12s
1745            1.2458          34.07s
1746            1.2458          34.02s
1747            1.2458          33.97s
1748            1.2458          33.92s
1749            1.2457          33.88s
1750            1.2457          33.84s
1751            1.2457          33.79s
1752            1.2457          33.74s
1753            1.2457          33.69s
1754            1.2456          33.64s
1755            1.2456          33.58s
1756            1.2456          33.53s
1757            1.2456          33.48s
1758            1.2456          33.44s
```

1759	1.2456	33.40s
1760	1.2455	33.35s
1761	1.2455	33.30s
1762	1.2455	33.26s
1763	1.2455	33.21s
1764	1.2455	33.17s
1765	1.2454	33.12s
1766	1.2454	33.07s
1767	1.2454	33.02s
1768	1.2454	32.97s
1769	1.2454	32.92s
1770	1.2454	32.86s
1771	1.2453	32.81s
1772	1.2453	32.76s
1773	1.2453	32.71s
1774	1.2453	32.66s
1775	1.2453	32.61s
1776	1.2452	32.56s
1777	1.2452	32.51s
1778	1.2452	32.46s
1779	1.2452	32.41s
1780	1.2452	32.36s
1781	1.2451	32.31s
1782	1.2451	32.26s
1783	1.2451	32.21s
1784	1.2451	32.16s
1785	1.2451	32.11s
1786	1.2451	32.06s
1787	1.2450	32.01s
1788	1.2450	31.96s
1789	1.2450	31.91s
1790	1.2450	31.86s
1791	1.2450	31.82s
1792	1.2449	31.77s
1793	1.2449	31.73s
1794	1.2449	31.68s
1795	1.2449	31.64s
1796	1.2449	31.59s
1797	1.2449	31.54s
1798	1.2448	31.51s
1799	1.2448	31.49s
1800	1.2448	31.45s
1801	1.2448	31.41s
1802	1.2448	31.39s
1803	1.2447	31.36s
1804	1.2447	31.32s
1805	1.2447	31.27s
1806	1.2447	31.22s
1807	1.2447	31.18s
1808	1.2446	31.14s
1809	1.2446	31.09s
1810	1.2446	31.04s
1811	1.2446	31.01s
1812	1.2446	30.96s
1813	1.2446	30.91s
1814	1.2445	30.86s
1815	1.2445	30.81s

```
1816            1.2445          30.76s
1817            1.2445          30.71s
1818            1.2445          30.66s
1819            1.2445          30.62s
1820            1.2444          30.57s
1821            1.2444          30.52s
1822            1.2444          30.47s
1823            1.2444          30.42s
1824            1.2444          30.38s
1825            1.2443          30.34s
1826            1.2443          30.29s
1827            1.2443          30.24s
1828            1.2443          30.19s
1829            1.2443          30.14s
1830            1.2443          30.09s
1831            1.2442          30.04s
1832            1.2442          29.99s
1833            1.2442          29.95s
1834            1.2442          29.92s
1835            1.2442          29.87s
1836            1.2441          29.82s
1837            1.2441          29.77s
1838            1.2441          29.72s
1839            1.2441          29.67s
1840            1.2441          29.62s
1841            1.2441          29.58s
1842            1.2440          29.53s
1843            1.2440          29.48s
1844            1.2440          29.43s
1845            1.2440          29.38s
1846            1.2440          29.33s
1847            1.2439          29.28s
1848            1.2439          29.23s
1849            1.2439          29.18s
1850            1.2439          29.13s
1851            1.2439          29.08s
1852            1.2439          29.03s
1853            1.2438          28.98s
1854            1.2438          28.93s
1855            1.2438          28.89s
1856            1.2438          28.84s
1857            1.2438          28.79s
1858            1.2437          28.74s
1859            1.2437          28.69s
1860            1.2437          28.64s
1861            1.2437          28.59s
1862            1.2437          28.54s
1863            1.2437          28.49s
1864            1.2436          28.44s
1865            1.2436          28.39s
1866            1.2436          28.34s
1867            1.2436          28.29s
1868            1.2436          28.24s
1869            1.2436          28.20s
1870            1.2435          28.15s
1871            1.2435          28.10s
1872            1.2435          28.05s
```

```
1873            1.2435          28.00s
1874            1.2435          27.95s
1875            1.2434          27.90s
1876            1.2434          27.85s
1877            1.2434          27.80s
1878            1.2434          27.75s
1879            1.2434          27.70s
1880            1.2434          27.65s
1881            1.2433          27.60s
1882            1.2433          27.55s
1883            1.2433          27.51s
1884            1.2433          27.46s
1885            1.2433          27.41s
1886            1.2433          27.36s
1887            1.2432          27.31s
1888            1.2432          27.26s
1889            1.2432          27.21s
1890            1.2432          27.17s
1891            1.2432          27.12s
1892            1.2431          27.07s
1893            1.2431          27.02s
1894            1.2431          26.97s
1895            1.2431          26.92s
1896            1.2431          26.87s
1897            1.2431          26.82s
1898            1.2430          26.77s
1899            1.2430          26.73s
1900            1.2430          26.68s
1901            1.2430          26.63s
1902            1.2430          26.58s
1903            1.2430          26.53s
1904            1.2429          26.48s
1905            1.2429          26.43s
1906            1.2429          26.38s
1907            1.2429          26.34s
1908            1.2429          26.29s
1909            1.2429          26.24s
1910            1.2428          26.19s
1911            1.2428          26.14s
1912            1.2428          26.09s
1913            1.2428          26.04s
1914            1.2428          25.99s
1915            1.2427          25.95s
1916            1.2427          25.90s
1917            1.2427          25.85s
1918            1.2427          25.80s
1919            1.2427          25.75s
1920            1.2427          25.70s
1921            1.2426          25.65s
1922            1.2426          25.61s
1923            1.2426          25.56s
1924            1.2426          25.51s
1925            1.2426          25.46s
1926            1.2426          25.41s
1927            1.2425          25.37s
1928            1.2425          25.32s
1929            1.2425          25.27s
```

```
1930            1.2425          25.22s
1931            1.2425          25.18s
1932            1.2425          25.13s
1933            1.2424          25.08s
1934            1.2424          25.03s
1935            1.2424          24.98s
1936            1.2424          24.93s
1937            1.2424          24.89s
1938            1.2423          24.84s
1939            1.2423          24.79s
1940            1.2423          24.74s
1941            1.2423          24.69s
1942            1.2423          24.65s
1943            1.2423          24.60s
1944            1.2422          24.55s
1945            1.2422          24.50s
1946            1.2422          24.45s
1947            1.2422          24.41s
1948            1.2422          24.36s
1949            1.2422          24.31s
1950            1.2421          24.26s
1951            1.2421          24.21s
1952            1.2421          24.17s
1953            1.2421          24.12s
1954            1.2421          24.07s
1955            1.2421          24.02s
1956            1.2420          23.98s
1957            1.2420          23.93s
1958            1.2420          23.88s
1959            1.2420          23.83s
1960            1.2420          23.79s
1961            1.2420          23.74s
1962            1.2419          23.69s
1963            1.2419          23.64s
1964            1.2419          23.59s
1965            1.2419          23.55s
1966            1.2419          23.50s
1967            1.2419          23.45s
1968            1.2418          23.41s
1969            1.2418          23.36s
1970            1.2418          23.31s
1971            1.2418          23.26s
1972            1.2418          23.21s
1973            1.2417          23.17s
1974            1.2417          23.12s
1975            1.2417          23.07s
1976            1.2417          23.02s
1977            1.2417          22.98s
1978            1.2417          22.93s
1979            1.2416          22.88s
1980            1.2416          22.84s
1981            1.2416          22.79s
1982            1.2416          22.74s
1983            1.2416          22.70s
1984            1.2416          22.65s
1985            1.2415          22.60s
1986            1.2415          22.55s
```

```
1987            1.2415          22.51s
1988            1.2415          22.46s
1989            1.2415          22.41s
1990            1.2415          22.37s
1991            1.2414          22.32s
1992            1.2414          22.27s
1993            1.2414          22.22s
1994            1.2414          22.18s
1995            1.2414          22.13s
1996            1.2414          22.08s
1997            1.2413          22.04s
1998            1.2413          21.99s
1999            1.2413          21.94s
2000            1.2413          21.89s
2001            1.2413          21.85s
2002            1.2413          21.80s
2003            1.2412          21.76s
2004            1.2412          21.72s
2005            1.2412          21.67s
2006            1.2412          21.62s
2007            1.2412          21.58s
2008            1.2412          21.53s
2009            1.2411          21.49s
2010            1.2411          21.44s
2011            1.2411          21.39s
2012            1.2411          21.35s
2013            1.2411          21.30s
2014            1.2411          21.25s
2015            1.2410          21.21s
2016            1.2410          21.16s
2017            1.2410          21.11s
2018            1.2410          21.07s
2019            1.2410          21.02s
2020            1.2410          20.98s
2021            1.2409          20.93s
2022            1.2409          20.88s
2023            1.2409          20.83s
2024            1.2409          20.79s
2025            1.2409          20.74s
2026            1.2409          20.69s
2027            1.2408          20.65s
2028            1.2408          20.60s
2029            1.2408          20.55s
2030            1.2408          20.51s
2031            1.2408          20.46s
2032            1.2408          20.41s
2033            1.2407          20.37s
2034            1.2407          20.32s
2035            1.2407          20.27s
2036            1.2407          20.23s
2037            1.2407          20.18s
2038            1.2407          20.13s
2039            1.2406          20.09s
2040            1.2406          20.04s
2041            1.2406          19.99s
2042            1.2406          19.95s
2043            1.2406          19.90s
```

```
2044            1.2406          19.85s
2045            1.2405          19.81s
2046            1.2405          19.76s
2047            1.2405          19.72s
2048            1.2405          19.67s
2049            1.2405          19.63s
2050            1.2405          19.58s
2051            1.2404          19.54s
2052            1.2404          19.49s
2053            1.2404          19.44s
2054            1.2404          19.40s
2055            1.2404          19.35s
2056            1.2404          19.30s
2057            1.2403          19.26s
2058            1.2403          19.21s
2059            1.2403          19.17s
2060            1.2403          19.12s
2061            1.2403          19.07s
2062            1.2403          19.03s
2063            1.2402          18.98s
2064            1.2402          18.93s
2065            1.2402          18.89s
2066            1.2402          18.84s
2067            1.2402          18.80s
2068            1.2402          18.75s
2069            1.2401          18.70s
2070            1.2401          18.66s
2071            1.2401          18.61s
2072            1.2401          18.57s
2073            1.2401          18.52s
2074            1.2401          18.47s
2075            1.2400          18.43s
2076            1.2400          18.38s
2077            1.2400          18.34s
2078            1.2400          18.29s
2079            1.2400          18.24s
2080            1.2400          18.20s
2081            1.2399          18.16s
2082            1.2399          18.11s
2083            1.2399          18.06s
2084            1.2399          18.02s
2085            1.2399          17.97s
2086            1.2399          17.92s
2087            1.2398          17.88s
2088            1.2398          17.83s
2089            1.2398          17.79s
2090            1.2398          17.74s
2091            1.2398          17.70s
2092            1.2398          17.65s
2093            1.2397          17.60s
2094            1.2397          17.56s
2095            1.2397          17.52s
2096            1.2397          17.47s
2097            1.2397          17.43s
2098            1.2397          17.38s
2099            1.2396          17.33s
2100            1.2396          17.29s
```

```
2101            1.2396          17.24s
2102            1.2396          17.20s
2103            1.2396          17.15s
2104            1.2396          17.10s
2105            1.2396          17.06s
2106            1.2395          17.01s
2107            1.2395          16.97s
2108            1.2395          16.92s
2109            1.2395          16.88s
2110            1.2395          16.83s
2111            1.2395          16.79s
2112            1.2394          16.74s
2113            1.2394          16.69s
2114            1.2394          16.65s
2115            1.2394          16.60s
2116            1.2394          16.56s
2117            1.2394          16.51s
2118            1.2393          16.47s
2119            1.2393          16.42s
2120            1.2393          16.37s
2121            1.2393          16.33s
2122            1.2393          16.28s
2123            1.2393          16.24s
2124            1.2392          16.19s
2125            1.2392          16.15s
2126            1.2392          16.10s
2127            1.2392          16.06s
2128            1.2392          16.01s
2129            1.2392          15.97s
2130            1.2391          15.92s
2131            1.2391          15.88s
2132            1.2391          15.84s
2133            1.2391          15.80s
2134            1.2391          15.76s
2135            1.2391          15.71s
2136            1.2391          15.67s
2137            1.2390          15.63s
2138            1.2390          15.58s
2139            1.2390          15.54s
2140            1.2390          15.50s
2141            1.2390          15.46s
2142            1.2390          15.41s
2143            1.2389          15.38s
2144            1.2389          15.33s
2145            1.2389          15.29s
2146            1.2389          15.24s
2147            1.2389          15.20s
2148            1.2389          15.15s
2149            1.2388          15.11s
2150            1.2388          15.06s
2151            1.2388          15.02s
2152            1.2388          14.97s
2153            1.2388          14.93s
2154            1.2388          14.88s
2155            1.2387          14.83s
2156            1.2387          14.79s
2157            1.2387          14.74s
```

```
2158            1.2387          14.70s
2159            1.2387          14.65s
2160            1.2387          14.61s
2161            1.2386          14.56s
2162            1.2386          14.52s
2163            1.2386          14.47s
2164            1.2386          14.43s
2165            1.2386          14.38s
2166            1.2386          14.34s
2167            1.2386          14.29s
2168            1.2385          14.25s
2169            1.2385          14.20s
2170            1.2385          14.16s
2171            1.2385          14.11s
2172            1.2385          14.07s
2173            1.2385          14.02s
2174            1.2384          13.98s
2175            1.2384          13.93s
2176            1.2384          13.89s
2177            1.2384          13.85s
2178            1.2384          13.80s
2179            1.2384          13.76s
2180            1.2383          13.71s
2181            1.2383          13.67s
2182            1.2383          13.62s
2183            1.2383          13.58s
2184            1.2383          13.53s
2185            1.2383          13.49s
2186            1.2383          13.44s
2187            1.2382          13.40s
2188            1.2382          13.35s
2189            1.2382          13.31s
2190            1.2382          13.26s
2191            1.2382          13.22s
2192            1.2382          13.17s
2193            1.2381          13.13s
2194            1.2381          13.08s
2195            1.2381          13.04s
2196            1.2381          12.99s
2197            1.2381          12.95s
2198            1.2381          12.90s
2199            1.2380          12.86s
2200            1.2380          12.81s
2201            1.2380          12.77s
2202            1.2380          12.72s
2203            1.2380          12.68s
2204            1.2380          12.63s
2205            1.2380          12.59s
2206            1.2379          12.54s
2207            1.2379          12.50s
2208            1.2379          12.45s
2209            1.2379          12.41s
2210            1.2379          12.36s
2211            1.2379          12.32s
2212            1.2378          12.28s
2213            1.2378          12.23s
2214            1.2378          12.19s
```

2215	1.2378	12.14s
2216	1.2378	12.10s
2217	1.2378	12.05s
2218	1.2377	12.01s
2219	1.2377	11.96s
2220	1.2377	11.92s
2221	1.2377	11.87s
2222	1.2377	11.83s
2223	1.2377	11.79s
2224	1.2377	11.74s
2225	1.2376	11.70s
2226	1.2376	11.65s
2227	1.2376	11.61s
2228	1.2376	11.56s
2229	1.2376	11.52s
2230	1.2376	11.47s
2231	1.2375	11.43s
2232	1.2375	11.39s
2233	1.2375	11.34s
2234	1.2375	11.30s
2235	1.2375	11.25s
2236	1.2375	11.21s
2237	1.2375	11.16s
2238	1.2374	11.12s
2239	1.2374	11.08s
2240	1.2374	11.03s
2241	1.2374	10.99s
2242	1.2374	10.94s
2243	1.2374	10.90s
2244	1.2373	10.85s
2245	1.2373	10.81s
2246	1.2373	10.77s
2247	1.2373	10.72s
2248	1.2373	10.68s
2249	1.2373	10.63s
2250	1.2372	10.59s
2251	1.2372	10.54s
2252	1.2372	10.50s
2253	1.2372	10.46s
2254	1.2372	10.41s
2255	1.2372	10.37s
2256	1.2372	10.32s
2257	1.2371	10.28s
2258	1.2371	10.23s
2259	1.2371	10.19s
2260	1.2371	10.15s
2261	1.2371	10.10s
2262	1.2371	10.06s
2263	1.2370	10.01s
2264	1.2370	9.97s
2265	1.2370	9.93s
2266	1.2370	9.88s
2267	1.2370	9.84s
2268	1.2370	9.79s
2269	1.2370	9.75s
2270	1.2369	9.71s
2271	1.2369	9.66s

```
2272            1.2369          9.62s
2273            1.2369          9.57s
2274            1.2369          9.53s
2275            1.2369          9.49s
2276            1.2368          9.44s
2277            1.2368          9.40s
2278            1.2368          9.35s
2279            1.2368          9.31s
2280            1.2368          9.27s
2281            1.2368          9.22s
2282            1.2368          9.18s
2283            1.2367          9.13s
2284            1.2367          9.09s
2285            1.2367          9.05s
2286            1.2367          9.00s
2287            1.2367          8.96s
2288            1.2367          8.91s
2289            1.2366          8.87s
2290            1.2366          8.83s
2291            1.2366          8.78s
2292            1.2366          8.74s
2293            1.2366          8.69s
2294            1.2366          8.65s
2295            1.2366          8.61s
2296            1.2365          8.56s
2297            1.2365          8.52s
2298            1.2365          8.48s
2299            1.2365          8.44s
2300            1.2365          8.39s
2301            1.2365          8.35s
2302            1.2364          8.30s
2303            1.2364          8.26s
2304            1.2364          8.22s
2305            1.2364          8.17s
2306            1.2364          8.13s
2307            1.2364          8.09s
2308            1.2364          8.04s
2309            1.2363          8.00s
2310            1.2363          7.96s
2311            1.2363          7.91s
2312            1.2363          7.87s
2313            1.2363          7.82s
2314            1.2363          7.78s
2315            1.2362          7.74s
2316            1.2362          7.69s
2317            1.2362          7.65s
2318            1.2362          7.61s
2319            1.2362          7.56s
2320            1.2362          7.52s
2321            1.2362          7.48s
2322            1.2361          7.43s
2323            1.2361          7.39s
2324            1.2361          7.35s
2325            1.2361          7.30s
2326            1.2361          7.26s
2327            1.2361          7.21s
2328            1.2361          7.17s
```

```
2329            1.2360          7.13s
2330            1.2360          7.09s
2331            1.2360          7.04s
2332            1.2360          7.00s
2333            1.2360          6.95s
2334            1.2360          6.91s
2335            1.2359          6.87s
2336            1.2359          6.83s
2337            1.2359          6.78s
2338            1.2359          6.74s
2339            1.2359          6.69s
2340            1.2359          6.65s
2341            1.2359          6.61s
2342            1.2358          6.56s
2343            1.2358          6.52s
2344            1.2358          6.48s
2345            1.2358          6.44s
2346            1.2358          6.39s
2347            1.2358          6.35s
2348            1.2357          6.31s
2349            1.2357          6.26s
2350            1.2357          6.22s
2351            1.2357          6.18s
2352            1.2357          6.13s
2353            1.2357          6.09s
2354            1.2357          6.05s
2355            1.2356          6.00s
2356            1.2356          5.96s
2357            1.2356          5.92s
2358            1.2356          5.87s
2359            1.2356          5.83s
2360            1.2356          5.79s
2361            1.2356          5.74s
2362            1.2355          5.70s
2363            1.2355          5.66s
2364            1.2355          5.61s
2365            1.2355          5.57s
2366            1.2355          5.53s
2367            1.2355          5.48s
2368            1.2355          5.44s
2369            1.2354          5.40s
2370            1.2354          5.36s
2371            1.2354          5.31s
2372            1.2354          5.27s
2373            1.2354          5.23s
2374            1.2354          5.18s
2375            1.2353          5.14s
2376            1.2353          5.10s
2377            1.2353          5.05s
2378            1.2353          5.01s
2379            1.2353          4.97s
2380            1.2353          4.93s
2381            1.2353          4.88s
2382            1.2352          4.84s
2383            1.2352          4.80s
2384            1.2352          4.75s
2385            1.2352          4.71s
```

```
2386            1.2352          4.67s
2387            1.2352          4.63s
2388            1.2352          4.58s
2389            1.2351          4.54s
2390            1.2351          4.50s
2391            1.2351          4.45s
2392            1.2351          4.41s
2393            1.2351          4.37s
2394            1.2351          4.33s
2395            1.2350          4.28s
2396            1.2350          4.24s
2397            1.2350          4.20s
2398            1.2350          4.16s
2399            1.2350          4.11s
2400            1.2350          4.07s
2401            1.2350          4.03s
2402            1.2349          3.98s
2403            1.2349          3.94s
2404            1.2349          3.90s
2405            1.2349          3.86s
2406            1.2349          3.81s
2407            1.2349          3.77s
2408            1.2349          3.73s
2409            1.2348          3.69s
2410            1.2348          3.64s
2411            1.2348          3.60s
2412            1.2348          3.56s
2413            1.2348          3.51s
2414            1.2348          3.47s
2415            1.2348          3.43s
2416            1.2347          3.39s
2417            1.2347          3.34s
2418            1.2347          3.30s
2419            1.2347          3.26s
2420            1.2347          3.22s
2421            1.2347          3.17s
2422            1.2346          3.13s
2423            1.2346          3.09s
2424            1.2346          3.05s
2425            1.2346          3.00s
2426            1.2346          2.96s
2427            1.2346          2.92s
2428            1.2346          2.88s
2429            1.2345          2.83s
2430            1.2345          2.79s
2431            1.2345          2.75s
2432            1.2345          2.70s
2433            1.2345          2.66s
2434            1.2345          2.62s
2435            1.2345          2.58s
2436            1.2344          2.53s
2437            1.2344          2.49s
2438            1.2344          2.45s
2439            1.2344          2.41s
2440            1.2344          2.36s
2441            1.2344          2.32s
2442            1.2344          2.28s
```

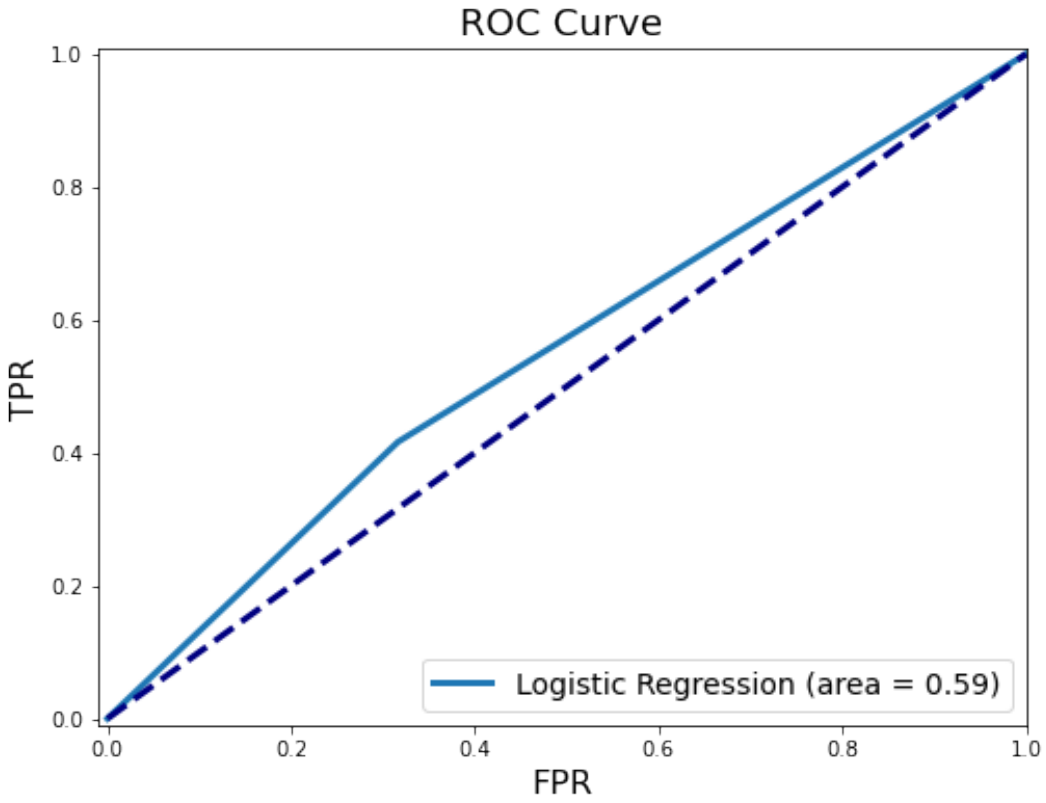
```
2443            1.2343          2.24s
2444            1.2343          2.19s
2445            1.2343          2.15s
2446            1.2343          2.11s
2447            1.2343          2.07s
2448            1.2343          2.03s
2449            1.2343          1.98s
2450            1.2342          1.94s
2451            1.2342          1.90s
2452            1.2342          1.86s
2453            1.2342          1.81s
2454            1.2342          1.77s
2455            1.2342          1.73s
2456            1.2341          1.69s
2457            1.2341          1.64s
2458            1.2341          1.60s
2459            1.2341          1.56s
2460            1.2341          1.52s
2461            1.2341          1.47s
2462            1.2341          1.43s
2463            1.2340          1.39s
2464            1.2340          1.35s
2465            1.2340          1.31s
2466            1.2340          1.26s
2467            1.2340          1.22s
2468            1.2340          1.18s
2469            1.2340          1.14s
2470            1.2339          1.09s
2471            1.2339          1.05s
2472            1.2339          1.01s
2473            1.2339          0.97s
2474            1.2339          0.93s
2475            1.2339          0.88s
2476            1.2339          0.84s
2477            1.2338          0.80s
2478            1.2338          0.76s
2479            1.2338          0.72s
2480            1.2338          0.67s
2481            1.2338          0.63s
2482            1.2338          0.59s
2483            1.2338          0.55s
2484            1.2337          0.50s
2485            1.2337          0.46s
2486            1.2337          0.42s
2487            1.2337          0.38s
2488            1.2337          0.34s
2489            1.2337          0.29s
2490            1.2337          0.25s
2491            1.2336          0.21s
2492            1.2336          0.17s
2493            1.2336          0.13s
2494            1.2336          0.08s
2495            1.2336          0.04s
2496            1.2336          0.00s
```

Out[85]:
```
GradientBoostingClassifier(learning_rate=0.01, max_leaf_nodes=2,
                           min_samples_leaf=10, n_estimators=2496,
                           random_state=88, verbose=3)
```

In [86]:
```
print('Accuracy:', round(accuracy_score(y_test, gbc_opt.predict(xd_test)), 5)
```

```
Accuracy: 0.62196
```

In [94]:
```
from sklearn.metrics import roc_curve, auc

fpr_opt, tpr_opt, _ = roc_curve(y_test, gbc_opt.predict(xd_test))

roc_auc = auc(fpr_opt, tpr_opt)
plt.figure(figsize=(8, 6))
plt.title('ROC Curve', fontsize=18)
plt.xlabel('FPR', fontsize=16)
plt.ylabel('TPR', fontsize=16)
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr, tpr, lw=3, label='Logistic Regression (area = {:0.2f})'.format(
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.legend(loc='lower right', fontsize=14)
plt.show()
print(fpr_opt,tpr_opt)
```

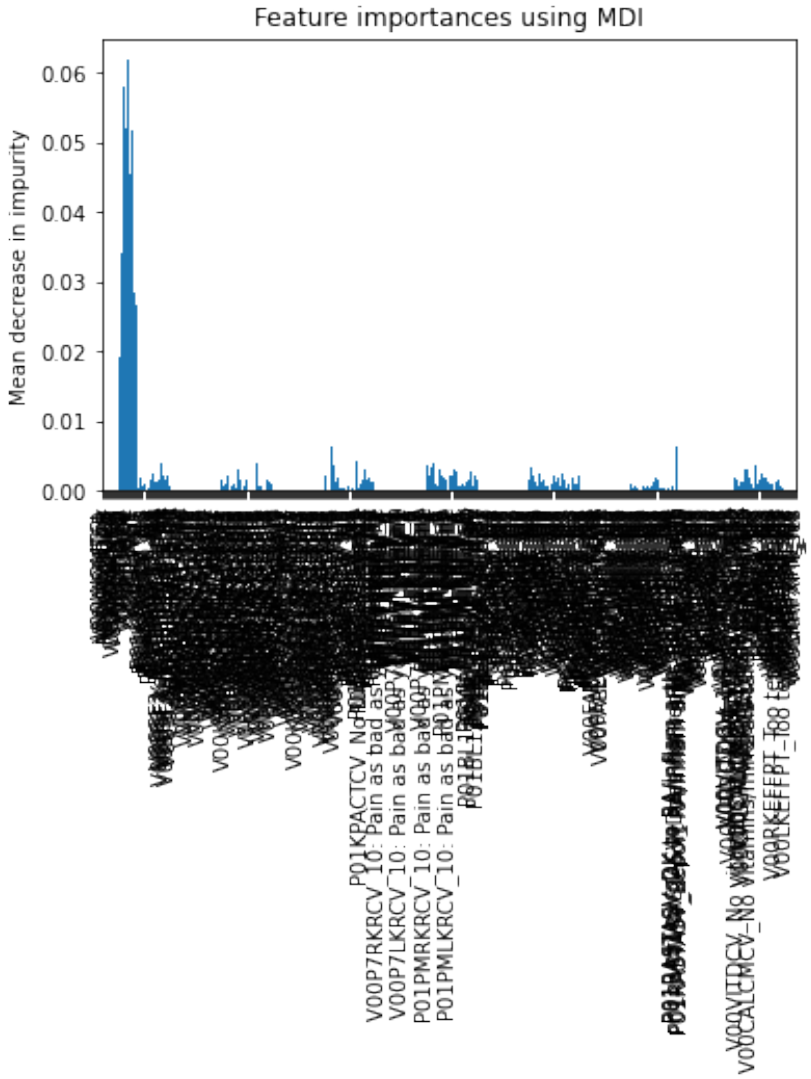



```
[0.         0.22170543 1.        ] [0.         0.4055794 1.        ]
```

Optimized Gradient Boosting Clasifier: accuracy = 0.62, fpr=0.22, tpr =0.41, AUC = 0.59

In [95]:
```
#3rd model: we only select the 25 most important features and build a booster
importances = gbc.feature_importances_



forest_importances = pd.Series(importances, index=xd.columns)

fig, ax = plt.subplots()
forest_importances.plot.bar(ax=ax)
ax.set_title("Feature importances using MDI")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()
```

```
<ipython-input-95-10412ee12b3e>:12: UserWarning: Tight layout not applied. The
bottom and top margins cannot be made large enough to accommodate all axes dec
orations.
  fig.tight_layout()
```



In [101…

```
print(feat_importances.nlargest(5).index)
```

```
Index(['V00DTCHOL', 'V00DTVITD', 'V00DTCALC', 'V00PASE', 'P01BMI'], dtype='obj
ect')
```

In [97]:

```
#Here, we only retain the 25 most important features
columns_red = feat_importances.nlargest(25).index
xd_red = pd.DataFrame()
for column in columns_red:
    xd_red[column] = xd[column]
```

In [98]:

```
xd_red.head()
```

Out[98]:

	V00DTCHOL	V00DTVITD	V00DTCALC	V00PASE	P01BMI	V00WOMTSR	V00WTMINKG
1489	300.6	260.66	984.8	50.0	43.9	25.2	59.1
364	157.5	164.02	904.2	199.0	27.8	0.0	67.7
2939	134.6	151.06	1071.8	340.0	21.3	0.0	52.3
1342	161.3	58.17	361.4	172.0	25.0	5.0	53.2
1425	334.4	161.20	1046.9	237.0	27.5	4.0	53.6

5 rows × 25 columns

In [90]:

```
gbc2 = GradientBoostingClassifier(n_estimators=2496, verbose=3,min_samples_le
                                  max_leaf_nodes=2,learning_rate = 0.01, random
gbc2.fit(xd_red, y_train)
```

```
      Iter       Train Loss   Remaining Time
         1           1.3576           31.21s
         2           1.3569           31.66s
         3           1.3563           33.53s
         4           1.3556           38.04s
         5           1.3550           40.17s
         6           1.3544           41.87s
         7           1.3538           42.57s
         8           1.3532           45.16s
         9           1.3526           45.25s
        10           1.3521           44.63s
        11           1.3515           44.99s
        12           1.3510           44.60s
        13           1.3505           44.24s
        14           1.3499           44.94s
        15           1.3494           46.17s
        16           1.3489           45.15s
        17           1.3484           44.23s
        18           1.3479           43.30s
        19           1.3474           42.47s
        20           1.3469           41.71s
        21           1.3464           41.01s
        22           1.3459           40.36s
        23           1.3454           39.80s
        24           1.3450           39.30s
        25           1.3445           38.87s
        26           1.3440           38.49s
        27           1.3436           38.07s
        28           1.3431           38.32s
        29           1.3427           38.73s
        30           1.3423           39.02s
        31           1.3418           39.10s
        32           1.3414           38.91s
        33           1.3410           39.28s
        34           1.3406           40.07s
```

```
35          1.3401          40.93s
36          1.3397          41.49s
37          1.3393          42.19s
38          1.3389          44.47s
39          1.3385          45.63s
40          1.3381          45.79s
41          1.3378          47.07s
42          1.3374          47.71s
43          1.3370          47.84s
44          1.3366          48.60s
45          1.3363          48.41s
46          1.3359          48.98s
47          1.3355          49.20s
48          1.3352          50.71s
49          1.3348          51.23s
50          1.3345          51.10s
51          1.3341          50.93s
52          1.3338          51.17s
53          1.3335          51.05s
54          1.3331          50.89s
55          1.3328          50.67s
56          1.3325          50.51s
57          1.3322          50.27s
58          1.3318          50.15s
59          1.3315          50.57s
60          1.3312          50.43s
61          1.3309          50.38s
62          1.3306          50.29s
63          1.3303          50.11s
64          1.3300          49.92s
65          1.3297          49.84s
66          1.3294          49.83s
67          1.3291          49.68s
68          1.3288          49.94s
69          1.3285          50.18s
70          1.3282          50.10s
71          1.3279          50.25s
72          1.3276          50.01s
73          1.3274          49.80s
74          1.3271          49.87s
75          1.3268          51.15s
76          1.3265          51.39s
77          1.3263          51.22s
78          1.3260          51.20s
79          1.3257          51.10s
80          1.3255          50.96s
81          1.3252          51.02s
82          1.3250          51.05s
83          1.3247          51.09s
84          1.3244          51.09s
85          1.3242          51.08s
86          1.3240          50.88s
87          1.3237          50.74s
88          1.3235          50.60s
89          1.3232          50.50s
90          1.3230          50.41s
91          1.3227          50.30s
```

92	1.3225	50.21s
93	1.3223	50.07s
94	1.3220	49.89s
95	1.3218	49.70s
96	1.3216	49.47s
97	1.3214	49.23s
98	1.3211	48.98s
99	1.3209	48.77s
100	1.3207	48.56s
101	1.3205	48.41s
102	1.3203	48.24s
103	1.3201	48.15s
104	1.3198	48.21s
105	1.3196	48.59s
106	1.3194	48.78s
107	1.3192	49.05s
108	1.3190	49.24s
109	1.3188	49.37s
110	1.3186	49.27s
111	1.3184	49.11s
112	1.3182	48.96s
113	1.3180	48.83s
114	1.3178	48.81s
115	1.3176	48.85s
116	1.3174	48.86s
117	1.3173	48.95s
118	1.3171	48.95s
119	1.3169	48.96s
120	1.3167	49.30s
121	1.3165	49.34s
122	1.3163	49.39s
123	1.3161	49.38s
124	1.3160	49.25s
125	1.3158	49.37s
126	1.3156	49.43s
127	1.3154	49.41s
128	1.3153	49.35s
129	1.3151	49.23s
130	1.3149	49.12s
131	1.3147	48.98s
132	1.3146	48.83s
133	1.3144	48.78s
134	1.3142	48.88s
135	1.3140	48.82s
136	1.3139	48.68s
137	1.3137	48.65s
138	1.3135	48.59s
139	1.3134	48.71s
140	1.3132	48.56s
141	1.3130	48.43s
142	1.3129	48.29s
143	1.3127	48.14s
144	1.3126	47.99s
145	1.3124	47.82s
146	1.3122	47.67s
147	1.3121	47.50s
148	1.3119	47.34s

```
149             1.3118          47.19s
150             1.3116          47.04s
151             1.3114          46.88s
152             1.3113          46.74s
153             1.3111          46.59s
154             1.3110          46.58s
155             1.3108          46.60s
156             1.3107          46.92s
157             1.3105          47.08s
158             1.3104          47.19s
159             1.3102          47.14s
160             1.3101          47.05s
161             1.3099          47.00s
162             1.3098          46.88s
163             1.3096          46.76s
164             1.3095          46.72s
165             1.3093          46.79s
166             1.3092          46.71s
167             1.3090          46.67s
168             1.3089          46.56s
169             1.3088          46.45s
170             1.3086          46.40s
171             1.3085          46.27s
172             1.3083          46.21s
173             1.3082          46.16s
174             1.3081          46.11s
175             1.3079          46.11s
176             1.3078          46.09s
177             1.3076          46.04s
178             1.3075          46.08s
179             1.3074          46.09s
180             1.3072          46.21s
181             1.3071          46.18s
182             1.3070          46.19s
183             1.3068          46.14s
184             1.3067          46.12s
185             1.3066          46.13s
186             1.3064          46.05s
187             1.3063          45.96s
188             1.3062          45.89s
189             1.3061          45.89s
190             1.3059          45.91s
191             1.3058          45.97s
192             1.3057          45.98s
193             1.3056          45.94s
194             1.3054          45.87s
195             1.3053          45.81s
196             1.3052          45.75s
197             1.3051          45.66s
198             1.3049          45.58s
199             1.3048          45.49s
200             1.3047          45.39s
201             1.3046          45.34s
202             1.3045          45.27s
203             1.3043          45.25s
204             1.3042          45.18s
205             1.3041          45.13s
```

```
206             1.3040          45.02s
207             1.3039          44.92s
208             1.3037          44.82s
209             1.3036          44.71s
210             1.3035          44.60s
211             1.3034          44.50s
212             1.3033          44.39s
213             1.3032          44.30s
214             1.3031          44.21s
215             1.3029          44.15s
216             1.3028          44.05s
217             1.3027          43.96s
218             1.3026          43.86s
219             1.3025          43.78s
220             1.3024          43.67s
221             1.3023          43.58s
222             1.3022          43.48s
223             1.3021          43.39s
224             1.3020          43.29s
225             1.3019          43.19s
226             1.3018          43.10s
227             1.3016          43.00s
228             1.3015          42.90s
229             1.3014          42.82s
230             1.3013          42.80s
231             1.3012          42.71s
232             1.3011          42.64s
233             1.3010          42.55s
234             1.3009          42.47s
235             1.3008          42.38s
236             1.3007          42.29s
237             1.3006          42.24s
238             1.3005          42.23s
239             1.3004          42.20s
240             1.3003          42.15s
241             1.3002          42.19s
242             1.3001          42.14s
243             1.3000          42.08s
244             1.2999          42.04s
245             1.2998          41.97s
246             1.2997          41.91s
247             1.2996          41.88s
248             1.2996          41.85s
249             1.2995          41.83s
250             1.2994          41.81s
251             1.2993          41.74s
252             1.2992          41.71s
253             1.2991          41.66s
254             1.2990          41.58s
255             1.2989          41.50s
256             1.2988          41.43s
257             1.2987          41.35s
258             1.2986          41.32s
259             1.2985          41.37s
260             1.2984          41.44s
261             1.2983          41.43s
262             1.2983          41.40s
```

```
263             1.2982          41.38s
264             1.2981          41.36s
265             1.2980          41.36s
266             1.2979          41.35s
267             1.2978          41.29s
268             1.2977          41.21s
269             1.2976          41.15s
270             1.2975          41.08s
271             1.2975          41.14s
272             1.2974          41.12s
273             1.2973          41.07s
274             1.2972          41.00s
275             1.2971          40.92s
276             1.2970          40.84s
277             1.2969          40.79s
278             1.2968          40.76s
279             1.2968          40.73s
280             1.2967          40.66s
281             1.2966          40.60s
282             1.2965          40.53s
283             1.2964          40.48s
284             1.2963          40.45s
285             1.2963          40.41s
286             1.2962          40.40s
287             1.2961          40.34s
288             1.2960          40.31s
289             1.2959          40.26s
290             1.2958          40.19s
291             1.2958          40.13s
292             1.2957          40.06s
293             1.2956          40.00s
294             1.2955          39.92s
295             1.2954          39.87s
296             1.2954          39.80s
297             1.2953          39.73s
298             1.2952          39.67s
299             1.2951          39.61s
300             1.2950          39.54s
301             1.2950          39.50s
302             1.2949          39.53s
303             1.2948          39.48s
304             1.2947          39.41s
305             1.2946          39.35s
306             1.2946          39.28s
307             1.2945          39.21s
308             1.2944          39.15s
309             1.2943          39.09s
310             1.2943          39.02s
311             1.2942          38.96s
312             1.2941          38.90s
313             1.2940          38.86s
314             1.2940          38.89s
315             1.2939          38.86s
316             1.2938          38.81s
317             1.2937          38.78s
318             1.2937          38.87s
319             1.2936          38.88s
```

```
320         1.2935          38.86s
321         1.2934          38.84s
322         1.2934          38.78s
323         1.2933          38.72s
324         1.2932          38.69s
325         1.2932          38.64s
326         1.2931          38.62s
327         1.2930          38.62s
328         1.2929          38.62s
329         1.2929          38.57s
330         1.2928          38.58s
331         1.2927          38.55s
332         1.2927          38.49s
333         1.2926          38.46s
334         1.2925          38.40s
335         1.2924          38.36s
336         1.2924          38.30s
337         1.2923          38.25s
338         1.2922          38.19s
339         1.2922          38.16s
340         1.2921          38.22s
341         1.2920          38.24s
342         1.2920          38.29s
343         1.2919          38.34s
344         1.2918          38.36s
345         1.2918          38.36s
346         1.2917          38.35s
347         1.2916          38.33s
348         1.2916          38.28s
349         1.2915          38.22s
350         1.2914          38.16s
351         1.2914          38.14s
352         1.2913          38.10s
353         1.2912          38.06s
354         1.2912          38.02s
355         1.2911          37.97s
356         1.2910          37.91s
357         1.2910          37.86s
358         1.2909          37.80s
359         1.2908          37.76s
360         1.2908          37.71s
361         1.2907          37.65s
362         1.2906          37.59s
363         1.2906          37.54s
364         1.2905          37.48s
365         1.2904          37.43s
366         1.2904          37.38s
367         1.2903          37.33s
368         1.2903          37.29s
369         1.2902          37.24s
370         1.2901          37.21s
371         1.2901          37.17s
372         1.2900          37.13s
373         1.2899          37.08s
374         1.2899          37.04s
375         1.2898          36.99s
376         1.2898          36.94s
```

377	1.2897	36.89s
378	1.2896	36.83s
379	1.2896	36.79s
380	1.2895	36.74s
381	1.2895	36.72s
382	1.2894	36.74s
383	1.2893	36.77s
384	1.2893	36.78s
385	1.2892	36.78s
386	1.2891	36.80s
387	1.2891	36.81s
388	1.2890	36.76s
389	1.2890	36.72s
390	1.2889	36.67s
391	1.2888	36.63s
392	1.2888	36.58s
393	1.2887	36.53s
394	1.2887	36.48s
395	1.2886	36.47s
396	1.2885	36.42s
397	1.2885	36.40s
398	1.2884	36.39s
399	1.2884	36.37s
400	1.2883	36.32s
401	1.2883	36.27s
402	1.2882	36.24s
403	1.2881	36.20s
404	1.2881	36.18s
405	1.2880	36.14s
406	1.2880	36.10s
407	1.2879	36.06s
408	1.2879	36.03s
409	1.2878	36.01s
410	1.2877	36.01s
411	1.2877	35.99s
412	1.2876	36.00s
413	1.2876	36.00s
414	1.2875	36.01s
415	1.2875	35.99s
416	1.2874	35.95s
417	1.2873	35.91s
418	1.2873	35.87s
419	1.2872	35.82s
420	1.2872	35.78s
421	1.2871	35.74s
422	1.2871	35.72s
423	1.2870	35.69s
424	1.2870	35.65s
425	1.2869	35.62s
426	1.2868	35.58s
427	1.2868	35.53s
428	1.2867	35.52s
429	1.2867	35.48s
430	1.2866	35.44s
431	1.2866	35.39s
432	1.2865	35.34s
433	1.2865	35.30s

```
434          1.2864          35.26s
435          1.2864          35.22s
436          1.2863          35.18s
437          1.2862          35.13s
438          1.2862          35.09s
439          1.2861          35.05s
440          1.2861          35.01s
441          1.2860          34.96s
442          1.2860          34.94s
443          1.2859          34.90s
444          1.2859          34.85s
445          1.2858          34.82s
446          1.2858          34.80s
447          1.2857          34.76s
448          1.2857          34.72s
449          1.2856          34.68s
450          1.2856          34.69s
451          1.2855          34.71s
452          1.2855          34.71s
453          1.2854          34.71s
454          1.2854          34.71s
455          1.2853          34.68s
456          1.2853          34.65s
457          1.2852          34.64s
458          1.2852          34.68s
459          1.2851          34.67s
460          1.2851          34.64s
461          1.2850          34.61s
462          1.2850          34.57s
463          1.2849          34.52s
464          1.2849          34.48s
465          1.2848          34.44s
466          1.2848          34.40s
467          1.2847          34.36s
468          1.2847          34.32s
469          1.2846          34.30s
470          1.2846          34.26s
471          1.2845          34.22s
472          1.2845          34.19s
473          1.2844          34.40s
474          1.2844          34.81s
475          1.2843          34.88s
476          1.2843          34.85s
477          1.2842          34.82s
478          1.2842          34.80s
479          1.2841          34.76s
480          1.2841          34.72s
481          1.2840          34.69s
482          1.2840          34.65s
483          1.2839          34.61s
484          1.2839          34.57s
485          1.2838          34.53s
486          1.2838          34.48s
487          1.2837          34.45s
488          1.2837          34.41s
489          1.2836          34.38s
490          1.2836          34.35s
```

```
491             1.2835          34.32s
492             1.2835          34.28s
493             1.2834          34.24s
494             1.2834          34.21s
495             1.2834          34.17s
496             1.2833          34.16s
497             1.2833          34.14s
498             1.2832          34.13s
499             1.2832          34.12s
500             1.2831          34.11s
501             1.2831          34.12s
502             1.2830          34.14s
503             1.2830          34.12s
504             1.2829          34.10s
505             1.2829          34.07s
506             1.2828          34.04s
507             1.2828          34.00s
508             1.2828          33.96s
509             1.2827          33.92s
510             1.2827          33.88s
511             1.2826          33.84s
512             1.2826          33.80s
513             1.2825          33.76s
514             1.2825          33.72s
515             1.2824          33.68s
516             1.2824          33.64s
517             1.2823          33.60s
518             1.2823          33.56s
519             1.2823          33.53s
520             1.2822          33.50s
521             1.2822          33.46s
522             1.2821          33.43s
523             1.2821          33.39s
524             1.2820          33.36s
525             1.2820          33.34s
526             1.2820          33.33s
527             1.2819          33.35s
528             1.2819          33.35s
529             1.2818          33.34s
530             1.2818          33.33s
531             1.2817          33.33s
532             1.2817          33.38s
533             1.2816          33.70s
534             1.2816          33.76s
535             1.2816          33.75s
536             1.2815          33.72s
537             1.2815          33.69s
538             1.2814          33.71s
539             1.2814          33.74s
540             1.2813          33.73s
541             1.2813          33.88s
542             1.2813          33.95s
543             1.2812          33.94s
544             1.2812          33.95s
545             1.2811          33.97s
546             1.2811          34.03s
547             1.2810          34.01s
```

```
548          1.2810          33.99s
549          1.2810          34.51s
550          1.2809          34.50s
551          1.2809          34.49s
552          1.2808          34.47s
553          1.2808          34.45s
554          1.2808          34.44s
555          1.2807          34.45s
556          1.2807          34.44s
557          1.2806          34.43s
558          1.2806          34.42s
559          1.2805          34.38s
560          1.2805          34.36s
561          1.2805          34.46s
562          1.2804          34.44s
563          1.2804          34.44s
564          1.2803          34.46s
565          1.2803          34.45s
566          1.2803          34.43s
567          1.2802          34.47s
568          1.2802          34.56s
569          1.2801          34.59s
570          1.2801          34.74s
571          1.2801          34.72s
572          1.2800          34.69s
573          1.2800          34.71s
574          1.2799          34.68s
575          1.2799          34.69s
576          1.2799          34.69s
577          1.2798          34.69s
578          1.2798          34.66s
579          1.2797          34.64s
580          1.2797          34.61s
581          1.2797          34.60s
582          1.2796          34.58s
583          1.2796          34.56s
584          1.2795          34.53s
585          1.2795          34.51s
586          1.2795          34.49s
587          1.2794          34.49s
588          1.2794          34.46s
589          1.2794          34.43s
590          1.2793          34.41s
591          1.2793          34.38s
592          1.2792          34.36s
593          1.2792          34.34s
594          1.2792          34.31s
595          1.2791          34.30s
596          1.2791          34.27s
597          1.2790          34.24s
598          1.2790          34.21s
599          1.2790          34.17s
600          1.2789          34.15s
601          1.2789          34.12s
602          1.2789          34.09s
603          1.2788          34.07s
604          1.2788          34.03s
```

```
605             1.2787          34.02s
606             1.2787          34.12s
607             1.2787          34.15s
608             1.2786          34.12s
609             1.2786          34.10s
610             1.2785          34.10s
611             1.2785          34.11s
612             1.2785          34.11s
613             1.2784          34.09s
614             1.2784          34.06s
615             1.2784          34.05s
616             1.2783          34.02s
617             1.2783          33.99s
618             1.2783          33.97s
619             1.2782          34.01s
620             1.2782          34.03s
621             1.2781          34.07s
622             1.2781          34.07s
623             1.2781          34.05s
624             1.2780          34.06s
625             1.2780          34.15s
626             1.2780          34.20s
627             1.2779          34.36s
628             1.2779          34.38s
629             1.2778          34.37s
630             1.2778          34.35s
631             1.2778          34.33s
632             1.2777          34.30s
633             1.2777          34.26s
634             1.2777          34.22s
635             1.2776          34.18s
636             1.2776          34.16s
637             1.2776          34.12s
638             1.2775          34.09s
639             1.2775          34.05s
640             1.2775          34.01s
641             1.2774          33.97s
642             1.2774          33.94s
643             1.2773          33.90s
644             1.2773          33.87s
645             1.2773          33.86s
646             1.2772          33.84s
647             1.2772          33.80s
648             1.2772          33.77s
649             1.2771          33.73s
650             1.2771          33.70s
651             1.2771          33.66s
652             1.2770          33.63s
653             1.2770          33.59s
654             1.2770          33.56s
655             1.2769          33.52s
656             1.2769          33.49s
657             1.2769          33.45s
658             1.2768          33.42s
659             1.2768          33.38s
660             1.2768          33.35s
661             1.2767          33.32s
```

```
662             1.2767          33.30s
663             1.2767          33.36s
664             1.2766          33.40s
665             1.2766          33.54s
666             1.2766          33.52s
667             1.2765          33.50s
668             1.2765          33.48s
669             1.2764          33.46s
670             1.2764          33.45s
671             1.2764          33.46s
672             1.2763          33.48s
673             1.2763          33.54s
674             1.2763          33.54s
675             1.2762          33.53s
676             1.2762          33.51s
677             1.2762          33.49s
678             1.2761          33.46s
679             1.2761          33.49s
680             1.2761          33.61s
681             1.2760          33.66s
682             1.2760          33.69s
683             1.2760          33.67s
684             1.2759          33.67s
685             1.2759          33.65s
686             1.2759          33.64s
687             1.2758          33.64s
688             1.2758          33.64s
689             1.2758          33.69s
690             1.2757          33.72s
691             1.2757          33.70s
692             1.2757          33.69s
693             1.2756          33.68s
694             1.2756          33.69s
695             1.2756          33.68s
696             1.2756          33.66s
697             1.2755          33.63s
698             1.2755          33.60s
699             1.2755          33.58s
700             1.2754          33.56s
701             1.2754          33.54s
702             1.2754          33.51s
703             1.2753          33.49s
704             1.2753          33.46s
705             1.2753          33.43s
706             1.2752          33.40s
707             1.2752          33.36s
708             1.2752          33.33s
709             1.2751          33.30s
710             1.2751          33.27s
711             1.2751          33.23s
712             1.2750          33.20s
713             1.2750          33.16s
714             1.2750          33.13s
715             1.2749          33.10s
716             1.2749          33.07s
717             1.2749          33.03s
718             1.2748          33.00s
```

```
719             1.2748          32.97s
720             1.2748          32.93s
721             1.2747          32.90s
722             1.2747          32.86s
723             1.2747          32.83s
724             1.2747          32.79s
725             1.2746          32.76s
726             1.2746          32.72s
727             1.2746          32.69s
728             1.2745          32.65s
729             1.2745          32.62s
730             1.2745          32.58s
731             1.2744          32.55s
732             1.2744          32.52s
733             1.2744          32.49s
734             1.2743          32.46s
735             1.2743          32.42s
736             1.2743          32.39s
737             1.2742          32.35s
738             1.2742          32.32s
739             1.2742          32.28s
740             1.2742          32.25s
741             1.2741          32.21s
742             1.2741          32.18s
743             1.2741          32.15s
744             1.2740          32.11s
745             1.2740          32.08s
746             1.2740          32.04s
747             1.2739          32.01s
748             1.2739          31.97s
749             1.2739          31.94s
750             1.2739          31.91s
751             1.2738          31.89s
752             1.2738          31.87s
753             1.2738          31.84s
754             1.2737          31.81s
755             1.2737          31.79s
756             1.2737          31.78s
757             1.2736          31.75s
758             1.2736          31.72s
759             1.2736          31.70s
760             1.2735          31.68s
761             1.2735          31.66s
762             1.2735          31.66s
763             1.2735          31.74s
764             1.2734          31.71s
765             1.2734          31.69s
766             1.2734          31.66s
767             1.2733          31.63s
768             1.2733          31.60s
769             1.2733          31.58s
770             1.2733          31.56s
771             1.2732          31.55s
772             1.2732          31.53s
773             1.2732          31.55s
774             1.2731          31.60s
775             1.2731          31.57s
```

776	1.2731	31.56s
777	1.2730	31.53s
778	1.2730	31.50s
779	1.2730	31.47s
780	1.2730	31.44s
781	1.2729	31.42s
782	1.2729	31.41s
783	1.2729	31.41s
784	1.2728	31.46s
785	1.2728	31.65s
786	1.2728	31.64s
787	1.2728	31.62s
788	1.2727	31.63s
789	1.2727	31.62s
790	1.2727	31.71s
791	1.2726	31.70s
792	1.2726	31.70s
793	1.2726	31.69s
794	1.2726	31.69s
795	1.2725	31.67s
796	1.2725	31.65s
797	1.2725	31.64s
798	1.2724	31.63s
799	1.2724	31.61s
800	1.2724	31.63s
801	1.2724	31.61s
802	1.2723	31.59s
803	1.2723	31.56s
804	1.2723	31.53s
805	1.2722	31.50s
806	1.2722	31.48s
807	1.2722	31.46s
808	1.2722	31.43s
809	1.2721	31.41s
810	1.2721	31.38s
811	1.2721	31.35s
812	1.2720	31.32s
813	1.2720	31.28s
814	1.2720	31.26s
815	1.2720	31.27s
816	1.2719	31.24s
817	1.2719	31.23s
818	1.2719	31.22s
819	1.2718	31.21s
820	1.2718	31.21s
821	1.2718	31.19s
822	1.2718	31.17s
823	1.2717	31.15s
824	1.2717	31.12s
825	1.2717	31.10s
826	1.2717	31.07s
827	1.2716	31.05s
828	1.2716	31.03s
829	1.2716	31.01s
830	1.2715	30.98s
831	1.2715	30.95s
832	1.2715	30.93s

833	1.2715	30.90s
834	1.2714	30.87s
835	1.2714	30.84s
836	1.2714	30.81s
837	1.2714	30.78s
838	1.2713	30.76s
839	1.2713	30.74s
840	1.2713	30.72s
841	1.2712	30.69s
842	1.2712	30.66s
843	1.2712	30.63s
844	1.2712	30.59s
845	1.2711	30.57s
846	1.2711	30.58s
847	1.2711	30.56s
848	1.2711	30.53s
849	1.2710	30.50s
850	1.2710	30.48s
851	1.2710	30.44s
852	1.2710	30.42s
853	1.2709	30.40s
854	1.2709	30.38s
855	1.2709	30.36s
856	1.2708	30.35s
857	1.2708	30.34s
858	1.2708	30.34s
859	1.2708	30.32s
860	1.2707	30.30s
861	1.2707	30.28s
862	1.2707	30.25s
863	1.2707	30.22s
864	1.2706	30.21s
865	1.2706	30.21s
866	1.2706	30.23s
867	1.2706	30.22s
868	1.2705	30.21s
869	1.2705	30.20s
870	1.2705	30.19s
871	1.2705	30.20s
872	1.2704	30.19s
873	1.2704	30.19s
874	1.2704	30.17s
875	1.2703	30.15s
876	1.2703	30.14s
877	1.2703	30.12s
878	1.2703	30.10s
879	1.2702	30.07s
880	1.2702	30.04s
881	1.2702	30.02s
882	1.2702	30.00s
883	1.2701	29.97s
884	1.2701	29.95s
885	1.2701	29.92s
886	1.2701	29.89s
887	1.2700	29.86s
888	1.2700	29.83s
889	1.2700	29.80s

```
890          1.2700          29.77s
891          1.2699          29.74s
892          1.2699          29.70s
893          1.2699          29.67s
894          1.2699          29.64s
895          1.2698          29.61s
896          1.2698          29.58s
897          1.2698          29.55s
898          1.2698          29.52s
899          1.2697          29.49s
900          1.2697          29.46s
901          1.2697          29.43s
902          1.2697          29.40s
903          1.2696          29.37s
904          1.2696          29.34s
905          1.2696          29.31s
906          1.2696          29.28s
907          1.2695          29.25s
908          1.2695          29.22s
909          1.2695          29.19s
910          1.2695          29.16s
911          1.2694          29.13s
912          1.2694          29.17s
913          1.2694          29.15s
914          1.2694          29.12s
915          1.2693          29.10s
916          1.2693          29.07s
917          1.2693          29.05s
918          1.2693          29.02s
919          1.2692          28.99s
920          1.2692          28.96s
921          1.2692          28.93s
922          1.2692          28.90s
923          1.2691          28.87s
924          1.2691          28.84s
925          1.2691          28.81s
926          1.2691          28.78s
927          1.2690          28.76s
928          1.2690          28.73s
929          1.2690          28.70s
930          1.2690          28.66s
931          1.2689          28.64s
932          1.2689          28.61s
933          1.2689          28.58s
934          1.2689          28.55s
935          1.2688          28.52s
936          1.2688          28.49s
937          1.2688          28.47s
938          1.2688          28.45s
939          1.2688          28.42s
940          1.2687          28.40s
941          1.2687          28.39s
942          1.2687          28.38s
943          1.2687          28.37s
944          1.2686          28.36s
945          1.2686          28.35s
946          1.2686          28.34s
```

947	1.2686	28.32s
948	1.2685	28.30s
949	1.2685	28.28s
950	1.2685	28.27s
951	1.2685	28.26s
952	1.2684	28.24s
953	1.2684	28.23s
954	1.2684	28.20s
955	1.2684	28.19s
956	1.2684	28.17s
957	1.2683	28.16s
958	1.2683	28.14s
959	1.2683	28.11s
960	1.2683	28.10s
961	1.2682	28.10s
962	1.2682	28.49s
963	1.2682	28.63s
964	1.2682	28.66s
965	1.2681	28.77s
966	1.2681	28.76s
967	1.2681	28.88s
968	1.2681	28.85s
969	1.2680	28.87s
970	1.2680	28.91s
971	1.2680	28.98s
972	1.2680	28.96s
973	1.2680	28.94s
974	1.2679	29.14s
975	1.2679	29.11s
976	1.2679	29.09s
977	1.2679	29.12s
978	1.2678	29.10s
979	1.2678	29.09s
980	1.2678	29.07s
981	1.2678	29.05s
982	1.2677	29.04s
983	1.2677	29.03s
984	1.2677	29.04s
985	1.2677	29.03s
986	1.2677	29.12s
987	1.2676	29.12s
988	1.2676	29.11s
989	1.2676	29.10s
990	1.2676	29.10s
991	1.2675	29.11s
992	1.2675	29.12s
993	1.2675	29.20s
994	1.2675	29.24s
995	1.2675	29.24s
996	1.2674	29.24s
997	1.2674	29.31s
998	1.2674	29.34s
999	1.2674	29.34s
1000	1.2673	29.34s
1001	1.2673	29.35s
1002	1.2673	29.33s
1003	1.2673	29.31s

```
1004            1.2673          29.31s
1005            1.2672          29.32s
1006            1.2672          29.32s
1007            1.2672          29.30s
1008            1.2672          29.32s
1009            1.2671          29.32s
1010            1.2671          29.32s
1011            1.2671          29.36s
1012            1.2671          29.36s
1013            1.2670          29.33s
1014            1.2670          29.31s
1015            1.2670          29.30s
1016            1.2670          29.27s
1017            1.2670          29.25s
1018            1.2669          29.23s
1019            1.2669          29.21s
1020            1.2669          29.18s
1021            1.2669          29.16s
1022            1.2669          29.13s
1023            1.2668          29.15s
1024            1.2668          29.13s
1025            1.2668          29.13s
1026            1.2668          29.13s
1027            1.2667          29.10s
1028            1.2667          29.10s
1029            1.2667          29.09s
1030            1.2667          29.09s
1031            1.2667          29.07s
1032            1.2666          29.06s
1033            1.2666          29.03s
1034            1.2666          29.00s
1035            1.2666          28.98s
1036            1.2665          28.95s
1037            1.2665          28.92s
1038            1.2665          28.90s
1039            1.2665          28.87s
1040            1.2665          28.85s
1041            1.2664          28.83s
1042            1.2664          28.81s
1043            1.2664          28.80s
1044            1.2664          28.77s
1045            1.2664          28.75s
1046            1.2663          28.72s
1047            1.2663          28.70s
1048            1.2663          28.68s
1049            1.2663          28.66s
1050            1.2662          28.63s
1051            1.2662          28.61s
1052            1.2662          28.59s
1053            1.2662          28.57s
1054            1.2662          28.55s
1055            1.2661          28.53s
1056            1.2661          28.50s
1057            1.2661          28.48s
1058            1.2661          28.45s
1059            1.2661          28.43s
1060            1.2660          28.41s
```

```
1061            1.2660          28.39s
1062            1.2660          28.37s
1063            1.2660          28.35s
1064            1.2659          28.32s
1065            1.2659          28.29s
1066            1.2659          28.27s
1067            1.2659          28.26s
1068            1.2659          28.23s
1069            1.2658          28.21s
1070            1.2658          28.20s
1071            1.2658          28.19s
1072            1.2658          28.17s
1073            1.2658          28.15s
1074            1.2657          28.13s
1075            1.2657          28.10s
1076            1.2657          28.08s
1077            1.2657          28.07s
1078            1.2657          28.05s
1079            1.2656          28.04s
1080            1.2656          28.02s
1081            1.2656          28.01s
1082            1.2656          27.98s
1083            1.2655          27.95s
1084            1.2655          27.93s
1085            1.2655          27.90s
1086            1.2655          27.87s
1087            1.2655          27.85s
1088            1.2654          27.82s
1089            1.2654          27.82s
1090            1.2654          27.80s
1091            1.2654          27.77s
1092            1.2654          27.75s
1093            1.2653          27.73s
1094            1.2653          27.71s
1095            1.2653          27.68s
1096            1.2653          27.65s
1097            1.2653          27.62s
1098            1.2652          27.59s
1099            1.2652          27.58s
1100            1.2652          27.58s
1101            1.2652          27.57s
1102            1.2652          27.54s
1103            1.2651          27.53s
1104            1.2651          27.50s
1105            1.2651          27.48s
1106            1.2651          27.46s
1107            1.2651          27.44s
1108            1.2650          27.41s
1109            1.2650          27.39s
1110            1.2650          27.36s
1111            1.2650          27.33s
1112            1.2650          27.30s
1113            1.2649          27.27s
1114            1.2649          27.24s
1115            1.2649          27.25s
1116            1.2649          27.25s
1117            1.2649          27.23s
```

```
1118            1.2648          27.21s
1119            1.2648          27.19s
1120            1.2648          27.16s
1121            1.2648          27.15s
1122            1.2648          27.13s
1123            1.2647          27.10s
1124            1.2647          27.08s
1125            1.2647          27.07s
1126            1.2647          27.04s
1127            1.2647          27.02s
1128            1.2646          27.00s
1129            1.2646          26.97s
1130            1.2646          26.96s
1131            1.2646          26.93s
1132            1.2646          26.91s
1133            1.2645          26.88s
1134            1.2645          26.86s
1135            1.2645          26.86s
1136            1.2645          26.84s
1137            1.2645          26.82s
1138            1.2644          26.79s
1139            1.2644          26.77s
1140            1.2644          26.75s
1141            1.2644          26.73s
1142            1.2644          26.71s
1143            1.2643          26.68s
1144            1.2643          26.66s
1145            1.2643          26.65s
1146            1.2643          26.64s
1147            1.2643          26.62s
1148            1.2642          26.60s
1149            1.2642          26.58s
1150            1.2642          26.55s
1151            1.2642          26.55s
1152            1.2642          26.54s
1153            1.2641          26.55s
1154            1.2641          26.55s
1155            1.2641          26.54s
1156            1.2641          26.52s
1157            1.2641          26.50s
1158            1.2640          26.50s
1159            1.2640          26.50s
1160            1.2640          26.48s
1161            1.2640          26.46s
1162            1.2640          26.48s
1163            1.2639          26.49s
1164            1.2639          26.50s
1165            1.2639          26.49s
1166            1.2639          26.48s
1167            1.2639          26.50s
1168            1.2638          26.48s
1169            1.2638          26.50s
1170            1.2638          26.53s
1171            1.2638          26.52s
1172            1.2638          26.50s
1173            1.2637          26.51s
1174            1.2637          26.48s
```

```
1175            1.2637          26.46s
1176            1.2637          26.43s
1177            1.2637          26.41s
1178            1.2636          26.42s
1179            1.2636          26.40s
1180            1.2636          26.38s
1181            1.2636          26.37s
1182            1.2636          26.36s
1183            1.2636          26.34s
1184            1.2635          26.32s
1185            1.2635          26.30s
1186            1.2635          26.28s
1187            1.2635          26.26s
1188            1.2635          26.24s
1189            1.2634          26.22s
1190            1.2634          26.20s
1191            1.2634          26.17s
1192            1.2634          26.15s
1193            1.2634          26.13s
1194            1.2633          26.11s
1195            1.2633          26.09s
1196            1.2633          26.07s
1197            1.2633          26.05s
1198            1.2633          26.03s
1199            1.2632          26.02s
1200            1.2632          26.03s
1201            1.2632          26.07s
1202            1.2632          26.06s
1203            1.2632          26.04s
1204            1.2632          26.02s
1205            1.2631          26.00s
1206            1.2631          25.97s
1207            1.2631          25.97s
1208            1.2631          25.98s
1209            1.2631          25.95s
1210            1.2630          25.93s
1211            1.2630          25.91s
1212            1.2630          25.89s
1213            1.2630          25.89s
1214            1.2630          25.89s
1215            1.2629          25.87s
1216            1.2629          25.85s
1217            1.2629          25.85s
1218            1.2629          25.85s
1219            1.2629          25.83s
1220            1.2629          25.81s
1221            1.2628          25.79s
1222            1.2628          25.76s
1223            1.2628          25.74s
1224            1.2628          25.71s
1225            1.2628          25.70s
1226            1.2627          25.67s
1227            1.2627          25.66s
1228            1.2627          25.71s
1229            1.2627          25.70s
1230            1.2627          25.71s
1231            1.2626          25.69s
```

```
1232            1.2626          25.67s
1233            1.2626          25.66s
1234            1.2626          25.67s
1235            1.2626          25.66s
1236            1.2626          25.64s
1237            1.2625          25.63s
1238            1.2625          25.61s
1239            1.2625          25.59s
1240            1.2625          25.59s
1241            1.2625          25.60s
1242            1.2624          25.58s
1243            1.2624          25.56s
1244            1.2624          25.54s
1245            1.2624          25.53s
1246            1.2624          25.51s
1247            1.2624          25.49s
1248            1.2623          25.51s
1249            1.2623          25.50s
1250            1.2623          25.49s
1251            1.2623          25.47s
1252            1.2623          25.45s
1253            1.2622          25.43s
1254            1.2622          25.41s
1255            1.2622          25.39s
1256            1.2622          25.39s
1257            1.2622          25.37s
1258            1.2622          25.36s
1259            1.2621          25.34s
1260            1.2621          25.32s
1261            1.2621          25.30s
1262            1.2621          25.28s
1263            1.2621          25.27s
1264            1.2620          25.29s
1265            1.2620          25.27s
1266            1.2620          25.30s
1267            1.2620          25.28s
1268            1.2620          25.25s
1269            1.2620          25.23s
1270            1.2619          25.21s
1271            1.2619          25.19s
1272            1.2619          25.17s
1273            1.2619          25.14s
1274            1.2619          25.13s
1275            1.2618          25.11s
1276            1.2618          25.12s
1277            1.2618          25.12s
1278            1.2618          25.10s
1279            1.2618          25.10s
1280            1.2618          25.09s
1281            1.2617          25.08s
1282            1.2617          25.06s
1283            1.2617          25.04s
1284            1.2617          25.03s
1285            1.2617          25.01s
1286            1.2616          25.00s
1287            1.2616          25.00s
1288            1.2616          24.97s
```

```
1289            1.2616          24.99s
1290            1.2616          24.97s
1291            1.2616          24.95s
1292            1.2615          24.93s
1293            1.2615          24.92s
1294            1.2615          24.90s
1295            1.2615          24.88s
1296            1.2615          24.86s
1297            1.2615          24.84s
1298            1.2614          24.83s
1299            1.2614          24.81s
1300            1.2614          24.80s
1301            1.2614          24.78s
1302            1.2614          24.75s
1303            1.2613          24.75s
1304            1.2613          24.72s
1305            1.2613          24.70s
1306            1.2613          24.67s
1307            1.2613          24.64s
1308            1.2613          24.62s
1309            1.2612          24.59s
1310            1.2612          24.57s
1311            1.2612          24.54s
1312            1.2612          24.52s
1313            1.2612          24.50s
1314            1.2611          24.48s
1315            1.2611          24.45s
1316            1.2611          24.43s
1317            1.2611          24.40s
1318            1.2611          24.38s
1319            1.2611          24.35s
1320            1.2610          24.33s
1321            1.2610          24.32s
1322            1.2610          24.31s
1323            1.2610          24.29s
1324            1.2610          24.26s
1325            1.2610          24.24s
1326            1.2609          24.23s
1327            1.2609          24.23s
1328            1.2609          24.24s
1329            1.2609          24.22s
1330            1.2609          24.19s
1331            1.2609          24.17s
1332            1.2608          24.15s
1333            1.2608          24.14s
1334            1.2608          24.12s
1335            1.2608          24.11s
1336            1.2608          24.09s
1337            1.2608          24.09s
1338            1.2607          24.07s
1339            1.2607          24.05s
1340            1.2607          24.03s
1341            1.2607          24.01s
1342            1.2607          23.99s
1343            1.2607          23.97s
1344            1.2606          23.94s
1345            1.2606          23.92s
```

```
1346            1.2606          23.90s
1347            1.2606          23.88s
1348            1.2606          23.85s
1349            1.2605          23.84s
1350            1.2605          23.83s
1351            1.2605          23.81s
1352            1.2605          23.79s
1353            1.2605          23.78s
1354            1.2605          23.76s
1355            1.2604          23.73s
1356            1.2604          23.71s
1357            1.2604          23.69s
1358            1.2604          23.67s
1359            1.2604          23.64s
1360            1.2604          23.62s
1361            1.2603          23.60s
1362            1.2603          23.59s
1363            1.2603          23.57s
1364            1.2603          23.54s
1365            1.2603          23.52s
1366            1.2603          23.49s
1367            1.2602          23.47s
1368            1.2602          23.46s
1369            1.2602          23.44s
1370            1.2602          23.42s
1371            1.2602          23.40s
1372            1.2602          23.38s
1373            1.2601          23.37s
1374            1.2601          23.35s
1375            1.2601          23.35s
1376            1.2601          23.33s
1377            1.2601          23.31s
1378            1.2601          23.28s
1379            1.2600          23.26s
1380            1.2600          23.25s
1381            1.2600          23.25s
1382            1.2600          23.22s
1383            1.2600          23.20s
1384            1.2600          23.17s
1385            1.2599          23.15s
1386            1.2599          23.12s
1387            1.2599          23.10s
1388            1.2599          23.08s
1389            1.2599          23.06s
1390            1.2599          23.04s
1391            1.2598          23.02s
1392            1.2598          22.99s
1393            1.2598          22.97s
1394            1.2598          22.95s
1395            1.2598          22.94s
1396            1.2598          22.94s
1397            1.2597          22.96s
1398            1.2597          22.96s
1399            1.2597          22.94s
1400            1.2597          22.93s
1401            1.2597          22.91s
1402            1.2597          22.92s
```

```
1403            1.2596          22.91s
1404            1.2596          22.91s
1405            1.2596          22.91s
1406            1.2596          22.89s
1407            1.2596          22.87s
1408            1.2596          22.85s
1409            1.2595          22.84s
1410            1.2595          22.83s
1411            1.2595          22.83s
1412            1.2595          22.80s
1413            1.2595          22.78s
1414            1.2595          22.75s
1415            1.2594          22.73s
1416            1.2594          22.71s
1417            1.2594          22.69s
1418            1.2594          22.66s
1419            1.2594          22.68s
1420            1.2594          22.66s
1421            1.2593          22.66s
1422            1.2593          22.65s
1423            1.2593          22.65s
1424            1.2593          22.63s
1425            1.2593          22.61s
1426            1.2593          22.59s
1427            1.2592          22.57s
1428            1.2592          22.55s
1429            1.2592          22.53s
1430            1.2592          22.51s
1431            1.2592          22.49s
1432            1.2592          22.47s
1433            1.2591          22.47s
1434            1.2591          22.62s
1435            1.2591          22.59s
1436            1.2591          22.57s
1437            1.2591          22.55s
1438            1.2591          22.53s
1439            1.2590          22.50s
1440            1.2590          22.49s
1441            1.2590          22.47s
1442            1.2590          22.47s
1443            1.2590          22.46s
1444            1.2590          22.48s
1445            1.2589          22.47s
1446            1.2589          22.46s
1447            1.2589          22.43s
1448            1.2589          22.43s
1449            1.2589          22.46s
1450            1.2589          22.44s
1451            1.2589          22.42s
1452            1.2588          22.40s
1453            1.2588          22.39s
1454            1.2588          22.41s
1455            1.2588          22.42s
1456            1.2588          22.41s
1457            1.2588          22.39s
1458            1.2587          22.39s
1459            1.2587          22.37s
```

```
1460            1.2587          22.34s
1461            1.2587          22.33s
1462            1.2587          22.32s
1463            1.2587          22.31s
1464            1.2586          22.46s
1465            1.2586          22.45s
1466            1.2586          22.52s
1467            1.2586          22.62s
1468            1.2586          22.63s
1469            1.2586          22.64s
1470            1.2585          22.85s
1471            1.2585          22.95s
1472            1.2585          22.94s
1473            1.2585          22.94s
1474            1.2585          22.97s
1475            1.2585          22.95s
1476            1.2584          23.03s
1477            1.2584          23.14s
1478            1.2584          23.19s
1479            1.2584          23.22s
1480            1.2584          23.21s
1481            1.2584          23.27s
1482            1.2584          23.25s
1483            1.2583          23.22s
1484            1.2583          23.21s
1485            1.2583          23.20s
1486            1.2583          23.20s
1487            1.2583          23.18s
1488            1.2583          23.17s
1489            1.2582          23.17s
1490            1.2582          23.16s
1491            1.2582          23.13s
1492            1.2582          23.12s
1493            1.2582          23.12s
1494            1.2582          23.14s
1495            1.2581          23.18s
1496            1.2581          23.16s
1497            1.2581          23.13s
1498            1.2581          23.25s
1499            1.2581          23.23s
1500            1.2581          23.22s
1501            1.2581          23.20s
1502            1.2580          23.19s
1503            1.2580          23.18s
1504            1.2580          23.19s
1505            1.2580          23.17s
1506            1.2580          23.16s
1507            1.2580          23.13s
1508            1.2579          23.18s
1509            1.2579          23.17s
1510            1.2579          23.16s
1511            1.2579          23.14s
1512            1.2579          23.12s
1513            1.2579          23.10s
1514            1.2578          23.09s
1515            1.2578          23.15s
1516            1.2578          23.22s
```

1517	1.2578	23.29s
1518	1.2578	23.36s
1519	1.2578	23.39s
1520	1.2578	23.37s
1521	1.2577	23.40s
1522	1.2577	23.49s
1523	1.2577	23.70s
1524	1.2577	23.71s
1525	1.2577	23.76s
1526	1.2577	23.79s
1527	1.2576	23.81s
1528	1.2576	23.78s
1529	1.2576	23.92s
1530	1.2576	23.91s
1531	1.2576	23.89s
1532	1.2576	23.87s
1533	1.2576	23.89s
1534	1.2575	23.88s
1535	1.2575	23.87s
1536	1.2575	23.87s
1537	1.2575	23.92s
1538	1.2575	23.90s
1539	1.2575	23.88s
1540	1.2574	23.86s
1541	1.2574	23.86s
1542	1.2574	23.87s
1543	1.2574	23.88s
1544	1.2574	23.87s
1545	1.2574	23.89s
1546	1.2574	23.88s
1547	1.2573	23.87s
1548	1.2573	23.86s
1549	1.2573	23.85s
1550	1.2573	23.86s
1551	1.2573	23.86s
1552	1.2573	23.88s
1553	1.2572	23.89s
1554	1.2572	23.88s
1555	1.2572	23.86s
1556	1.2572	23.87s
1557	1.2572	23.90s
1558	1.2572	23.91s
1559	1.2572	23.88s
1560	1.2571	23.85s
1561	1.2571	23.83s
1562	1.2571	23.87s
1563	1.2571	23.86s
1564	1.2571	23.87s
1565	1.2571	23.87s
1566	1.2570	23.85s
1567	1.2570	23.83s
1568	1.2570	23.89s
1569	1.2570	23.90s
1570	1.2570	23.88s
1571	1.2570	23.87s
1572	1.2570	23.85s
1573	1.2569	23.88s

```
1574            1.2569          23.87s
1575            1.2569          23.84s
1576            1.2569          23.85s
1577            1.2569          23.83s
1578            1.2569          23.80s
1579            1.2569          23.80s
1580            1.2568          23.78s
1581            1.2568          23.76s
1582            1.2568          23.74s
1583            1.2568          23.73s
1584            1.2568          23.71s
1585            1.2568          23.73s
1586            1.2567          23.70s
1587            1.2567          23.66s
1588            1.2567          23.65s
1589            1.2567          23.63s
1590            1.2567          23.61s
1591            1.2567          23.60s
1592            1.2567          23.62s
1593            1.2566          23.60s
1594            1.2566          23.58s
1595            1.2566          23.56s
1596            1.2566          23.58s
1597            1.2566          23.57s
1598            1.2566          23.55s
1599            1.2565          23.53s
1600            1.2565          23.52s
1601            1.2565          23.53s
1602            1.2565          23.51s
1603            1.2565          23.49s
1604            1.2565          23.46s
1605            1.2565          23.47s
1606            1.2564          23.46s
1607            1.2564          23.45s
1608            1.2564          23.42s
1609            1.2564          23.39s
1610            1.2564          23.36s
1611            1.2564          23.33s
1612            1.2564          23.30s
1613            1.2563          23.29s
1614            1.2563          23.26s
1615            1.2563          23.27s
1616            1.2563          23.25s
1617            1.2563          23.24s
1618            1.2563          23.22s
1619            1.2563          23.20s
1620            1.2562          23.19s
1621            1.2562          23.16s
1622            1.2562          23.13s
1623            1.2562          23.11s
1624            1.2562          23.11s
1625            1.2562          23.08s
1626            1.2561          23.06s
1627            1.2561          23.05s
1628            1.2561          23.02s
1629            1.2561          22.99s
1630            1.2561          22.96s
```

```
1631            1.2561          22.97s
1632            1.2561          22.94s
1633            1.2560          22.92s
1634            1.2560          22.90s
1635            1.2560          22.89s
1636            1.2560          22.87s
1637            1.2560          22.84s
1638            1.2560          22.81s
1639            1.2560          22.80s
1640            1.2559          22.78s
1641            1.2559          22.75s
1642            1.2559          22.73s
1643            1.2559          22.70s
1644            1.2559          22.68s
1645            1.2559          22.65s
1646            1.2559          22.62s
1647            1.2558          22.59s
1648            1.2558          22.56s
1649            1.2558          22.53s
1650            1.2558          22.50s
1651            1.2558          22.47s
1652            1.2558          22.44s
1653            1.2558          22.41s
1654            1.2557          22.38s
1655            1.2557          22.35s
1656            1.2557          22.33s
1657            1.2557          22.30s
1658            1.2557          22.27s
1659            1.2557          22.24s
1660            1.2557          22.21s
1661            1.2556          22.17s
1662            1.2556          22.14s
1663            1.2556          22.11s
1664            1.2556          22.09s
1665            1.2556          22.06s
1666            1.2556          22.03s
1667            1.2556          21.99s
1668            1.2555          21.96s
1669            1.2555          21.93s
1670            1.2555          21.90s
1671            1.2555          21.87s
1672            1.2555          21.84s
1673            1.2555          21.81s
1674            1.2555          21.79s
1675            1.2554          21.75s
1676            1.2554          21.72s
1677            1.2554          21.70s
1678            1.2554          21.67s
1679            1.2554          21.64s
1680            1.2554          21.61s
1681            1.2554          21.58s
1682            1.2553          21.55s
1683            1.2553          21.52s
1684            1.2553          21.50s
1685            1.2553          21.47s
1686            1.2553          21.43s
1687            1.2553          21.40s
```

1688	1.2553	21.37s
1689	1.2552	21.34s
1690	1.2552	21.31s
1691	1.2552	21.28s
1692	1.2552	21.25s
1693	1.2552	21.22s
1694	1.2552	21.19s
1695	1.2552	21.16s
1696	1.2551	21.13s
1697	1.2551	21.10s
1698	1.2551	21.07s
1699	1.2551	21.04s
1700	1.2551	21.03s
1701	1.2551	21.01s
1702	1.2551	20.99s
1703	1.2550	20.98s
1704	1.2550	20.95s
1705	1.2550	20.92s
1706	1.2550	20.89s
1707	1.2550	20.86s
1708	1.2550	20.83s
1709	1.2550	20.80s
1710	1.2549	20.78s
1711	1.2549	20.76s
1712	1.2549	20.73s
1713	1.2549	20.70s
1714	1.2549	20.70s
1715	1.2549	20.68s
1716	1.2549	20.68s
1717	1.2548	20.66s
1718	1.2548	20.64s
1719	1.2548	20.61s
1720	1.2548	20.62s
1721	1.2548	20.60s
1722	1.2548	20.56s
1723	1.2548	20.53s
1724	1.2547	20.50s
1725	1.2547	20.47s
1726	1.2547	20.44s
1727	1.2547	20.41s
1728	1.2547	20.38s
1729	1.2547	20.35s
1730	1.2547	20.32s
1731	1.2546	20.29s
1732	1.2546	20.29s
1733	1.2546	20.26s
1734	1.2546	20.24s
1735	1.2546	20.21s
1736	1.2546	20.18s
1737	1.2546	20.16s
1738	1.2545	20.18s
1739	1.2545	20.17s
1740	1.2545	20.14s
1741	1.2545	20.11s
1742	1.2545	20.09s
1743	1.2545	20.07s
1744	1.2545	20.04s

```
1745            1.2545          20.01s
1746            1.2544          19.98s
1747            1.2544          19.95s
1748            1.2544          19.94s
1749            1.2544          19.92s
1750            1.2544          19.88s
1751            1.2544          19.85s
1752            1.2544          19.82s
1753            1.2543          19.79s
1754            1.2543          19.77s
1755            1.2543          19.76s
1756            1.2543          19.75s
1757            1.2543          19.72s
1758            1.2543          19.69s
1759            1.2543          19.68s
1760            1.2542          19.65s
1761            1.2542          19.63s
1762            1.2542          19.61s
1763            1.2542          19.58s
1764            1.2542          19.55s
1765            1.2542          19.52s
1766            1.2542          19.49s
1767            1.2541          19.46s
1768            1.2541          19.43s
1769            1.2541          19.40s
1770            1.2541          19.37s
1771            1.2541          19.34s
1772            1.2541          19.31s
1773            1.2541          19.29s
1774            1.2541          19.26s
1775            1.2540          19.23s
1776            1.2540          19.20s
1777            1.2540          19.17s
1778            1.2540          19.14s
1779            1.2540          19.11s
1780            1.2540          19.08s
1781            1.2540          19.05s
1782            1.2539          19.02s
1783            1.2539          18.99s
1784            1.2539          18.98s
1785            1.2539          18.95s
1786            1.2539          18.92s
1787            1.2539          18.89s
1788            1.2539          18.86s
1789            1.2538          18.83s
1790            1.2538          18.80s
1791            1.2538          18.78s
1792            1.2538          18.75s
1793            1.2538          18.72s
1794            1.2538          18.69s
1795            1.2538          18.66s
1796            1.2538          18.64s
1797            1.2537          18.61s
1798            1.2537          18.58s
1799            1.2537          18.56s
1800            1.2537          18.52s
1801            1.2537          18.50s
```

```
1802            1.2537          18.47s
1803            1.2537          18.44s
1804            1.2536          18.41s
1805            1.2536          18.38s
1806            1.2536          18.35s
1807            1.2536          18.32s
1808            1.2536          18.29s
1809            1.2536          18.26s
1810            1.2536          18.23s
1811            1.2536          18.20s
1812            1.2535          18.17s
1813            1.2535          18.14s
1814            1.2535          18.11s
1815            1.2535          18.08s
1816            1.2535          18.05s
1817            1.2535          18.02s
1818            1.2535          17.99s
1819            1.2534          17.96s
1820            1.2534          17.93s
1821            1.2534          17.90s
1822            1.2534          17.87s
1823            1.2534          17.84s
1824            1.2534          17.81s
1825            1.2534          17.78s
1826            1.2534          17.75s
1827            1.2533          17.72s
1828            1.2533          17.70s
1829            1.2533          17.67s
1830            1.2533          17.66s
1831            1.2533          17.63s
1832            1.2533          17.60s
1833            1.2533          17.58s
1834            1.2532          17.55s
1835            1.2532          17.53s
1836            1.2532          17.50s
1837            1.2532          17.47s
1838            1.2532          17.45s
1839            1.2532          17.43s
1840            1.2532          17.40s
1841            1.2532          17.37s
1842            1.2531          17.34s
1843            1.2531          17.31s
1844            1.2531          17.29s
1845            1.2531          17.26s
1846            1.2531          17.23s
1847            1.2531          17.20s
1848            1.2531          17.17s
1849            1.2531          17.14s
1850            1.2530          17.12s
1851            1.2530          17.09s
1852            1.2530          17.06s
1853            1.2530          17.03s
1854            1.2530          17.00s
1855            1.2530          16.97s
1856            1.2530          16.95s
1857            1.2529          16.93s
1858            1.2529          16.90s
```

1859	1.2529	16.88s
1860	1.2529	16.85s
1861	1.2529	16.83s
1862	1.2529	16.80s
1863	1.2529	16.77s
1864	1.2529	16.74s
1865	1.2528	16.72s
1866	1.2528	16.72s
1867	1.2528	16.70s
1868	1.2528	16.68s
1869	1.2528	16.68s
1870	1.2528	16.70s
1871	1.2528	16.67s
1872	1.2528	16.65s
1873	1.2527	16.62s
1874	1.2527	16.64s
1875	1.2527	16.61s
1876	1.2527	16.58s
1877	1.2527	16.55s
1878	1.2527	16.53s
1879	1.2527	16.50s
1880	1.2526	16.48s
1881	1.2526	16.46s
1882	1.2526	16.43s
1883	1.2526	16.40s
1884	1.2526	16.37s
1885	1.2526	16.34s
1886	1.2526	16.31s
1887	1.2526	16.28s
1888	1.2525	16.25s
1889	1.2525	16.22s
1890	1.2525	16.20s
1891	1.2525	16.17s
1892	1.2525	16.14s
1893	1.2525	16.11s
1894	1.2525	16.08s
1895	1.2525	16.05s
1896	1.2524	16.02s
1897	1.2524	16.00s
1898	1.2524	15.97s
1899	1.2524	15.94s
1900	1.2524	15.91s
1901	1.2524	15.88s
1902	1.2524	15.85s
1903	1.2524	15.82s
1904	1.2523	15.79s
1905	1.2523	15.77s
1906	1.2523	15.74s
1907	1.2523	15.71s
1908	1.2523	15.68s
1909	1.2523	15.65s
1910	1.2523	15.63s
1911	1.2523	15.61s
1912	1.2522	15.58s
1913	1.2522	15.56s
1914	1.2522	15.53s
1915	1.2522	15.50s

```
1916            1.2522          15.47s
1917            1.2522          15.44s
1918            1.2522          15.41s
1919            1.2522          15.38s
1920            1.2521          15.35s
1921            1.2521          15.33s
1922            1.2521          15.30s
1923            1.2521          15.27s
1924            1.2521          15.24s
1925            1.2521          15.21s
1926            1.2521          15.18s
1927            1.2520          15.15s
1928            1.2520          15.12s
1929            1.2520          15.09s
1930            1.2520          15.06s
1931            1.2520          15.04s
1932            1.2520          15.01s
1933            1.2520          14.98s
1934            1.2520          14.95s
1935            1.2519          14.92s
1936            1.2519          14.89s
1937            1.2519          14.87s
1938            1.2519          14.84s
1939            1.2519          14.81s
1940            1.2519          14.78s
1941            1.2519          14.76s
1942            1.2519          14.73s
1943            1.2518          14.70s
1944            1.2518          14.67s
1945            1.2518          14.64s
1946            1.2518          14.61s
1947            1.2518          14.59s
1948            1.2518          14.56s
1949            1.2518          14.53s
1950            1.2518          14.50s
1951            1.2517          14.47s
1952            1.2517          14.44s
1953            1.2517          14.41s
1954            1.2517          14.38s
1955            1.2517          14.35s
1956            1.2517          14.32s
1957            1.2517          14.30s
1958            1.2517          14.27s
1959            1.2516          14.24s
1960            1.2516          14.21s
1961            1.2516          14.18s
1962            1.2516          14.15s
1963            1.2516          14.12s
1964            1.2516          14.09s
1965            1.2516          14.06s
1966            1.2516          14.04s
1967            1.2515          14.01s
1968            1.2515          13.98s
1969            1.2515          13.95s
1970            1.2515          13.92s
1971            1.2515          13.89s
1972            1.2515          13.87s
```

1973	1.2515	13.84s
1974	1.2515	13.81s
1975	1.2515	13.78s
1976	1.2514	13.75s
1977	1.2514	13.72s
1978	1.2514	13.70s
1979	1.2514	13.67s
1980	1.2514	13.64s
1981	1.2514	13.61s
1982	1.2514	13.58s
1983	1.2514	13.56s
1984	1.2513	13.53s
1985	1.2513	13.50s
1986	1.2513	13.47s
1987	1.2513	13.44s
1988	1.2513	13.41s
1989	1.2513	13.39s
1990	1.2513	13.38s
1991	1.2513	13.35s
1992	1.2512	13.32s
1993	1.2512	13.29s
1994	1.2512	13.27s
1995	1.2512	13.24s
1996	1.2512	13.21s
1997	1.2512	13.18s
1998	1.2512	13.16s
1999	1.2512	13.13s
2000	1.2511	13.10s
2001	1.2511	13.08s
2002	1.2511	13.05s
2003	1.2511	13.02s
2004	1.2511	12.99s
2005	1.2511	12.96s
2006	1.2511	12.94s
2007	1.2511	12.91s
2008	1.2510	12.88s
2009	1.2510	12.85s
2010	1.2510	12.82s
2011	1.2510	12.80s
2012	1.2510	12.77s
2013	1.2510	12.74s
2014	1.2510	12.72s
2015	1.2510	12.71s
2016	1.2509	12.68s
2017	1.2509	12.66s
2018	1.2509	12.63s
2019	1.2509	12.60s
2020	1.2509	12.57s
2021	1.2509	12.56s
2022	1.2509	12.53s
2023	1.2509	12.50s
2024	1.2509	12.47s
2025	1.2508	12.44s
2026	1.2508	12.42s
2027	1.2508	12.39s
2028	1.2508	12.36s
2029	1.2508	12.33s

```
2030            1.2508          12.30s
2031            1.2508          12.28s
2032            1.2508          12.25s
2033            1.2507          12.22s
2034            1.2507          12.19s
2035            1.2507          12.16s
2036            1.2507          12.14s
2037            1.2507          12.11s
2038            1.2507          12.08s
2039            1.2507          12.05s
2040            1.2507          12.02s
2041            1.2506          12.00s
2042            1.2506          11.97s
2043            1.2506          11.94s
2044            1.2506          11.91s
2045            1.2506          11.88s
2046            1.2506          11.85s
2047            1.2506          11.82s
2048            1.2506          11.79s
2049            1.2506          11.77s
2050            1.2505          11.74s
2051            1.2505          11.71s
2052            1.2505          11.68s
2053            1.2505          11.65s
2054            1.2505          11.63s
2055            1.2505          11.60s
2056            1.2505          11.57s
2057            1.2505          11.54s
2058            1.2504          11.51s
2059            1.2504          11.49s
2060            1.2504          11.46s
2061            1.2504          11.43s
2062            1.2504          11.40s
2063            1.2504          11.38s
2064            1.2504          11.35s
2065            1.2504          11.33s
2066            1.2503          11.30s
2067            1.2503          11.27s
2068            1.2503          11.25s
2069            1.2503          11.22s
2070            1.2503          11.20s
2071            1.2503          11.17s
2072            1.2503          11.15s
2073            1.2503          11.12s
2074            1.2503          11.10s
2075            1.2502          11.08s
2076            1.2502          11.05s
2077            1.2502          11.02s
2078            1.2502          11.00s
2079            1.2502          10.97s
2080            1.2502          10.95s
2081            1.2502          10.92s
2082            1.2502          10.89s
2083            1.2501          10.86s
2084            1.2501          10.83s
2085            1.2501          10.81s
2086            1.2501          10.79s
```

```
2087            1.2501          10.76s
2088            1.2501          10.73s
2089            1.2501          10.71s
2090            1.2501          10.68s
2091            1.2501          10.65s
2092            1.2500          10.62s
2093            1.2500          10.60s
2094            1.2500          10.57s
2095            1.2500          10.54s
2096            1.2500          10.52s
2097            1.2500          10.49s
2098            1.2500          10.47s
2099            1.2500          10.44s
2100            1.2499          10.41s
2101            1.2499          10.38s
2102            1.2499          10.36s
2103            1.2499          10.33s
2104            1.2499          10.30s
2105            1.2499          10.28s
2106            1.2499          10.25s
2107            1.2499          10.22s
2108            1.2499          10.19s
2109            1.2498          10.16s
2110            1.2498          10.13s
2111            1.2498          10.11s
2112            1.2498          10.08s
2113            1.2498          10.05s
2114            1.2498          10.03s
2115            1.2498          10.00s
2116            1.2498           9.97s
2117            1.2497           9.94s
2118            1.2497           9.91s
2119            1.2497           9.89s
2120            1.2497           9.86s
2121            1.2497           9.83s
2122            1.2497           9.80s
2123            1.2497           9.78s
2124            1.2497           9.75s
2125            1.2497           9.72s
2126            1.2496           9.69s
2127            1.2496           9.66s
2128            1.2496           9.64s
2129            1.2496           9.61s
2130            1.2496           9.58s
2131            1.2496           9.55s
2132            1.2496           9.53s
2133            1.2496           9.50s
2134            1.2496           9.47s
2135            1.2495           9.45s
2136            1.2495           9.42s
2137            1.2495           9.40s
2138            1.2495           9.37s
2139            1.2495           9.35s
2140            1.2495           9.32s
2141            1.2495           9.29s
2142            1.2495           9.27s
2143            1.2494           9.24s
```

2144	1.2494	9.21s
2145	1.2494	9.18s
2146	1.2494	9.16s
2147	1.2494	9.13s
2148	1.2494	9.10s
2149	1.2494	9.08s
2150	1.2494	9.05s
2151	1.2494	9.02s
2152	1.2493	8.99s
2153	1.2493	8.96s
2154	1.2493	8.94s
2155	1.2493	8.91s
2156	1.2493	8.88s
2157	1.2493	8.85s
2158	1.2493	8.83s
2159	1.2493	8.80s
2160	1.2493	8.77s
2161	1.2492	8.74s
2162	1.2492	8.72s
2163	1.2492	8.69s
2164	1.2492	8.66s
2165	1.2492	8.63s
2166	1.2492	8.61s
2167	1.2492	8.58s
2168	1.2492	8.55s
2169	1.2492	8.53s
2170	1.2491	8.50s
2171	1.2491	8.47s
2172	1.2491	8.45s
2173	1.2491	8.42s
2174	1.2491	8.39s
2175	1.2491	8.36s
2176	1.2491	8.33s
2177	1.2491	8.31s
2178	1.2490	8.28s
2179	1.2490	8.25s
2180	1.2490	8.23s
2181	1.2490	8.20s
2182	1.2490	8.17s
2183	1.2490	8.14s
2184	1.2490	8.12s
2185	1.2490	8.09s
2186	1.2490	8.06s
2187	1.2489	8.03s
2188	1.2489	8.01s
2189	1.2489	7.98s
2190	1.2489	7.95s
2191	1.2489	7.93s
2192	1.2489	7.90s
2193	1.2489	7.87s
2194	1.2489	7.85s
2195	1.2489	7.82s
2196	1.2488	7.79s
2197	1.2488	7.77s
2198	1.2488	7.74s
2199	1.2488	7.71s
2200	1.2488	7.69s

```
2201            1.2488          7.66s
2202            1.2488          7.63s
2203            1.2488          7.61s
2204            1.2488          7.58s
2205            1.2487          7.55s
2206            1.2487          7.53s
2207            1.2487          7.50s
2208            1.2487          7.47s
2209            1.2487          7.45s
2210            1.2487          7.42s
2211            1.2487          7.39s
2212            1.2487          7.36s
2213            1.2487          7.34s
2214            1.2486          7.31s
2215            1.2486          7.28s
2216            1.2486          7.26s
2217            1.2486          7.23s
2218            1.2486          7.21s
2219            1.2486          7.18s
2220            1.2486          7.15s
2221            1.2486          7.13s
2222            1.2486          7.10s
2223            1.2485          7.07s
2224            1.2485          7.05s
2225            1.2485          7.02s
2226            1.2485          6.99s
2227            1.2485          6.97s
2228            1.2485          6.94s
2229            1.2485          6.92s
2230            1.2485          6.89s
2231            1.2485          6.86s
2232            1.2484          6.84s
2233            1.2484          6.81s
2234            1.2484          6.78s
2235            1.2484          6.76s
2236            1.2484          6.73s
2237            1.2484          6.70s
2238            1.2484          6.68s
2239            1.2484          6.65s
2240            1.2484          6.62s
2241            1.2483          6.60s
2242            1.2483          6.57s
2243            1.2483          6.54s
2244            1.2483          6.52s
2245            1.2483          6.49s
2246            1.2483          6.46s
2247            1.2483          6.44s
2248            1.2483          6.41s
2249            1.2483          6.38s
2250            1.2482          6.35s
2251            1.2482          6.33s
2252            1.2482          6.30s
2253            1.2482          6.28s
2254            1.2482          6.25s
2255            1.2482          6.22s
2256            1.2482          6.19s
2257            1.2482          6.17s
```

2258	1.2482	6.14s
2259	1.2481	6.12s
2260	1.2481	6.09s
2261	1.2481	6.06s
2262	1.2481	6.03s
2263	1.2481	6.01s
2264	1.2481	5.98s
2265	1.2481	5.96s
2266	1.2481	5.93s
2267	1.2481	5.90s
2268	1.2480	5.87s
2269	1.2480	5.85s
2270	1.2480	5.82s
2271	1.2480	5.79s
2272	1.2480	5.77s
2273	1.2480	5.74s
2274	1.2480	5.71s
2275	1.2480	5.69s
2276	1.2480	5.66s
2277	1.2479	5.63s
2278	1.2479	5.61s
2279	1.2479	5.58s
2280	1.2479	5.55s
2281	1.2479	5.53s
2282	1.2479	5.50s
2283	1.2479	5.47s
2284	1.2479	5.45s
2285	1.2479	5.42s
2286	1.2479	5.40s
2287	1.2478	5.37s
2288	1.2478	5.35s
2289	1.2478	5.32s
2290	1.2478	5.29s
2291	1.2478	5.27s
2292	1.2478	5.24s
2293	1.2478	5.21s
2294	1.2478	5.19s
2295	1.2478	5.16s
2296	1.2477	5.13s
2297	1.2477	5.11s
2298	1.2477	5.08s
2299	1.2477	5.05s
2300	1.2477	5.03s
2301	1.2477	5.00s
2302	1.2477	4.97s
2303	1.2477	4.95s
2304	1.2477	4.92s
2305	1.2476	4.90s
2306	1.2476	4.87s
2307	1.2476	4.85s
2308	1.2476	4.82s
2309	1.2476	4.79s
2310	1.2476	4.77s
2311	1.2476	4.74s
2312	1.2476	4.71s
2313	1.2476	4.69s
2314	1.2475	4.66s

```
2315            1.2475          4.64s
2316            1.2475          4.61s
2317            1.2475          4.58s
2318            1.2475          4.56s
2319            1.2475          4.53s
2320            1.2475          4.50s
2321            1.2475          4.48s
2322            1.2475          4.45s
2323            1.2474          4.43s
2324            1.2474          4.40s
2325            1.2474          4.37s
2326            1.2474          4.35s
2327            1.2474          4.32s
2328            1.2474          4.30s
2329            1.2474          4.27s
2330            1.2474          4.24s
2331            1.2474          4.22s
2332            1.2474          4.19s
2333            1.2473          4.17s
2334            1.2473          4.14s
2335            1.2473          4.11s
2336            1.2473          4.09s
2337            1.2473          4.06s
2338            1.2473          4.03s
2339            1.2473          4.01s
2340            1.2473          3.98s
2341            1.2473          3.96s
2342            1.2472          3.93s
2343            1.2472          3.90s
2344            1.2472          3.88s
2345            1.2472          3.85s
2346            1.2472          3.83s
2347            1.2472          3.80s
2348            1.2472          3.77s
2349            1.2472          3.75s
2350            1.2472          3.72s
2351            1.2471          3.70s
2352            1.2471          3.67s
2353            1.2471          3.64s
2354            1.2471          3.62s
2355            1.2471          3.59s
2356            1.2471          3.57s
2357            1.2471          3.54s
2358            1.2471          3.51s
2359            1.2471          3.49s
2360            1.2471          3.46s
2361            1.2470          3.43s
2362            1.2470          3.41s
2363            1.2470          3.38s
2364            1.2470          3.36s
2365            1.2470          3.33s
2366            1.2470          3.30s
2367            1.2470          3.28s
2368            1.2470          3.25s
2369            1.2470          3.23s
2370            1.2469          3.20s
2371            1.2469          3.17s
```

```
2372            1.2469          3.15s
2373            1.2469          3.12s
2374            1.2469          3.10s
2375            1.2469          3.07s
2376            1.2469          3.04s
2377            1.2469          3.02s
2378            1.2469          2.99s
2379            1.2469          2.97s
2380            1.2468          2.94s
2381            1.2468          2.92s
2382            1.2468          2.89s
2383            1.2468          2.86s
2384            1.2468          2.84s
2385            1.2468          2.81s
2386            1.2468          2.79s
2387            1.2468          2.76s
2388            1.2468          2.73s
2389            1.2467          2.71s
2390            1.2467          2.68s
2391            1.2467          2.66s
2392            1.2467          2.63s
2393            1.2467          2.61s
2394            1.2467          2.58s
2395            1.2467          2.55s
2396            1.2467          2.53s
2397            1.2467          2.50s
2398            1.2467          2.48s
2399            1.2466          2.45s
2400            1.2466          2.43s
2401            1.2466          2.40s
2402            1.2466          2.38s
2403            1.2466          2.35s
2404            1.2466          2.32s
2405            1.2466          2.30s
2406            1.2466          2.27s
2407            1.2466          2.25s
2408            1.2465          2.22s
2409            1.2465          2.20s
2410            1.2465          2.17s
2411            1.2465          2.14s
2412            1.2465          2.12s
2413            1.2465          2.09s
2414            1.2465          2.07s
2415            1.2465          2.04s
2416            1.2465          2.02s
2417            1.2465          1.99s
2418            1.2464          1.97s
2419            1.2464          1.94s
2420            1.2464          1.91s
2421            1.2464          1.89s
2422            1.2464          1.86s
2423            1.2464          1.84s
2424            1.2464          1.81s
2425            1.2464          1.79s
2426            1.2464          1.76s
2427            1.2463          1.74s
2428            1.2463          1.71s
```

```
2429            1.2463          1.68s
2430            1.2463          1.66s
2431            1.2463          1.63s
2432            1.2463          1.61s
2433            1.2463          1.58s
2434            1.2463          1.56s
2435            1.2463          1.53s
2436            1.2463          1.51s
2437            1.2462          1.48s
2438            1.2462          1.46s
2439            1.2462          1.45s
2440            1.2462          1.43s
2441            1.2462          1.41s
2442            1.2462          1.39s
2443            1.2462          1.36s
2444            1.2462          1.34s
2445            1.2462          1.31s
2446            1.2462          1.29s
2447            1.2461          1.26s
2448            1.2461          1.25s
2449            1.2461          1.22s
2450            1.2461          1.20s
2451            1.2461          1.17s
2452            1.2461          1.15s
2453            1.2461          1.13s
2454            1.2461          1.10s
2455            1.2461          1.08s
2456            1.2461          1.05s
2457            1.2460          1.02s
2458            1.2460          1.00s
2459            1.2460          0.97s
2460            1.2460          0.95s
2461            1.2460          0.92s
2462            1.2460          0.90s
2463            1.2460          0.87s
2464            1.2460          0.85s
2465            1.2460          0.82s
2466            1.2459          0.79s
2467            1.2459          0.77s
2468            1.2459          0.74s
2469            1.2459          0.72s
2470            1.2459          0.69s
2471            1.2459          0.66s
2472            1.2459          0.64s
2473            1.2459          0.61s
2474            1.2459          0.59s
2475            1.2459          0.56s
2476            1.2458          0.53s
2477            1.2458          0.51s
2478            1.2458          0.48s
2479            1.2458          0.45s
2480            1.2458          0.43s
2481            1.2458          0.40s
2482            1.2458          0.37s
2483            1.2458          0.35s
2484            1.2458          0.32s
2485            1.2458          0.30s
```

```
2486          1.2457          0.27s
2487          1.2457          0.24s
2488          1.2457          0.22s
2489          1.2457          0.19s
2490          1.2457          0.16s
2491          1.2457          0.13s
2492          1.2457          0.11s
2493          1.2457          0.08s
2494          1.2457          0.05s
2495          1.2457          0.03s
2496          1.2456          0.00s
```

Out[90]:
```
GradientBoostingClassifier(learning_rate=0.01, max_leaf_nodes=2,
                           min_samples_leaf=10, n_estimators=2496,
                           random_state=88, verbose=3)
```

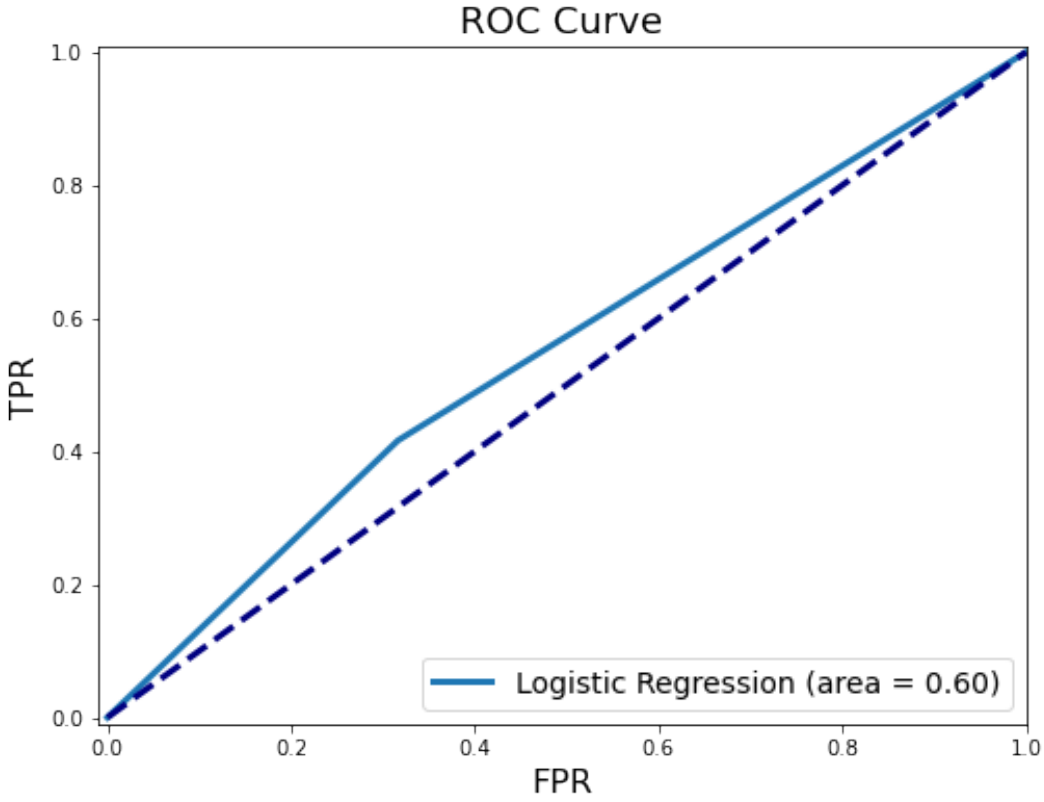
In [91]:
```
xd_red_test = pd.DataFrame()
for column in columns_red:
    xd_red_test[column] = xd_test[column]
```

In [92]:
```
from sklearn.metrics import accuracy_score


print('Accuracy:', round(accuracy_score(y_test, gbc2.predict(xd_red_test)), 5
```

```
Accuracy: 0.63366
```

In [93]:
```
from sklearn.metrics import roc_curve, auc

fpr2, tpr2, _ = roc_curve(y_test, gbc2.predict(xd_red_test))

print (fpr,tpr)
roc_auc = auc(fpr2, tpr2)
plt.figure(figsize=(8, 6))
plt.title('ROC Curve', fontsize=18)
plt.xlabel('FPR', fontsize=16)
plt.ylabel('TPR', fontsize=16)
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr, tpr, lw=3, label='Logistic Regression (area = {:0.2f})'.format(
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.legend(loc='lower right', fontsize=14)
plt.show()
```

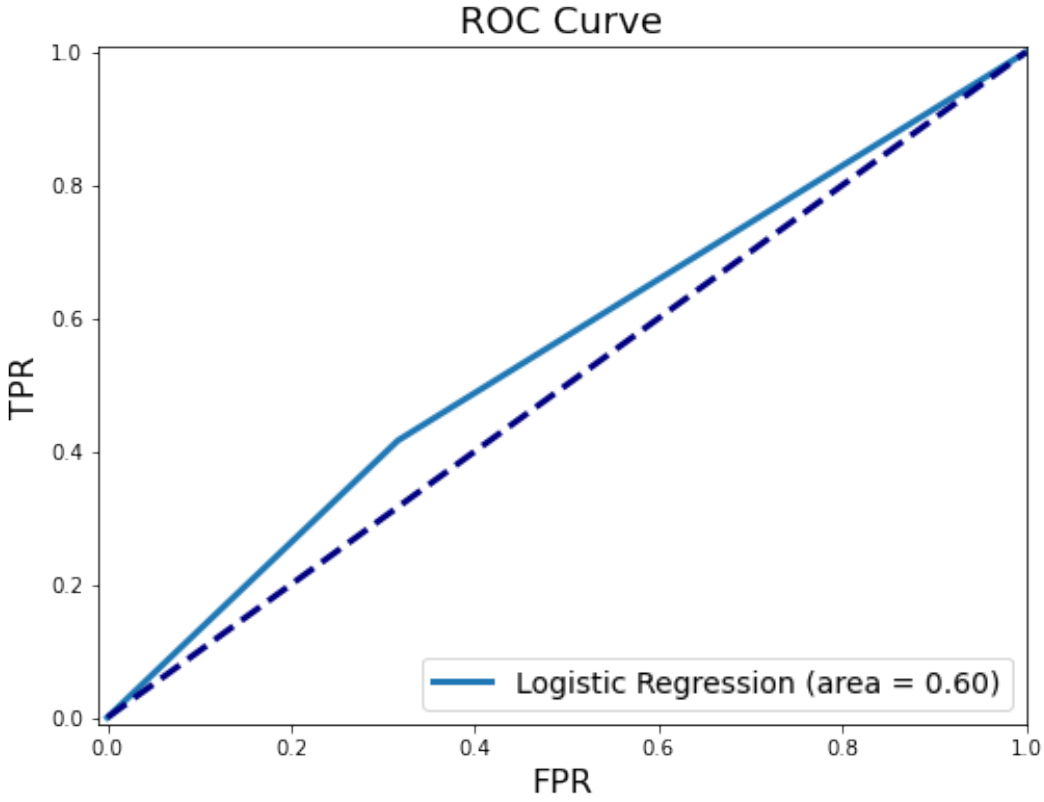
```
[0.         0.31627907 1.        ] [0.         0.41630901 1.        ]
```



In [93]:

```
from sklearn.metrics import roc_curve, auc

fpr2, tpr2, _ = roc_curve(y_test, gbc2.predict(xd_red_test))

print (fpr,tpr)
roc_auc = auc(fpr2, tpr2)
plt.figure(figsize=(8, 6))
plt.title('ROC Curve', fontsize=18)
plt.xlabel('FPR', fontsize=16)
plt.ylabel('TPR', fontsize=16)
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr, tpr, lw=3, label='Logistic Regression (area = {:0.2f})'.format(
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.legend(loc='lower right', fontsize=14)
plt.show()
```

```
[0.         0.31627907 1.        ] [0.         0.41630901 1.        ]
```



Final model: AUC = 0.60, TPR =0.42, FPR = 0.32, AUC = 0.60

In [100…

```
d = 0
for y in y_test:
    if y==0:
        d+=1
print (d/len(y_test))
```

```
0.5805580558055805
```

In [ ]:

In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
```

In [6]:
```
import statsmodels.formula.api as smf
```

In [7]:
```
#Helper Function: accuracy
```

In [8]:
```
def accuracy(y_test, y_pred):
    a=0
    for i in len(y_test):
        if y_test[i]==y_pred[i]:
            a+=1
    return a/len(y_test)
```

In [9]:
```
data = pd.read_csv('/Users/william/Downloads/cleaned_data.csv')
from sklearn.model_selection import train_test_split

data_train, data_test = train_test_split(data, test_size=0.3, random_state=95
```

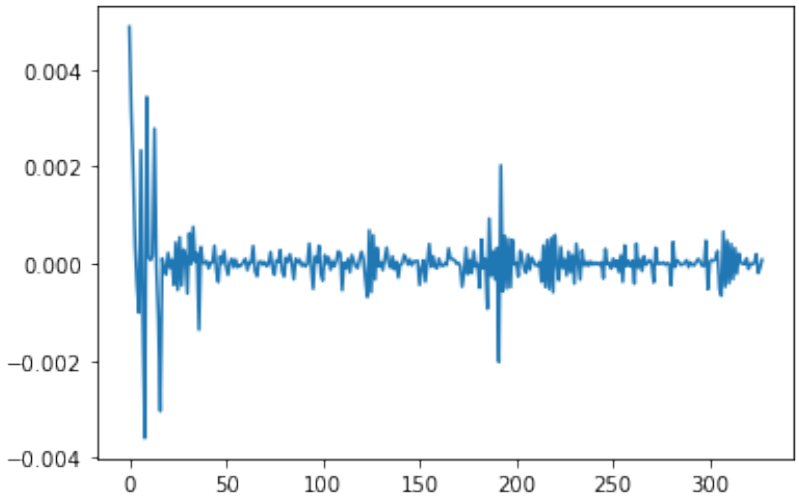
In [10]:
```
y_train = data_train['cumulative_outcome']
y_train = [1 if x=='Yes' else 0 for x in y_train]
x_train = data_train[data_train.columns[1:]]
x_train = x_train.drop(columns = ['cumulative_outcome'],axis=1)
xd = pd.get_dummies(x_train)
```

In [11]:
```
for column in xd.columns:
    xd[column]=xd[column].fillna(xd[column].mode()[0])
```

In [12]:
```
from sklearn.linear_model import RidgeClassifier

alpha_max = 10**5
rr = RidgeClassifier(alpha=alpha_max, random_state=88)
rr.fit(xd, y_train)
```

Out[12]:
```
RidgeClassifier(alpha=100000, random_state=88)
```

In [13]:
```
plt.plot(rr.coef_[0])
plt.show()
print(max(abs(rr.coef_[0])))
```



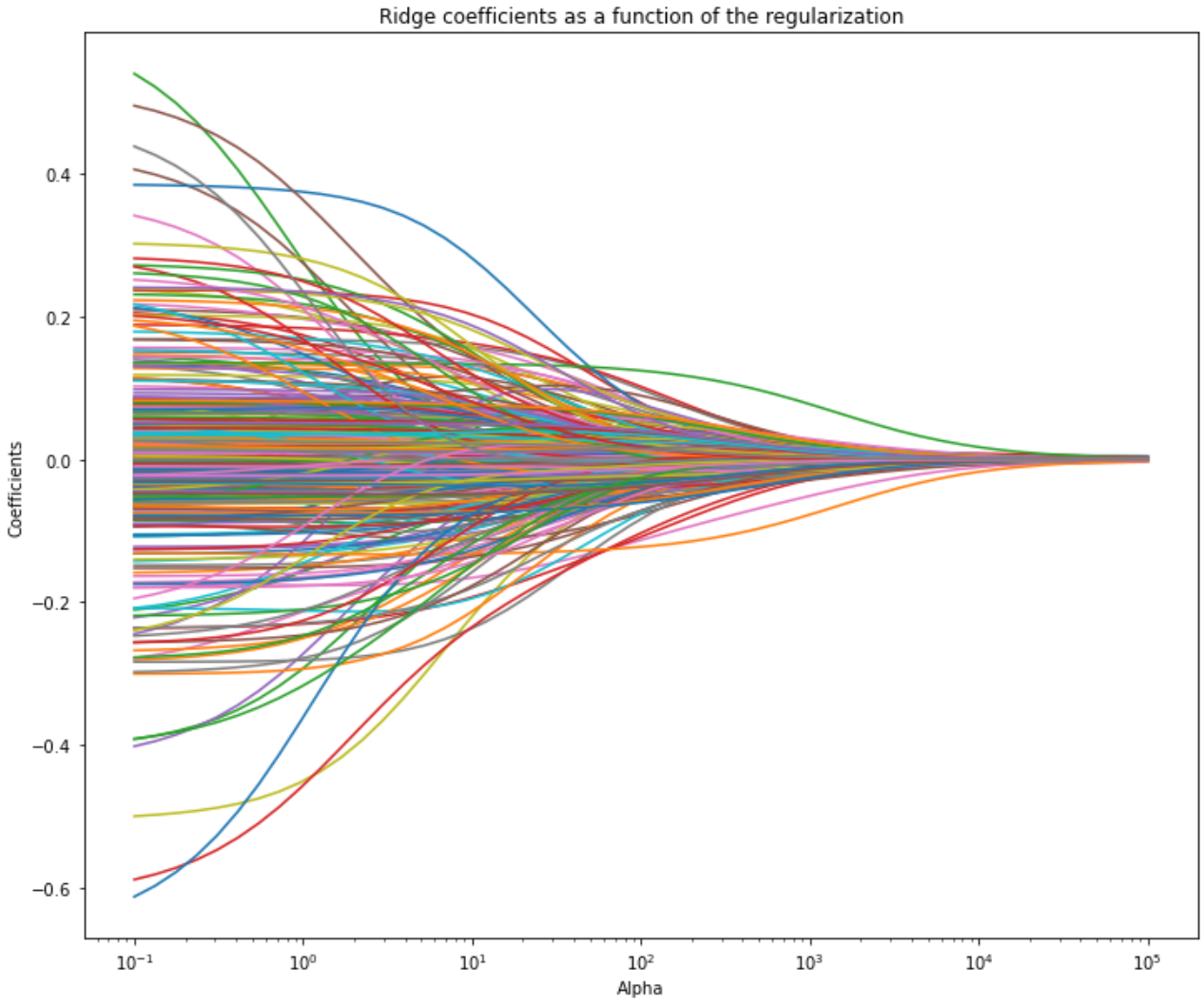

```
0.0048814152535055135
```

In [14]:
```
rr.coef_[0][1]
```

Out[14]:
```
0.0031545544766037755
```

In [15]:

```
alpha_grid = np.logspace(-1, 5, num=50, base=10)
coefs = []
for a in alpha_grid:
    rr = RidgeClassifier(alpha=a, fit_intercept=False, random_state=88)
    rr.fit(xd, y_train)
    coefs.append(rr.coef_[0])

plt.figure(figsize=(12, 10))
ax = plt.gca()
ax.plot(alpha_grid, coefs)
ax.set_xscale('log')
plt.xlabel('Alpha')
plt.ylabel('Coefficients')
plt.title('Ridge coefficients as a function of the regularization')
plt.show()
```



In [16]:

```
from sklearn.model_selection import GridSearchCV


alpha_grid = {'alpha': np.logspace(-1, 5, num=50, base=10)}

rr = RidgeClassifier(random_state=88)
rr_cv = GridSearchCV(rr, alpha_grid, scoring='accuracy', cv=10, verbose = 4)
rr_cv.fit(xd, y_train)
```

```
Fitting 10 folds for each of 50 candidates, totalling 500 fits
[CV 1/10] END .....................................alpha=0.1; total time=   0.
1s
[CV 2/10] END .....................................alpha=0.1; total time=   0.
0s
[CV 3/10] END .....................................alpha=0.1; total time=   0.
0s
[CV 4/10] END .....................................alpha=0.1; total time=   0.
0s
[CV 5/10] END .....................................alpha=0.1; total time=   0.
0s
[CV 6/10] END .....................................alpha=0.1; total time=   0.
0s
[CV 7/10] END .....................................alpha=0.1; total time=   0.
0s
[CV 8/10] END .....................................alpha=0.1; total time=   0.
0s
[CV 9/10] END .....................................alpha=0.1; total time=   0.
0s
[CV 10/10] END ....................................alpha=0.1; total time=   0.
0s
[CV 1/10] END ..................alpha=0.13257113655901093; total time=   0.
1s
[CV 2/10] END ..................alpha=0.13257113655901093; total time=   0.
0s
[CV 3/10] END ..................alpha=0.13257113655901093; total time=   0.
0s
[CV 4/10] END ..................alpha=0.13257113655901093; total time=   0.
0s
[CV 5/10] END ..................alpha=0.13257113655901093; total time=   0.
0s
[CV 6/10] END ..................alpha=0.13257113655901093; total time=   0.
1s
[CV 7/10] END ..................alpha=0.13257113655901093; total time=   0.
1s
[CV 8/10] END ..................alpha=0.13257113655901093; total time=   0.
1s
[CV 9/10] END ..................alpha=0.13257113655901093; total time=   0.
1s
[CV 10/10] END .................alpha=0.13257113655901093; total time=   0.
0s
[CV 1/10] END ..................alpha=0.1757510624854792; total time=   0.
1s
[CV 2/10] END ..................alpha=0.1757510624854792; total time=   0.
0s
[CV 3/10] END ..................alpha=0.1757510624854792; total time=   0.
```

```
0s
[CV 4/10] END .....................alpha=0.1757510624854792; total time=   0.
0s
[CV 5/10] END .....................alpha=0.1757510624854792; total time=   0.
0s
[CV 6/10] END .....................alpha=0.1757510624854792; total time=   0.
0s
[CV 7/10] END .....................alpha=0.1757510624854792; total time=   0.
1s
[CV 8/10] END .....................alpha=0.1757510624854792; total time=   0.
2s
[CV 9/10] END .....................alpha=0.1757510624854792; total time=   0.
1s
[CV 10/10] END .....................alpha=0.1757510624854792; total time=   0.
0s
[CV 1/10] END .....................alpha=0.2329951810515372; total time=   0.
0s
[CV 2/10] END .....................alpha=0.2329951810515372; total time=   0.
0s
[CV 3/10] END .....................alpha=0.2329951810515372; total time=   0.
0s
[CV 4/10] END .....................alpha=0.2329951810515372; total time=   0.
1s
[CV 5/10] END .....................alpha=0.2329951810515372; total time=   0.
0s
[CV 6/10] END .....................alpha=0.2329951810515372; total time=   0.
0s
[CV 7/10] END .....................alpha=0.2329951810515372; total time=   0.
1s
[CV 8/10] END .....................alpha=0.2329951810515372; total time=   0.
1s
[CV 9/10] END .....................alpha=0.2329951810515372; total time=   0.
0s
[CV 10/10] END .....................alpha=0.2329951810515372; total time=   0.
0s
[CV 1/10] END .....................alpha=0.3088843596477481; total time=   0.
0s
[CV 2/10] END .....................alpha=0.3088843596477481; total time=   0.
1s
[CV 3/10] END .....................alpha=0.3088843596477481; total time=   0.
0s
[CV 4/10] END .....................alpha=0.3088843596477481; total time=   0.
0s
[CV 5/10] END .....................alpha=0.3088843596477481; total time=   0.
1s
[CV 6/10] END .....................alpha=0.3088843596477481; total time=   0.
1s
[CV 7/10] END .....................alpha=0.3088843596477481; total time=   0.
1s
[CV 8/10] END .....................alpha=0.3088843596477481; total time=   0.
0s
[CV 9/10] END .....................alpha=0.3088843596477481; total time=   0.
0s
[CV 10/10] END .....................alpha=0.3088843596477481; total time=   0.
0s
[CV 1/10] END .....................alpha=0.40949150623804254; total time=   0.
1s
```

```
[CV 2/10] END .....................alpha=0.40949150623804254; total time=   0.
0s
[CV 3/10] END .....................alpha=0.40949150623804254; total time=   0.
0s
[CV 4/10] END .....................alpha=0.40949150623804254; total time=   0.
0s
[CV 5/10] END .....................alpha=0.40949150623804254; total time=   0.
0s
[CV 6/10] END .....................alpha=0.40949150623804254; total time=   0.
0s
[CV 7/10] END .....................alpha=0.40949150623804254; total time=   0.
1s
[CV 8/10] END .....................alpha=0.40949150623804254; total time=   0.
0s
[CV 9/10] END .....................alpha=0.40949150623804254; total time=   0.
0s
[CV 10/10] END ....................alpha=0.40949150623804254; total time=   0.
0s
[CV 1/10] END .....................alpha=0.5428675439323859; total time=   0.
0s
[CV 2/10] END .....................alpha=0.5428675439323859; total time=   0.
0s
[CV 3/10] END .....................alpha=0.5428675439323859; total time=   0.
1s
[CV 4/10] END .....................alpha=0.5428675439323859; total time=   0.
0s
[CV 5/10] END .....................alpha=0.5428675439323859; total time=   0.
0s
[CV 6/10] END .....................alpha=0.5428675439323859; total time=   0.
0s
[CV 7/10] END .....................alpha=0.5428675439323859; total time=   0.
0s
[CV 8/10] END .....................alpha=0.5428675439323859; total time=   0.
0s
[CV 9/10] END .....................alpha=0.5428675439323859; total time=   0.
1s
[CV 10/10] END ....................alpha=0.5428675439323859; total time=   0.
0s
[CV 1/10] END .....................alpha=0.7196856730011519; total time=   0.
0s
[CV 2/10] END .....................alpha=0.7196856730011519; total time=   0.
0s
[CV 3/10] END .....................alpha=0.7196856730011519; total time=   0.
0s
[CV 4/10] END .....................alpha=0.7196856730011519; total time=   0.
1s
[CV 5/10] END .....................alpha=0.7196856730011519; total time=   0.
0s
[CV 6/10] END .....................alpha=0.7196856730011519; total time=   0.
0s
[CV 7/10] END .....................alpha=0.7196856730011519; total time=   0.
0s
[CV 8/10] END .....................alpha=0.7196856730011519; total time=   0.
0s
[CV 9/10] END .....................alpha=0.7196856730011519; total time=   0.
1s
[CV 10/10] END ....................alpha=0.7196856730011519; total time=   0.
```

```
0s
[CV 1/10] END .....................alpha=0.9540954763499939; total time=   0.
0s
[CV 2/10] END .....................alpha=0.9540954763499939; total time=   0.
0s
[CV 3/10] END .....................alpha=0.9540954763499939; total time=   0.
1s
[CV 4/10] END .....................alpha=0.9540954763499939; total time=   0.
0s
[CV 5/10] END .....................alpha=0.9540954763499939; total time=   0.
0s
[CV 6/10] END .....................alpha=0.9540954763499939; total time=   0.
0s
[CV 7/10] END .....................alpha=0.9540954763499939; total time=   0.
0s
[CV 8/10] END .....................alpha=0.9540954763499939; total time=   0.
0s
[CV 9/10] END .....................alpha=0.9540954763499939; total time=   0.
1s
[CV 10/10] END .....................alpha=0.9540954763499939; total time=   0.
0s
[CV 1/10] END .....................alpha=1.2648552168552958; total time=   0.
0s
[CV 2/10] END .....................alpha=1.2648552168552958; total time=   0.
0s
[CV 3/10] END .....................alpha=1.2648552168552958; total time=   0.
0s
[CV 4/10] END .....................alpha=1.2648552168552958; total time=   0.
0s
[CV 5/10] END .....................alpha=1.2648552168552958; total time=   0.
1s
[CV 6/10] END .....................alpha=1.2648552168552958; total time=   0.
0s
[CV 7/10] END .....................alpha=1.2648552168552958; total time=   0.
0s
[CV 8/10] END .....................alpha=1.2648552168552958; total time=   0.
0s
[CV 9/10] END .....................alpha=1.2648552168552958; total time=   0.
0s
[CV 10/10] END .....................alpha=1.2648552168552958; total time=   0.
1s
[CV 1/10] END .....................alpha=1.6768329368110082; total time=   0.
1s
[CV 2/10] END .....................alpha=1.6768329368110082; total time=   0.
1s
[CV 3/10] END .....................alpha=1.6768329368110082; total time=   0.
1s
[CV 4/10] END .....................alpha=1.6768329368110082; total time=   0.
0s
[CV 5/10] END .....................alpha=1.6768329368110082; total time=   0.
0s
[CV 6/10] END .....................alpha=1.6768329368110082; total time=   0.
1s
[CV 7/10] END .....................alpha=1.6768329368110082; total time=   0.
1s
[CV 8/10] END .....................alpha=1.6768329368110082; total time=   0.
1s
```

```
[CV 9/10] END .....................alpha=1.6768329368110082; total time=   0.
0s
[CV 10/10] END .....................alpha=1.6768329368110082; total time=   0.
0s
[CV 1/10] END .....................alpha=2.2229964825261943; total time=   0.
0s
[CV 2/10] END .....................alpha=2.2229964825261943; total time=   0.
1s
[CV 3/10] END .....................alpha=2.2229964825261943; total time=   0.
0s
[CV 4/10] END .....................alpha=2.2229964825261943; total time=   0.
0s
[CV 5/10] END .....................alpha=2.2229964825261943; total time=   0.
0s
[CV 6/10] END .....................alpha=2.2229964825261943; total time=   0.
0s
[CV 7/10] END .....................alpha=2.2229964825261943; total time=   0.
0s
[CV 8/10] END .....................alpha=2.2229964825261943; total time=   0.
0s
[CV 9/10] END .....................alpha=2.2229964825261943; total time=   0.
0s
[CV 10/10] END .....................alpha=2.2229964825261943; total time=   0.
0s
[CV 1/10] END .....................alpha=2.9470517025518097; total time=   0.
1s
[CV 2/10] END .....................alpha=2.9470517025518097; total time=   0.
1s
[CV 3/10] END .....................alpha=2.9470517025518097; total time=   0.
1s
[CV 4/10] END .....................alpha=2.9470517025518097; total time=   0.
0s
[CV 5/10] END .....................alpha=2.9470517025518097; total time=   0.
0s
[CV 6/10] END .....................alpha=2.9470517025518097; total time=   0.
0s
[CV 7/10] END .....................alpha=2.9470517025518097; total time=   0.
1s
[CV 8/10] END .....................alpha=2.9470517025518097; total time=   0.
2s
[CV 9/10] END .....................alpha=2.9470517025518097; total time=   0.
0s
[CV 10/10] END .....................alpha=2.9470517025518097; total time=   0.
0s
[CV 1/10] END .....................alpha=3.906939937054617; total time=   0.
0s
[CV 2/10] END .....................alpha=3.906939937054617; total time=   0.
0s
[CV 3/10] END .....................alpha=3.906939937054617; total time=   0.
0s
[CV 4/10] END .....................alpha=3.906939937054617; total time=   0.
0s
[CV 5/10] END .....................alpha=3.906939937054617; total time=   0.
0s
[CV 6/10] END .....................alpha=3.906939937054617; total time=   0.
0s
[CV 7/10] END .....................alpha=3.906939937054617; total time=   0.
```

```
0s
[CV 8/10] END .....................alpha=3.906939937054617; total time=   0.
1s
[CV 9/10] END .....................alpha=3.906939937054617; total time=   0.
1s
[CV 10/10] END .....................alpha=3.906939937054617; total time=   0.
1s
[CV 1/10] END .....................alpha=5.17947467923121; total time=   0.
0s
[CV 2/10] END .....................alpha=5.17947467923121; total time=   0.
0s
[CV 3/10] END .....................alpha=5.17947467923121; total time=   0.
0s
[CV 4/10] END .....................alpha=5.17947467923121; total time=   0.
0s
[CV 5/10] END .....................alpha=5.17947467923121; total time=   0.
0s
[CV 6/10] END .....................alpha=5.17947467923121; total time=   0.
0s
[CV 7/10] END .....................alpha=5.17947467923121; total time=   0.
1s
[CV 8/10] END .....................alpha=5.17947467923121; total time=   0.
0s
[CV 9/10] END .....................alpha=5.17947467923121; total time=   0.
0s
[CV 10/10] END .....................alpha=5.17947467923121; total time=   0.
0s
[CV 1/10] END .....................alpha=6.8664884500430015; total time=   0.
0s
[CV 2/10] END .....................alpha=6.8664884500430015; total time=   0.
0s
[CV 3/10] END .....................alpha=6.8664884500430015; total time=   0.
1s
[CV 4/10] END .....................alpha=6.8664884500430015; total time=   0.
1s
[CV 5/10] END .....................alpha=6.8664884500430015; total time=   0.
1s
[CV 6/10] END .....................alpha=6.8664884500430015; total time=   0.
0s
[CV 7/10] END .....................alpha=6.8664884500430015; total time=   0.
0s
[CV 8/10] END .....................alpha=6.8664884500430015; total time=   0.
1s
[CV 9/10] END .....................alpha=6.8664884500430015; total time=   0.
0s
[CV 10/10] END .....................alpha=6.8664884500430015; total time=   0.
0s
[CV 1/10] END .....................alpha=9.102981779915218; total time=   0.
0s
[CV 2/10] END .....................alpha=9.102981779915218; total time=   0.
0s
[CV 3/10] END .....................alpha=9.102981779915218; total time=   0.
0s
[CV 4/10] END .....................alpha=9.102981779915218; total time=   0.
1s
[CV 5/10] END .....................alpha=9.102981779915218; total time=   0.
1s
```

```
[CV 6/10] END .....................alpha=9.102981779915218; total time=   0.
1s
[CV 7/10] END .....................alpha=9.102981779915218; total time=   0.
1s
[CV 8/10] END .....................alpha=9.102981779915218; total time=   0.
1s
[CV 9/10] END .....................alpha=9.102981779915218; total time=   0.
0s
[CV 10/10] END ....................alpha=9.102981779915218; total time=   0.
0s
[CV 1/10] END .....................alpha=12.067926406393289; total time=   0.
0s
[CV 2/10] END .....................alpha=12.067926406393289; total time=   0.
0s
[CV 3/10] END .....................alpha=12.067926406393289; total time=   0.
0s
[CV 4/10] END .....................alpha=12.067926406393289; total time=   0.
0s
[CV 5/10] END .....................alpha=12.067926406393289; total time=   0.
0s
[CV 6/10] END .....................alpha=12.067926406393289; total time=   0.
0s
[CV 7/10] END .....................alpha=12.067926406393289; total time=   0.
0s
[CV 8/10] END .....................alpha=12.067926406393289; total time=   0.
0s
[CV 9/10] END .....................alpha=12.067926406393289; total time=   0.
0s
[CV 10/10] END ....................alpha=12.067926406393289; total time=   0.
0s
[CV 1/10] END .....................alpha=15.998587196060573; total time=   0.
0s
[CV 2/10] END .....................alpha=15.998587196060573; total time=   0.
0s
[CV 3/10] END .....................alpha=15.998587196060573; total time=   0.
0s
[CV 4/10] END .....................alpha=15.998587196060573; total time=   0.
0s
[CV 5/10] END .....................alpha=15.998587196060573; total time=   0.
0s
[CV 6/10] END .....................alpha=15.998587196060573; total time=   0.
1s
[CV 7/10] END .....................alpha=15.998587196060573; total time=   0.
1s
[CV 8/10] END .....................alpha=15.998587196060573; total time=   0.
1s
[CV 9/10] END .....................alpha=15.998587196060573; total time=   0.
1s
[CV 10/10] END ....................alpha=15.998587196060573; total time=   0.
3s
[CV 1/10] END .....................alpha=21.209508879201906; total time=   0.
5s
[CV 2/10] END .....................alpha=21.209508879201906; total time=   0.
1s
[CV 3/10] END .....................alpha=21.209508879201906; total time=   0.
1s
[CV 4/10] END .....................alpha=21.209508879201906; total time=   0.
```

```
1s
[CV 5/10] END .....................alpha=21.209508879201906; total time=   0.
1s
[CV 6/10] END .....................alpha=21.209508879201906; total time=   0.
1s
[CV 7/10] END .....................alpha=21.209508879201906; total time=   0.
0s
[CV 8/10] END .....................alpha=21.209508879201906; total time=   0.
1s
[CV 9/10] END .....................alpha=21.209508879201906; total time=   0.
1s
[CV 10/10] END .....................alpha=21.209508879201906; total time=   0.
0s
[CV 1/10] END .....................alpha=28.11768697974231; total time=   0.
0s
[CV 2/10] END .....................alpha=28.11768697974231; total time=   0.
1s
[CV 3/10] END .....................alpha=28.11768697974231; total time=   0.
0s
[CV 4/10] END .....................alpha=28.11768697974231; total time=   0.
1s
[CV 5/10] END .....................alpha=28.11768697974231; total time=   0.
1s
[CV 6/10] END .....................alpha=28.11768697974231; total time=   0.
1s
[CV 7/10] END .....................alpha=28.11768697974231; total time=   0.
1s
[CV 8/10] END .....................alpha=28.11768697974231; total time=   0.
1s
[CV 9/10] END .....................alpha=28.11768697974231; total time=   0.
0s
[CV 10/10] END .....................alpha=28.11768697974231; total time=   0.
0s
[CV 1/10] END .....................alpha=37.27593720314938; total time=   0.
0s
[CV 2/10] END .....................alpha=37.27593720314938; total time=   0.
1s
[CV 3/10] END .....................alpha=37.27593720314938; total time=   0.
1s
[CV 4/10] END .....................alpha=37.27593720314938; total time=   0.
0s
[CV 5/10] END .....................alpha=37.27593720314938; total time=   0.
0s
[CV 6/10] END .....................alpha=37.27593720314938; total time=   0.
0s
[CV 7/10] END .....................alpha=37.27593720314938; total time=   0.
0s
[CV 8/10] END .....................alpha=37.27593720314938; total time=   0.
1s
[CV 9/10] END .....................alpha=37.27593720314938; total time=   0.
1s
[CV 10/10] END .....................alpha=37.27593720314938; total time=   0.
1s
[CV 1/10] END .....................alpha=49.417133613238335; total time=   0.
1s
[CV 2/10] END .....................alpha=49.417133613238335; total time=   0.
1s
```

```
[CV 3/10] END .....................alpha=49.417133613238335; total time=   0.
1s
[CV 4/10] END .....................alpha=49.417133613238335; total time=   0.
1s
[CV 5/10] END .....................alpha=49.417133613238335; total time=   0.
1s
[CV 6/10] END .....................alpha=49.417133613238335; total time=   0.
1s
[CV 7/10] END .....................alpha=49.417133613238335; total time=   0.
1s
[CV 8/10] END .....................alpha=49.417133613238335; total time=   0.
1s
[CV 9/10] END .....................alpha=49.417133613238335; total time=   0.
1s
[CV 10/10] END .....................alpha=49.417133613238335; total time=   0.
0s
[CV 1/10] END .....................alpha=65.51285568595509; total time=   0.
0s
[CV 2/10] END .....................alpha=65.51285568595509; total time=   0.
0s
[CV 3/10] END .....................alpha=65.51285568595509; total time=   0.
0s
[CV 4/10] END .....................alpha=65.51285568595509; total time=   0.
1s
[CV 5/10] END .....................alpha=65.51285568595509; total time=   0.
0s
[CV 6/10] END .....................alpha=65.51285568595509; total time=   0.
0s
[CV 7/10] END .....................alpha=65.51285568595509; total time=   0.
1s
[CV 8/10] END .....................alpha=65.51285568595509; total time=   0.
0s
[CV 9/10] END .....................alpha=65.51285568595509; total time=   0.
1s
[CV 10/10] END .....................alpha=65.51285568595509; total time=   0.
1s
[CV 1/10] END .....................alpha=86.85113737513521; total time=   0.
0s
[CV 2/10] END .....................alpha=86.85113737513521; total time=   0.
0s
[CV 3/10] END .....................alpha=86.85113737513521; total time=   0.
0s
[CV 4/10] END .....................alpha=86.85113737513521; total time=   0.
0s
[CV 5/10] END .....................alpha=86.85113737513521; total time=   0.
0s
[CV 6/10] END .....................alpha=86.85113737513521; total time=   0.
0s
[CV 7/10] END .....................alpha=86.85113737513521; total time=   0.
1s
[CV 8/10] END .....................alpha=86.85113737513521; total time=   0.
0s
[CV 9/10] END .....................alpha=86.85113737513521; total time=   0.
0s
[CV 10/10] END .....................alpha=86.85113737513521; total time=   0.
0s
[CV 1/10] END .....................alpha=115.1395399326447; total time=   0.
```

```
0s
[CV 2/10] END .....................alpha=115.1395399326447; total time=   0.
0s
[CV 3/10] END .....................alpha=115.1395399326447; total time=   0.
0s
[CV 4/10] END .....................alpha=115.1395399326447; total time=   0.
1s
[CV 5/10] END .....................alpha=115.1395399326447; total time=   0.
1s
[CV 6/10] END .....................alpha=115.1395399326447; total time=   0.
1s
[CV 7/10] END .....................alpha=115.1395399326447; total time=   0.
0s
[CV 8/10] END .....................alpha=115.1395399326447; total time=   0.
0s
[CV 9/10] END .....................alpha=115.1395399326447; total time=   0.
1s
[CV 10/10] END .....................alpha=115.1395399326447; total time=   0.
0s
[CV 1/10] END .....................alpha=152.64179671752333; total time=   0.
2s
[CV 2/10] END .....................alpha=152.64179671752333; total time=   0.
0s
[CV 3/10] END .....................alpha=152.64179671752333; total time=   0.
0s
[CV 4/10] END .....................alpha=152.64179671752333; total time=   0.
0s
[CV 5/10] END .....................alpha=152.64179671752333; total time=   0.
1s
[CV 6/10] END .....................alpha=152.64179671752333; total time=   0.
1s
[CV 7/10] END .....................alpha=152.64179671752333; total time=   0.
1s
[CV 8/10] END .....................alpha=152.64179671752333; total time=   0.
0s
[CV 9/10] END .....................alpha=152.64179671752333; total time=   0.
0s
[CV 10/10] END .....................alpha=152.64179671752333; total time=   0.
1s
[CV 1/10] END .....................alpha=202.35896477251575; total time=   0.
1s
[CV 2/10] END .....................alpha=202.35896477251575; total time=   0.
3s
[CV 3/10] END .....................alpha=202.35896477251575; total time=   0.
1s
[CV 4/10] END .....................alpha=202.35896477251575; total time=   0.
1s
[CV 5/10] END .....................alpha=202.35896477251575; total time=   0.
1s
[CV 6/10] END .....................alpha=202.35896477251575; total time=   0.
1s
[CV 7/10] END .....................alpha=202.35896477251575; total time=   0.
1s
[CV 8/10] END .....................alpha=202.35896477251575; total time=   0.
1s
[CV 9/10] END .....................alpha=202.35896477251575; total time=   0.
1s
```

```
[CV 10/10] END .....................alpha=202.35896477251575; total time=   0.
1s
[CV 1/10] END .....................alpha=268.26957952797244; total time=   0.
2s
[CV 2/10] END .....................alpha=268.26957952797244; total time=   0.
1s
[CV 3/10] END .....................alpha=268.26957952797244; total time=   0.
1s
[CV 4/10] END .....................alpha=268.26957952797244; total time=   0.
1s
[CV 5/10] END .....................alpha=268.26957952797244; total time=   0.
1s
[CV 6/10] END .....................alpha=268.26957952797244; total time=   0.
1s
[CV 7/10] END .....................alpha=268.26957952797244; total time=   0.
1s
[CV 8/10] END .....................alpha=268.26957952797244; total time=   0.
2s
[CV 9/10] END .....................alpha=268.26957952797244; total time=   0.
1s
[CV 10/10] END .....................alpha=268.26957952797244; total time=   0.
2s
[CV 1/10] END .....................alpha=355.64803062231283; total time=   0.
1s
[CV 2/10] END .....................alpha=355.64803062231283; total time=   0.
1s
[CV 3/10] END .....................alpha=355.64803062231283; total time=   0.
1s
[CV 4/10] END .....................alpha=355.64803062231283; total time=   0.
1s
[CV 5/10] END .....................alpha=355.64803062231283; total time=   0.
3s
[CV 6/10] END .....................alpha=355.64803062231283; total time=   0.
1s
[CV 7/10] END .....................alpha=355.64803062231283; total time=   0.
2s
[CV 8/10] END .....................alpha=355.64803062231283; total time=   0.
1s
[CV 9/10] END .....................alpha=355.64803062231283; total time=   0.
1s
[CV 10/10] END .....................alpha=355.64803062231283; total time=   0.
0s
[CV 1/10] END .....................alpha=471.48663634573944; total time=   0.
1s
[CV 2/10] END .....................alpha=471.48663634573944; total time=   0.
1s
[CV 3/10] END .....................alpha=471.48663634573944; total time=   0.
0s
[CV 4/10] END .....................alpha=471.48663634573944; total time=   0.
1s
[CV 5/10] END .....................alpha=471.48663634573944; total time=   0.
3s
[CV 6/10] END .....................alpha=471.48663634573944; total time=   0.
1s
[CV 7/10] END .....................alpha=471.48663634573944; total time=   0.
1s
[CV 8/10] END .....................alpha=471.48663634573944; total time=   0.
```

```
0s
[CV 9/10] END .....................alpha=471.48663634573944; total time=   0.
0s
[CV 10/10] END .....................alpha=471.48663634573944; total time=   0.
1s
[CV 1/10] END .....................alpha=625.0551925273969; total time=   0.
1s
[CV 2/10] END .....................alpha=625.0551925273969; total time=   0.
1s
[CV 3/10] END .....................alpha=625.0551925273969; total time=   0.
1s
[CV 4/10] END .....................alpha=625.0551925273969; total time=   0.
1s
[CV 5/10] END .....................alpha=625.0551925273969; total time=   0.
0s
[CV 6/10] END .....................alpha=625.0551925273969; total time=   0.
0s
[CV 7/10] END .....................alpha=625.0551925273969; total time=   0.
0s
[CV 8/10] END .....................alpha=625.0551925273969; total time=   0.
0s
[CV 9/10] END .....................alpha=625.0551925273969; total time=   0.
1s
[CV 10/10] END .....................alpha=625.0551925273969; total time=   0.
0s
[CV 1/10] END .....................alpha=828.6427728546843; total time=   0.
1s
[CV 2/10] END .....................alpha=828.6427728546843; total time=   0.
1s
[CV 3/10] END .....................alpha=828.6427728546843; total time=   0.
1s
[CV 4/10] END .....................alpha=828.6427728546843; total time=   0.
0s
[CV 5/10] END .....................alpha=828.6427728546843; total time=   0.
0s
[CV 6/10] END .....................alpha=828.6427728546843; total time=   0.
0s
[CV 7/10] END .....................alpha=828.6427728546843; total time=   0.
0s
[CV 8/10] END .....................alpha=828.6427728546843; total time=   0.
0s
[CV 9/10] END .....................alpha=828.6427728546843; total time=   0.
0s
[CV 10/10] END .....................alpha=828.6427728546843; total time=   0.
0s
[CV 1/10] END .....................alpha=1098.5411419875572; total time=   0.
0s
[CV 2/10] END .....................alpha=1098.5411419875572; total time=   0.
0s
[CV 3/10] END .....................alpha=1098.5411419875572; total time=   0.
0s
[CV 4/10] END .....................alpha=1098.5411419875572; total time=   0.
0s
[CV 5/10] END .....................alpha=1098.5411419875572; total time=   0.
1s
[CV 6/10] END .....................alpha=1098.5411419875572; total time=   0.
2s
```

```
[CV 7/10] END .....................alpha=1098.5411419875572; total time=   0.
1s
[CV 8/10] END .....................alpha=1098.5411419875572; total time=   0.
1s
[CV 9/10] END .....................alpha=1098.5411419875572; total time=   0.
1s
[CV 10/10] END .....................alpha=1098.5411419875572; total time=   0.
0s
[CV 1/10] END .....................alpha=1456.3484775012444; total time=   0.
0s
[CV 2/10] END .....................alpha=1456.3484775012444; total time=   0.
0s
[CV 3/10] END .....................alpha=1456.3484775012444; total time=   0.
0s
[CV 4/10] END .....................alpha=1456.3484775012444; total time=   0.
0s
[CV 5/10] END .....................alpha=1456.3484775012444; total time=   0.
0s
[CV 6/10] END .....................alpha=1456.3484775012444; total time=   0.
0s
[CV 7/10] END .....................alpha=1456.3484775012444; total time=   0.
0s
[CV 8/10] END .....................alpha=1456.3484775012444; total time=   0.
0s
[CV 9/10] END .....................alpha=1456.3484775012444; total time=   0.
0s
[CV 10/10] END .....................alpha=1456.3484775012444; total time=   0.
1s
[CV 1/10] END .....................alpha=1930.6977288832495; total time=   0.
0s
[CV 2/10] END .....................alpha=1930.6977288832495; total time=   0.
0s
[CV 3/10] END .....................alpha=1930.6977288832495; total time=   0.
0s
[CV 4/10] END .....................alpha=1930.6977288832495; total time=   0.
0s
[CV 5/10] END .....................alpha=1930.6977288832495; total time=   0.
0s
[CV 6/10] END .....................alpha=1930.6977288832495; total time=   0.
0s
[CV 7/10] END .....................alpha=1930.6977288832495; total time=   0.
0s
[CV 8/10] END .....................alpha=1930.6977288832495; total time=   0.
0s
[CV 9/10] END .....................alpha=1930.6977288832495; total time=   0.
0s
[CV 10/10] END .....................alpha=1930.6977288832495; total time=   0.
1s
[CV 1/10] END .....................alpha=2559.547922699533; total time=   0.
1s
[CV 2/10] END .....................alpha=2559.547922699533; total time=   0.
1s
[CV 3/10] END .....................alpha=2559.547922699533; total time=   0.
2s
[CV 4/10] END .....................alpha=2559.547922699533; total time=   0.
2s
[CV 5/10] END .....................alpha=2559.547922699533; total time=   0.
```

```
1s
[CV 6/10] END .....................alpha=2559.547922699533; total time=   0.
0s
[CV 7/10] END .....................alpha=2559.547922699533; total time=   0.
0s
[CV 8/10] END .....................alpha=2559.547922699533; total time=   0.
1s
[CV 9/10] END .....................alpha=2559.547922699533; total time=   0.
1s
[CV 10/10] END .....................alpha=2559.547922699533; total time=   0.
1s
[CV 1/10] END .....................alpha=3393.2217718953298; total time=   0.
1s
[CV 2/10] END .....................alpha=3393.2217718953298; total time=   0.
2s
[CV 3/10] END .....................alpha=3393.2217718953298; total time=   0.
2s
[CV 4/10] END .....................alpha=3393.2217718953298; total time=   0.
1s
[CV 5/10] END .....................alpha=3393.2217718953298; total time=   0.
1s
[CV 6/10] END .....................alpha=3393.2217718953298; total time=   0.
1s
[CV 7/10] END .....................alpha=3393.2217718953298; total time=   0.
1s
[CV 8/10] END .....................alpha=3393.2217718953298; total time=   0.
2s
[CV 9/10] END .....................alpha=3393.2217718953298; total time=   0.
1s
[CV 10/10] END .....................alpha=3393.2217718953298; total time=   0.
2s
[CV 1/10] END .....................alpha=4498.432668969444; total time=   0.
1s
[CV 2/10] END .....................alpha=4498.432668969444; total time=   0.
1s
[CV 3/10] END .....................alpha=4498.432668969444; total time=   0.
1s
[CV 4/10] END .....................alpha=4498.432668969444; total time=   0.
1s
[CV 5/10] END .....................alpha=4498.432668969444; total time=   0.
1s
[CV 6/10] END .....................alpha=4498.432668969444; total time=   0.
1s
[CV 7/10] END .....................alpha=4498.432668969444; total time=   0.
0s
[CV 8/10] END .....................alpha=4498.432668969444; total time=   0.
1s
[CV 9/10] END .....................alpha=4498.432668969444; total time=   0.
2s
[CV 10/10] END .....................alpha=4498.432668969444; total time=   0.
1s
[CV 1/10] END .....................alpha=5963.623316594637; total time=   0.
0s
[CV 2/10] END .....................alpha=5963.623316594637; total time=   0.
0s
[CV 3/10] END .....................alpha=5963.623316594637; total time=   0.
0s
```

```
[CV 4/10] END .....................alpha=5963.623316594637; total time=   0.
0s
[CV 5/10] END .....................alpha=5963.623316594637; total time=   0.
0s
[CV 6/10] END .....................alpha=5963.623316594637; total time=   0.
0s
[CV 7/10] END .....................alpha=5963.623316594637; total time=   0.
0s
[CV 8/10] END .....................alpha=5963.623316594637; total time=   0.
0s
[CV 9/10] END .....................alpha=5963.623316594637; total time=   0.
1s
[CV 10/10] END .....................alpha=5963.623316594637; total time=   0.
0s
[CV 1/10] END .....................alpha=7906.043210907702; total time=   0.
0s
[CV 2/10] END .....................alpha=7906.043210907702; total time=   0.
1s
[CV 3/10] END .....................alpha=7906.043210907702; total time=   0.
0s
[CV 4/10] END .....................alpha=7906.043210907702; total time=   0.
1s
[CV 5/10] END .....................alpha=7906.043210907702; total time=   0.
0s
[CV 6/10] END .....................alpha=7906.043210907702; total time=   0.
0s
[CV 7/10] END .....................alpha=7906.043210907702; total time=   0.
0s
[CV 8/10] END .....................alpha=7906.043210907702; total time=   0.
0s
[CV 9/10] END .....................alpha=7906.043210907702; total time=   0.
0s
[CV 10/10] END .....................alpha=7906.043210907702; total time=   0.
0s
[CV 1/10] END .....................alpha=10481.131341546852; total time=   0.
0s
[CV 2/10] END .....................alpha=10481.131341546852; total time=   0.
0s
[CV 3/10] END .....................alpha=10481.131341546852; total time=   0.
1s
[CV 4/10] END .....................alpha=10481.131341546852; total time=   0.
1s
[CV 5/10] END .....................alpha=10481.131341546852; total time=   0.
1s
[CV 6/10] END .....................alpha=10481.131341546852; total time=   0.
1s
[CV 7/10] END .....................alpha=10481.131341546852; total time=   0.
1s
[CV 8/10] END .....................alpha=10481.131341546852; total time=   0.
1s
[CV 9/10] END .....................alpha=10481.131341546852; total time=   0.
1s
[CV 10/10] END .....................alpha=10481.131341546852; total time=   0.
1s
[CV 1/10] END .....................alpha=13894.95494373136; total time=   0.
1s
[CV 2/10] END .....................alpha=13894.95494373136; total time=   0.
```

```
1s
[CV 3/10] END .....................alpha=13894.95494373136; total time=   0.
1s
[CV 4/10] END .....................alpha=13894.95494373136; total time=   0.
1s
[CV 5/10] END .....................alpha=13894.95494373136; total time=   0.
2s
[CV 6/10] END .....................alpha=13894.95494373136; total time=   0.
1s
[CV 7/10] END .....................alpha=13894.95494373136; total time=   0.
1s
[CV 8/10] END .....................alpha=13894.95494373136; total time=   0.
2s
[CV 9/10] END .....................alpha=13894.95494373136; total time=   0.
1s
[CV 10/10] END .....................alpha=13894.95494373136; total time=   0.
1s
[CV 1/10] END .....................alpha=18420.699693267165; total time=   0.
1s
[CV 2/10] END .....................alpha=18420.699693267165; total time=   0.
3s
[CV 3/10] END .....................alpha=18420.699693267165; total time=   0.
2s
[CV 4/10] END .....................alpha=18420.699693267165; total time=   0.
1s
[CV 5/10] END .....................alpha=18420.699693267165; total time=   0.
1s
[CV 6/10] END .....................alpha=18420.699693267165; total time=   0.
1s
[CV 7/10] END .....................alpha=18420.699693267165; total time=   0.
2s
[CV 8/10] END .....................alpha=18420.699693267165; total time=   0.
1s
[CV 9/10] END .....................alpha=18420.699693267165; total time=   0.
1s
[CV 10/10] END .....................alpha=18420.699693267165; total time=   0.
1s
[CV 1/10] END .....................alpha=24420.5309454865; total time=   0.
0s
[CV 2/10] END .....................alpha=24420.5309454865; total time=   0.
0s
[CV 3/10] END .....................alpha=24420.5309454865; total time=   0.
0s
[CV 4/10] END .....................alpha=24420.5309454865; total time=   0.
0s
[CV 5/10] END .....................alpha=24420.5309454865; total time=   0.
1s
[CV 6/10] END .....................alpha=24420.5309454865; total time=   0.
1s
[CV 7/10] END .....................alpha=24420.5309454865; total time=   0.
1s
[CV 8/10] END .....................alpha=24420.5309454865; total time=   0.
0s
[CV 9/10] END .....................alpha=24420.5309454865; total time=   0.
0s
[CV 10/10] END .....................alpha=24420.5309454865; total time=   0.
0s
```

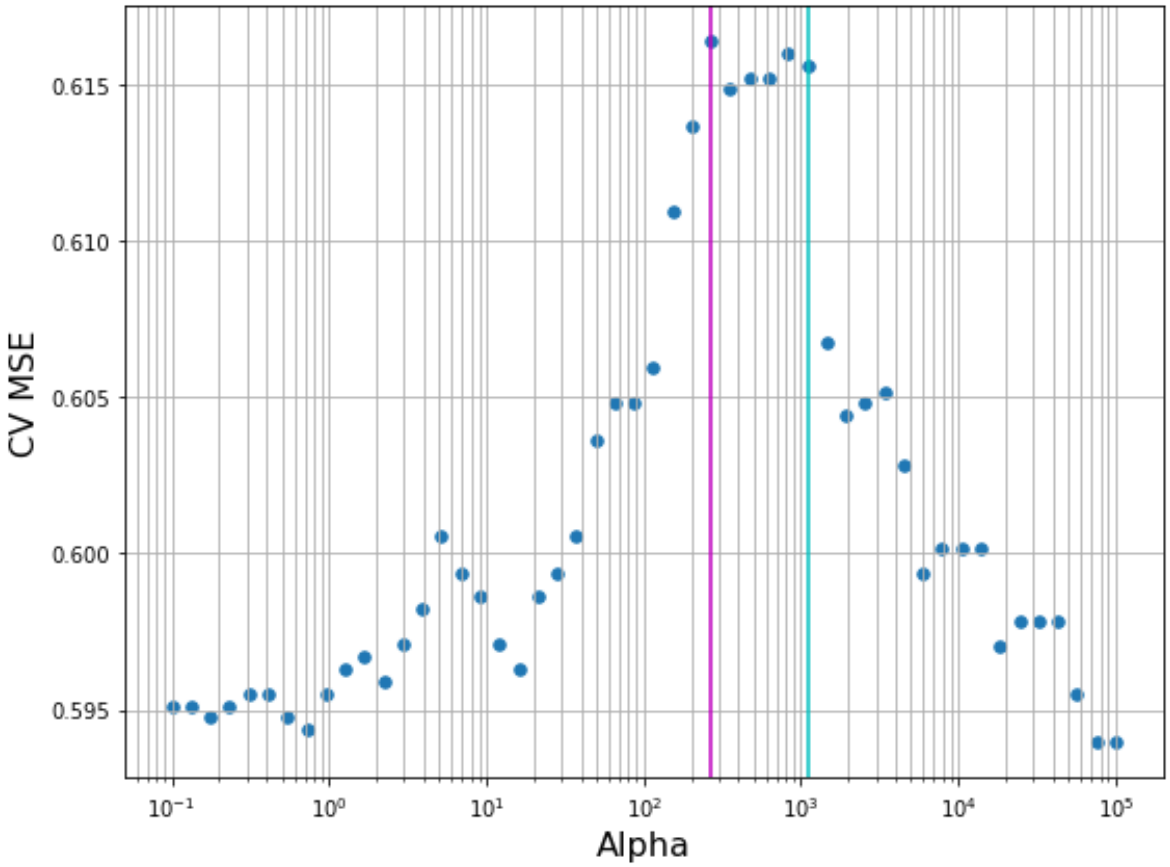
```
[CV 1/10] END .....................alpha=32374.5754281764; total time=   0.
0s
[CV 2/10] END .....................alpha=32374.5754281764; total time=   0.
0s
[CV 3/10] END .....................alpha=32374.5754281764; total time=   0.
0s
[CV 4/10] END .....................alpha=32374.5754281764; total time=   0.
0s
[CV 5/10] END .....................alpha=32374.5754281764; total time=   0.
0s
[CV 6/10] END .....................alpha=32374.5754281764; total time=   0.
0s
[CV 7/10] END .....................alpha=32374.5754281764; total time=   0.
1s
[CV 8/10] END .....................alpha=32374.5754281764; total time=   0.
1s
[CV 9/10] END .....................alpha=32374.5754281764; total time=   0.
1s
[CV 10/10] END .....................alpha=32374.5754281764; total time=   0.
0s
[CV 1/10] END .....................alpha=42919.34260128778; total time=   0.
0s
[CV 2/10] END .....................alpha=42919.34260128778; total time=   0.
0s
[CV 3/10] END .....................alpha=42919.34260128778; total time=   0.
0s
[CV 4/10] END .....................alpha=42919.34260128778; total time=   0.
0s
[CV 5/10] END .....................alpha=42919.34260128778; total time=   0.
0s
[CV 6/10] END .....................alpha=42919.34260128778; total time=   0.
0s
[CV 7/10] END .....................alpha=42919.34260128778; total time=   0.
0s
[CV 8/10] END .....................alpha=42919.34260128778; total time=   0.
0s
[CV 9/10] END .....................alpha=42919.34260128778; total time=   0.
0s
[CV 10/10] END .....................alpha=42919.34260128778; total time=   0.
0s
[CV 1/10] END .....................alpha=56898.66029018293; total time=   0.
1s
[CV 2/10] END .....................alpha=56898.66029018293; total time=   0.
1s
[CV 3/10] END .....................alpha=56898.66029018293; total time=   0.
0s
[CV 4/10] END .....................alpha=56898.66029018293; total time=   0.
0s
[CV 5/10] END .....................alpha=56898.66029018293; total time=   0.
0s
[CV 6/10] END .....................alpha=56898.66029018293; total time=   0.
1s
[CV 7/10] END .....................alpha=56898.66029018293; total time=   0.
0s
[CV 8/10] END .....................alpha=56898.66029018293; total time=   0.
0s
[CV 9/10] END .....................alpha=56898.66029018293; total time=   0.
```

```
0s
[CV 10/10] END .....................alpha=56898.66029018293; total time=   0.
1s
[CV 1/10] END ......................alpha=75431.20063354608; total time=   0.
1s
[CV 2/10] END ......................alpha=75431.20063354608; total time=   0.
1s
[CV 3/10] END ......................alpha=75431.20063354608; total time=   0.
1s
[CV 4/10] END ......................alpha=75431.20063354608; total time=   0.
0s
[CV 5/10] END ......................alpha=75431.20063354608; total time=   0.
0s
[CV 6/10] END ......................alpha=75431.20063354608; total time=   0.
0s
[CV 7/10] END ......................alpha=75431.20063354608; total time=   0.
0s
[CV 8/10] END ......................alpha=75431.20063354608; total time=   0.
1s
[CV 9/10] END ......................alpha=75431.20063354608; total time=   0.
2s
[CV 10/10] END .....................alpha=75431.20063354608; total time=   0.
1s
[CV 1/10] END ..............................alpha=100000.0; total time=   0.
1s
[CV 2/10] END ..............................alpha=100000.0; total time=   0.
1s
[CV 3/10] END ..............................alpha=100000.0; total time=   0.
1s
[CV 4/10] END ..............................alpha=100000.0; total time=   0.
1s
[CV 5/10] END ..............................alpha=100000.0; total time=   0.
1s
[CV 6/10] END ..............................alpha=100000.0; total time=   0.
1s
[CV 7/10] END ..............................alpha=100000.0; total time=   0.
1s
[CV 8/10] END ..............................alpha=100000.0; total time=   0.
0s
[CV 9/10] END ..............................alpha=100000.0; total time=   0.
0s
[CV 10/10] END .............................alpha=100000.0; total time=   0.
0s
```

Out[16]:
```
GridSearchCV(cv=10, estimator=RidgeClassifier(random_state=88),
             param_grid={'alpha': array([1.00000000e-01, 1.32571137e-01, 1.757
51062e-01, 2.32995181e-01,
       3.08884360e-01, 4.09491506e-01, 5.42867544e-01, 7.19685673e-01,
       9.54095476e-01, 1.26485522e+00, 1.67683294e+00, 2.22299648e+00,
       2.94705170e+00, 3.90693994e+00, 5.17947468e+00, 6.86648845e+00,
       9.10298178e+00, 1.2067...
       2.68269580e+02, 3.55648031e+02, 4.71486636e+02, 6.25055193e+02,
       8.28642773e+02, 1.09854114e+03, 1.45634848e+03, 1.93069773e+03,
       2.55954792e+03, 3.39322177e+03, 4.49843267e+03, 5.96362332e+03,
       7.90604321e+03, 1.04811313e+04, 1.38949549e+04, 1.84206997e+04,
       2.44205309e+04, 3.23745754e+04, 4.29193426e+04, 5.68986603e+04,
       7.54312006e+04, 1.00000000e+05])},
             scoring='accuracy', verbose=4)
```

In [17]:
```
def one_standard_error_rule(model, results, param_grid, n_splits):

    #assert neg_mean_squared_error == True # function is defined specifically

    #find model with minimum error, then select the simplest model
    #whose mean falls within 1 standard deviation of the minimum

    range_x = param_grid # results['param_'+list(param_grid.keys())[0]].data
    std_vs_x  = pd.Series(results['std_test_score'], index = range_x)
    sem_vs_x  = std_vs_x/np.sqrt(n_splits)

    mean_vs_x = pd.Series(results['mean_test_score'], index = range_x)
    mean_vs_x = mean_vs_x*(-1)

    x_min = mean_vs_x.idxmin()
    sem = sem_vs_x[x_min]


    if (model=='pcr'):
        x_1se = mean_vs_x[mean_vs_x <= min(mean_vs_x) + sem].index.min()
    elif (model=='ridge') | (model=='lasso'):
        x_1se = mean_vs_x[mean_vs_x <= min(mean_vs_x) + sem].index.max()

    #x_1se_idx = int(np.argwhere(range_x == x_1se)[0])

    return x_min, x_1se
```

In [18]:
```
range_alpha = rr_cv.cv_results_['param_alpha'].data
accuracy_scores = rr_cv.cv_results_['mean_test_score']
x_min, x_1se = one_standard_error_rule(model='ridge',
                                       results=rr_cv.cv_results_,
                                       param_grid=range_alpha,
                                       n_splits=10,
                                       )
plt.figure(figsize=(8, 6))
ax = plt.gca()
ax.set_xscale('log')
plt.xlabel('Alpha', fontsize=16)
plt.ylabel('CV MSE', fontsize=16)
plt.scatter(range_alpha, accuracy_scores, s=30)
plt.axvline(x=x_min, color='m')
plt.axvline(x=x_1se, color='c')
plt.grid(True, which='both')

plt.tight_layout()
plt.show()
```



In [19]:
```
print('Alpha one standard error rule:', x_1se)
```

```
Alpha one standard error rule: 1098.5411419875572
```

In [20]:
```
rr_cv = GridSearchCV(rr, {'alpha': [x_1se]}, scoring='accuracy', cv=10)
rr_cv.fit(xd, y_train)
```

Out[20]:
```
GridSearchCV(cv=10, estimator=RidgeClassifier(random_state=88),
             param_grid={'alpha': [1098.5411419875572]}, scoring='accuracy')
```

In [21]:
```
y_test = data_test['cumulative_outcome']
y_test = [1 if x=='Yes' else 0 for x in y_test]
x_test = data_test[data_test.columns[1:]]
x_test = x_test.drop(columns = ['cumulative_outcome'],axis=1)
xd_test = pd.get_dummies(x_test)

#treatment of missing data: we put the most common observation in place of th
for column in xd_test.columns:
    xd_test[column]=xd_test[column].fillna(xd_test[column].mode()[0])
```

In [22]:
```
from sklearn.metrics import accuracy_score


print('Accuracy:', round(accuracy_score(y_test, rr_cv.predict(xd_test)), 5))
```

```
Accuracy: 0.61926
```

In [23]:
```
from sklearn.metrics import roc_curve, auc

fpr, tpr, _ = roc_curve(y_test, rr_cv.predict(xd_test))

print(fpr,tpr)
```

```
[0.         0.18449612 1.        ] [0.         0.34763948 1.        ]
```

In [39]:
```
def create_polynomial_features(df, n_degree):

    new_df = None

    for i in range(2, n_degree+1):

        tmp = df.pow(i)

        affix = '_pow_'+str(i)
        tmp.columns = list(map(lambda x: x + affix, df.columns))

        if new_df is not None:
            new_df = pd.concat([new_df, tmp], axis=1)
        else:
            new_df = tmp

    return new_df
```

In [40]:
```
n_degree = 3

X_train_poly = pd.concat([xd, create_polynomial_features(xd[xd.columns], n_de
X_test_poly = pd.concat([xd_test, create_polynomial_features(xd[xd.columns],


print(X_train_poly.shape, X_test_poly.shape)
```

```
(2591, 984) (3702, 984)
```

In [ ]:

In [ ]:

In [ ]:

In [41]:
```
from sklearn.linear_model import LogisticRegression

LogReg = LogisticRegression(penalty='l2')
```

In [42]:
```
LogReg.fit(xd, y_train)
```

```
/Users/william/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/
_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
```

Out[42]:
```
LogisticRegression()
```

In [43]:
```
print('Accuracy:', round(accuracy_score(y_test, LogReg.predict(xd_test)), 5))
```

```
Accuracy: 0.58596
```

In [44]:
```
rr_cv.fit(X_train_poly , y_train)
```

```
/Users/william/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/
_ridge.py:147: LinAlgWarning: Ill-conditioned matrix (rcond=1.21138e-19): resu
lt may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True,
/Users/william/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/
_ridge.py:147: LinAlgWarning: Ill-conditioned matrix (rcond=1.19124e-19): resu
lt may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True,
/Users/william/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/
_ridge.py:147: LinAlgWarning: Ill-conditioned matrix (rcond=1.95134e-19): resu
lt may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True,
/Users/william/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/
_ridge.py:147: LinAlgWarning: Ill-conditioned matrix (rcond=1.20616e-19): resu
lt may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True,
/Users/william/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/
_ridge.py:147: LinAlgWarning: Ill-conditioned matrix (rcond=1.25064e-19): resu
lt may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True,
/Users/william/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/
_ridge.py:147: LinAlgWarning: Ill-conditioned matrix (rcond=1.19266e-19): resu
lt may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True,
/Users/william/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/
_ridge.py:147: LinAlgWarning: Ill-conditioned matrix (rcond=1.20408e-19): resu
lt may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True,
/Users/william/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/
_ridge.py:147: LinAlgWarning: Ill-conditioned matrix (rcond=1.2072e-19): resul
t may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True,
/Users/william/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/
_ridge.py:147: LinAlgWarning: Ill-conditioned matrix (rcond=1.28986e-19): resu
lt may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True,
/Users/william/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/
_ridge.py:147: LinAlgWarning: Ill-conditioned matrix (rcond=1.26559e-19): resu
lt may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True,
/Users/william/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/
_ridge.py:147: LinAlgWarning: Ill-conditioned matrix (rcond=1.14262e-19): resu
lt may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True,
```

Out[44]:
```
GridSearchCV(cv=10, estimator=RidgeClassifier(random_state=88),
             param_grid={'alpha': [1098.5411419875572]}, scoring='accuracy')
```

In [ ]:

Project_242_Clustering

December 17, 2021

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
from sklearn.model_selection import train_test_split
from google.colab import drive

drive.mount('/content/drive')

df = pd.read_csv('/content/drive/MyDrive/Osteoarthritis dataset/All Clinical/
 ↪cleaned_data.csv')

df.head()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:

	ID	P01KPNREV	P01KPNLEV	…	V00LLWGT	V00RLWGT	cumulative_outcome
0	9000296	Yes	No	…	12.0	18.0	No
1	9000622	Yes	No	…	14.0	14.0	Yes
2	9001695	Yes	No	…	13.0	13.0	Yes
3	9001897	No	No	…	16.0	17.0	Yes
4	9002411	Yes	No	…	18.0	22.0	No

[5 rows x 122 columns]

1 Pre-processing

[ ]:
```
## Replace 'No' and 'Yes' by 0 and 1 in the outcome column

df['cumulative_outcome'] = df['cumulative_outcome'].apply(lambda x: 1 if x ==␣
 ↪'Yes' else 0)
df.head()
```

[ ]:

	ID	P01KPNREV	P01KPNLEV	…	V00LLWGT	V00RLWGT	cumulative_outcome
0	9000296	Yes	No	…	12.0	18.0	0
1	9000622	Yes	No	…	14.0	14.0	1

```
2  9001695       Yes        No  …     13.0     13.0                  1
3  9001897        No        No  …     16.0     17.0                  1
4  9002411       Yes        No  …     18.0     22.0                  0

[5 rows x 122 columns]
```

We drop the useless columns and we transform the columns with string into dummy variables.

[ ]:
```
## Drop first column (ID of the patient)

df = df.drop('ID', axis='columns')


## Check columns to convert to dummies

columns_dummies = []

for col in df.columns:
  if df[col].dtype != 'float64' and df[col].dtype != 'int64':
    columns_dummies.append(col)

print(f'Number of columns to dummy encode = {len(columns_dummies)} out of␣
 ↪{len(df.columns)}')


df_enc = pd.get_dummies(df, columns = columns_dummies, drop_first = True)
```

```
Number of columns to dummy encode = 103 out of 121
```

[ ]:
```
df_enc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3702 entries, 0 to 3701
Columns: 228 entries, V00WOMTSL to V00LKRFXPN_Yes
dtypes: float64(17), int64(1), uint8(210)
memory usage: 1.2 MB
```

[ ]:
```
df_enc.head()
```

[ ]:
```
   V00WOMTSL  V00WOMTSR  …  V00RKRFXPN_Yes  V00LKRFXPN_Yes
0        0.0        0.0  …               0               0
1        0.0       20.9  …               1               0
2        0.0        NaN  …               0               0
3        0.0        0.0  …               0               0
4        1.1        2.1  …               0               0

[5 rows x 228 columns]
```

Now, we are looking for Nan values in the data set, and we are going to replace them by the mean

of the variable.

[ ]:
```
# Search for the columns with Nan values

columns_with_nan_values = []
for col in df_enc.columns:
  if df_enc[col].isnull().values.any() == True:
    columns_with_nan_values.append(col)

print(columns_with_nan_values)
```

```
['V00WOMTSL', 'V00WOMTSR', 'P01KPACDCV', 'V00COMORB', 'V00HT25MM', 'V00WT25KG',
'V00WTMAXKG', 'V00WTMINKG', 'V00BPDIAS', 'V00BPSYS', 'V00DTVITD', 'V00DTCALC',
'V00DTCHOL', 'P01BMI', 'V00PASE', 'V00LLWGT', 'V00RLWGT']
```

[ ]:
```
# Replacement of the Nan values by the mean of the column

for col in columns_with_nan_values:
  col_mean = df[col].describe()['mean']
  df_enc[col] = df[col].fillna(col_mean)

df_enc.head()
```

[ ]:

	V00WOMTSL	V00WOMTSR	…	V00RKRFXPN_Yes	V00LKRFXPN_Yes
0	0.0	0.000000	…	0	0
1	0.0	20.900000	…	1	0
2	0.0	10.076808	…	0	0
3	0.0	0.000000	…	0	0
4	1.1	2.100000	…	0	0

[5 rows x 228 columns]

[ ]:
```
## Check if there are still NaN values in data frame
df_enc.isnull().values.any()
```

[ ]: False

[ ]:
```
## Rename columns to have format usable with the Stats Model Formula input

print('Before = ', df_enc.columns)

# Replace all spaces and symbols by underscores

df_enc.columns = df_enc.columns.str.replace(' ', '_')
df_enc.columns = df_enc.columns.str.replace(',', '_')
df_enc.columns = df_enc.columns.str.replace(':', '_')
df_enc.columns = df_enc.columns.str.replace('/', '_')
df_enc.columns = df_enc.columns.str.replace('-', '_')
```

```
print('After = ', df_enc.columns)
```

```
Before =  Index(['V00WOMTSL', 'V00WOMTSR', 'P01KPACDCV', 'V00COMORB',
'V00HT25MM',
       'V00WT25KG', 'V00WTMAXKG', 'V00WTMINKG', 'V00BPDIAS', 'V00BPSYS',
       …
       'P01FAMKR_Yes', 'P02WTGA_Yes', 'V00RKEFFB_Yes',
       'V00RKEFFPT_Too tender to examine', 'V00RKEFFPT_Yes', 'V00LKEFFB_Yes',
       'V00LKEFFPT_Too tender to examine', 'V00LKEFFPT_Yes', 'V00RKRFXPN_Yes',
       'V00LKRFXPN_Yes'],
      dtype='object', length=228)
After =  Index(['V00WOMTSL', 'V00WOMTSR', 'P01KPACDCV', 'V00COMORB',
'V00HT25MM',
       'V00WT25KG', 'V00WTMAXKG', 'V00WTMINKG', 'V00BPDIAS', 'V00BPSYS',
       …
       'P01FAMKR_Yes', 'P02WTGA_Yes', 'V00RKEFFB_Yes',
       'V00RKEFFPT_Too_tender_to_examine', 'V00RKEFFPT_Yes', 'V00LKEFFB_Yes',
       'V00LKEFFPT_Too_tender_to_examine', 'V00LKEFFPT_Yes', 'V00RKRFXPN_Yes',
       'V00LKRFXPN_Yes'],
      dtype='object', length=228)
```

[ ]:
```
df_enc.describe()
```

[ ]:

	V00WOMTSL	V00WOMTSR	…	V00RKRFXPN_Yes	V00LKRFXPN_Yes
count	3702.000000	3702.000000	…	3702.000000	3702.000000
mean	9.774844	10.076808	…	0.135062	0.148568
std	14.058002	13.081182	…	0.341836	0.355711
min	0.000000	0.000000	…	0.000000	0.000000
25%	0.000000	0.000000	…	0.000000	0.000000
50%	3.000000	5.000000	…	0.000000	0.000000
75%	14.000000	15.000000	…	0.000000	0.000000
max	96.000000	93.900000	…	1.000000	1.000000

[8 rows x 228 columns]

We normalized each column before performing the clustering, so each variable has the same weight when computing the clusters.
First, center the data (substract the mean to each column) => mean becomes 0 for each column

Then, scale the data, by dividing by the standard deviation => std becomes 1 for each column

[ ]:
```
from sklearn import preprocessing

scaler = preprocessing.StandardScaler()
df_enc_features = df_enc.drop('cumulative_outcome', axis='columns')
df_scaled = scaler.fit_transform(df_enc_features)

df_scaled = pd.DataFrame(df_scaled, columns=df_enc_features.columns)
```

```
df_scaled.describe()
```

[ ]:

	V00WOMTSL	V00WOMTSR	…	V00RKRFXPN_Yes	V00LKRFXPN_Yes
count	3.702000e+03	3.702000e+03	…	3.702000e+03	3.702000e+03
mean	7.462966e-17	2.181909e-16	…	2.495453e-16	1.616751e-16
std	1.000135e+00	1.000135e+00	…	1.000135e+00	1.000135e+00
min	-6.954164e-01	-7.704327e-01	…	-3.951612e-01	-4.177229e-01
25%	-6.954164e-01	-7.704327e-01	…	-3.951612e-01	-4.177229e-01
50%	-4.819859e-01	-3.881526e-01	…	-3.951612e-01	-4.177229e-01
75%	3.005923e-01	3.764076e-01	…	-3.951612e-01	-4.177229e-01
max	6.134357e+00	6.408788e+00	…	2.530613e+00	2.393932e+00

[8 rows x 227 columns]

[ ]:
```
df_normalized = preprocessing.normalize(df_scaled)
df_normalized = pd.DataFrame(df_normalized, columns=df_scaled.columns)

df_normalized.describe()
```

[ ]:

	V00WOMTSL	V00WOMTSR	…	V00RKRFXPN_Yes	V00LKRFXPN_Yes
count	3702.000000	3702.000000	…	3702.000000	3702.000000
mean	-0.013706	-0.014290	…	-0.006360	-0.006459
std	0.059204	0.060842	…	0.066169	0.067200
min	-0.094583	-0.104786	…	-0.053746	-0.056814
25%	-0.058564	-0.061627	…	-0.036446	-0.038481
50%	-0.032862	-0.031160	…	-0.029492	-0.031005
75%	0.019169	0.022912	…	-0.021970	-0.022756
max	0.211807	0.212003	…	0.300912	0.284660

[8 rows x 227 columns]

2 Split the data set

We first need to split the data set into a training set and a test set. Then, we will apply clustering only on the training set. To predict the outcome of the testing set, each observations will be assigned to a cluster and then the specific model of this cluster will be applied on the observation.

[ ]:
```
## Divide data set in training and testing set
df_normalized_outcome = pd.concat([df_normalized,␣
 ↪df_enc['cumulative_outcome']], axis = 1)
#print(df_normalized_outcome.head())

df_train, df_test = train_test_split(df_normalized_outcome, test_size=0.3,␣
 ↪random_state=88)
df_train.shape, df_test.shape
```

[ ]: ((2591, 228), (1111, 228))

[ ]:
```
y_train = df_train['cumulative_outcome']
y_test = df_test['cumulative_outcome']

x_train = df_train.drop('cumulative_outcome', axis='columns')
x_test = df_test.drop('cumulative_outcome', axis='columns')
```

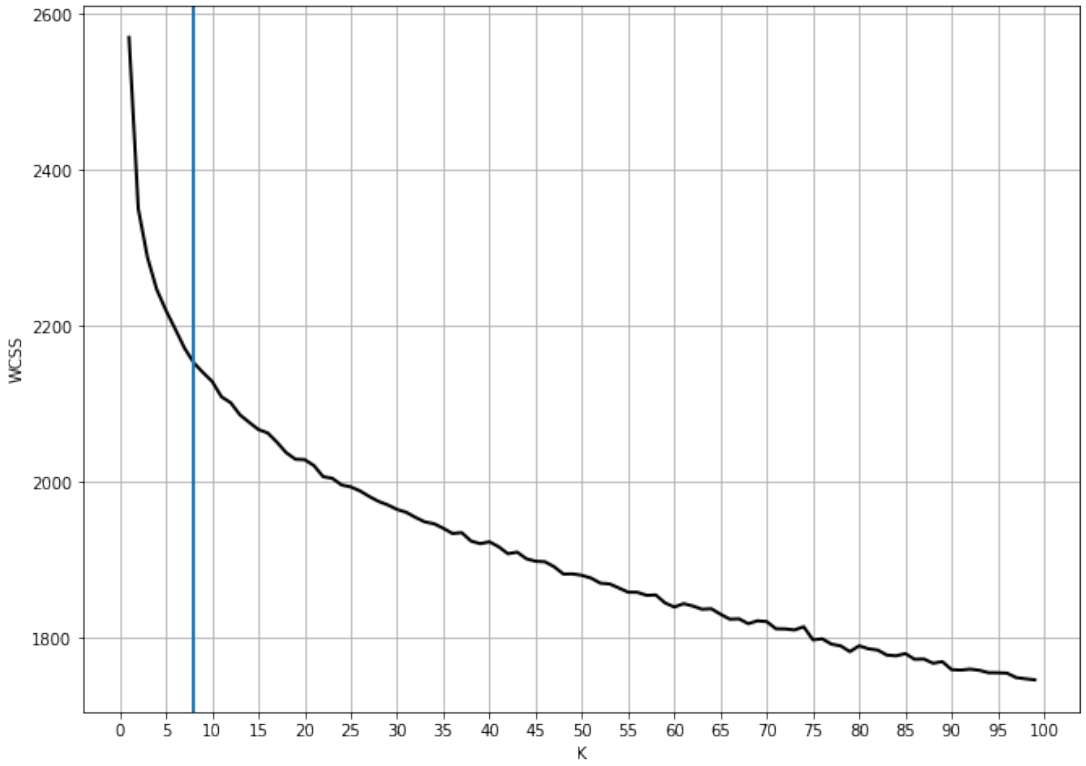
3 K-means clustering

3.1 Scree plot (to select the value of K)

To create the plot we need to compute the cluster dissimilarity for different values of k

[ ]:
```
from sklearn.cluster import KMeans

# Cross-validation to find the optimal k value

wcss = []
for k in range(1, 100):
    kmeans = KMeans(init='k-means++', n_clusters=k, n_init=10, max_iter=100,␣
 ↪random_state=88)
    kmeans.fit(x_train)
    wcss.append(kmeans.inertia_)
```

[ ]:
```
plt.figure(figsize=(11,8))
plt.plot(range(1, 100), wcss, linewidth=2, color='black')
plt.axvline(x=8, linewidth=2)
plt.xticks(np.linspace(0,100,21))
plt.xlabel('K')
plt.ylabel('WCSS')
plt.grid(axis='both')
plt.show()
```



We choose k = 8 clusters, as it is a knee in the WCSS curve above.

[ ]:
```
# Fitting of the kmeans model for 8 clusters

kmeans = KMeans(init='k-means++', n_clusters=8, n_init=10, max_iter=100,␣
 ↪random_state=88)
kmeans.fit(x_train)
wcss_train = kmeans.inertia_
```

[ ]:
```
# We compute the centroid and the size of each cluster, which will be useful␣
 ↪later

print('k-means center is: \n', kmeans.cluster_centers_)
kmeans_size = []
clusters = np.unique(kmeans.labels_)
for cluster in clusters:
    kmeans_size.append(len(np.where(kmeans.labels_ == cluster)[0]))
kmeans_size
```

```
k-means center is:
 [[ 1.66886546e-02 -4.52399483e-02 -1.98475466e-03 … -1.58281687e-03
  -2.10225776e-02  3.64570467e-02]
 [-6.91319112e-02 -7.59399903e-02 -4.17256035e-02 … -1.28761278e-02
```

```
 -3.78199814e-02 -3.70274057e-02]
[ 1.85091911e-03  5.48941521e-03 -1.54929475e-02 …  6.59621665e-03
 -7.37328460e-03 -9.99549614e-03]
…
[ 6.97766059e-02  5.99528742e-02  2.98608045e-02 …  4.51966322e-03
  2.87379871e-02  3.32891002e-02]
[-6.14809192e-02 -6.45144949e-02 -3.67775292e-02 …  5.85539873e-05
 -2.95185789e-02 -3.11825516e-02]
[-6.10319463e-02 -6.65398438e-02 -3.56719631e-02 … -7.01042643e-03
 -3.50427494e-02 -2.75475167e-02]]
```

[ ]:
```
[297, 327, 331, 346, 415, 494, 227, 154]
```

[ ]:
```
# We had a column Cluster with the cluster number of each observation

df_train_cluster = df_train.copy()
df_train_cluster['Cluster'] = kmeans.labels_
df_train_cluster.head()
```

[ ]:
```
      V00WOMTSL  V00WOMTSR  …  cumulative_outcome  Cluster
558   -0.062097  -0.076635  …                   0        7
2651  -0.045562  -0.028329  …                   0        3
2673  -0.080512  -0.089197  …                   0        6
3499   0.003663   0.007725  …                   1        2
815   -0.004304  -0.012397  …                   1        2

[5 rows x 229 columns]
```

[ ]:
```
#df_train_cluster.loc[2909].values
```

3.2 Split into the different clusters

We create one data set for each cluster, we split the observations according to which cluster it belongs. It will be useful to apply a model for each cluster.

[ ]:
```
df_cluster_0 = df_train_cluster[df_train_cluster['Cluster'] == 0]
df_cluster_1 = df_train_cluster[df_train_cluster['Cluster'] == 1]
df_cluster_2 = df_train_cluster[df_train_cluster['Cluster'] == 2]
df_cluster_3 = df_train_cluster[df_train_cluster['Cluster'] == 3]
df_cluster_4 = df_train_cluster[df_train_cluster['Cluster'] == 4]
df_cluster_5 = df_train_cluster[df_train_cluster['Cluster'] == 5]
df_cluster_6 = df_train_cluster[df_train_cluster['Cluster'] == 6]
df_cluster_7 = df_train_cluster[df_train_cluster['Cluster'] == 7]
```

[ ]:
```
df_cluster_0 = df_cluster_0.drop('Cluster', axis = 1)
df_cluster_1 = df_cluster_1.drop('Cluster', axis = 1)
df_cluster_2 = df_cluster_2.drop('Cluster', axis = 1)
df_cluster_3 = df_cluster_3.drop('Cluster', axis = 1)
```

```
df_cluster_4 = df_cluster_4.drop('Cluster', axis = 1)
df_cluster_5 = df_cluster_5.drop('Cluster', axis = 1)
df_cluster_6 = df_cluster_6.drop('Cluster', axis = 1)
df_cluster_7 = df_cluster_7.drop('Cluster', axis = 1)
```

[ ]:
```
#df_train_cluster_0 = cluster_data(df_train_cluster, 0)
len(df_cluster_1)
df_cluster_0.head(10)
```

[ ]:

	V00WOMTSL	V00WOMTSR	…	V00LKRFXPN_Yes	cumulative_outcome
1620	0.025892	-0.053191	…	-0.035981	0
2191	0.023447	-0.014838	…	-0.026348	1
2754	0.001217	-0.029500	…	-0.031747	0
2582	0.087363	-0.046737	…	-0.031615	0
2963	0.009648	-0.085281	…	-0.046239	1
983	0.049346	-0.044072	…	0.136944	0
2924	0.152516	-0.043208	…	0.134259	1
1526	-0.031348	-0.058785	…	-0.031873	1
357	0.055625	-0.048338	…	-0.029096	0
2378	-0.009016	-0.055012	…	-0.029827	0

[10 rows x 228 columns]

[ ]:
```
#print(df_train_cluster_0['cumulative_outcome'])
```

[ ]:
```
print(len(df_cluster_0), len(df_cluster_1), len(df_cluster_2),␣
↪len(df_cluster_3), len(df_cluster_4), len(df_cluster_5), len(df_cluster_6),␣
↪len(df_cluster_7))
```

```
297 327 331 346 415 494 227 154
```

[ ]:
```
print(len(df_cluster_0.columns), len(df_cluster_1.columns), len(df_cluster_2.
↪columns), len(df_cluster_3.columns), len(df_cluster_4.columns),␣
↪len(df_cluster_5.columns), len(df_cluster_6.columns), len(df_cluster_7.
↪columns))
```

```
228 228 228 228 228 228 228 228
```

3.3 Logistic Regression: training

We choose to apply a logistic regression model as it is the one which has the best performance among the model that we did before. We will train one model for each cluster.

3.3.1 Cluster 0

[ ]:
```
print(np.linalg.matrix_rank(df_cluster_0))
```

```
183
```

We will use sklearn as a library for the logistic regression model, because our dataframes are not invertible matrix, and sklearn can handle it.

[ ]:
```
## Logistic Regression with sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import metrics

x_train_cluster_0 = df_cluster_0.drop('cumulative_outcome', axis='columns')
y_train_cluster_0 = df_cluster_0['cumulative_outcome']

logreg0 = LogisticRegression()
logreg0.fit(x_train_cluster_0, y_train_cluster_0.ravel())
print('Accuracy = ', logreg0.score(x_train_cluster_0, y_train_cluster_0.
 ↪ravel()))
```

```
Accuracy =  0.7542087542087542
```

[ ]:
```
# We create a list with the accuracy of each cluster to analyze it later
Accuracy_train = []
Accuracy_train.append(logreg0.score(x_train_cluster_0, y_train_cluster_0.
 ↪ravel()))
```

[ ]:
```
## Coefficients of the logistic regression
logreg0.coef_
```

[ ]:
```
array([[ 1.39222375e-01, -1.23004859e-01,  2.35835486e-02,
         3.08020079e-01, -2.07616960e-01,  9.99436309e-02,
         4.08202312e-02,  2.76164301e-01, -4.61090499e-01,
        -3.19480732e-01, -1.28648742e-01,  3.65021573e-01,
        -4.36454488e-01,  2.81718164e-01,  2.45292835e-01,
        -9.55878264e-02,  7.66807918e-02, -2.11671981e-01,
        -1.20264298e-01, -2.15421403e-01,  4.26039321e-01,
         6.49420590e-01,  1.29928639e-01,  9.05589847e-02,
        -4.37634639e-02, -3.65211922e-01,  1.42539947e-03,
         2.20177445e-01, -5.40297314e-01,  5.13581684e-02,
        -7.84557207e-02,  6.05213116e-03,  6.52687634e-02,
         1.96659836e-03, -2.44738075e-01, -1.11835524e-01,
         3.01258042e-01,  4.14854598e-03,  5.96100121e-02,
        -2.39316089e-01, -3.19099224e-02,  2.38695235e-01,
        -1.03438760e-02,  2.22605800e-03,  9.36711175e-03,
        -1.12560399e-01, -3.45379512e-03, -1.25572212e-01,
         7.87528043e-02,  2.48170019e-03, -1.84189859e-01,
        -1.70811574e-01,  2.70777084e-01,  3.35728602e-03,
        -1.37613031e-01,  4.36726806e-03,  1.18255091e-01,
         1.30750386e-03,  7.68082694e-02,  4.99387890e-01,
        -4.74100793e-01,  4.11545826e-01, -2.56937839e-01,
         3.57228328e-01,  2.29241926e-02, -8.99360713e-02,
```

```
 8.34301381e-01, -1.35970129e-01,  1.95991876e-01,
-8.71021909e-01,  1.07706919e-01,  3.54328533e-01,
-4.00423353e-01,  1.53133765e-01,  2.97700622e-01,
 3.30709741e-01, -4.17035170e-01, -1.58939563e-01,
 8.43882140e-01, -7.70638614e-01, -1.15359733e-01,
-2.00755585e-01,  2.37054233e-01,  1.82316431e-01,
-3.03975458e-01,  1.76339330e-03, -4.17134713e-01,
 4.91329383e-01,  3.04683431e-01, -6.96827597e-01,
 4.32318025e-02,  1.12071628e-01, -9.08157725e-02,
-1.04433447e+00, -2.15421403e-01,  1.10805315e-02,
 4.19133457e-01,  5.77976372e-01, -1.10805315e-02,
 1.56913271e-03, -3.60778208e-01,  2.94345685e-01,
-2.94454238e-01,  3.42186836e-01,  2.79965565e-01,
 4.00684665e-03,  2.93048857e-03,  1.60309925e-03,
 2.11446620e-01,  1.96659836e-03,  1.40346687e-01,
-9.83101679e-03, -1.15113266e+00,  4.46363903e-01,
-4.53215973e-01,  1.64230610e-01,  1.07969043e-01,
 1.96659836e-03,  1.44293589e-01, -2.12685932e-01,
-4.22534024e-01, -3.41523190e-03,  9.77119172e-02,
 2.29789064e-01,  1.62609851e-01,  4.00684665e-03,
-2.18210430e-01,  1.88210465e-03,  9.87876457e-02,
 3.06929664e-01,  5.55768575e-02, -4.71265712e-01,
 3.88725125e-01,  3.25393884e-01, -3.19178620e-01,
-3.69063781e-01, -1.01413973e-01,  2.41766067e-01,
 3.21123996e-01,  1.01461030e+00,  2.24488831e-01,
-6.09776294e-01,  2.54818902e-01, -3.97452997e-02,
-2.58662176e-03, -4.62877700e-01, -2.37913158e-01,
 7.71386167e-01, -1.99327933e-01,  1.94426204e-01,
 3.80166619e-01,  2.53669153e-01,  1.81097156e-02,
-1.20424880e-01, -6.20078413e-01,  6.55185666e-02,
 4.85020014e-01,  6.24860672e-01,  1.96351584e-01,
-2.32277034e-01,  1.17808873e-03, -5.29810350e-01,
 6.55378467e-01,  1.57045501e-01,  7.92957094e-01,
 1.64602950e-01,  3.25575176e-01,  7.89700356e-02,
 4.79242264e-01,  1.72360485e-01,  4.91144941e-01,
-7.70133293e-02,  4.62848538e-01, -6.88880496e-01,
 1.03283229e-03,  2.90491091e-01, -3.84260039e-01,
 2.57074409e-02, -2.18891571e-01, -2.18353392e-01,
-4.05341038e-01,  1.82368282e-03, -3.31058837e-01,
-2.56593838e-01, -9.47101548e-02, -4.75584358e-01,
-5.03419263e-02,  1.92571568e-01, -7.73085148e-02,
 4.35346629e-01,  3.26212884e-04, -1.04943864e-01,
 3.26212884e-04,  1.97020634e-01,  7.99595418e-04,
-3.06262867e-01,  2.51778689e-01,  4.92171134e-01,
 4.74345324e-01, -1.42195739e-01, -7.95259727e-03,
 7.29828686e-04,  3.60787068e-01, -3.48562473e-01,
-1.40376270e-01, -3.85628871e-01, -5.03379793e-01,
```

```
 3.17867829e-01, -4.31154207e-01,  9.19328703e-02,
-5.80247700e-01,  2.16312688e-01, -2.58066629e-01,
-3.65130617e-02,  9.19328703e-02,  6.05708779e-01,
 5.18376285e-02,  6.16618162e-01,  1.13321614e+00,
 2.83882613e-01,  7.99595418e-04,  5.14325221e-02,
 5.67140759e-01, -2.54113553e-01,  4.08980553e-02,
 1.90845605e-01,  5.79793333e-01]])
```

[ ]:
```
## Computation of the p-values of the features, to perform features selection␣
↪later
import statsmodels.api as sm
mod0 = sm.OLS(y_train_cluster_0,x_train_cluster_0)
fii0 = mod0.fit()
p_values0 = fii.summary2().tables[1]['P>|t|']
p_values0
```

[ ]:
```
V00WOMTSL                          0.422254
V00WOMTSR                          0.594368
P01KPACDCV                         0.490362
V00COMORB                          0.441689
V00HT25MM                          0.407957
                                     …
V00LKEFFB_Yes                      0.573021
V00LKEFFPT_Too_tender_to_examine   0.602212
V00LKEFFPT_Yes                     0.278842
V00RKRFXPN_Yes                     0.911536
V00LKRFXPN_Yes                     0.144860
Name: P>|t|, Length: 227, dtype: float64
```

3.3.2 Cluster 1: Training

[ ]:
```
## Logistic Regression with sklearn

x_train_cluster_1 = df_cluster_1.drop('cumulative_outcome', axis='columns')
y_train_cluster_1 = df_cluster_1['cumulative_outcome']

logreg1 = LogisticRegression()
logreg1.fit(x_train_cluster_1, y_train_cluster_1.ravel())
print('Accuracy = ', logreg1.score(x_train_cluster_1, y_train_cluster_1.
↪ravel()))
```

```
Accuracy =  0.7400611620795107
```

[ ]:
```
Accuracy_train.append(logreg1.score(x_train_cluster_1, y_train_cluster_1.
↪ravel()))
```

[ ]:
```
## Coefficients of the logistic regression
logreg1.coef_
```

```
[ ]: array([[ 1.16524432e-01, -9.13812496e-03,  6.50551406e-04,
              -7.64911841e-03, -5.33054310e-01, -3.14173876e-01,
              -5.55627771e-01,  2.23396436e-01, -2.43692261e-01,
              -1.26258567e-02, -2.67180233e-01, -3.83685327e-01,
              -2.55194938e-01, -1.43922234e-01, -2.95793584e-01,
              -8.75224679e-02,  1.73411355e-01, -1.67441484e-01,
               3.96366964e-01, -7.04799940e-02,  3.25926451e-01,
              -1.62600046e-01,  6.82702282e-01,  4.81108907e-01,
               5.82293664e-01,  3.84912932e-01,  2.04317123e-03,
               7.18174522e-02, -8.16632863e-01,  6.95240252e-01,
               3.63035451e-02,  8.67514019e-03, -3.80439794e-02,
               2.81892709e-03, -2.43365388e-01, -5.09030152e-02,
               2.55177523e-01,  5.94653635e-03,  1.43863545e-02,
               1.37899202e-01,  8.37860045e-02, -2.66052265e-01,
               1.75249286e-02,  3.19083719e-03,  2.84841280e-02,
              -6.67815223e-02, -1.94942854e-01,  9.03161131e-03,
               1.66355066e-01,  3.55727536e-03, -2.11422099e-01,
               9.88691918e-03,  1.79421149e-01,  4.81234231e-03,
               6.98221092e-02,  6.26005314e-03, -6.50343276e-02,
               1.87417935e-03,  8.75041335e-02,  7.83441802e-03,
              -8.11413505e-02,  3.65119037e-03, -4.14040907e-01,
               1.08610246e-02,  3.59223297e-01,  5.55530116e-03,
              -1.31069351e-01, -2.33833794e-01,  4.90390819e-01,
              -2.43662822e-01, -2.11858336e-01,  3.77300571e-03,
               2.26693128e-01,  2.75894792e-01, -8.21663446e-02,
               9.50775925e-03, -5.58047251e-02,  3.82651736e-01,
               1.64505969e-02,  1.01761659e-02, -2.32955165e-02,
               5.40514277e-03, -2.81600017e-01,  6.14402912e-03,
               2.42770245e-01,  2.52765245e-03,  3.81566918e-01,
               5.30864857e-01, -6.41585913e-01, -1.18033150e-01,
               2.70494863e-01,  1.99118131e-01,  1.66188028e-02,
               8.38316862e-02, -7.04799940e-02, -4.22814270e-01,
               1.90996386e-01,  3.94609700e-01,  4.22814270e-01,
               2.24919882e-03, -6.50252008e-02, -2.40833152e-01,
               8.08981182e-03,  7.43963068e-03,  6.16349061e-03,
               5.74342417e-03,  3.87638069e-01,  2.29788654e-03,
               2.50961936e-02,  2.81892709e-03, -2.10371604e-01,
              -2.74171302e-01,  7.99471727e-03, -1.16792750e-01,
               5.78452481e-03,  5.29565273e-03,  4.49030878e-03,
               2.81892709e-03,  2.85954517e-01,  2.57121335e-03,
              -4.17113844e-01, -1.41848481e-01, -9.06557317e-02,
               8.64500418e-03,  6.33643695e-03,  5.74342417e-03,
               4.69082758e-03,  2.69781358e-03,  2.51783784e-01,
               3.04728396e-03, -2.73777793e-01, -7.03889567e-02,
               3.68636103e-01,  2.92389233e-01,  6.16349061e-03,
               5.61860255e-03, -1.81782267e-01,  3.11984710e-03,
              -5.14460664e-01,  3.44817976e-01, -2.32389561e-01,
```

```
 2.36078169e-01,  5.83243876e-01, -3.12369210e-01,
 9.97296813e-02,  5.16089799e-01, -1.60923179e-01,
 3.77464143e-01,  3.07551473e-02, -3.52408475e-01,
 1.00666530e+00,  7.38495778e-01,  1.39159064e-02,
-5.22341252e-02, -1.21613515e-01, -9.25021416e-02,
-7.34705829e-01,  4.60723142e-01,  4.93229944e-01,
-1.43590573e-01,  1.68867538e-03,  7.65974212e-01,
-1.85904045e-01, -1.00848232e-01,  1.38183757e-01,
 2.89160271e-01, -1.83818398e-01,  7.16476936e-01,
 4.11833546e-03,  1.73696695e-01, -2.88275172e-01,
 3.20195562e-01,  3.01037700e-03,  7.61831281e-02,
 1.48046443e-03, -1.97491918e-01, -3.33270484e-01,
 1.48046443e-03,  2.51710613e-01,  1.93212196e-03,
-1.24208151e-02, -1.80230043e-01,  7.04872212e-01,
 1.91215271e-01, -2.79502497e-01,  2.27098765e-01,
 7.31602876e-01,  3.04728396e-03,  7.53482479e-02,
 5.80332443e-01,  4.67594378e-04, -1.49257208e-01,
 4.67594378e-04, -3.57616654e-01,  1.14614210e-03,
 2.19946094e-03,  1.48046443e-03,  1.93212196e-03,
-2.02087842e-01,  1.71070432e-01,  6.44963441e-03,
 1.04613829e-03, -3.11527036e-01,  1.93212196e-03,
 1.68679538e-01, -4.00993618e-01, -7.47800992e-02,
-5.68634779e-01,  7.01954860e-01,  6.67086418e-02,
-4.48986399e-01, -1.95145774e-01,  1.12466312e-01,
 1.33031807e-01,  6.67086418e-02,  1.83950864e-01,
 2.18405304e-01, -1.08928126e-01, -3.61988845e-02,
-4.51659530e-01,  1.14614210e-03, -1.83920105e-01,
-1.08730623e-01,  1.40430171e-03, -2.39021095e-01,
-3.88582085e-01, -1.73482759e-01]])
```

[ ]:
```
## p-values of the logistic regression to perform features selection
import statsmodels.api as sm
mod1 = sm.OLS(y_train_cluster_1,x_train_cluster_1)
fii1 = mod1.fit()
p_values1 = fii.summary2().tables[1]['P>|t|']
p_values1
```

[ ]:
```
V00WOMTSL                          0.422254
V00WOMTSR                          0.594368
P01KPACDCV                         0.490362
V00COMORB                          0.441689
V00HT25MM                          0.407957
                                     ...
V00LKEFFB_Yes                      0.573021
V00LKEFFPT_Too_tender_to_examine   0.602212
V00LKEFFPT_Yes                     0.278842
V00RKRFXPN_Yes                     0.911536
```

```
V00LKRFXPN_Yes                        0.144860
Name: P>|t|, Length: 227, dtype: float64
```

3.3.3 Cluster 2: Training

[ ]:
```
## Logistic Regression with sklearn

x_train_cluster_2 = df_cluster_2.drop('cumulative_outcome', axis='columns')
y_train_cluster_2 = df_cluster_2['cumulative_outcome']

logreg2 = LogisticRegression()
logreg2.fit(x_train_cluster_2, y_train_cluster_2.ravel())
print('Accuracy = ', logreg2.score(x_train_cluster_2, y_train_cluster_2.
 ↪ravel()))
```

```
Accuracy =  0.7069486404833837
```

[ ]:
```
Accuracy_train.append(logreg2.score(x_train_cluster_2, y_train_cluster_2.
 ↪ravel()))
```

[ ]:
```
## Coefficients of the logistic regression
logreg2.coef_
```

[ ]:
```
array([[-2.00402107e-01,  6.56619578e-02,  8.83331617e-02,
         1.35970781e-01, -8.73048877e-01, -3.83702474e-01,
         1.71937294e-01, -3.07576717e-01, -1.72240023e-01,
         1.94630853e-02,  2.53088711e-01, -3.97607200e-01,
        -4.54022144e-01,  8.49054074e-01, -1.03205526e+00,
        -2.63595164e-01, -2.78338850e-01, -3.77348158e-01,
        -4.38306563e-01,  7.11769752e-03,  2.73593468e-01,
         2.75810341e-01, -4.90331103e-01,  5.10645137e-01,
         4.91384030e-01, -5.44876208e-01, -1.28969684e-03,
        -5.21256554e-01, -3.52675105e-01, -8.52805322e-02,
        -5.02732774e-01, -1.20398008e-02,  4.52446898e-01,
        -1.77937185e-03, -2.98485992e-01, -5.84300056e-02,
         3.46653920e-01, -8.25676544e-02, -2.54360159e-01,
         4.25663761e-01, -1.47977937e-01, -3.87081597e-02,
         1.63249264e-01, -2.01413009e-03, -6.84296575e-01,
         1.79631627e-01,  3.37022353e-02,  1.88194714e-01,
        -1.96881594e-01,  2.01869005e-01, -7.75001659e-02,
         1.86964838e-01, -1.09849518e-01,  1.77288173e-01,
        -1.57026811e-01,  2.27384875e-01,  1.52724728e-02,
        -1.18302527e-03, -4.31030050e-01, -7.95428659e-03,
         3.76660577e-01, -2.30471565e-03,  2.97175162e-01,
        -3.70613820e-01,  2.71985527e-02, -1.13262741e-01,
        -1.28966446e-02,  1.20598132e-01, -1.25259744e-01,
        -7.08047390e-02, -2.15363276e-01,  1.87111711e-01,
        -8.90339890e-02, -1.03404813e-01,  3.21498567e-01,
```

```
-5.72813161e-02, -2.40987652e-01, -2.75262408e-03,
-2.12186923e-01, -1.83460761e-01,  2.94406035e-01,
 2.27769076e-02, -3.60002091e-01,  1.86937307e-01,
 2.17337779e-01, -1.59551257e-03, -2.31574604e-01,
-8.37924953e-02,  8.73418802e-02,  1.40675574e-01,
-1.74806506e-01, -2.83161817e-01, -1.49818907e-01,
-2.14261586e-01,  7.11769752e-03, -2.60506214e-01,
 3.24218848e-01,  5.53711390e-01,  2.60506214e-01,
-1.41974622e-03, -1.70199110e-01,  3.26191196e-01,
 4.76964975e-01,  4.00714374e-02,  3.43810533e-02,
-3.33973306e-01, -1.49978398e-01,  3.52287473e-01,
-1.80838728e-01, -1.77937185e-03,  2.90591708e-01,
-9.42464523e-02,  2.57629087e-01, -3.59765236e-02,
 1.23212272e-01, -4.70151966e-01, -2.25584341e-01,
-1.77937185e-03, -1.19625330e-01, -1.62300922e-03,
-5.44862017e-01,  4.93376327e-02, -9.64561201e-02,
 7.07377737e-02, -2.79520632e-01, -3.35246996e-01,
 1.88296507e-01, -1.70292221e-03,  3.96559188e-01,
-1.92351598e-03, -5.63948590e-01, -1.09426654e-01,
-3.18601343e-01,  4.17271249e-02,  1.65947146e-01,
-3.80792722e-02, -1.09273366e-01,  2.96364303e-01,
 2.35505855e-01,  4.43379632e-01,  4.99541083e-01,
-4.23701023e-01, -2.91562458e-02,  5.51457370e-01,
-7.04421540e-01,  1.62777424e-01,  1.17514125e-01,
 4.78354636e-01,  2.29998443e-01,  3.51074995e-01,
 5.31562578e-01,  4.57308187e-01, -9.91617078e-02,
 6.41825560e-02,  1.73643788e-01,  3.68087715e-01,
-5.64872957e-01, -1.85977418e-01,  1.11703452e-01,
-2.51067941e-01, -1.06593088e-03,  2.41490838e-01,
 1.92975759e-01,  1.03320945e+00,  6.58351340e-01,
 2.56202078e-01, -4.93700114e-01,  2.16408462e-01,
 4.59514515e-01,  5.74633114e-01,  2.30962584e-01,
-3.30961197e-02, -4.56754228e-01,  5.91107305e-02,
-9.34503322e-04,  3.37116996e-01, -3.19498606e-02,
-2.81400447e-01,  9.44754833e-01,  2.25684584e-01,
 2.71827485e-01, -3.08552132e-01,  4.95116532e-01,
-9.82691117e-02,  1.02467689e-01, -6.97108512e-01,
 6.00206243e-01, -1.64230865e-01, -7.00664608e-02,
 3.13660336e-02, -2.95156365e-04, -3.88208520e-01,
-2.95156365e-04,  3.56181970e-01, -7.23471354e-04,
-1.38835052e-03, -9.34503322e-04, -1.21959999e-03,
 9.57822406e-02,  9.71321148e-02,  1.56468321e-01,
-6.60346640e-04,  1.09889700e-01, -1.21959999e-03,
 5.33104157e-01, -2.95300801e-01, -1.77673051e-01,
-4.39635905e-01,  4.57360979e-01,  3.18953708e-02,
 6.96675126e-02,  1.47439150e-01, -3.22674002e-01,
-1.49082482e-01,  3.18953708e-02, -2.86919024e-01,
```

```
         3.38223886e-01,  5.20559113e-01,  1.08538422e+00,
        -1.84501606e-01, -7.23471354e-04,  4.83826362e-01,
        -5.00328808e-01, -8.86427655e-04,  5.34989821e-01,
        -3.96662751e-01, -4.24761323e-02]])
```

[ ]:
```
## p-values of the logistic regression to perform features selection
import statsmodels.api as sm
mod2 = sm.OLS(y_train_cluster_2,x_train_cluster_2)
fii2 = mod2.fit()
p_values2 = fii.summary2().tables[1]['P>|t|']
p_values2
```

[ ]:
```
V00WOMTSL                          0.422254
V00WOMTSR                          0.594368
P01KPACDCV                         0.490362
V00COMORB                          0.441689
V00HT25MM                          0.407957
                                     …
V00LKEFFB_Yes                      0.573021
V00LKEFFPT_Too_tender_to_examine   0.602212
V00LKEFFPT_Yes                     0.278842
V00RKRFXPN_Yes                     0.911536
V00LKRFXPN_Yes                     0.144860
Name: P>|t|, Length: 227, dtype: float64
```

3.3.4 Cluster 3: Training

[ ]:
```
## Logistic Regression with sklearn

x_train_cluster_3 = df_cluster_3.drop('cumulative_outcome', axis='columns')
y_train_cluster_3 = df_cluster_3['cumulative_outcome']

logreg3 = LogisticRegression()
logreg3.fit(x_train_cluster_3, y_train_cluster_3.ravel())
print('Accuracy = ', logreg3.score(x_train_cluster_3, y_train_cluster_3.
 ↪ravel()))
```

```
Accuracy =  0.7832369942196532
```

[ ]:
```
Accuracy_train.append(logreg3.score(x_train_cluster_3, y_train_cluster_3.
 ↪ravel()))
```

[ ]:
```
## Coefficients of the logistic regression
logreg3.coef_
```

[ ]:
```
array([[ 0.15222227, -0.01291461, -0.16989793,  0.09497085, -0.79897101,
        -0.08909227,  0.19523345, -0.01450771, -0.0632383 ,  0.05744524,
         0.27201943,  0.40736777,  0.72343072,  0.48821612,  0.60221453,
```

```
       -0.04250753, -0.44785716,  0.01492169,  0.0482295 , -0.36486817,
        0.62415724,  0.16989762, -0.13698212,  0.73505391,  0.61416334,
        0.23672655,  0.39857176,  0.81104823, -0.21144243,  0.5390284 ,
       -0.37328352,  0.75202681, -0.23872771,  0.34585163, -0.21106139,
       -0.44746085,  0.29520768,  0.60398974,  0.15577124, -0.25646265,
       -0.06389761,  0.11362505, -0.05313398,  0.47977948, -0.27716797,
       -0.6538812 , -0.02951153, -0.1293954 , -0.07027502,  0.54833588,
        0.68184759, -0.18973791, -0.52383204, -0.2385236 ,  0.29421369,
       -0.13777259, -0.24450387,  0.35768045,  0.00550698,  0.06739823,
       -0.05437136,  0.0314106 ,  0.18115186,  0.09768538, -0.24905579,
        0.04779136, -0.1360402 ,  0.09850416, -0.24681157,  0.36767403,
       -0.02488403,  0.26425061,  0.41971786, -0.42538084,  0.11421777,
       -0.0516659 , -0.1259006 ,  0.03751507,  0.06863206, -0.00757338,
       -0.1195732 ,  0.04649957,  0.37250049,  0.25303975, -0.46476706,
        0.02174499,  0.19431236, -0.06245168, -0.57319426,  0.38097094,
       -0.00840921, -0.25376227, -0.8323965 , -0.18756255, -0.36486817,
       -0.12472407,  0.19554104,  0.37760393,  0.12472407,  0.0761814 ,
       -0.08954814, -0.1521688 ,  0.29786917,  0.08074025, -0.18768454,
        0.44430403, -0.56984446,  0.60737458, -0.02838814,  0.02425077,
        0.18840899,  0.46505086, -0.49557405,  0.16664232,  0.05718503,
        0.05914847,  0.0386294 ,  0.20173027, -0.60137535,  0.48774912,
       -0.61204549, -0.5338489 ,  0.36270435,  0.07760604,  0.44701334,
       -0.2848346 ,  0.22448793,  0.82170832,  0.30970614,  0.3445535 ,
       -0.35453767,  0.15012028,  0.27818799, -0.02014749, -0.40216564,
       -0.06515042, -0.15613301,  0.29906246,  0.06872199,  0.09365635,
        0.17771861,  0.23331973,  0.01050782,  0.04780997,  0.04204939,
        0.11814653, -0.29917253,  0.28240969, -1.04446252,  0.22839454,
        0.92391687,  0.18990305,  0.27707457,  0.76194978, -1.13238084,
        0.15036104,  0.05586157,  0.53788285, -0.46429796,  0.86552942,
        0.55577408,  0.14984519,  0.52740664,  0.54679851,  0.41308673,
        0.13567702,  0.20178463,  0.38565011, -0.49689874,  0.25213922,
       -0.29605217, -0.13679385, -0.23237642,  0.61899611,  0.58409633,
        0.25351973, -0.67201803,  0.15370282,  0.79022407, -0.2351597 ,
        0.11390599, -0.31655561,  0.06673317,  0.06090807,  0.177688  ,
       -0.14064908, -0.19900887,  0.46610476,  0.02927415,  0.2134692 ,
        0.00402264, -1.0143326 ,  0.00402264,  0.17896531,  0.31371823,
        0.43664912, -0.17049826, -0.1240461 ,  0.0933254 ,  0.17873976,
       -0.14177054,  0.28796209,  0.34289355,  0.3399684 , -0.25951676,
        0.61581614,  0.44864028, -0.16952722,  0.13153433, -0.34454578,
        0.19400946, -0.07049805, -0.21136288,  0.49078896, -0.34454578,
        0.51755892,  0.34114297, -0.18293341, -0.40265245, -0.0546139 ,
       -0.28779192,  0.23315633,  0.27802014,  0.01208098, -0.2727859 ,
       -0.60485004, -0.71699086]])
```

[ ]:
```
## p-values of the logistic regression to perform features selection
import statsmodels.api as sm
mod3 = sm.OLS(y_train_cluster_3,x_train_cluster_3)
```

```
fii3 = mod3.fit()
p_values3 = fii.summary2().tables[1]['P>|t|']
p_values3
```

[ ]:
```
V00WOMTSL                          0.422254
V00WOMTSR                          0.594368
P01KPACDCV                         0.490362
V00COMORB                          0.441689
V00HT25MM                          0.407957
                                     …
V00LKEFFB_Yes                      0.573021
V00LKEFFPT_Too_tender_to_examine   0.602212
V00LKEFFPT_Yes                     0.278842
V00RKRFXPN_Yes                     0.911536
V00LKRFXPN_Yes                     0.144860
Name: P>|t|, Length: 227, dtype: float64
```

3.3.5 Cluster 4: Training

[ ]:
```
## Logistic Regression with sklearn

x_train_cluster_4 = df_cluster_4.drop('cumulative_outcome', axis='columns')
y_train_cluster_4 = df_cluster_4['cumulative_outcome']

logreg4 = LogisticRegression()
logreg4.fit(x_train_cluster_4, y_train_cluster_4.ravel())
print('Accuracy = ', logreg4.score(x_train_cluster_4, y_train_cluster_4.
 ↪ravel()))
```

```
Accuracy =  0.7614457831325301
```

[ ]:
```
Accuracy_train.append(logreg4.score(x_train_cluster_4, y_train_cluster_4.
 ↪ravel()))
```

[ ]:
```
## Coefficients of the logistic regression
logreg4.coef_
```

[ ]:
```
array([[ 2.13494685e-01,  3.77873162e-01,  2.18580024e-02,
        -1.85752228e-01, -2.11801467e-02, -2.20374256e-01,
         8.40732892e-02, -4.04060877e-02, -5.71851636e-01,
         4.52604591e-01, -2.53694492e-01,  4.38593459e-01,
        -5.11091227e-01,  7.70681102e-01,  1.90777355e-01,
         1.73593334e-01, -1.55004238e-02, -6.64641425e-01,
         3.50298372e-01,  2.61011257e-01, -3.08346160e-01,
         1.17889105e+00, -1.40647726e-01, -2.79137609e-01,
         6.59664135e-02, -6.78735388e-01, -1.70542895e-01,
         6.96338382e-01, -8.88060883e-01,  1.75192000e-01,
        -4.49386421e-01, -2.89360653e-02,  4.14704530e-01,
```

```
 1.38832461e-03,  9.61280338e-02,  1.01542133e-01,
-1.62013459e-01,  2.92867552e-03, -2.47870876e-01,
 1.41526412e-01,  3.38636363e-01, -2.09092151e-01,
-1.14306114e-01,  1.57149074e-03,  2.87783057e-01,
-2.60570015e-01, -2.65693292e-02,  1.25868311e-01,
-9.09212159e-02,  1.45129577e-01, -2.74202453e-01,
 1.15644798e-01,  1.32568340e-01,  1.09709199e-01,
-3.16210538e-01,  7.09519822e-02,  2.39194421e-01,
 9.23035339e-04,  2.89165523e-01, -9.34504857e-02,
-1.95417574e-01,  1.79821517e-03, -2.67800377e-01,
-1.56625445e-01,  3.42634274e-01,  2.73599178e-03,
 3.83007993e-02, -2.55211307e-01, -4.62914691e-02,
 2.96770585e-01,  3.00292348e-01,  1.85820936e-03,
-2.92433552e-01,  5.34315066e-02, -9.66641027e-02,
 4.68258162e-03,  4.27312196e-02,  1.20012408e-01,
-8.53797961e-03, -8.79275357e-03, -2.44178537e-02,
 9.89243274e-02,  3.23683183e-01,  3.02594092e-03,
-2.85387627e-01,  1.24487154e-03,  5.64075748e-01,
-2.02171658e-01, -3.57978238e-01,  6.58640703e-02,
-2.75007462e-01,  1.69476580e-01, -3.96515454e-01,
-2.04992373e-02,  2.61011257e-01, -1.30872993e-01,
 3.28661711e-01,  5.05099829e-01,  1.30872993e-01,
-2.37818681e-01,  2.35673881e-01,  7.52065044e-01,
 2.85356689e-01, -1.38158655e-01, -1.60109246e-01,
-2.37572075e-01, -3.59921985e-01,  1.13171158e-03,
-1.80810723e-02,  1.38832461e-03, -3.34733684e-01,
 7.91882799e-01, -2.63905682e-01, -2.51439865e-01,
 2.84888467e-03,  2.13093121e-01, -1.42323237e-01,
 1.38832461e-03, -2.17524586e-01,  1.26632533e-03,
-4.07358871e-01,  1.61151553e-01,  2.48882618e-02,
-3.32917619e-01, -2.84332964e-01, -4.62712086e-01,
 4.01984465e-01,  9.10284715e-01,  5.21089766e-01,
 1.50079062e-03,  6.32906686e-02, -3.71036024e-01,
-1.81108945e-01,  2.50739704e-01,  1.83632130e-01,
 7.79902272e-02,  4.38385140e-01, -1.93515477e-01,
 1.48407949e-01, -8.59459027e-02, -9.00879183e-02,
-1.31101882e-01, -6.72417999e-02,  4.27948535e-01,
-3.95305798e-01,  1.75802382e-01,  1.19244788e-01,
 2.01260758e-01,  5.46119872e-01,  1.00539968e-01,
 2.48346780e+00,  2.76310562e-01,  2.05589224e-01,
-4.87206214e-01, -2.85683414e-01,  9.10480556e-01,
-9.77886280e-02, -4.59163858e-01, -1.24303783e-01,
 1.53652804e-03,  6.18975490e-01,  1.70492894e+00,
 6.97157605e-02,  6.53869933e-01,  1.81759631e-01,
 1.26056289e+00, -4.59399081e-01, -2.98894127e-01,
 7.19660854e-01,  3.17419453e-01, -5.72895688e-01,
 2.51128104e-01,  1.94168860e-01, -8.38913469e-02,
```

```
 7.29130317e-04,  1.20085125e-03,  5.14221225e-02,
 3.71225029e-01,  3.30082234e-01, -1.08326286e-01,
 1.71793365e-01,  6.73292652e-01,  1.78882818e-01,
 4.31473330e-01, -9.25177712e-02, -7.04477047e-01,
 5.30859837e-01, -1.78531124e-02,  1.16252894e-01,
-2.72031562e-01,  2.30290732e-04,  5.32386284e-01,
 2.30290732e-04,  1.43512441e-01, -2.48924128e-01,
-1.82456665e-01, -5.76280730e-01, -7.88634104e-01,
 7.38874093e-01,  5.06165230e-01,  2.84812718e-01,
 1.35890849e-01, -6.60825290e-01, -2.81237605e-01,
 3.19045169e-01, -1.11997603e+00, -9.35344764e-02,
-3.75851330e-01,  7.77555785e-01,  2.14493820e-02,
-4.31964596e-01,  9.03976253e-02, -3.69001453e-02,
 2.12123125e-01,  2.14493820e-02,  2.98089562e-01,
 3.11806170e-01,  1.98060876e-01,  1.64584737e-01,
 7.31053742e-01,  5.16231888e-01, -1.92155593e-02,
-9.69163698e-01,  6.91620096e-04, -7.36305996e-01,
 6.56407035e-02,  1.21098862e-01]])
```

[ ]:
```
## p-values of the logistic regression to perform features selection
import statsmodels.api as sm
mod4 = sm.OLS(y_train_cluster_4,x_train_cluster_4)
fii4 = mod4.fit()
p_values4 = fii.summary2().tables[1]['P>|t|']
p_values4
```

[ ]:
```
V00WOMTSL                          0.422254
V00WOMTSR                          0.594368
P01KPACDCV                         0.490362
V00COMORB                          0.441689
V00HT25MM                          0.407957
                                     ...
V00LKEFFB_Yes                      0.573021
V00LKEFFPT_Too_tender_to_examine   0.602212
V00LKEFFPT_Yes                     0.278842
V00RKRFXPN_Yes                     0.911536
V00LKRFXPN_Yes                     0.144860
Name: P>|t|, Length: 227, dtype: float64
```

3.3.6 Cluster 5: Training

[ ]:
```
## Logistic Regression with sklearn

x_train_cluster_5 = df_cluster_5.drop('cumulative_outcome', axis='columns')
y_train_cluster_5 = df_cluster_5['cumulative_outcome']

logreg5 = LogisticRegression()
```

```
logreg5.fit(x_train_cluster_5, y_train_cluster_5.ravel())
print('Accuracy = ', logreg5.score(x_train_cluster_5, y_train_cluster_5.
 ↪ravel()))
```

```
Accuracy =  0.7226720647773279
```

[ ]:
```
Accuracy_train.append(logreg5.score(x_train_cluster_5, y_train_cluster_5.
 ↪ravel()))
```

[ ]:
```
## Coefficients of the logistic regression
logreg5.coef_
```

[ ]:
```
array([[ 0.45286626, -0.12767409, -0.1855959 , -0.51129373,  0.40008584,
         0.07525755, -0.19155081, -0.28775398,  0.44826567, -0.00884279,
         0.16273168,  0.18065849,  0.16400087,  0.54086787,  0.32018528,
        -0.28597249, -0.3638088 , -0.69470225,  0.11405321, -0.46224979,
        -0.07026457, -0.21056297,  0.12167148,  0.22423481,  0.43413994,
         0.53478925,  0.78193503, -0.41007259,  0.11339178,  0.3307658 ,
        -0.50529942,  0.44382192,  0.08569345,  0.39675516,  0.01753221,
        -0.24904895, -0.2074535 ,  0.38839615, -0.04796672, -0.05392646,
         0.24799221, -0.51815622,  0.31286338,  0.38596427, -0.20489169,
         0.00893264, -0.28109293, -0.06630699,  0.36790305,  0.00212043,
         0.46821311, -0.88058316, -0.01257985,  0.41010689, -0.1734156 ,
         0.06161137,  0.19247629, -0.43563738,  0.251328  , -0.05766713,
        -0.38740149,  0.52035152, -0.34599105,  0.23586656, -0.24750933,
         0.54586836, -0.09715624, -0.38904507,  0.15222342,  0.16522604,
        -0.51110272,  0.00411113,  0.55531764,  0.53344764, -0.23366485,
         0.35262769, -0.28683622,  0.78201358, -0.39183527,  0.01778701,
        -0.05551059,  0.62896154,  0.52775915, -0.11540691, -0.35349162,
        -0.32810956,  0.30647435, -0.25716942,  0.03628847, -0.36452923,
        -0.45867851, -0.29367079, -0.40594244,  0.31798404, -0.46224979,
         0.66360356,  0.90383775, -0.22492013, -0.66360356,  0.34734874,
        -0.09487717, -0.02548893, -0.11240567,  0.39430609,  0.26187672,
        -0.23741689, -1.03277431, -0.27469212,  0.26995988, -0.15569413,
         0.15716249, -0.06935717,  0.27581435,  0.6348052 , -0.21730027,
        -0.87826535,  0.47279958,  0.28477307, -0.01421099, -0.19760236,
        -0.31086976,  0.32083619, -0.14452647, -0.59125548,  0.44548635,
         0.35727634, -1.22134023,  0.14610854,  0.34421525,  0.08645384,
         0.73316572,  0.15840264, -0.25960821, -0.34641285, -0.04384101,
         0.43371589, -0.15814598, -0.30663013, -0.18782245, -0.17549354,
         0.33352673,  0.04340614,  0.47860778,  0.33413065,  0.24225708,
         0.23253672, -0.19040279,  0.31555996,  0.53921086,  0.67726258,
         0.33752619,  0.80499511, -0.19043563,  0.3123385 ,  0.97111742,
         0.33557727, -0.06034495, -0.03057915, -0.54114856, -0.05243886,
        -0.18534807,  0.29058588, -0.59201542,  0.73420358,  0.61951514,
         0.38764149, -0.04335982,  0.49198483,  0.53193711,  0.25083119,
        -0.3228864 ,  0.43252377, -0.5752708 , -0.16538096,  0.41958073,
         0.11586685,  0.11800197,  0.00602779,  0.57244452, -0.17765527,
```

```
 0.60342474,  0.26276462,  0.09358675,  0.55370067,  0.15703425,
 0.23065389,  0.22846828,  1.0069386 ,  0.37005225, -0.01136374,
 0.00190383, -0.6485655 ,  0.00190383,  0.70031738, -0.08147339,
-0.0483129 , -0.48318439,  0.06498685,  0.20197174,  0.22434724,
-0.36513196,  0.00425941, -0.37327937, -0.21797762,  0.11520742,
 0.29432876, -0.13376615,  0.35529096, -0.21927294, -0.40713126,
 0.40199788,  0.06598336,  0.00481756,  0.09747999, -0.40713126,
-0.11746598,  0.3732049 ,  0.37104192, -0.10143224,  0.14226426,
 0.55460321,  0.38461854,  0.56843959, -0.02849578,  0.47448832,
-0.40874519,  0.69137879]])
```

[ ]:
```
## p-values of the logistic regression to perform features selection
import statsmodels.api as sm
mod5 = sm.OLS(y_train_cluster_5,x_train_cluster_5)
fii5 = mod5.fit()
p_values5 = fii.summary2().tables[1]['P>|t|']
p_values5
```

[ ]:
```
V00WOMTSL                          0.422254
V00WOMTSR                          0.594368
P01KPACDCV                         0.490362
V00COMORB                          0.441689
V00HT25MM                          0.407957
                                     ...
V00LKEFFB_Yes                      0.573021
V00LKEFFPT_Too_tender_to_examine   0.602212
V00LKEFFPT_Yes                     0.278842
V00RKRFXPN_Yes                     0.911536
V00LKRFXPN_Yes                     0.144860
Name: P>|t|, Length: 227, dtype: float64
```

3.3.7 Cluster 6: Training

[ ]:
```
## Logistic Regression with sklearn

x_train_cluster_6 = df_cluster_6.drop('cumulative_outcome', axis='columns')
y_train_cluster_6 = df_cluster_6['cumulative_outcome']

logreg6 = LogisticRegression()
logreg6.fit(x_train_cluster_6, y_train_cluster_6.ravel())
print('Accuracy = ', logreg6.score(x_train_cluster_6, y_train_cluster_6.
 ↪ravel()))
```

```
Accuracy =  0.7048458149779736
```

[ ]:
```
Accuracy_train.append(logreg6.score(x_train_cluster_6, y_train_cluster_6.
 ↪ravel()))
```

[ ]:
```
## Coefficients of the logistic regression
logreg6.coef_
```

[ ]:
```
array([[-1.69366502e-03, -1.71462874e-02, -3.94245807e-04,
        -5.63182592e-01, -3.28767749e-01, -2.40417868e-01,
         2.31751344e-01, -1.58582280e-01,  9.21856648e-03,
        -5.44523046e-01, -2.02433209e-01, -3.08988625e-01,
         3.46699379e-01,  2.40749107e-01, -3.09631408e-01,
         7.68410367e-01,  3.41936173e-01,  6.25128213e-02,
        -3.39244723e-01,  1.16525757e-01, -2.34528404e-01,
         2.16388282e-01,  5.13360563e-01, -9.99907136e-02,
         8.24872344e-02, -3.14008043e-01, -7.78921179e-04,
        -2.97757741e-01, -5.09550791e-01,  1.91246627e-01,
         9.89915772e-02, -3.30723648e-03, -8.53113725e-02,
        -1.07466373e-03,  4.93581127e-01,  1.16933228e-01,
        -5.31861044e-01, -2.26700682e-03, -1.29847494e-01,
         4.07167790e-01, -2.44344155e-01, -2.85088683e-02,
        -1.72049450e-01, -1.21644757e-03,  7.71311903e-02,
         7.48543775e-02,  1.44496764e-01, -3.44313449e-03,
        -1.25695883e-01, -1.35614533e-03,  1.17444207e-02,
         1.54967679e-01, -1.08294991e-01, -1.83461635e-03,
        -6.24447897e-02, -8.97015083e-02,  1.03424277e-01,
        -7.14496157e-04,  2.81411524e-01, -2.98672673e-03,
        -2.40197429e-01, -1.39194869e-03,  2.10776924e-01,
        -4.14056442e-03, -1.84129819e-01, -2.11785565e-03,
        -8.50508765e-02,  7.64315690e-02,  4.21870122e-02,
        -4.98643605e-02,  9.67542616e-02, -1.43838853e-03,
        -1.41543227e-01, -2.72592893e-03,  4.73882793e-01,
        -3.62465707e-03, -4.07261254e-01, -1.66246603e-03,
         4.40022776e-01,  5.59598276e-02, -4.15201217e-01,
        -2.06061055e-03,  7.63628749e-02, -2.34229728e-03,
        -6.54084517e-02, -9.63620668e-04, -4.21084132e-01,
         6.31224152e-01, -1.42087486e-01, -3.38128637e-03,
         6.92571296e-02, -6.56928601e-02, -9.08115593e-02,
         5.41164109e-03,  1.16525757e-01,  4.12311951e-01,
         2.31277724e-01,  4.85624624e-01, -4.12311951e-01,
        -8.57465382e-04, -3.45708474e-01, -1.47952237e-01,
         3.22580692e-01, -8.45452563e-02, -2.34971661e-03,
         4.23482830e-02, -1.60138940e-03, -8.76026673e-04,
        -1.14633626e-01, -1.07466373e-03, -8.25082199e-01,
         3.51856967e-01,  2.00345484e-01,  1.42070657e-01,
        -2.20524292e-03,  2.24289983e-01, -1.71184704e-03,
        -1.07466373e-03,  6.88615281e-02, -9.80227455e-04,
         6.06957960e-02, -4.83246531e-01,  2.55543437e-01,
         1.31045017e-01, -2.41564920e-03, -2.18957406e-03,
        -1.78829111e-03, -1.02849145e-03,  3.25775003e-01,
        -1.16172056e-03,  7.17825106e-02, -3.78532739e-01,
```

```
 2.90751432e-01,  5.73279880e-02,  4.89307697e-01,
 2.75248481e-01, -2.01043149e-03, -1.18938391e-03,
-2.03405347e-01,  2.83825293e-01,  5.78423824e-02,
 8.56968605e-03,  3.33523330e-01,  9.12862880e-02,
-2.88118874e-01,  4.05512245e-02, -6.52753330e-01,
-3.17797834e-01, -3.44373081e-02,  1.30284612e-01,
 6.40463209e-01,  5.07463387e-01,  4.20414750e-01,
 5.09122401e-01, -4.54954497e-01,  7.80402664e-02,
-4.56458288e-01,  4.65198468e-01,  9.32241654e-01,
-2.68143840e-01, -2.76568126e-01, -6.69407038e-02,
-3.42665318e-01, -2.90504765e-01,  6.91824433e-02,
 4.86182382e-01,  2.68841942e-01, -5.42263042e-01,
-1.57003910e-03,  1.97171130e-01,  1.83824073e-01,
 1.76361650e-02, -1.57166851e-01, -7.35898800e-02,
-5.64399636e-04, -9.29545776e-04, -1.08963600e-03,
-5.64399636e-04, -3.93115157e-01, -7.36585705e-04,
-1.78132664e-01, -2.29510001e-01,  1.44023899e-01,
 3.26070736e-01, -1.26892861e-03,  5.23472948e-02,
-1.62375540e-01, -1.16172056e-03, -5.20645889e-01,
 1.89624782e-01, -1.78261694e-04,  1.83380603e-01,
-1.78261694e-04,  1.32212640e-01, -4.36945444e-04,
-8.38503737e-04, -5.64399636e-04, -7.36585705e-04,
-1.11548956e-01, -3.90349135e-01, -1.28684696e-01,
-3.98820844e-04, -9.90637820e-02, -7.36585705e-04,
-2.68442708e-01, -1.71184704e-03,  6.91629024e-02,
-1.41736150e-01, -1.59808963e-01, -5.07464888e-03,
-4.97637524e-01,  3.11434843e-01, -3.04195935e-02,
-3.30115603e-01, -5.07464888e-03,  3.55083296e-01,
 9.52360140e-01,  7.85303087e-02,  2.30453116e-02,
 4.68657738e-01, -4.36945444e-04, -1.64655591e-01,
 4.35016305e-02, -5.35364010e-04, -3.25597930e-01,
-1.83960165e-02,  1.73008505e-01]])
```

[ ]:
```
## p-values of the logistic regression to perform features selection
import statsmodels.api as sm
mod6 = sm.OLS(y_train_cluster_6,x_train_cluster_6)
fii6 = mod6.fit()
p_values6 = fii.summary2().tables[1]['P>|t|']
p_values6
```

[ ]:
```
V00WOMTSL                    0.422254
V00WOMTSR                    0.594368
P01KPACDCV                   0.490362
V00COMORB                    0.441689
V00HT25MM                    0.407957
                               …
V00LKEFFB_Yes                0.573021
```

```
V00LKEFFPT_Too_tender_to_examine    0.602212
V00LKEFFPT_Yes                      0.278842
V00RKRFXPN_Yes                      0.911536
V00LKRFXPN_Yes                      0.144860
Name: P>|t|, Length: 227, dtype: float64
```

3.3.8 Cluster 7: Training

[ ]:
```
## Logistic Regression with sklearn

x_train_cluster_7 = df_cluster_7.drop('cumulative_outcome', axis='columns')
y_train_cluster_7 = df_cluster_7['cumulative_outcome']

logreg7 = LogisticRegression()
logreg7.fit(x_train_cluster_7, y_train_cluster_7.ravel())
print('Accuracy = ',logreg7.score(x_train_cluster_7, y_train_cluster_7.ravel()))
```

```
Accuracy =  0.7337662337662337
```

[ ]:
```
Accuracy_train.append(logreg7.score(x_train_cluster_7, y_train_cluster_7.
↪ravel()))
```

[ ]:
```
## Coefficients of the logistic regression
logreg7.coef_
```

[ ]:
```
array([[ 0.1408727 ,  0.19222347,  0.21111394,  0.03854913,  0.2698563 ,
        -0.0448699 ,  0.24515991,  0.01232008, -0.01705201,  0.67313005,
        -0.34003229, -0.27139969,  0.14506051,  0.49888018, -0.53692493,
         0.19684381,  0.39482   , -0.02057835, -0.16934771,  0.21573129,
         0.83986522,  0.01149528,  0.01518505, -0.19930721,  0.51113901,
         0.08147762,  0.00706956,  0.32791319, -0.7393787 ,  0.48141261,
        -0.04663204,  0.03001676,  0.05331872, -0.14343583,  0.4728237 ,
        -0.00801118, -0.43307041,  0.02057555, -0.00320854, -0.53015363,
         0.63911871, -0.30145528,  0.05710126,  0.01104058, -0.1074985 ,
         0.02333486,  0.2809983 ,  0.03125018, -0.27232018,  0.01230849,
        -0.25602105, -0.02407774,  0.23601977,  0.01665113,  0.0163018 ,
         0.02166034, -0.0271805 ,  0.00648483,  0.24622155,  0.23820918,
        -0.35461494,  0.01263344, -0.06514589,  0.03758012,  0.02399368,
         0.01922184,  0.11855154, -0.59052725,  0.58343104, -0.23445824,
         0.08267468,  0.01305494,  0.157222  , -0.09319688,  0.21109663,
         0.0328977 , -0.20941378,  0.01508868,  0.03639345,  0.03521044,
        -0.0628942 ,  0.01870228, -0.04200666,  0.02125889,  0.02343593,
         0.0087459 , -0.13480209, -0.00116773, -0.0102455 ,  0.21473826,
         0.03783177,  0.39337906, -0.05851078,  0.0894343 ,  0.21573129,
        -0.27165865,  0.25345345,  0.09897595,  0.27165865,  0.00778243,
         0.27740523,  0.15395765,  0.01232669,  0.15006312,  0.02132623,
         0.01987276, -0.18483272,  0.00795089,  0.13857315,  0.00975374,
        -0.06260053, -0.03697111, -0.10435809,  0.02562555,  0.02001497,
```

```
 0.01832343, -0.0739268 ,  0.00975374,  0.05532276,  0.00889663,
-0.02525962, -0.31838109, -0.02640005,  0.02991249, -0.06324668,
-0.08091704,  0.01623068,  0.00933468,  0.08134861,  0.01054387,
 0.02525073,  0.39388861, -0.34095409,  0.02864158,  0.19510992,
 0.01944087,  0.28889833,  0.33189692, -0.28876427,  0.52382349,
-0.41681556,  0.0786269 ,  0.2189371 ,  0.07200886, -0.19718332,
 0.08695051, -0.00176501,  0.43848051, -0.44371176,  0.27097842,
 0.96986941,  0.90760036,  0.27670592, -0.24266468, -0.12976387,
-0.24836248,  0.12141361,  0.63445149,  0.17471407,  0.01079495,
 0.00584297,  0.94920405,  0.15987229,  0.0130375 ,  0.0562929 ,
 0.2704699 ,  0.31717997,  0.52259224,  0.01424981,  0.03990293,
 0.42134761, -0.01008126,  0.40245163, -0.60622861,  0.00512254,
 0.00843664,  0.00988963,  0.00512254, -0.2123024 ,  0.00668531,
 0.0164841 ,  0.00904492,  0.02166034,  0.27812225,  0.0115169 ,
 0.01698082,  0.14642232,  0.01054387,  0.01754455, -0.11067911,
 0.00161792, -0.30975953,  0.00161792,  0.27403889,  0.00396575,
 0.00761033,  0.00512254,  0.00668531,  0.2704329 ,  0.16920576,
 0.02231631,  0.00361973,  0.13194746,  0.00668531,  0.16656961,
 0.01553687,  0.01424981,  0.10786457,  0.05126654, -0.21034283,
 0.02251008,  0.03001676,  0.06557348,  0.07074403, -0.21034283,
 0.53452539,  0.0800955 , -0.16000372,  0.32601897,  0.023972  ,
 0.00396575, -0.18691544, -0.26241871,  0.00485901,  0.16517902,
-0.04741857, -0.11738065]])
```

[ ]:
```
## p-values of the logistic regression to perform features selection
import statsmodels.api as sm
mod7 = sm.OLS(y_train_cluster_7,x_train_cluster_7)
fii7 = mod7.fit()
p_values7 = fii.summary2().tables[1]['P>|t|']
p_values7
```

[ ]:
```
V00WOMTSL                          0.422254
V00WOMTSR                          0.594368
P01KPACDCV                         0.490362
V00COMORB                          0.441689
V00HT25MM                          0.407957
                                     …
V00LKEFFB_Yes                      0.573021
V00LKEFFPT_Too_tender_to_examine   0.602212
V00LKEFFPT_Yes                     0.278842
V00RKRFXPN_Yes                     0.911536
V00LKRFXPN_Yes                     0.144860
Name: P>|t|, Length: 227, dtype: float64
```

4 Logistic Regression: Test

For each observation of the test data set, we are going to assign a cluster, according to which cluster the observation is the closest, and then to apply the logistic regression model of this specific cluster.

[ ]:
```
len(x_test)
```

[ ]:
```
1111
```

[ ]:
```
## Assign each point of the test set to a cluster
from tqdm import tqdm

Clusters_test = pd.DataFrame(data = len(x_test)*[0], index = x_test.index,␣
↪columns = ['Cluster'])
for i in tqdm(range(len(x_test))):
  dist_clusters = 8*[0] #First element will be the distance to the first␣
↪cluster, second element to the second, ...
  for j in range(227):
    for k in range(8):
      #We use a metric to compute the distance between tow points, that we sum␣
↪for each point of the vector
      dist_clusters[k] = dist_clusters[k] + (x_test.iloc[i,j] - kmeans.
↪cluster_centers_[k][j])**2
  #We select the cluster for which the distance between the observation and the␣
↪centroid is minimal
  K = dist_clusters.index(min(dist_clusters))
  Clusters_test.iloc[i,0] = K
```

```
100%|      | 1111/1111 [00:56<00:00, 19.61it/s]
```

[ ]:
```
## Number of observations in each cluster

Clusters_test['Cluster'].value_counts()
```

[ ]:
```
5    235
4    164
2    145
1    139
3    133
0    119
6    105
7     71
Name: Cluster, dtype: int64
```

[ ]:
```
## We split the observation according to the cluster it belongs to
## Then, the aim is to predict the outcomes of one data set using the specific␣
↪model of the cluster involved
```

```
df_test_cluster_0 = df_test[Clusters_test['Cluster'] == 0]
df_test_cluster_1 = df_test[Clusters_test['Cluster'] == 1]
df_test_cluster_2 = df_test[Clusters_test['Cluster'] == 2]
df_test_cluster_3 = df_test[Clusters_test['Cluster'] == 3]
df_test_cluster_4 = df_test[Clusters_test['Cluster'] == 4]
df_test_cluster_5 = df_test[Clusters_test['Cluster'] == 5]
df_test_cluster_6 = df_test[Clusters_test['Cluster'] == 6]
df_test_cluster_7 = df_test[Clusters_test['Cluster'] == 7]
```

[ ]:
```
# We check the len of each created data sets
print(len(df_test_cluster_0), len(df_test_cluster_1), len(df_test_cluster_2),␣
↪len(df_test_cluster_3), len(df_test_cluster_4), len(df_test_cluster_5),␣
↪len(df_test_cluster_6), len(df_test_cluster_7))
```

```
119 139 145 133 164 235 105 71
```

[ ]:
```
df_test_cluster_0.head()
```

[ ]:

	V00WOMTSL	V00WOMTSR	…	V00LKRFXPN_Yes	cumulative_outcome
1732	-0.007473	-0.013923	…	-0.024724	0
2080	0.062570	-0.049925	…	0.172221	0
3065	0.039535	-0.056148	…	0.174467	0
1906	0.042137	-0.029830	…	-0.026819	1
283	0.047794	-0.050617	…	0.157281	1

[5 rows x 228 columns]

Then, we are going to predict the outcomes for each cluster, using the models trained before. We assess its performance by computing the accuracy.

4.0.1 Cluster 0

[ ]:
```
x_test_cluster_0 = df_test_cluster_0.drop('cumulative_outcome', axis='columns')
y_test_cluster_0 = df_test_cluster_0['cumulative_outcome']

logreg0.predict(x_test_cluster_0)
print('Accuracy = ',logreg0.score(x_test_cluster_0, y_test_cluster_0.ravel()))
```

```
Accuracy =  0.5966386554621849
```

[ ]:
```
#We create a list with the accuracy of each cluster to analyze it further
Accuracy_test = []
Accuracy_test.append(logreg0.score(x_test_cluster_0, y_test_cluster_0.ravel()))
```

4.0.2 Cluster 1

[ ]:
```
x_train_cluster_1.shape
```

[ ]:
```
(327, 227)
```

[ ]:
```
x_test_cluster_1 = df_test_cluster_1.drop('cumulative_outcome', axis='columns')
y_test_cluster_1 = df_test_cluster_1['cumulative_outcome']

logreg1.predict(x_test_cluster_1)
print('Accuracy = ', logreg1.score(x_test_cluster_1, y_test_cluster_1.ravel()))
```

```
Accuracy =  0.6258992805755396
```

[ ]:
```
Accuracy_test.append(logreg1.score(x_test_cluster_1, y_test_cluster_1.ravel()))
```

4.0.3 Cluster 2

[ ]:
```
x_test_cluster_2 = df_test_cluster_2.drop('cumulative_outcome', axis='columns')
y_test_cluster_2 = df_test_cluster_2['cumulative_outcome']

logreg2.predict(x_test_cluster_2)
print('Accuracy = ',logreg2.score(x_test_cluster_2, y_test_cluster_2.ravel()))
```

```
Accuracy =  0.5724137931034483
```

[ ]:
```
Accuracy_test.append(logreg2.score(x_test_cluster_2, y_test_cluster_2.ravel()))
```

4.0.4 Cluster 3

[ ]:
```
x_test_cluster_3 = df_test_cluster_3.drop('cumulative_outcome', axis='columns')
y_test_cluster_3 = df_test_cluster_3['cumulative_outcome']

logreg3.predict(x_test_cluster_3)
print('Accuracy = ',logreg3.score(x_test_cluster_3, y_test_cluster_3.ravel()))
```

```
Accuracy =  0.6090225563909775
```

[ ]:
```
Accuracy_test.append(logreg3.score(x_test_cluster_3, y_test_cluster_3.ravel()))
```

4.0.5 Cluster 4

[ ]:
```
x_test_cluster_4 = df_test_cluster_4.drop('cumulative_outcome', axis='columns')
y_test_cluster_4 = df_test_cluster_4['cumulative_outcome']

logreg4.predict(x_test_cluster_4)
print('Accuracy = ',logreg4.score(x_test_cluster_4, y_test_cluster_4.ravel()))
```

```
Accuracy =  0.524390243902439
```

[ ]:
```
Accuracy_test.append(logreg4.score(x_test_cluster_4, y_test_cluster_4.ravel()))
```

4.0.6 Cluster 5

[ ]:
```
x_test_cluster_5 = df_test_cluster_5.drop('cumulative_outcome', axis='columns')
y_test_cluster_5 = df_test_cluster_5['cumulative_outcome']

logreg5.predict(x_test_cluster_5)
print('Accuracy = ', logreg5.score(x_test_cluster_5, y_test_cluster_5.ravel()))
```

```
Accuracy =  0.5702127659574469
```

[ ]:
```
Accuracy_test.append(logreg5.score(x_test_cluster_5, y_test_cluster_5.ravel()))
```

4.0.7 Cluster 6

[ ]:
```
x_test_cluster_6 = df_test_cluster_6.drop('cumulative_outcome', axis='columns')
y_test_cluster_6 = df_test_cluster_6['cumulative_outcome']

logreg6.predict(x_test_cluster_6)
print('Accuracy = ', logreg6.score(x_test_cluster_6, y_test_cluster_6.ravel()))
```

```
Accuracy =  0.6571428571428571
```

[ ]:
```
Accuracy_test.append(logreg6.score(x_test_cluster_6, y_test_cluster_6.ravel()))
```

4.0.8 Cluster 7

[ ]:
```
x_test_cluster_7 = df_test_cluster_7.drop('cumulative_outcome', axis='columns')
y_test_cluster_7 = df_test_cluster_7['cumulative_outcome']

logreg7.predict(x_test_cluster_7)
print('Accuracy = ',logreg7.score(x_test_cluster_7, y_test_cluster_7.ravel()))
```

```
Accuracy =  0.7464788732394366
```

[ ]:
```
Accuracy_test.append(logreg7.score(x_test_cluster_7, y_test_cluster_7.ravel()))
```

4.0.9 Summary

[ ]:
```
summary1 = pd.DataFrame(data = [Accuracy_train, Accuracy_test],
                        index = ['Accuracy_train', 'Accuracy_test'],
                        columns = ['Cluster 0','Cluster 1','Cluster 2','Cluster␣
 ↪3','Cluster 4','Cluster 5','Cluster 6','Cluster 7'])
summary1
```

[ ]:

	Cluster 0	Cluster 1	…	Cluster 6	Cluster 7
Accuracy_train	0.754209	0.740061	…	0.704846	0.733766
Accuracy_test	0.596639	0.625899	…	0.657143	0.746479

[2 rows x 8 columns]

5 Features selection

As we have a huge difference between the accuracy on the training and the testing data, we are going to perform features selection to reduce the overfitting of the training data.

5.1 Cluster 0

5.1.1 Feature selection: coefficients value

We are going to remove the features which have a coefficient value too low

[ ]:
```
#We look for the columns which have low coefficients' values

low_coef_col = []
for i in range(len(logreg0.coef_[0])):
  if abs(logreg0.coef_[0][i]) < 0.1:
    low_coef_col.append(df_cluster_0.iloc[:,i].name)
print(len(low_coef_col))
print(low_coef_col)
```

```
71
['P01KPACDCV', 'V00WT25KG', 'V00WTMAXKG', 'V00LLWGT', 'V00RLWGT',
'V00SF2_Yes__limited_a_lot', 'V00SF3_Yes__limited_a_little', 'V00SF8_Extremely',
'V00SF8_Quite_a_bit', 'V00WPRKN1_Mild', 'V00WPRKN1_Moderate', 'V00WPRKN1_None',
'V00WPRKN1_Severe', 'V00WPRKN2_Severe', 'V00P7RKFR_Daily', 'V00P7RKFR_Never',
'V00KSXRKN1_Never', 'V00KSXRKN1_Often', 'V00KSXRKN1_Rarely', 'V00DIRKN1_Mild',
'V00DIRKN1_None', 'V00DIRKN1_Severe', 'V00DIRKN2_Severe', 'V00DIRKN14_Moderate',
'V00DIRKN14_Severe', 'V00WPLKN1_Mild', 'V00WPLKN2_None', 'V00WPLKN2_Severe',
'V00DILKN14_Severe', 'P02KPNRCV_Yes', 'P01KPR30CV_Yes',
'P01KPACTCV_No_Limits_or_avoidance', 'P01KPA30CV_Yes',
'V00P7RKRCV_10__Pain_as_bad_as_you_can_imagine', 'V00P7RKRCV_7', 'V00P7RKRCV_8',
'V00P7RKRCV_9', 'V00P7LKRCV_10__Pain_as_bad_as_you_can_imagine', 'V00P7LKRCV_3',
'V00P7LKRCV_9', 'P01PMRKRCV_3', 'P01PMRKRCV_4', 'P01PMRKRCV_7', 'P01PMRKRCV_9',
'P01PMRKRCV_No_pain', 'P01PMLKRCV_2', 'P01BL12SXR_Neither',
'P01BL12SXR_SV_only', 'P01KPMED_Yes', 'P01KSURGL_Yes', 'V00POLYRH_Yes',
'P01OTARTCV_Yes', 'V00FALLCV_One', 'V00HYINJCV_Yes', 'V00ACUSCV_Yes',
'V00DIETCV_Yes', 'V00CAPSNCV_Yes', 'V00YOGACV_Yes', 'V00RELACV_Yes',
'P01RASTASV_DK_to_RA_inflam_arth__no_dk_to_meds',
'P01RASTASV_Report_RA_inflam_arth__dk_to_all_meds',
'P01RASTASV_Report_RA_inflam_arth__no_dk_to_meds', 'V00LFXPN_Yes',
'V00EKRSL_Yes', 'V00VITDCV_No_vitamins_minerals_taken_in_past_year',
'V00CALCMCV_Every_day', 'V00CALCMCV_No_vitamins_minerals_taken_in_past_year',
'V00VIT1_Yes', 'V00RKEFFPT_Too_tender_to_examine', 'V00RKEFFPT_Yes',
'V00LKEFFPT_Yes']
```

[ ]:
```
# And we remove those features from the dataset

df2_cluster0 = df_cluster_0.copy()
for col in low_coef_col:
```

```
  df2_cluster0 = df2_cluster0.drop(col, axis = 1)
df2_cluster0.shape
```

[ ]: (297, 157)

Training

[ ]:
```
## Logistic Regression with sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import metrics

x_train_cluster_0_2 = df2_cluster0.drop('cumulative_outcome', axis='columns')
y_train_cluster_0_2 = df2_cluster0['cumulative_outcome']

logreg0_2 = LogisticRegression()
logreg0_2.fit(x_train_cluster_0_2, y_train_cluster_0_2.ravel())
logreg0_2.score(x_train_cluster_0_2, y_train_cluster_0_2.ravel())
```

[ ]: 0.7474747474747475

Testing

[ ]:
```
# And we remove the features from the dataset

df2_test_cluster0 = df_test_cluster_0.copy()
for col in low_coef_col:
  df2_test_cluster0 = df2_test_cluster0.drop(col, axis = 1)
df2_test_cluster0.shape
```

[ ]: (119, 157)

[ ]:
```
#Predictions

x_test_cluster_0_2 = df2_test_cluster0.drop('cumulative_outcome',␣
 ↪axis='columns')
y_test_cluster_0_2 = df2_test_cluster0['cumulative_outcome']

logreg0_2.predict(x_test_cluster_0_2)
logreg0_2.score(x_test_cluster_0_2, y_test_cluster_0_2)
```

[ ]: 0.5966386554621849

[ ]:
```
## Coefficients of the logistic regression
logreg0_2.coef_
```

[ ]:
```
array([[ 0.13780324, -0.12352875,  0.31309742, -0.1820646 ,  0.30969361,
        -0.45251176, -0.31429697, -0.13089856,  0.36566243, -0.42813797,
         0.29278037,  0.24871173, -0.21602049, -0.12069133, -0.21702404,
```

```
 0.42919984,  0.65025787,  0.12862233, -0.34686778,  0.21197352,
-0.53802045, -0.24276687, -0.11393682,  0.30100911, -0.22726107,
 0.23792969, -0.11350771, -0.1265626 , -0.18489191, -0.17247442,
 0.27247423, -0.13183365,  0.11321266,  0.48593589, -0.49434005,
 0.41090723, -0.24924106,  0.35763799,  0.83873242, -0.13299622,
 0.19537591, -0.87388998,  0.10824349,  0.35390605, -0.397907  ,
 0.14733212,  0.30031   ,  0.33215434, -0.41811245, -0.16557908,
 0.84087584, -0.77144852, -0.11023735, -0.20246612,  0.23774995,
 0.18165088, -0.30422907, -0.41188465,  0.49273809,  0.3040687 ,
-0.70611086,  0.11558581, -1.04713194, -0.21702404,  0.42180154,
 0.5687488 , -0.35433547,  0.29482442, -0.29501539,  0.3441796 ,
 0.27752642,  0.20787636,  0.15612093, -1.15478075,  0.45468172,
-0.45198897,  0.16042803,  0.10075829,  0.14374137, -0.21245217,
-0.42116809,  0.22525458,  0.16144616, -0.21966164,  0.30329596,
-0.46630384,  0.38782239,  0.32663529, -0.31927576, -0.37244765,
-0.10133901,  0.24157162,  0.31857075,  1.01871311,  0.22161631,
-0.61064813,  0.2579599 , -0.46093918, -0.24212327,  0.77561051,
-0.1714534 ,  0.19533629,  0.37721038,  0.25069742, -0.11769287,
-0.62059947,  0.49035605,  0.62683037,  0.19148903, -0.2365919 ,
-0.53136455,  0.64918444,  0.1516107 ,  0.79560517,  0.16612914,
 0.32118037,  0.48246326,  0.17894212,  0.50998506,  0.45505717,
-0.68655798,  0.28907089, -0.3842792 , -0.22356564, -0.218149  ,
-0.40887369, -0.34563782, -0.26930702, -0.47793844,  0.19125919,
 0.43269161, -0.10573165,  0.19794557, -0.30734029,  0.25220834,
 0.49048037,  0.47856133, -0.14112913,  0.36020569, -0.34978228,
-0.13932292, -0.38968953, -0.50829898,  0.29790714, -0.45394757,
-0.58840486,  0.21506685, -0.26700317,  0.58255989,  0.61568069,
 1.14297476,  0.28167535,  0.58001871, -0.25549436,  0.18800763,
 0.57916534]])
```

5.1.2 Features selection: P-values

[ ]:
```
#We look for the feature which have a p-value too high

high_pvalue_col = []
for i in range(len(p_values0)):
  if p_values0[i] > 0.5:
    high_pvalue_col.append(p_values0.index[i])
print(len(high_pvalue_col))
print(high_pvalue_col)
```

```
99
['V00WOMTSR', 'V00WT25KG', 'V00BPDIAS', 'V00DTCHOL', 'P01BMI', 'V00PASE',
'P01KPNREV_Yes', 'P01KPNLEV_Yes', 'P01HPNR12_Yes', 'P01HPNL12_Yes',
'V00SF2_Yes__limited_a_little', 'V00SF8_Moderately', 'V00WPRKN2_Mild',
'V00WPRKN2_Moderate', 'V00WPRKN2_None', 'V00P7RKFR_Monthly', 'V00P7RKFR_Never',
'V00KSXRKN1_Never', 'V00KSXRKN1_Rarely', 'V00DIRKN2_None', 'V00WPLKN1_Mild',
'V00WPLKN1_Moderate', 'V00WPLKN2_Mild', 'V00WPLKN2_Moderate', 'V00WPLKN2_None',
```

```
'V00P7LKFR_Daily', 'V00P7LKFR_Monthly', 'V00P7LKFR_Never', 'V00P7LKFR_Weekly',
'V00KSXLKN1_Often', 'V00KSXLKN1_Sometimes', 'V00DILKN1_Mild',
'V00DILKN1_Moderate', 'V00DILKN1_None', 'V00DILKN2_Moderate', 'V00DILKN14_Mild',
'V00DILKN14_None', 'V00KOOSFX5_Mild', 'V00KOOSFX5_Moderate', 'V00KOOSFX5_None',
'P02KPNRCV_Yes', 'P02KPNLCV_Yes', 'P01KPR30CV_Yes',
'P01KPACTCV_No_Limits_or_avoidance', 'P01HPR12CV_Yes', 'P01HPL12CV_Yes',
'P01KPA30CV_Yes', 'V00P7LKRCV_2', 'V00P7LKRCV_5', 'V00P7LKRCV_7',
'V00P7LKRCV_No_pain', 'P01PMRKRCV_6', 'P01PMLKRCV_3', 'P01PMLKRCV_6',
'P01PMLKRCV_7', 'P01PMLKRCV_8', 'P01BL12SXL_IEI_only', 'P01BL12SXL_Neither',
'P01BL12SXL_SV_only', 'P01BL12SXR_IEI_only', 'P01BL12SXR_Neither',
'P01BL12SXR_SV_only', 'P01LKP30CV_Yes', 'P01RKP30CV_Yes', 'P02KSURG_Yes',
'P01RAIA_Yes', 'P01ARTHOTH_Yes', 'P01ARTDOC_Yes', 'P01INJR_Yes',
'P01KSURGR_Yes', 'P01KSURGL_Yes', 'V00BONEFX_Yes', 'P01OAHIPCV_Yes',
'P01OTARTCV_Yes', 'P01KPMEDCV_Yes', 'V00FALLCV_None', 'V00FALLCV_One',
'V00ACUSCV_Yes', 'V00VITMCV_Yes', 'V00RUBCV_Yes', 'V00CAPSNCV_Yes',
'V00BRACCV_Yes', 'V00RELACV_Yes', 'V00SPIRCV_Yes',
'P01RASTASV_Does_not_report_RA_inflam_arth',
'P01RASTASV_Report_RA_inflam_arth__no_to_all_meds', 'V00OTHCAMC_Yes',
'V00DISCOMF_Yes', 'V00REXPN_Yes',
'V00VITDCV_No_vitamins_minerals_taken_in_past_year',
'V00CALCMCV_4_6_days_per_week', 'V00CALCMCV_A_few_days_per_month',
'V00CALCMCV_No_vitamins_minerals_taken_in_past_year', 'P01FAMKR_Yes',
'V00RKEFFB_Yes', 'V00RKEFFPT_Yes', 'V00LKEFFB_Yes',
'V00LKEFFPT_Too_tender_to_examine', 'V00RKRFXPN_Yes']
```

[ ]:
```
# And we remove those features from the dataset

df3_cluster0 = df2_cluster0.copy()
for col in high_pvalue_col:
  if col in df3_cluster0.columns:
    df3_cluster0 = df3_cluster0.drop(col, axis = 1)
df3_cluster0.shape
```

[ ]:
```
(297, 79)
```

Training

[ ]:
```
## Logistic Regression with sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import metrics

x_train_cluster_0_3 = df3_cluster0.drop('cumulative_outcome', axis='columns')
y_train_cluster_0_3 = df3_cluster0['cumulative_outcome']

logreg0_3 = LogisticRegression()
logreg0_3.fit(x_train_cluster_0_3, y_train_cluster_0_3.ravel())
logreg0_3.score(x_train_cluster_0_3, y_train_cluster_0_3.ravel())
```

[ ]: 0.7104377104377104

Testing

[ ]:
```
# And we remove the features from the dataset

df3_test_cluster0 = df2_test_cluster0.copy()
for col in high_pvalue_col:
  if col in df3_test_cluster0.columns:
    df3_test_cluster0 = df3_test_cluster0.drop(col, axis = 1)
df3_test_cluster0.shape
```

[ ]: (119, 79)

[ ]:
```
#Predictions

x_test_cluster_0_3 = df3_test_cluster0.drop('cumulative_outcome',␣
↪axis='columns')
y_test_cluster_0_3 = df3_test_cluster0['cumulative_outcome']

logreg0_3.predict(x_test_cluster_0_3)
logreg0_3.score(x_test_cluster_0_3, y_test_cluster_0_3.ravel())
```

[ ]: 0.6218487394957983

[ ]:
```
# We create two lists with the accuracy on the training and testing set for␣
↪each cluster, for further analysis
Accuracy_train2 = []
Accuracy_test2 = []
```

[ ]:
```
Accuracy_train2.append(logreg0_3.score(x_train_cluster_0_3, y_train_cluster_0_3.
↪ravel()))
Accuracy_test2.append(logreg0_3.score(x_test_cluster_0_3, y_test_cluster_0_3.
↪ravel()))
```

5.2 Cluster 1

5.2.1 Feature selection: coefficients value

[ ]:
```
#We look for the columns which have low coefficients' values

low_coef_col1 = []
for i in range(len(logreg1.coef_[0])):
  if abs(logreg1.coef_[0][i]) < 0.1:
    low_coef_col1.append(df_cluster_1.iloc[:,i].name)
print(len(low_coef_col1))
print(low_coef_col1)
```

```
103
['V00WOMTSR', 'P01KPACDCV', 'V00COMORB', 'V00BPSYS', 'V00LLWGT',
'P01KPACT30_Yes', 'V00SF8_Extremely', 'V00SF8_Moderately', 'V00WPRKN1_Mild',
'V00WPRKN1_Moderate', 'V00WPRKN1_None', 'V00WPRKN1_Severe',
'V00WPRKN2_Moderate', 'V00WPRKN2_Severe', 'V00P7RKFR_Daily', 'V00P7RKFR_Never',
'V00KSXRKN1_Never', 'V00KSXRKN1_Often', 'V00KSXRKN1_Rarely',
'V00KSXRKN1_Sometimes', 'V00DIRKN1_Moderate', 'V00DIRKN1_Severe',
'V00DIRKN2_Moderate', 'V00DIRKN2_Severe', 'V00DIRKN14_Mild',
'V00DIRKN14_Moderate', 'V00DIRKN14_None', 'V00DIRKN14_Severe', 'V00WPLKN1_Mild',
'V00WPLKN1_Moderate', 'V00WPLKN1_None', 'V00WPLKN1_Severe',
'V00WPLKN2_Moderate', 'V00WPLKN2_Severe', 'V00KSXLKN1_Often', 'V00DILKN1_Mild',
'V00DILKN1_Moderate', 'V00DILKN1_None', 'V00DILKN2_Mild', 'V00DILKN2_Moderate',
'V00DILKN2_None', 'V00DILKN2_Severe', 'V00DILKN14_Moderate',
'V00DILKN14_Severe', 'P01KPR30CV_Yes', 'P01KPL30CV_Yes', 'P01KPACTCV_Limits',
'V00P7RKRCV_10__Pain_as_bad_as_you_can_imagine', 'V00P7RKRCV_2', 'V00P7RKRCV_4',
'V00P7RKRCV_5', 'V00P7RKRCV_6', 'V00P7RKRCV_7', 'V00P7RKRCV_9',
'V00P7RKRCV_No_pain', 'V00P7LKRCV_10__Pain_as_bad_as_you_can_imagine',
'V00P7LKRCV_4', 'V00P7LKRCV_6', 'V00P7LKRCV_7', 'V00P7LKRCV_8', 'V00P7LKRCV_9',
'P01PMRKRCV_10__Pain_as_bad_as_you_can_imagine', 'P01PMRKRCV_4', 'P01PMRKRCV_5',
'P01PMRKRCV_6', 'P01PMRKRCV_7', 'P01PMRKRCV_8', 'P01PMRKRCV_9',
'P01PMLKRCV_10__Pain_as_bad_as_you_can_imagine', 'P01PMLKRCV_3', 'P01PMLKRCV_6',
'P01PMLKRCV_7', 'P01PMLKRCV_9', 'P01BL12SXR_SV_only', 'P02KSURG_Yes',
'P01KPMED_Yes', 'P01INJR_Yes', 'P01KSURGL_Yes', 'V00POLYRH_Yes',
'P01ARTDRCV_Yes', 'V00FALLCV_Six_or_more', 'V00FALLCV_Two_or_Three',
'V00HYINJCV_Yes', 'V00ACUSCV_Yes', 'V00HOMECV_Yes', 'V00MASSCV_Yes',
'V00HERBCV_Yes', 'V00RELACV_Yes',
'P01RASTASV_DK_to_RA_inflam_arth__no_dk_to_meds',
'P01RASTASV_Report_RA_inflam_arth__dk_to_all_meds',
'P01RASTASV_Report_RA_inflam_arth__no_dk_to_meds', 'P02KSURGCV_Yes',
'V00OTHCAMC_Yes', 'V00OTHCAM_Yes', 'V00LFXPN_Yes', 'V00EKRSL_Yes',
'V00EKRSR_Yes', 'V00VITDCV_A_few_days_per_month',
'V00VITDCV_No_vitamins_minerals_taken_in_past_year',
'V00CALCMCV_No_vitamins_minerals_taken_in_past_year', 'P02WTGA_Yes',
'V00RKEFFPT_Too_tender_to_examine', 'V00LKEFFPT_Too_tender_to_examine']
```

[ ]:
```
# And we remove those features from the dataset

df2_cluster1 = df_cluster_1.copy()
for col in low_coef_col1:
  df2_cluster1 = df2_cluster1.drop(col, axis = 1)
df2_cluster1.shape
```

[ ]:
```
(327, 125)
```

Training

[ ]:
```
## Logistic Regression with sklearn
```

```
x_train_cluster_1_2 = df2_cluster1.drop('cumulative_outcome', axis='columns')
y_train_cluster_1_2 = df2_cluster1['cumulative_outcome']

logreg1_2 = LogisticRegression()
logreg1_2.fit(x_train_cluster_1_2, y_train_cluster_1_2.ravel())
logreg1_2.score(x_train_cluster_1_2, y_train_cluster_1_2.ravel())
```

[ ]: 0.7400611620795107

Testing

```
[ ]: # And we remove the features from the dataset

df2_test_cluster1 = df_test_cluster_1.copy()
for col in low_coef_col1:
  df2_test_cluster1 = df2_test_cluster1.drop(col, axis = 1)
df2_test_cluster1.shape
```

[ ]: (139, 125)

```
[ ]: # Prediction of the outcomes

x_test_cluster_1_2 = df2_test_cluster1.drop('cumulative_outcome',␣
 ↪axis='columns')
y_test_cluster_1_2 = df2_test_cluster1['cumulative_outcome']

logreg1_2.predict(x_test_cluster_1_2)
logreg1_2.score(x_test_cluster_1_2, y_test_cluster_1_2)
```

[ ]: 0.6330935251798561

```
[ ]: ## Coefficients of the logistic regression
logreg1_2.coef_
```

```
[ ]: array([[ 0.11866382, -0.53734453, -0.31565543, -0.56124246,  0.22022874,
        -0.25088347, -0.26372501, -0.38164911, -0.25530582, -0.14931521,
        -0.29431313,  0.15716768, -0.17017085,  0.39288304,  0.32391749,
        -0.15912997,  0.6892358 ,  0.48149547,  0.58620942,  0.38538849,
        -0.82089384,  0.69424342, -0.24647382,  0.25832092,  0.11560059,
        -0.26806769, -0.19527504,  0.16628885, -0.21201437,  0.17950329,
        -0.41211897,  0.35697439, -0.1298124 , -0.23534013,  0.48955001,
        -0.24234191, -0.21383291,  0.22784115,  0.27681953,  0.38808989,
        -0.28078718,  0.24184841,  0.38145921,  0.53368665, -0.64415755,
        -0.11749632,  0.26990591,  0.20093688, -0.42122286,  0.18765409,
         0.39976055,  0.42122286, -0.24296719,  0.38816362, -0.20657486,
        -0.27402539, -0.11642321,  0.28189559, -0.41937405, -0.14123131,
         0.25600266, -0.27278443,  0.36781609,  0.29321074, -0.18176625,
        -0.51348866,  0.3456739 , -0.23805852,  0.24456475,  0.58085386,
```

```
-0.31578572,  0.51909328, -0.16356982,  0.375715  , -0.3512818 ,
 1.01053083,  0.74497081, -0.12274634, -0.734093  ,  0.45980949,
 0.49600076, -0.14272482,  0.76195786, -0.19015122, -0.09476004,
 0.13658967,  0.29196425, -0.18395963,  0.71863669,  0.17455921,
-0.30023369,  0.31240121, -0.19783508, -0.33338771,  0.25771137,
-0.17990515,  0.71525795,  0.19337261, -0.27890883,  0.22745908,
 0.73509442,  0.58338437, -0.14861035, -0.35648618, -0.20638144,
 0.17085649, -0.31105949,  0.16846248, -0.39837738, -0.568599  ,
 0.7005078 , -0.45196806, -0.20034701,  0.11389897,  0.13045642,
 0.17709043,  0.21732952, -0.11825334, -0.44670338, -0.18159715,
-0.10547958, -0.24014728, -0.38884607, -0.17138452]])
```

5.2.2 Features selection: P-values

[ ]:
```
#We look for the feature which have a p-value too high

high_pvalue_col1 = []
for i in range(len(p_values1)):
  if p_values1[i] > 0.6:
    high_pvalue_col1.append(p_values1.index[i])
print(len(high_pvalue_col1))
print(high_pvalue_col1)
```

```
71
['V00WT25KG', 'P01BMI', 'P01KPNLEV_Yes', 'P01HPNR12_Yes', 'P01HPNL12_Yes',
'V00SF2_Yes__limited_a_little', 'V00SF8_Moderately', 'V00WPRKN2_Moderate',
'V00P7RKFR_Never', 'V00KSXRKN1_Never', 'V00KSXRKN1_Rarely', 'V00DIRKN2_None',
'V00WPLKN1_Mild', 'V00WPLKN2_Mild', 'V00WPLKN2_None', 'V00P7LKFR_Daily',
'V00P7LKFR_Monthly', 'V00KSXLKN1_Often', 'V00KSXLKN1_Sometimes',
'V00DILKN1_Mild', 'V00DILKN1_Moderate', 'V00DILKN1_None', 'V00DILKN14_Mild',
'P02KPNLCV_Yes', 'P01KPR30CV_Yes', 'P01KPACTCV_No_Limits_or_avoidance',
'P01HPR12CV_Yes', 'P01HPL12CV_Yes', 'P01KPA30CV_Yes', 'V00P7LKRCV_2',
'V00P7LKRCV_5', 'V00P7LKRCV_7', 'V00P7LKRCV_No_pain', 'P01PMRKRCV_6',
'P01PMLKRCV_3', 'P01PMLKRCV_7', 'P01PMLKRCV_8', 'P01BL12SXL_IEI_only',
'P01BL12SXL_Neither', 'P01BL12SXR_Neither', 'P01BL12SXR_SV_only',
'P01LKP30CV_Yes', 'P01RKP30CV_Yes', 'P01RAIA_Yes', 'P01ARTHOTH_Yes',
'P01INJR_Yes', 'P01KSURGL_Yes', 'V00BONEFX_Yes', 'P01OAHIPCV_Yes',
'P01OTARTCV_Yes', 'P01KPMEDCV_Yes', 'V00FALLCV_None', 'V00FALLCV_One',
'V00ACUSCV_Yes', 'V00VITMCV_Yes', 'V00CAPSNCV_Yes', 'V00BRACCV_Yes',
'V00RELACV_Yes', 'V00SPIRCV_Yes', 'P01RASTASV_Does_not_report_RA_inflam_arth',
'V00OTHCAMC_Yes', 'V00DISCOMF_Yes', 'V00REXPN_Yes',
'V00VITDCV_No_vitamins_minerals_taken_in_past_year',
'V00CALCMCV_4_6_days_per_week',
'V00CALCMCV_No_vitamins_minerals_taken_in_past_year', 'P01FAMKR_Yes',
'V00RKEFFB_Yes', 'V00RKEFFPT_Yes', 'V00LKEFFPT_Too_tender_to_examine',
'V00RKRFXPN_Yes']
```

[ ]:
```
# And we remove those features from the dataset

df3_cluster1 = df2_cluster1.copy()
for col in high_pvalue_col1:
  if col in df3_cluster1.columns:
    df3_cluster1 = df3_cluster1.drop(col, axis = 1)
df3_cluster1.shape
```

[ ]:
```
(327, 78)
```

Training

[ ]:
```
## Logistic Regression with sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import metrics

x_train_cluster_1_3 = df3_cluster1.drop('cumulative_outcome', axis='columns')
y_train_cluster_1_3 = df3_cluster1['cumulative_outcome']

logreg1_3 = LogisticRegression()
logreg1_3.fit(x_train_cluster_1_3, y_train_cluster_1_3.ravel())
logreg1_3.score(x_train_cluster_1_3, y_train_cluster_1_3.ravel())
```

[ ]:
```
0.7186544342507645
```

Testing

[ ]:
```
#We remove the features from the testing data set, as we did for the training␣
 ↪data set

df3_test_cluster1 = df2_test_cluster1.copy()
for col in high_pvalue_col1:
  if col in df3_test_cluster1.columns:
    df3_test_cluster1 = df3_test_cluster1.drop(col, axis = 1)
df3_test_cluster1.shape
```

[ ]:
```
(139, 78)
```

[ ]:
```
# Prediction of the outcomes

x_test_cluster_1_3 = df3_test_cluster1.drop('cumulative_outcome',␣
 ↪axis='columns')
y_test_cluster_1_3 = df3_test_cluster1['cumulative_outcome']

logreg1_3.predict(x_test_cluster_1_3)
logreg1_3.score(x_test_cluster_1_3, y_test_cluster_1_3.ravel())
```

[ ]:
```
0.6330935251798561
```

[ ]:
```
Accuracy_train2.append(logreg1_3.score(x_train_cluster_1_3, y_train_cluster_1_3.
 ↪ravel()))
Accuracy_test2.append(logreg1_3.score(x_test_cluster_1_3, y_test_cluster_1_3.
 ↪ravel()))
```

5.3 Cluster 2

5.3.1 Feature selection: coefficients value

[ ]:
```
#We look for the columns which have low coefficients' values

low_coef_col2 = []
for i in range(len(logreg2.coef_[0])):
  if abs(logreg2.coef_[0][i]) < 0.1:
    low_coef_col2.append(df_cluster_2.iloc[:,i].name)
print(len(low_coef_col2))
print(low_coef_col2)
```

```
71
['V00WOMTSR', 'P01KPACDCV', 'V00BPSYS', 'P01KPACT30_Yes', 'V00SF8_Extremely',
'V00SF8_Quite_a_bit', 'V00WPRKN1_Moderate', 'V00WPRKN1_Severe',
'V00WPRKN2_Moderate', 'V00WPRKN2_Severe', 'V00P7RKFR_Weekly',
'V00KSXRKN1_Often', 'V00DIRKN1_Mild', 'V00DIRKN2_Mild', 'V00DIRKN14_None',
'V00DIRKN14_Severe', 'V00WPLKN1_Moderate', 'V00WPLKN1_Severe', 'V00WPLKN2_None',
'V00P7LKFR_Daily', 'V00P7LKFR_Weekly', 'V00KSXLKN1_Rarely',
'V00DILKN1_Moderate', 'V00DILKN1_Severe', 'V00DILKN2_Severe',
'V00DILKN14_Severe', 'V00KOOSFX5_Moderate', 'V00KOOSFX5_None',
'P01KPACTCV_Limits', 'V00P7RKRCV_10__Pain_as_bad_as_you_can_imagine',
'V00P7RKRCV_5', 'V00P7RKRCV_6', 'V00P7LKRCV_10__Pain_as_bad_as_you_can_imagine',
'V00P7LKRCV_3', 'V00P7LKRCV_5', 'V00P7LKRCV_9',
'P01PMRKRCV_10__Pain_as_bad_as_you_can_imagine', 'P01PMRKRCV_3', 'P01PMRKRCV_4',
'P01PMRKRCV_5', 'P01PMRKRCV_9', 'P01PMLKRCV_10__Pain_as_bad_as_you_can_imagine',
'P01PMLKRCV_5', 'P01PMLKRCV_7', 'P01BL12SXR_IEI_only', 'P01KPMED_Yes',
'P01INJR_Yes', 'V00POLYRH_Yes', 'V00FALLCV_One', 'V00FALLCV_Two_or_Three',
'V00HYINJCV_Yes', 'V00ACUTCV_Yes', 'V00RUBCV_Yes', 'V00RELACV_Yes',
'V00SPIRCV_Yes', 'P01RASTASV_DK_to_RA_inflam_arth__no_dk_to_meds',
'P01RASTASV_Report_RA_inflam_arth__dk_to_all_meds',
'P01RASTASV_Report_RA_inflam_arth__no_dk_to_meds', 'P02KSURGCV_Yes',
'V00OTHCAMC_Yes', 'V00OTHCAM_Yes', 'V00DISCOMF_Yes', 'V00LFXCOMP_Yes',
'V00EKRSL_Yes', 'V00EKRSR_Yes',
'V00VITDCV_No_vitamins_minerals_taken_in_past_year',
'V00CALCMCV_4_6_days_per_week',
'V00CALCMCV_No_vitamins_minerals_taken_in_past_year',
'V00RKEFFPT_Too_tender_to_examine', 'V00LKEFFPT_Too_tender_to_examine',
'V00LKRFXPN_Yes']
```

[ ]:
```
# And we remove those features from the dataset

df2_cluster2 = df_cluster_2.copy()
for col in low_coef_col2:
  df2_cluster2 = df2_cluster2.drop(col, axis = 1)
df2_cluster2.shape
```

[ ]:
```
(331, 157)
```

Training

[ ]:
```
## Logistic Regression with sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import metrics

x_train_cluster_2_2 = df2_cluster2.drop('cumulative_outcome', axis='columns')
y_train_cluster_2_2 = df2_cluster2['cumulative_outcome']

logreg2_2 = LogisticRegression()
logreg2_2.fit(x_train_cluster_2_2, y_train_cluster_2_2.ravel())
logreg2_2.score(x_train_cluster_2_2, y_train_cluster_2_2.ravel())
```

[ ]:
```
0.716012084592145
```

Testing

[ ]:
```
#We remove the features from the testing data set, as we did for the training␣
 ↪data set

df2_test_cluster2 = df_test_cluster_2.copy()
for col in low_coef_col2:
  df2_test_cluster2 = df2_test_cluster2.drop(col, axis = 1)
df2_test_cluster2.shape
```

[ ]:
```
(145, 157)
```

[ ]:
```
# Prediction of the outcomes

x_test_cluster_2_2 = df2_test_cluster2.drop('cumulative_outcome',␣
 ↪axis='columns')
y_test_cluster_2_2 = df2_test_cluster2['cumulative_outcome']

logreg2_2.predict(x_test_cluster_2_2)
logreg2_2.score(x_test_cluster_2_2, y_test_cluster_2_2)
```

[ ]:
```
0.5793103448275863
```

[ ]:
```
## Coefficients of the logistic regression
logreg2_2.coef_
```

[ ]:
```
array([[-0.20653967,  0.14188415, -0.8682224 , -0.38420269,  0.17233682,
        -0.31088509, -0.17374885,  0.25889259, -0.39032669, -0.44763211,
         0.84911294, -1.03721832, -0.25411045, -0.27026343, -0.37850017,
        -0.44266566,  0.26267006,  0.26970853, -0.48196114,  0.50880162,
         0.4979326 , -0.54814455, -0.51524177, -0.34872926, -0.50166552,
         0.45325761, -0.27990927,  0.3452398 , -0.23474974,  0.43913556,
        -0.14659644,  0.17195068, -0.68819678,  0.17060989,  0.18617539,
        -0.20299902,  0.20167788,  0.19060158, -0.09805756,  0.18107369,
        -0.16441247,  0.22832814, -0.4266106 ,  0.37321044,  0.28802545,
        -0.38134691, -0.11328034,  0.14531357, -0.11940651, -0.1864824 ,
         0.18606357, -0.10714936,  0.32629706, -0.23554871, -0.21594824,
        -0.19476548,  0.3025416 , -0.36465834,  0.18654753,  0.22168088,
        -0.23347736,  0.14094208, -0.1741984 , -0.28036218, -0.14802591,
        -0.21628802, -0.25584814,  0.32213827,  0.55712812,  0.25584814,
        -0.16606884,  0.31168189,  0.48012278, -0.33665813, -0.15250562,
         0.3574604 , -0.17861454,  0.30701275,  0.26425502,  0.12404656,
        -0.47472609, -0.22662656, -0.11195594, -0.55307038, -0.28818454,
        -0.33733648,  0.18655986,  0.40139177, -0.56583685, -0.10120801,
        -0.33366883,  0.15491107, -0.10910251,  0.30258068,  0.24163251,
         0.44409733,  0.49791178, -0.42562496,  0.55478417, -0.71159283,
         0.16035731,  0.11427731,  0.46542882,  0.23627927,  0.34791004,
         0.51839746,  0.46381477,  0.18169578,  0.36667559, -0.56148195,
        -0.18071151,  0.11303013, -0.24923308,  0.24315931,  0.19185282,
         1.03030752,  0.65448006,  0.26184057, -0.49498087,  0.2190868 ,
         0.45828727,  0.56921573,  0.23729072, -0.45693699,  0.33724703,
        -0.28603048,  0.94353138,  0.22464066,  0.26851268, -0.30778614,
         0.48623001,  0.10821015, -0.70085527,  0.58738226, -0.16248826,
        -0.38771392,  0.35299362,  0.16114648,  0.11364694,  0.5599995 ,
        -0.27910921, -0.17472573, -0.45038937,  0.4501078 ,  0.14599725,
        -0.33478622, -0.16073489, -0.29347463,  0.33440507,  0.53296764,
         1.08520602, -0.18444161,  0.49000634, -0.5015506 ,  0.54523501,
        -0.40120901]])
```

5.3.2 Features selection: P-values

[ ]:
```
#We look for the feature which have a p-value too high

high_pvalue_col2 = []
for i in range(len(p_values2)):
  if p_values2[i] > 0.7:
    high_pvalue_col2.append(p_values2.index[i])
print(len(high_pvalue_col2))
print(high_pvalue_col2)
```

```
54
```

```
['V00WT25KG', 'P01BMI', 'P01HPNL12_Yes', 'V00SF2_Yes__limited_a_little',
'V00SF8_Moderately', 'V00WPRKN2_Moderate', 'V00KSXRKN1_Never', 'V00DIRKN2_None',
'V00WPLKN1_Mild', 'V00WPLKN2_Mild', 'V00WPLKN2_None', 'V00P7LKFR_Daily',
'V00P7LKFR_Monthly', 'V00KSXLKN1_Often', 'V00KSXLKN1_Sometimes',
'V00DILKN1_Mild', 'V00DILKN1_Moderate', 'V00DILKN14_Mild', 'P02KPNLCV_Yes',
'P01KPR30CV_Yes', 'P01KPACTCV_No_Limits_or_avoidance', 'P01HPR12CV_Yes',
'P01KPA30CV_Yes', 'V00P7LKRCV_2', 'V00P7LKRCV_5', 'V00P7LKRCV_7',
'V00P7LKRCV_No_pain', 'P01PMLKRCV_7', 'P01PMLKRCV_8', 'P01BL12SXL_Neither',
'P01BL12SXR_SV_only', 'P01LKP30CV_Yes', 'P01RKP30CV_Yes', 'P01INJR_Yes',
'V00BONEFX_Yes', 'P01OAHIPCV_Yes', 'P01OTARTCV_Yes', 'P01KPMEDCV_Yes',
'V00FALLCV_None', 'V00ACUSCV_Yes', 'V00VITMCV_Yes', 'V00CAPSNCV_Yes',
'V00BRACCV_Yes', 'V00SPIRCV_Yes', 'P01RASTASV_Does_not_report_RA_inflam_arth',
'V00OTHCAMC_Yes', 'V00DISCOMF_Yes', 'V00REXPN_Yes',
'V00VITDCV_No_vitamins_minerals_taken_in_past_year',
'V00CALCMCV_4_6_days_per_week',
'V00CALCMCV_No_vitamins_minerals_taken_in_past_year', 'V00RKEFFB_Yes',
'V00RKEFFPT_Yes', 'V00RKRFXPN_Yes']
```

[ ]:
```
# And we remove those features from the dataset

df3_cluster2 = df2_cluster2.copy()
for col in high_pvalue_col2:
  if col in df3_cluster2.columns:
    df3_cluster2 = df3_cluster2.drop(col, axis = 1)
df3_cluster2.shape
```

[ ]:
```
(331, 116)
```

Training

[ ]:
```
## Logistic Regression with sklearn

x_train_cluster_2_3 = df3_cluster2.drop('cumulative_outcome', axis='columns')
y_train_cluster_2_3 = df3_cluster2['cumulative_outcome']

logreg2_3 = LogisticRegression()
logreg2_3.fit(x_train_cluster_2_3, y_train_cluster_2_3.ravel())
logreg2_3.score(x_train_cluster_2_3, y_train_cluster_2_3.ravel())
```

[ ]:
```
0.6918429003021148
```

Testing

[ ]:
```
#We remove the features from the testing data set, as we did for the training␣
 ↪data set

df3_test_cluster2 = df2_test_cluster2.copy()
for col in high_pvalue_col2:
  if col in df3_test_cluster2.columns:
```

```
    df3_test_cluster2 = df3_test_cluster2.drop(col, axis = 1)
df3_test_cluster2.shape
```

[ ]: (145, 116)

```
#Prediction of the outcomes

x_test_cluster_2_3 = df3_test_cluster2.drop('cumulative_outcome',␣
 ↪axis='columns')
y_test_cluster_2_3 = df3_test_cluster2['cumulative_outcome']

logreg2_3.predict(x_test_cluster_2_3)
logreg2_3.score(x_test_cluster_2_3, y_test_cluster_2_3)
```

[ ]: 0.5655172413793104

As the accuracy is better if we don't select the features according to their p-values, we are going to keep the accuracy with features selection only on the coefficient values.

```
Accuracy_train2.append(logreg2_2.score(x_train_cluster_2_2, y_train_cluster_2_2.
 ↪ravel()))
Accuracy_test2.append(logreg2_2.score(x_test_cluster_2_2, y_test_cluster_2_2))
```

5.4 Cluster 3

5.4.1 Feature selection: coefficients value

```
#We look for the columns which have low coefficients' values

low_coef_col3 = []
for i in range(len(logreg3.coef_[0])):
  if abs(logreg3.coef_[0][i]) < 0.18:
    low_coef_col3.append(df_cluster_3.iloc[:,i].name)
print(len(low_coef_col3))
print(low_coef_col3)
```

```
90
['V00WOMTSL', 'V00WOMTSR', 'P01KPACDCV', 'V00COMORB', 'V00WT25KG', 'V00WTMINKG',
'V00BPDIAS', 'V00BPSYS', 'V00LLWGT', 'P01KPNREV_Yes', 'P01KPNLEV_Yes',
'P01HPNL12_Yes', 'V00SF2_Yes__limited_a_little', 'V00P7RKFR_Daily',
'V00P7RKFR_Never', 'V00P7RKFR_Weekly', 'V00KSXRKN1_Never', 'V00DIRKN1_Mild',
'V00DIRKN1_Moderate', 'V00DIRKN1_None', 'V00DIRKN14_Moderate', 'V00WPLKN1_Mild',
'V00WPLKN1_Moderate', 'V00WPLKN1_None', 'V00WPLKN1_Severe',
'V00WPLKN2_Moderate', 'V00WPLKN2_Severe', 'V00P7LKFR_Daily',
'V00P7LKFR_Monthly', 'V00KSXLKN1_Never', 'V00DILKN1_Mild', 'V00DILKN1_Moderate',
'V00DILKN1_None', 'V00DILKN1_Severe', 'V00DILKN2_Mild', 'V00DILKN2_Moderate',
'V00DILKN2_None', 'V00DILKN2_Severe', 'V00DILKN14_Severe',
'V00KOOSFX5_Moderate', 'P02KPNRCV_Yes', 'P01KPACTCV_No_Limits_or_avoidance',
'P01KPA30CV_Yes', 'V00P7RKRCV_10__Pain_as_bad_as_you_can_imagine',
```

```
'V00P7RKRCV_2', 'V00P7RKRCV_3', 'V00P7RKRCV_5', 'V00P7RKRCV_No_pain',
'V00P7LKRCV_10__Pain_as_bad_as_you_can_imagine', 'V00P7LKRCV_5', 'V00P7LKRCV_6',
'V00P7LKRCV_7', 'V00P7LKRCV_8', 'P01PMRKRCV_5', 'P01PMLKRCV_3', 'P01PMLKRCV_5',
'P01PMLKRCV_7', 'P01PMLKRCV_8', 'P01PMLKRCV_No_pain', 'P01BL12SXL_IEI_only',
'P01BL12SXL_Neither', 'P01BL12SXR_IEI_only', 'P01BL12SXR_Neither',
'P01BL12SXR_SV_only', 'P01LKP30CV_Yes', 'P01KSURGL_Yes', 'V00HRTAT_Yes',
'V00BONEFX_Yes', 'P01OAHIPCV_Yes', 'V00FALLCV_One', 'V00ACUSCV_Yes',
'V00MASSCV_Yes', 'V00VITMCV_Yes', 'V00RUBCV_Yes', 'V00CAPSNCV_Yes',
'V00BRACCV_Yes', 'V00RELACV_Yes',
'P01RASTASV_DK_to_RA_inflam_arth__no_dk_to_meds',
'P01RASTASV_Report_RA_inflam_arth__dk_to_all_meds',
'P01RASTASV_Report_RA_inflam_arth__no_to_all_meds', 'V00OTHCAMC_Yes',
'V00OTHCAM_Yes', 'V00DISCOMF_Yes', 'V00LFXCOMP_Yes', 'V00LFXPN_Yes',
'V00VITDCV_Didn_t_take', 'V00VITDCV_Every_day',
'V00CALCMCV_A_few_days_per_month', 'V00RKEFFB_Yes',
'V00LKEFFPT_Too_tender_to_examine']
```

[ ]:
```
# And we remove those features from the dataset

df2_cluster3 = df_cluster_3.copy()
for col in low_coef_col3:
  df2_cluster3 = df2_cluster3.drop(col, axis = 1)
df2_cluster3.shape
```

[ ]:
```
(346, 138)
```

Training

[ ]:
```
## Logistic Regression with sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import metrics

x_train_cluster_3_2 = df2_cluster3.drop('cumulative_outcome', axis='columns')
y_train_cluster_3_2 = df2_cluster3['cumulative_outcome']

logreg3_2 = LogisticRegression()
logreg3_2.fit(x_train_cluster_3_2, y_train_cluster_3_2.ravel())
logreg3_2.score(x_train_cluster_3_2, y_train_cluster_3_2)
```

[ ]:
```
0.7861271676300579
```

Testing

[ ]:
```
#We remove the features from the testing data set, as we did for the training␣
 ↪data set

df2_test_cluster3 = df_test_cluster_3.copy()
for col in low_coef_col3:
```

```
  df2_test_cluster3 = df2_test_cluster3.drop(col, axis = 1)
df2_test_cluster3.shape
```

[ ]: (133, 138)

[ ]:
```
# Prediction of the outcomes

x_test_cluster_3_2 = df2_test_cluster3.drop('cumulative_outcome',␣
 ↪axis='columns')
y_test_cluster_3_2 = df2_test_cluster3['cumulative_outcome']

logreg3_2.predict(x_test_cluster_3_2)
logreg3_2.score(x_test_cluster_3_2, y_test_cluster_3_2)
```

[ ]: 0.6240601503759399

[ ]:
```
## Coefficients of the logistic regression
logreg3_2.coef_
```

[ ]:
```
array([[-0.81843819,  0.1759312 ,  0.27915353,  0.42073369,  0.72852567,
         0.48985625,  0.5952005 , -0.46574876, -0.34663273,  0.65185128,
         0.73996883,  0.61159396,  0.24649259,  0.39742573,  0.81160663,
        -0.22707824,  0.55456893, -0.36542151,  0.74799499, -0.24594731,
         0.35463295, -0.21343534, -0.43319438,  0.28481571,  0.61008734,
        -0.27782627,  0.48274646, -0.27424002, -0.64687229,  0.56535148,
         0.68989024, -0.20111412, -0.5240446 , -0.25076255,  0.29913526,
        -0.22104082,  0.36136808,  0.18934698, -0.25772564, -0.29176459,
         0.38711738,  0.26215332,  0.42324349, -0.42669603,  0.3779483 ,
         0.25174105, -0.46897425,  0.19395774, -0.54966741,  0.39580175,
        -0.27316125, -0.85599343, -0.19154636, -0.34663273,  0.22053575,
         0.40635756,  0.29995479, -0.1997922 ,  0.45457274, -0.55428282,
         0.61057898,  0.21052155,  0.47074774, -0.49929609,  0.20705961,
        -0.6335064 ,  0.47461628, -0.63632345, -0.55063183,  0.34350441,
         0.46313383, -0.29643128,  0.22058942,  0.83144031,  0.30975358,
         0.34372591, -0.37079635,  0.28460554, -0.40136674,  0.31061096,
         0.23010807, -0.30089087,  0.27253304, -1.03129005,  0.27728749,
         0.95556531,  0.20697003,  0.2902726 ,  0.7621314 , -1.14846587,
         0.54923527, -0.45413371,  0.8724367 ,  0.5670986 ,  0.51131339,
         0.54741432,  0.40499413,  0.21654971,  0.37515536, -0.50525959,
         0.2819424 , -0.27926251, -0.23974179,  0.6369503 ,  0.58858175,
         0.25152266, -0.68934999,  0.79201079, -0.21168355, -0.30640829,
        -0.20364885,  0.47909066,  0.20989425, -1.0528691 ,  0.31109881,
         0.44423353,  0.29556159,  0.35104277,  0.3444624 , -0.22461762,
         0.6157837 ,  0.47488205, -0.33644671,  0.21460675, -0.24127023,
         0.52403228, -0.33644671,  0.54386788,  0.34140708, -0.18400662,
        -0.39402967, -0.28565867,  0.23179907,  0.28852099, -0.26266774,
        -0.58082861, -0.71368975]])
```

5.4.2 Features selection: P-values

[ ]:
```
#We look for the feature which have a p-value too high

high_pvalue_col3 = []
for i in range(len(p_values3)):
  if p_values3[i] > 0.5:
    high_pvalue_col3.append(p_values3.index[i])
print(len(high_pvalue_col3))
print(high_pvalue_col3)
```

```
99
['V00WOMTSR', 'V00WT25KG', 'V00BPDIAS', 'V00DTCHOL', 'P01BMI', 'V00PASE',
'P01KPNREV_Yes', 'P01KPNLEV_Yes', 'P01HPNR12_Yes', 'P01HPNL12_Yes',
'V00SF2_Yes__limited_a_little', 'V00SF8_Moderately', 'V00WPRKN2_Mild',
'V00WPRKN2_Moderate', 'V00WPRKN2_None', 'V00P7RKFR_Monthly', 'V00P7RKFR_Never',
'V00KSXRKN1_Never', 'V00KSXRKN1_Rarely', 'V00DIRKN2_None', 'V00WPLKN1_Mild',
'V00WPLKN1_Moderate', 'V00WPLKN2_Mild', 'V00WPLKN2_Moderate', 'V00WPLKN2_None',
'V00P7LKFR_Daily', 'V00P7LKFR_Monthly', 'V00P7LKFR_Never', 'V00P7LKFR_Weekly',
'V00KSXLKN1_Often', 'V00KSXLKN1_Sometimes', 'V00DILKN1_Mild',
'V00DILKN1_Moderate', 'V00DILKN1_None', 'V00DILKN2_Moderate', 'V00DILKN14_Mild',
'V00DILKN14_None', 'V00KOOSFX5_Mild', 'V00KOOSFX5_Moderate', 'V00KOOSFX5_None',
'P02KPNRCV_Yes', 'P02KPNLCV_Yes', 'P01KPR30CV_Yes',
'P01KPACTCV_No_Limits_or_avoidance', 'P01HPR12CV_Yes', 'P01HPL12CV_Yes',
'P01KPA30CV_Yes', 'V00P7LKRCV_2', 'V00P7LKRCV_5', 'V00P7LKRCV_7',
'V00P7LKRCV_No_pain', 'P01PMRKRCV_6', 'P01PMLKRCV_3', 'P01PMLKRCV_6',
'P01PMLKRCV_7', 'P01PMLKRCV_8', 'P01BL12SXL_IEI_only', 'P01BL12SXL_Neither',
'P01BL12SXL_SV_only', 'P01BL12SXR_IEI_only', 'P01BL12SXR_Neither',
'P01BL12SXR_SV_only', 'P01LKP30CV_Yes', 'P01RKP30CV_Yes', 'P02KSURG_Yes',
'P01RAIA_Yes', 'P01ARTHOTH_Yes', 'P01ARTDOC_Yes', 'P01INJR_Yes',
'P01KSURGR_Yes', 'P01KSURGL_Yes', 'V00BONEFX_Yes', 'P01OAHIPCV_Yes',
'P01OTARTCV_Yes', 'P01KPMEDCV_Yes', 'V00FALLCV_None', 'V00FALLCV_One',
'V00ACUSCV_Yes', 'V00VITMCV_Yes', 'V00RUBCV_Yes', 'V00CAPSNCV_Yes',
'V00BRACCV_Yes', 'V00RELACV_Yes', 'V00SPIRCV_Yes',
'P01RASTASV_Does_not_report_RA_inflam_arth',
'P01RASTASV_Report_RA_inflam_arth__no_to_all_meds', 'V00OTHCAMC_Yes',
'V00DISCOMF_Yes', 'V00REXPN_Yes',
'V00VITDCV_No_vitamins_minerals_taken_in_past_year',
'V00CALCMCV_4_6_days_per_week', 'V00CALCMCV_A_few_days_per_month',
'V00CALCMCV_No_vitamins_minerals_taken_in_past_year', 'P01FAMKR_Yes',
'V00RKEFFB_Yes', 'V00RKEFFPT_Yes', 'V00LKEFFB_Yes',
'V00LKEFFPT_Too_tender_to_examine', 'V00RKRFXPN_Yes']
```

[ ]:
```
# And we remove those features from the dataset

df3_cluster3 = df2_cluster3.copy()
for col in high_pvalue_col3:
  if col in df3_cluster3.columns:
```

```
    df3_cluster3 = df3_cluster3.drop(col, axis = 1)
df3_cluster3.shape
```

[ ]: (346, 88)

Training

[ ]:
```
## Logistic Regression with sklearn

x_train_cluster_3_3 = df3_cluster3.drop('cumulative_outcome', axis='columns')
y_train_cluster_3_3 = df3_cluster3['cumulative_outcome']

logreg3_3 = LogisticRegression()
logreg3_3.fit(x_train_cluster_3_3, y_train_cluster_3_3.ravel())
logreg3_3.score(x_train_cluster_3_3, y_train_cluster_3_3.ravel())
```

[ ]: 0.7254335260115607

Testing

[ ]:
```
#We remove the features from the testing data set, as we did for the training␣
 ↪data set

df3_test_cluster3 = df2_test_cluster3.copy()
for col in high_pvalue_col3:
  if col in df3_test_cluster3.columns:
    df3_test_cluster3 = df3_test_cluster3.drop(col, axis = 1)
df3_test_cluster3.shape
```

[ ]: (133, 88)

[ ]:
```
#Prediction of the outcomes

x_test_cluster_3_3 = df3_test_cluster3.drop('cumulative_outcome',␣
 ↪axis='columns')
y_test_cluster_3_3 = df3_test_cluster3['cumulative_outcome']

logreg3_3.predict(x_test_cluster_3_3)
logreg3_3.score(x_test_cluster_3_3, y_test_cluster_3_3)
```

[ ]: 0.6240601503759399

[ ]:
```
Accuracy_train2.append(logreg3_3.score(x_train_cluster_3_3, y_train_cluster_3_3.
 ↪ravel()))
Accuracy_test2.append(logreg3_3.score(x_test_cluster_3_3, y_test_cluster_3_3.
 ↪ravel()))
```

5.5 Cluster 4

5.5.1 Feature selection: coefficients value

[ ]:
```
#We look for the columns which have low coefficients' values

low_coef_col4 = []
for i in range(len(logreg4.coef_[0])):
  if abs(logreg4.coef_[0][i]) < 0.5:
    low_coef_col4.append(df_cluster_4.iloc[:,i].name)
print(len(low_coef_col4))
print(low_coef_col4)
```

```
189
['V00WOMTSL', 'V00WOMTSR', 'P01KPACDCV', 'V00COMORB', 'V00HT25MM', 'V00WT25KG',
'V00WTMAXKG', 'V00WTMINKG', 'V00BPSYS', 'V00DTVITD', 'V00DTCALC', 'V00PASE',
'V00LLWGT', 'V00RLWGT', 'P01KPNLEV_Yes', 'P01KPACT30_Yes', 'P01HPNR12_Yes',
'V00SF2_Yes__limited_a_little', 'V00SF2_Yes__limited_a_lot',
'V00SF3_Yes__limited_a_little', 'V00SF8_Extremely', 'V00SF8_Quite_a_bit',
'V00WPRKN1_Mild', 'V00WPRKN1_Moderate', 'V00WPRKN1_None', 'V00WPRKN1_Severe',
'V00WPRKN2_Mild', 'V00WPRKN2_Moderate', 'V00WPRKN2_None', 'V00WPRKN2_Severe',
'V00P7RKFR_Daily', 'V00P7RKFR_Monthly', 'V00P7RKFR_Never', 'V00P7RKFR_Weekly',
'V00KSXRKN1_Never', 'V00KSXRKN1_Often', 'V00KSXRKN1_Rarely',
'V00KSXRKN1_Sometimes', 'V00DIRKN1_Mild', 'V00DIRKN1_Moderate',
'V00DIRKN1_None', 'V00DIRKN1_Severe', 'V00DIRKN2_Mild', 'V00DIRKN2_Moderate',
'V00DIRKN2_None', 'V00DIRKN2_Severe', 'V00DIRKN14_Mild', 'V00DIRKN14_Moderate',
'V00DIRKN14_None', 'V00DIRKN14_Severe', 'V00WPLKN1_Mild', 'V00WPLKN1_Moderate',
'V00WPLKN1_None', 'V00WPLKN1_Severe', 'V00WPLKN2_Mild', 'V00WPLKN2_Moderate',
'V00WPLKN2_None', 'V00WPLKN2_Severe', 'V00P7LKFR_Daily', 'V00P7LKFR_Monthly',
'V00P7LKFR_Never', 'V00P7LKFR_Weekly', 'V00KSXLKN1_Never', 'V00KSXLKN1_Often',
'V00KSXLKN1_Rarely', 'V00KSXLKN1_Sometimes', 'V00DILKN1_Mild',
'V00DILKN1_Moderate', 'V00DILKN1_None', 'V00DILKN1_Severe', 'V00DILKN2_Mild',
'V00DILKN2_Moderate', 'V00DILKN2_None', 'V00DILKN2_Severe', 'V00DILKN14_Mild',
'V00DILKN14_Moderate', 'V00DILKN14_None', 'V00DILKN14_Severe',
'V00KOOSFX5_Moderate', 'V00KOOSFX5_None', 'V00KOOSFX5_Severe', 'P02KPNRCV_Yes',
'P02KPNLCV_Yes', 'P01KPR30CV_Yes', 'P01KPL30CV_Yes', 'P01KPACTCV_Limits',
'P01KPACTCV_No_Limits_or_avoidance', 'P01HPR12CV_Yes', 'P01KPA30CV_Yes',
'V00P7RKRCV_10__Pain_as_bad_as_you_can_imagine', 'V00P7RKRCV_2', 'V00P7RKRCV_4',
'V00P7RKRCV_5', 'V00P7RKRCV_6', 'V00P7RKRCV_7', 'V00P7RKRCV_8', 'V00P7RKRCV_9',
'V00P7RKRCV_No_pain', 'V00P7LKRCV_10__Pain_as_bad_as_you_can_imagine',
'V00P7LKRCV_2', 'V00P7LKRCV_4', 'V00P7LKRCV_5', 'V00P7LKRCV_6', 'V00P7LKRCV_7',
'V00P7LKRCV_8', 'V00P7LKRCV_9', 'V00P7LKRCV_No_pain',
'P01PMRKRCV_10__Pain_as_bad_as_you_can_imagine', 'P01PMRKRCV_2', 'P01PMRKRCV_3',
'P01PMRKRCV_4', 'P01PMRKRCV_5', 'P01PMRKRCV_6', 'P01PMRKRCV_7', 'P01PMRKRCV_8',
'P01PMLKRCV_10__Pain_as_bad_as_you_can_imagine', 'P01PMLKRCV_2', 'P01PMLKRCV_3',
'P01PMLKRCV_4', 'P01PMLKRCV_5', 'P01PMLKRCV_6', 'P01PMLKRCV_7', 'P01PMLKRCV_8',
'P01PMLKRCV_9', 'P01PMLKRCV_No_pain', 'P01BL12SXL_IEI_only',
'P01BL12SXL_Neither', 'P01BL12SXL_SV_only', 'P01BL12SXR_IEI_only',
'P01BL12SXR_Neither', 'P01BL12SXR_SV_only', 'P01LKP30CV_Yes', 'P01RKP30CV_Yes',
'P02KPMED_Yes', 'P01RAIA_Yes', 'P01ARTDOC_Yes', 'P01KPMED_Yes', 'P01INJR_Yes',
```

```
'P01KSURGR_Yes', 'V00HRTAT_Yes', 'V00STROKE_Yes', 'V00DIAB_Yes', 'V00RA_Yes',
'V00SMOKE_Yes', 'P02KPMEDCV_Yes', 'P01GOUTCV_Yes', 'P01OTARTCV_Yes',
'P01KPMEDCV_Yes', 'V00FALLCV_One', 'V00FALLCV_Six_or_more',
'V00FALLCV_Two_or_Three', 'V00HYINJCV_Yes', 'V00STINJCV_Yes', 'V00ACUTCV_Yes',
'V00ACUSCV_Yes', 'V00CHIRCV_Yes', 'V00HOMECV_Yes', 'V00MASSCV_Yes',
'V00VITMCV_Yes', 'V00RUBCV_Yes', 'V00CAPSNCV_Yes', 'V00HERBCV_Yes',
'V00RELACV_Yes', 'V00SPIRCV_Yes',
'P01RASTASV_DK_to_RA_inflam_arth__no_dk_to_meds',
'P01RASTASV_Report_RA_inflam_arth__dk_to_all_meds',
'P01RASTASV_Report_RA_inflam_arth__no_to_all_meds',
'P01RASTASV_Report_RA_inflam_arth__no_dk_to_meds', 'P02KSURGCV_Yes',
'V00LFXPN_Yes', 'V00EKRSL_Yes', 'V00EKRSR_Yes', 'V00RFXCOMP_Yes',
'V00VITDCV_A_few_days_per_month', 'V00VITDCV_Didn_t_take',
'V00VITDCV_No_vitamins_minerals_taken_in_past_year',
'V00CALCMCV_4_6_days_per_week', 'V00CALCMCV_A_few_days_per_month',
'V00CALCMCV_Didn_t_take', 'V00CALCMCV_Every_day',
'V00CALCMCV_No_vitamins_minerals_taken_in_past_year', 'V00VIT9_Yes',
'V00VIT1_Yes', 'P01FAMKR_Yes', 'P02WTGA_Yes', 'V00RKEFFPT_Yes',
'V00LKEFFPT_Too_tender_to_examine', 'V00RKRFXPN_Yes', 'V00LKRFXPN_Yes']
```

[ ]:
```
# And we remove those features from the dataset

df2_cluster4 = df_cluster_4.copy()
for col in low_coef_col4:
  df2_cluster4 = df2_cluster4.drop(col, axis = 1)
df2_cluster4.shape
```

[ ]:
```
(415, 39)
```

Training

[ ]:
```
## Logistic Regression with sklearn

x_train_cluster_4_2 = df2_cluster4.drop('cumulative_outcome', axis='columns')
y_train_cluster_4_2 = df2_cluster4['cumulative_outcome']

logreg4_2 = LogisticRegression()
logreg4_2.fit(x_train_cluster_4_2, y_train_cluster_4_2.ravel())
logreg4_2.score(x_train_cluster_4_2, y_train_cluster_4_2.ravel())
```

[ ]:
```
0.7228915662650602
```

Testing

[ ]:
```
#We remove the features from the testing data set, as we did for the training␣
 ↪data set

df2_test_cluster4 = df_test_cluster_4.copy()
for col in low_coef_col4:
```

```
  df2_test_cluster4 = df2_test_cluster4.drop(col, axis = 1)
df2_test_cluster4.shape
```

[ ]: (164, 39)

[ ]:
```
#Prediction of the outcomes

x_test_cluster_4_2 = df2_test_cluster4.drop('cumulative_outcome',␣
 ↪axis='columns')
y_test_cluster_4_2 = df2_test_cluster4['cumulative_outcome']

logreg4_2.predict(x_test_cluster_4_2)
logreg4_2.score(x_test_cluster_4_2, y_test_cluster_4_2)
```

[ ]: 0.5548780487804879

[ ]:
```
## Coefficients of the logistic regression
logreg4_2.coef_
```

[ ]:
```
array([[-0.4064876 , -0.43084041,  0.88107317, -0.83626109,  1.22986999,
        -0.72268921,  0.6481113 , -0.95515315,  0.6693788 ,  0.54205997,
         0.66363181,  0.82262607,  0.88756572,  0.53535651,  0.40951013,
         2.52054748,  0.98486286,  0.57348646,  1.75060699,  0.81298851,
         1.28935531,  0.752831  , -0.67548755,  0.77156828, -0.69465308,
         0.54896725,  0.49844191, -0.56228799, -0.7669594 ,  0.73437141,
         0.5777814 , -0.65031753, -1.1809342 ,  0.999373  ,  0.80906431,
         0.49435356, -0.93070592, -0.75613397]])
```

5.5.2 Features selection: P-values

[ ]:
```
#We look for the feature which have a p-value too high

high_pvalue_col4 = []
for i in range(len(p_values4)):
  if p_values4[i] > 0.6:
    high_pvalue_col4.append(p_values4.index[i])
print(len(high_pvalue_col4))
print(high_pvalue_col4)
```

```
71
['V00WT25KG', 'P01BMI', 'P01KPNLEV_Yes', 'P01HPNR12_Yes', 'P01HPNL12_Yes',
'V00SF2_Yes__limited_a_little', 'V00SF8_Moderately', 'V00WPRKN2_Moderate',
'V00P7RKFR_Never', 'V00KSXRKN1_Never', 'V00KSXRKN1_Rarely', 'V00DIRKN2_None',
'V00WPLKN1_Mild', 'V00WPLKN2_Mild', 'V00WPLKN2_None', 'V00P7LKFR_Daily',
'V00P7LKFR_Monthly', 'V00KSXLKN1_Often', 'V00KSXLKN1_Sometimes',
'V00DILKN1_Mild', 'V00DILKN1_Moderate', 'V00DILKN1_None', 'V00DILKN14_Mild',
'P02KPNLCV_Yes', 'P01KPR30CV_Yes', 'P01KPACTCV_No_Limits_or_avoidance',
'P01HPR12CV_Yes', 'P01HPL12CV_Yes', 'P01KPA30CV_Yes', 'V00P7LKRCV_2',
```

```
'V00P7LKRCV_5', 'V00P7LKRCV_7', 'V00P7LKRCV_No_pain', 'P01PMRKRCV_6',
'P01PMLKRCV_3', 'P01PMLKRCV_7', 'P01PMLKRCV_8', 'P01BL12SXL_IEI_only',
'P01BL12SXL_Neither', 'P01BL12SXR_Neither', 'P01BL12SXR_SV_only',
'P01LKP30CV_Yes', 'P01RKP30CV_Yes', 'P01RAIA_Yes', 'P01ARTHOTH_Yes',
'P01INJR_Yes', 'P01KSURGL_Yes', 'V00BONEFX_Yes', 'P01OAHIPCV_Yes',
'P01OTARTCV_Yes', 'P01KPMEDCV_Yes', 'V00FALLCV_None', 'V00FALLCV_One',
'V00ACUSCV_Yes', 'V00VITMCV_Yes', 'V00CAPSNCV_Yes', 'V00BRACCV_Yes',
'V00RELACV_Yes', 'V00SPIRCV_Yes', 'P01RASTASV_Does_not_report_RA_inflam_arth',
'V00OTHCAMC_Yes', 'V00DISCOMF_Yes', 'V00REXPN_Yes',
'V00VITDCV_No_vitamins_minerals_taken_in_past_year',
'V00CALCMCV_4_6_days_per_week',
'V00CALCMCV_No_vitamins_minerals_taken_in_past_year', 'P01FAMKR_Yes',
'V00RKEFFB_Yes', 'V00RKEFFPT_Yes', 'V00LKEFFPT_Too_tender_to_examine',
'V00RKRFXPN_Yes']
```

[ ]:
```
# And we remove those features from the dataset

df3_cluster4 = df2_cluster4.copy()
for col in high_pvalue_col4:
  if col in df3_cluster4.columns:
    df3_cluster4 = df3_cluster4.drop(col, axis = 1)
df3_cluster4.shape
```

[ ]:
```
(415, 24)
```

Training

[ ]:
```
## Logistic Regression with sklearn

x_train_cluster_4_3 = df3_cluster4.drop('cumulative_outcome', axis='columns')
y_train_cluster_4_3 = df3_cluster4['cumulative_outcome']

logreg4_3 = LogisticRegression()
logreg4_3.fit(x_train_cluster_4_3, y_train_cluster_4_3.ravel())
logreg4_3.score(x_train_cluster_4_3, y_train_cluster_4_3.ravel())
```

[ ]:
```
0.653012048192771
```

Testing

[ ]:
```
#We remove the features from the testing data set, as we did for the training␣
 ↪data set

df3_test_cluster4 = df2_test_cluster4.copy()
for col in high_pvalue_col4:
  if col in df3_test_cluster4.columns:
    df3_test_cluster4 = df3_test_cluster4.drop(col, axis = 1)
df3_test_cluster4.shape
```

[ ]:
```
(164, 24)
```

[ ]:
```
# Predictions of the outcomes

x_test_cluster_4_3 = df3_test_cluster4.drop('cumulative_outcome',␣
↪axis='columns')
y_test_cluster_4_3 = df3_test_cluster4['cumulative_outcome']

logreg4_3.predict(x_test_cluster_4_3)
logreg4_3.score(x_test_cluster_4_3, y_test_cluster_4_3)
```

[ ]:
```
0.5365853658536586
```

As the accuracy is better if we don't select the features according to their p-values, we are going to keep the accuracy with features selection only on the coefficient values.

[ ]:
```
Accuracy_train2.append(logreg4_2.score(x_train_cluster_4_2, y_train_cluster_4_2.
↪ravel()))
Accuracy_test2.append(logreg4_2.score(x_test_cluster_4_2, y_test_cluster_4_2.
↪ravel()))
```

5.6 Cluster 5

5.6.1 Feature selection: coefficients value

[ ]:
```
#We look for the columns which have low coefficients' values

low_coef_col5 = []
for i in range(len(logreg5.coef_[0])):
  if abs(logreg5.coef_[0][i]) < 0.1:
    low_coef_col5.append(df_cluster_5.iloc[:,i].name)
print(len(low_coef_col5))
print(low_coef_col5)
```

```
42
['V00WT25KG', 'V00BPSYS', 'P01HPNR12_Yes', 'V00WPRKN1_None', 'V00WPRKN2_Mild',
'V00P7RKFR_Daily', 'V00P7RKFR_Monthly', 'V00KSXRKN1_Sometimes',
'V00DIRKN1_Moderate', 'V00DIRKN1_Severe', 'V00DIRKN2_None',
'V00DIRKN14_Moderate', 'V00WPLKN1_Moderate', 'V00P7LKFR_Daily',
'V00KSXLKN1_Often', 'V00DILKN2_Moderate', 'V00DILKN2_None', 'V00KOOSFX5_None',
'V00P7RKRCV_2', 'V00P7RKRCV_3', 'V00P7LKRCV_3', 'V00P7LKRCV_No_pain',
'P01PMLKRCV_10__Pain_as_bad_as_you_can_imagine', 'P01PMLKRCV_6',
'P01BL12SXL_SV_only', 'V00HRTAT_Yes', 'V00STROKE_Yes', 'V00RA_Yes',
'P01GOUTCV_Yes', 'V00ACUSCV_Yes', 'V00VITMCV_Yes', 'V00SPIRCV_Yes',
'P01RASTASV_DK_to_RA_inflam_arth__no_dk_to_meds',
'P01RASTASV_Report_RA_inflam_arth__dk_to_all_meds',
'P01RASTASV_Report_RA_inflam_arth__no_dk_to_meds', 'P02KSURGCV_Yes',
'V00OTHCAM_Yes', 'V00EKRSL_Yes', 'V00CALCMCV_A_few_days_per_month',
```

```
'V00CALCMCV_Didn_t_take', 'V00CALCMCV_Every_day',
'V00LKEFFPT_Too_tender_to_examine']
```

[ ]:
```
# And we remove those features from the dataset

df2_cluster5 = df_cluster_5.copy()
for col in low_coef_col5:
  df2_cluster5 = df2_cluster5.drop(col, axis = 1)
df2_cluster5.shape
```

[ ]:
```
(494, 186)
```

Training

[ ]:
```
## Logistic Regression with sklearn

x_train_cluster_5_2 = df2_cluster5.drop('cumulative_outcome', axis='columns')
y_train_cluster_5_2 = df2_cluster5['cumulative_outcome']

logreg5_2 = LogisticRegression()
logreg5_2.fit(x_train_cluster_5_2, y_train_cluster_5_2.ravel())
logreg5_2.score(x_train_cluster_5_2, y_train_cluster_5_2.ravel())
```

[ ]:
```
0.7186234817813765
```

Testing

[ ]:
```
#We remove the features from the testing data set, as we did for the training␣
 ↪data set

df2_test_cluster5 = df_test_cluster_5.copy()
for col in low_coef_col5:
  df2_test_cluster5 = df2_test_cluster5.drop(col, axis = 1)
df2_test_cluster5.shape
```

[ ]:
```
(235, 186)
```

[ ]:
```
#Prediction of the outcomes

x_test_cluster_5_2 = df2_test_cluster5.drop('cumulative_outcome',␣
 ↪axis='columns')
y_test_cluster_5_2 = df2_test_cluster5['cumulative_outcome']

logreg5_2.predict(x_test_cluster_5_2)
logreg5_2.score(x_test_cluster_5_2, y_test_cluster_5_2)
```

[ ]:
```
0.5531914893617021
```

[ ]:
```
## Coefficients of the logistic regression
logreg5_2.coef_
```

[ ]:
```
array([[ 0.45468687, -0.12811831, -0.18518962, -0.51797002,  0.40117844,
        -0.18514458, -0.28303673,  0.45141005,  0.16740197,  0.18032991,
         0.16709478,  0.54692302,  0.32599809, -0.28645304, -0.36190868,
        -0.69641604,  0.11501085, -0.46197816, -0.21233134,  0.12177374,
         0.22881148,  0.44065718,  0.53683593,  0.79153677, -0.40765198,
         0.11276048,  0.33220146, -0.51530196,  0.42983592,  0.40187647,
        -0.25492728, -0.20493649,  0.38552049,  0.253195  , -0.5053205 ,
         0.30004785,  0.39089556, -0.20647168, -0.26990971,  0.37315867,
         0.46905805, -0.87527228,  0.4077238 , -0.18473761,  0.18286008,
        -0.4371855 ,  0.25039354, -0.37473831,  0.52588295, -0.34382832,
         0.22935403, -0.24527546,  0.54982519, -0.38086491,  0.15575889,
         0.19041269, -0.50898185,  0.55815131,  0.53098424, -0.23703073,
         0.35341045, -0.28613672,  0.78585005, -0.39067647,  0.6312947 ,
         0.52628174, -0.10566831, -0.35988657, -0.31684515,  0.30483143,
        -0.26437979, -0.36015789, -0.45859895, -0.30450561, -0.40707943,
         0.31586245, -0.46197816,  0.66200459,  0.89643113, -0.22329411,
        -0.66200459,  0.35192869, -0.09906853,  0.404107  ,  0.27114119,
        -0.23760374, -1.03368612, -0.26651966,  0.27558302, -0.15008126,
         0.15671259,  0.2831354 ,  0.64116779, -0.21511784, -0.87285824,
         0.47693347,  0.28544276, -0.17987736, -0.30916045,  0.3172521 ,
        -0.14633224, -0.59000281,  0.43201835,  0.35791788, -1.20906719,
         0.15258349,  0.34313765,  0.73382211,  0.15573073, -0.25174603,
        -0.34025147,  0.43319997, -0.1596785 , -0.3022724 , -0.18671591,
        -0.18176802,  0.33516411,  0.48009474,  0.33180753,  0.24600764,
         0.2313364 , -0.19012061,  0.31709358,  0.53738865,  0.68030678,
         0.3321055 ,  0.80483084, -0.19393122,  0.31325618,  0.97021827,
         0.33307705, -0.5397782 , -0.18226674,  0.28730988, -0.59363848,
         0.7401267 ,  0.62182857,  0.39187079,  0.48833613,  0.53379641,
         0.2508283 , -0.3199831 ,  0.42702639, -0.57551049, -0.16621052,
         0.41969726,  0.1172878 ,  0.12370322,  0.57604672, -0.17584925,
         0.60720422,  0.26792418,  0.55090368,  0.1573915 ,  0.2385836 ,
         0.22717613,  1.01621753,  0.36387597, -0.65241929,  0.70635881,
        -0.47674572,  0.19843471,  0.22055991, -0.36240924, -0.36812013,
        -0.2248091 ,  0.11094264,  0.29368449, -0.12797799,  0.35154664,
        -0.21274874, -0.41251395,  0.39360482, -0.41251395, -0.10039949,
         0.37649654,  0.3635202 , -0.09474141,  0.14522041,  0.55244982,
         0.38633608,  0.57402084,  0.47199167, -0.41051834,  0.69280346]])
```

5.6.2 Features selection: P-values

[ ]:
```
#We look for the feature which have a p-value too high

high_pvalue_col5 = []
for i in range(len(p_values5)):
```

```
  if p_values5[i] > 0.5:
    high_pvalue_col5.append(p_values5.index[i])
print(len(high_pvalue_col5))
print(high_pvalue_col5)
```

```
99
['V00WOMTSR', 'V00WT25KG', 'V00BPDIAS', 'V00DTCHOL', 'P01BMI', 'V00PASE',
'P01KPNREV_Yes', 'P01KPNLEV_Yes', 'P01HPNR12_Yes', 'P01HPNL12_Yes',
'V00SF2_Yes__limited_a_little', 'V00SF8_Moderately', 'V00WPRKN2_Mild',
'V00WPRKN2_Moderate', 'V00WPRKN2_None', 'V00P7RKFR_Monthly', 'V00P7RKFR_Never',
'V00KSXRKN1_Never', 'V00KSXRKN1_Rarely', 'V00DIRKN2_None', 'V00WPLKN1_Mild',
'V00WPLKN1_Moderate', 'V00WPLKN2_Mild', 'V00WPLKN2_Moderate', 'V00WPLKN2_None',
'V00P7LKFR_Daily', 'V00P7LKFR_Monthly', 'V00P7LKFR_Never', 'V00P7LKFR_Weekly',
'V00KSXLKN1_Often', 'V00KSXLKN1_Sometimes', 'V00DILKN1_Mild',
'V00DILKN1_Moderate', 'V00DILKN1_None', 'V00DILKN2_Moderate', 'V00DILKN14_Mild',
'V00DILKN14_None', 'V00KOOSFX5_Mild', 'V00KOOSFX5_Moderate', 'V00KOOSFX5_None',
'P02KPNRCV_Yes', 'P02KPNLCV_Yes', 'P01KPR30CV_Yes',
'P01KPACTCV_No_Limits_or_avoidance', 'P01HPR12CV_Yes', 'P01HPL12CV_Yes',
'P01KPA30CV_Yes', 'V00P7LKRCV_2', 'V00P7LKRCV_5', 'V00P7LKRCV_7',
'V00P7LKRCV_No_pain', 'P01PMRKRCV_6', 'P01PMLKRCV_3', 'P01PMLKRCV_6',
'P01PMLKRCV_7', 'P01PMLKRCV_8', 'P01BL12SXL_IEI_only', 'P01BL12SXL_Neither',
'P01BL12SXL_SV_only', 'P01BL12SXR_IEI_only', 'P01BL12SXR_Neither',
'P01BL12SXR_SV_only', 'P01LKP30CV_Yes', 'P01RKP30CV_Yes', 'P02KSURG_Yes',
'P01RAIA_Yes', 'P01ARTHOTH_Yes', 'P01ARTDOC_Yes', 'P01INJR_Yes',
'P01KSURGR_Yes', 'P01KSURGL_Yes', 'V00BONEFX_Yes', 'P01OAHIPCV_Yes',
'P01OTARTCV_Yes', 'P01KPMEDCV_Yes', 'V00FALLCV_None', 'V00FALLCV_One',
'V00ACUSCV_Yes', 'V00VITMCV_Yes', 'V00RUBCV_Yes', 'V00CAPSNCV_Yes',
'V00BRACCV_Yes', 'V00RELACV_Yes', 'V00SPIRCV_Yes',
'P01RASTASV_Does_not_report_RA_inflam_arth',
'P01RASTASV_Report_RA_inflam_arth__no_to_all_meds', 'V00OTHCAMC_Yes',
'V00DISCOMF_Yes', 'V00REXPN_Yes',
'V00VITDCV_No_vitamins_minerals_taken_in_past_year',
'V00CALCMCV_4_6_days_per_week', 'V00CALCMCV_A_few_days_per_month',
'V00CALCMCV_No_vitamins_minerals_taken_in_past_year', 'P01FAMKR_Yes',
'V00RKEFFB_Yes', 'V00RKEFFPT_Yes', 'V00LKEFFB_Yes',
'V00LKEFFPT_Too_tender_to_examine', 'V00RKRFXPN_Yes']
```

[ ]:
```
# And we remove those features from the dataset

df3_cluster5 = df2_cluster5.copy()
for col in high_pvalue_col5:
  if col in df3_cluster5.columns:
    df3_cluster5 = df3_cluster5.drop(col, axis = 1)
df3_cluster5.shape
```

[ ]: (494, 105)

Training

[ ]:
```
## Logistic Regression with sklearn

x_train_cluster_5_3 = df3_cluster5.drop('cumulative_outcome', axis='columns')
y_train_cluster_5_3 = df3_cluster5['cumulative_outcome']

logreg5_3 = LogisticRegression()
logreg5_3.fit(x_train_cluster_5_3, y_train_cluster_5_3.ravel())
logreg5_3.score(x_train_cluster_5_3, y_train_cluster_5_3)
```

[ ]:
```
0.6700404858299596
```

Testing

[ ]:
```
#We remove the features from the testing data set, as we did for the training␣
 ↪data set

df3_test_cluster5 = df2_test_cluster5.copy()
for col in high_pvalue_col5:
  if col in df3_test_cluster5.columns:
    df3_test_cluster5 = df3_test_cluster5.drop(col, axis = 1)
df3_test_cluster5.shape
```

[ ]:
```
(235, 105)
```

[ ]:
```
#Predictions of the outcomes

x_test_cluster_5_3 = df3_test_cluster5.drop('cumulative_outcome',␣
 ↪axis='columns')
y_test_cluster_5_3 = df3_test_cluster5['cumulative_outcome']

logreg5_3.predict(x_test_cluster_5_3)
logreg5_3.score(x_test_cluster_5_3, y_test_cluster_5_3)
```

[ ]:
```
0.5872340425531914
```

[ ]:
```
Accuracy_train2.append(logreg5_3.score(x_train_cluster_5_3, y_train_cluster_5_3.
 ↪ravel()))
Accuracy_test2.append(logreg5_3.score(x_test_cluster_5_3, y_test_cluster_5_3.
 ↪ravel()))
```

5.7 Cluster 6

5.7.1 Feature selection: coefficients value

[ ]:
```
#We look for the columns which have low coefficients' values

low_coef_col6 = []
for i in range(len(logreg6.coef_[0])):
```

```
  if abs(logreg6.coef_[0][i]) < 0.1:
    low_coef_col6.append(df_cluster_6.iloc[:,i].name)
print(len(low_coef_col6))
print(low_coef_col6)
```

```
109
['V00WOMTSL', 'V00WOMTSR', 'P01KPACDCV', 'V00BPDIAS', 'P01KPNREV_Yes',
'V00SF2_Yes__limited_a_lot', 'V00SF3_Yes__limited_a_little', 'V00SF8_Extremely',
'V00WPRKN1_Mild', 'V00WPRKN1_Moderate', 'V00WPRKN1_None', 'V00WPRKN1_Severe',
'V00WPRKN2_Severe', 'V00P7RKFR_Weekly', 'V00KSXRKN1_Often', 'V00KSXRKN1_Rarely',
'V00KSXRKN1_Sometimes', 'V00DIRKN1_Moderate', 'V00DIRKN1_Severe',
'V00DIRKN2_Mild', 'V00DIRKN2_Severe', 'V00DIRKN14_Mild', 'V00DIRKN14_Moderate',
'V00DIRKN14_Severe', 'V00WPLKN1_Moderate', 'V00WPLKN1_Severe',
'V00WPLKN2_Moderate', 'V00WPLKN2_Severe', 'V00P7LKFR_Daily',
'V00P7LKFR_Monthly', 'V00P7LKFR_Never', 'V00P7LKFR_Weekly', 'V00KSXLKN1_Never',
'V00KSXLKN1_Often', 'V00KSXLKN1_Sometimes', 'V00DILKN1_Moderate',
'V00DILKN1_Severe', 'V00DILKN2_Moderate', 'V00DILKN2_Severe', 'V00DILKN14_Mild',
'V00DILKN14_Moderate', 'V00DILKN14_None', 'V00DILKN14_Severe',
'V00KOOSFX5_Severe', 'P02KPNRCV_Yes', 'P02KPNLCV_Yes', 'P01KPR30CV_Yes',
'P01KPL30CV_Yes', 'V00P7RKRCV_10__Pain_as_bad_as_you_can_imagine',
'V00P7RKRCV_5', 'V00P7RKRCV_6', 'V00P7RKRCV_7', 'V00P7RKRCV_8', 'V00P7RKRCV_9',
'V00P7LKRCV_10__Pain_as_bad_as_you_can_imagine', 'V00P7LKRCV_6', 'V00P7LKRCV_8',
'V00P7LKRCV_9', 'V00P7LKRCV_No_pain',
'P01PMRKRCV_10__Pain_as_bad_as_you_can_imagine', 'P01PMRKRCV_2', 'P01PMRKRCV_6',
'P01PMRKRCV_7', 'P01PMRKRCV_8', 'P01PMRKRCV_9',
'P01PMLKRCV_10__Pain_as_bad_as_you_can_imagine', 'P01PMLKRCV_2', 'P01PMLKRCV_5',
'P01PMLKRCV_8', 'P01PMLKRCV_9', 'P01BL12SXL_Neither', 'P01BL12SXL_SV_only',
'P01BL12SXR_Neither', 'P01LKP30CV_Yes', 'P02KSURG_Yes', 'P01KSURGL_Yes',
'V00BONEFX_Yes', 'P02KPMEDCV_Yes', 'P01ARTDRCV_Yes', 'V00FALLCV_One',
'V00FALLCV_Two_or_Three', 'V00HYINJCV_Yes', 'V00STINJCV_Yes', 'V00ACUTCV_Yes',
'V00ACUSCV_Yes', 'V00HOMECV_Yes', 'V00CAPSNCV_Yes', 'V00BRACCV_Yes',
'V00HERBCV_Yes', 'P01RASTASV_DK_to_RA_inflam_arth__no_dk_to_meds',
'P01RASTASV_Report_RA_inflam_arth__dk_to_all_meds',
'P01RASTASV_Report_RA_inflam_arth__no_dk_to_meds', 'P02KSURGCV_Yes',
'V00OTHCAMC_Yes', 'V00OTHCAM_Yes', 'V00EKRSL_Yes', 'V00REXPN_Yes',
'V00EKRSR_Yes', 'V00VITDCV_4_6_days_per_week', 'V00VITDCV_A_few_days_per_month',
'V00VITDCV_No_vitamins_minerals_taken_in_past_year', 'V00CALCMCV_Didn_t_take',
'V00CALCMCV_No_vitamins_minerals_taken_in_past_year', 'P01FAMKR_Yes',
'P02WTGA_Yes', 'V00RKEFFPT_Too_tender_to_examine', 'V00LKEFFB_Yes',
'V00LKEFFPT_Too_tender_to_examine', 'V00RKRFXPN_Yes']
```

[ ]:
```
#And we remove those features from the dataset

df2_cluster6 = df_cluster_6.copy()
for col in low_coef_col6:
  df2_cluster6 = df2_cluster6.drop(col, axis = 1)
df2_cluster6.shape
```

[ ]: (227, 119)

Training

[ ]:
```
## Logistic Regression with sklearn

x_train_cluster_6_2 = df2_cluster6.drop('cumulative_outcome', axis='columns')
y_train_cluster_6_2 = df2_cluster6['cumulative_outcome']

logreg6_2 = LogisticRegression()
logreg6_2.fit(x_train_cluster_6_2, y_train_cluster_6_2.ravel())
logreg6_2.score(x_train_cluster_6_2, y_train_cluster_6_2.ravel())
```

[ ]: 0.7004405286343612

Testing

[ ]:
```
#We remove the features from the testing data set, as we did for the training␣
 ↪data set

df2_test_cluster6 = df_test_cluster_6.copy()
for col in low_coef_col6:
  df2_test_cluster6 = df2_test_cluster6.drop(col, axis = 1)
df2_test_cluster6.shape
```

[ ]: (105, 119)

[ ]:
```
# Prediction of the outcomes

x_test_cluster_6_2 = df2_test_cluster6.drop('cumulative_outcome',␣
 ↪axis='columns')
y_test_cluster_6_2 = df2_test_cluster6['cumulative_outcome']

logreg6_2.predict(x_test_cluster_6_2)
logreg6_2.score(x_test_cluster_6_2, y_test_cluster_6_2)
```

[ ]: 0.6571428571428571

[ ]:
```
## Coefficients of the logistic regression
logreg6_2.coef_
```

[ ]:
```
array([[-0.56247449, -0.33337545, -0.23701079,  0.23047702, -0.15627739,
        -0.54073716, -0.20221651, -0.31360458,  0.34948935,  0.24267628,
        -0.30698217,  0.77535176,  0.34499232, -0.33752305,  0.11619637,
        -0.23624446,  0.21634494,  0.51638943, -0.31928136, -0.3045373 ,
        -0.51036244,  0.19060369,  0.50201941,  0.11761102, -0.54009669,
        -0.12956809,  0.41196104, -0.24848864, -0.18337522,  0.15051861,
        -0.13123696,  0.15728427, -0.11422346,  0.10630182,  0.28242113,
        -0.24123857,  0.20789475, -0.18185348, -0.141543  ,  0.47695296,
```

```
-0.41014813,  0.43905242, -0.41604151, -0.4236629 ,  0.63370301,
-0.1420481 ,  0.11619637,  0.41222739,  0.22760515,  0.48524161,
-0.41222739, -0.345318  , -0.15154012,  0.32370471, -0.12054966,
-0.83022469,  0.34699932,  0.19883978,  0.13891133,  0.22716478,
-0.48929846,  0.25227785,  0.13979419,  0.31916273, -0.37900605,
 0.29001929,  0.48759206,  0.27731535, -0.21617687,  0.27386663,
 0.32102005, -0.2953493 , -0.65108178, -0.31434916,  0.12948422,
 0.64181547,  0.50882064,  0.41940374,  0.51577782, -0.4605117 ,
-0.45404   ,  0.46377453,  0.93245631, -0.26893441, -0.27369479,
-0.33020573, -0.29090246,  0.48401618,  0.26039766, -0.54689505,
 0.19654082,  0.1855725 , -0.15943041, -0.39305558, -0.17858744,
-0.22690892,  0.14810538,  0.33135821, -0.16265218, -0.52026224,
 0.18950871,  0.18482166,  0.13139428, -0.11272217, -0.39394795,
-0.12775171, -0.27023736, -0.14625273, -0.16027409, -0.49561889,
 0.3187419 , -0.3292625 ,  0.36166578,  0.95010168,  0.4725091 ,
-0.16251909, -0.32873366,  0.16767671]])
```

5.7.2 Features selection: P-values

[ ]:
```
#We look for the feature which have a p-value too high

high_pvalue_col6 = []
for i in range(len(p_values6)):
  if p_values6[i] > 0.1:
    high_pvalue_col6.append(p_values6.index[i])
print(len(high_pvalue_col6))
print(high_pvalue_col6)
```

```
182
['V00WOMTSL', 'V00WOMTSR', 'P01KPACDCV', 'V00COMORB', 'V00HT25MM', 'V00WT25KG',
'V00WTMAXKG', 'V00WTMINKG', 'V00BPDIAS', 'V00BPSYS', 'V00DTVITD', 'V00DTCALC',
'V00DTCHOL', 'P01BMI', 'V00PASE', 'V00LLWGT', 'V00RLWGT', 'P01KPNREV_Yes',
'P01KPNLEV_Yes', 'P01KPACT30_Yes', 'P01HPNR12_Yes', 'P01HPNL12_Yes',
'V00SF2_Yes__limited_a_little', 'V00SF2_Yes__limited_a_lot',
'V00SF3_Yes__limited_a_little', 'V00SF3_Yes__limited_a_lot',
'V00SF8_Moderately', 'V00SF8_Not_at_all', 'V00SF8_Quite_a_bit',
'V00WPRKN1_Mild', 'V00WPRKN2_Mild', 'V00WPRKN2_Moderate', 'V00WPRKN2_None',
'V00P7RKFR_Daily', 'V00P7RKFR_Monthly', 'V00P7RKFR_Never', 'V00P7RKFR_Weekly',
'V00KSXRKN1_Never', 'V00KSXRKN1_Rarely', 'V00KSXRKN1_Sometimes',
'V00DIRKN1_Mild', 'V00DIRKN1_Moderate', 'V00DIRKN2_Mild', 'V00DIRKN2_Moderate',
'V00DIRKN2_None', 'V00DIRKN14_Mild', 'V00WPLKN1_Mild', 'V00WPLKN1_Moderate',
'V00WPLKN1_None', 'V00WPLKN1_Severe', 'V00WPLKN2_Mild', 'V00WPLKN2_Moderate',
'V00WPLKN2_None', 'V00WPLKN2_Severe', 'V00P7LKFR_Daily', 'V00P7LKFR_Monthly',
'V00P7LKFR_Never', 'V00P7LKFR_Weekly', 'V00KSXLKN1_Never', 'V00KSXLKN1_Often',
'V00KSXLKN1_Rarely', 'V00KSXLKN1_Sometimes', 'V00DILKN1_Mild',
'V00DILKN1_Moderate', 'V00DILKN1_None', 'V00DILKN1_Severe', 'V00DILKN2_Mild',
'V00DILKN2_Moderate', 'V00DILKN2_None', 'V00DILKN2_Severe', 'V00DILKN14_Mild',
'V00DILKN14_None', 'V00KOOSFX5_Mild', 'V00KOOSFX5_Moderate', 'V00KOOSFX5_None',
```

```
'V00KOOSFX5_Severe', 'P02KPNRCV_Yes', 'P02KPNLCV_Yes', 'P01KPR30CV_Yes',
'P01KPL30CV_Yes', 'P01KPACTCV_Limits', 'P01KPACTCV_No_Limits_or_avoidance',
'P01HPR12CV_Yes', 'P01HPL12CV_Yes', 'P01KPA30CV_Yes', 'V00P7RKRCV_2',
'V00P7RKRCV_3', 'V00P7RKRCV_5', 'V00P7RKRCV_6', 'V00P7RKRCV_No_pain',
'V00P7LKRCV_2', 'V00P7LKRCV_3', 'V00P7LKRCV_5', 'V00P7LKRCV_6', 'V00P7LKRCV_7',
'V00P7LKRCV_8', 'V00P7LKRCV_No_pain', 'P01PMRKRCV_2', 'P01PMRKRCV_3',
'P01PMRKRCV_4', 'P01PMRKRCV_5', 'P01PMRKRCV_6', 'P01PMRKRCV_8',
'P01PMRKRCV_No_pain', 'P01PMLKRCV_2', 'P01PMLKRCV_3', 'P01PMLKRCV_5',
'P01PMLKRCV_6', 'P01PMLKRCV_7', 'P01PMLKRCV_8', 'P01PMLKRCV_9',
'P01PMLKRCV_No_pain', 'P01BL12SXL_IEI_only', 'P01BL12SXL_Neither',
'P01BL12SXL_SV_only', 'P01BL12SXR_IEI_only', 'P01BL12SXR_Neither',
'P01BL12SXR_SV_only', 'P01LKP30CV_Yes', 'P01RKP30CV_Yes', 'P02KPMED_Yes',
'P02KSURG_Yes', 'P01RAIA_Yes', 'P01ARTHOTH_Yes', 'P01ARTDOC_Yes',
'P01KPMED_Yes', 'P01INJR_Yes', 'P01KSURGR_Yes', 'P01KSURGL_Yes',
'V00STROKE_Yes', 'V00DIAB_Yes', 'V00RA_Yes', 'V00BONEFX_Yes', 'V00SMOKE_Yes',
'P02KPMEDCV_Yes', 'P01OAHIPCV_Yes', 'P01GOUTCV_Yes', 'P01OTARTCV_Yes',
'P01KPMEDCV_Yes', 'V00FALLCV_None', 'V00FALLCV_One', 'V00FALLCV_Six_or_more',
'V00FALLCV_Two_or_Three', 'V00STINJCV_Yes', 'V00ACUTCV_Yes', 'V00ACUSCV_Yes',
'V00HOMECV_Yes', 'V00MASSCV_Yes', 'V00VITMCV_Yes', 'V00RUBCV_Yes',
'V00CAPSNCV_Yes', 'V00BRACCV_Yes', 'V00YOGACV_Yes', 'V00HERBCV_Yes',
'V00RELACV_Yes', 'V00SPIRCV_Yes', 'P01RASTASV_Does_not_report_RA_inflam_arth',
'P01RASTASV_Report_RA_inflam_arth__no_to_all_meds', 'V00OTHCAMC_Yes',
'V00DISCOMF_Yes', 'V00LFXCOMP_Yes', 'V00LFXPN_Yes', 'V00REXPN_Yes',
'V00EKRSR_Yes', 'V00RFXCOMP_Yes', 'V00VITDCV_4_6_days_per_week',
'V00VITDCV_Didn_t_take', 'V00VITDCV_No_vitamins_minerals_taken_in_past_year',
'V00CALCMCV_4_6_days_per_week', 'V00CALCMCV_A_few_days_per_month',
'V00CALCMCV_Didn_t_take', 'V00CALCMCV_Every_day',
'V00CALCMCV_No_vitamins_minerals_taken_in_past_year', 'V00VIT1_Yes',
'P01FAMKR_Yes', 'V00RKEFFB_Yes', 'V00RKEFFPT_Yes', 'V00LKEFFB_Yes',
'V00LKEFFPT_Too_tender_to_examine', 'V00LKEFFPT_Yes', 'V00RKRFXPN_Yes',
'V00LKRFXPN_Yes']
```

[ ]:
```
# And we remove those features from the dataset

df3_cluster6 = df2_cluster6.copy()
for col in high_pvalue_col6:
  if col in df3_cluster6.columns:
    df3_cluster6 = df3_cluster6.drop(col, axis = 1)
df3_cluster6.shape
```

[ ]:
```
(227, 13)
```

Training

[ ]:
```
## Logistic Regression with sklearn

x_train_cluster_6_3 = df3_cluster6.drop('cumulative_outcome', axis='columns')
y_train_cluster_6_3 = df3_cluster6['cumulative_outcome']
```

```
logreg6_3 = LogisticRegression()
logreg6_3.fit(x_train_cluster_6_3, y_train_cluster_6_3.ravel())
logreg6_3.score(x_train_cluster_6_3, y_train_cluster_6_3.ravel())
```

[ ]: 0.6784140969162996

Testing

```
[ ]: #We remove the features from the testing data set, as we did for the training␣
     ↪data set

     df3_test_cluster6 = df2_test_cluster6.copy()
     for col in high_pvalue_col6:
       if col in df3_test_cluster6.columns:
         df3_test_cluster6 = df3_test_cluster6.drop(col, axis = 1)
     df3_test_cluster6.shape
```

[ ]: (105, 13)

```
[ ]: #Predictions of the outcomes

     x_test_cluster_6_3 = df3_test_cluster6.drop('cumulative_outcome',␣
      ↪axis='columns')
     y_test_cluster_6_3 = df3_test_cluster6['cumulative_outcome']

     logreg6_3.predict(x_test_cluster_6_3)
     logreg6_3.score(x_test_cluster_6_3, y_test_cluster_6_3)
```

[ ]: 0.6476190476190476

As the accuracy is better if we don't select the features according to their p-values, we are going to keep the accuracy with features selection only on the coefficient values.

```
[ ]: Accuracy_train2.append(logreg6_2.score(x_train_cluster_6_2, y_train_cluster_6_2.
      ↪ravel()))
     Accuracy_test2.append(logreg6_2.score(x_test_cluster_6_2, y_test_cluster_6_2.
      ↪ravel()))
```

5.8 Cluster 7

5.8.1 Feature selection: coefficients value

```
[ ]: #We look for the columns which have low coefficients' values

     low_coef_col7 = []
     for i in range(len(logreg7.coef_[0])):
       if abs(logreg7.coef_[0][i]) < 0.1:
         low_coef_col7.append(df_cluster_7.iloc[:,i].name)
```

```
print(len(low_coef_col7))
print(low_coef_col7)
```

```
121
['V00COMORB', 'V00WT25KG', 'V00WTMINKG', 'V00BPDIAS', 'P01KPNREV_Yes',
'P01HPNL12_Yes', 'V00SF2_Yes__limited_a_little', 'V00SF3_Yes__limited_a_lot',
'V00SF8_Extremely', 'V00WPRKN1_Mild', 'V00WPRKN1_Moderate', 'V00WPRKN1_None',
'V00WPRKN2_Moderate', 'V00WPRKN2_Severe', 'V00P7RKFR_Daily', 'V00KSXRKN1_Never',
'V00KSXRKN1_Often', 'V00KSXRKN1_Sometimes', 'V00DIRKN1_Moderate',
'V00DIRKN1_Severe', 'V00DIRKN2_Moderate', 'V00DIRKN2_Severe', 'V00DIRKN14_Mild',
'V00DIRKN14_Moderate', 'V00DIRKN14_None', 'V00DIRKN14_Severe',
'V00WPLKN1_Severe', 'V00WPLKN2_Mild', 'V00WPLKN2_Moderate', 'V00WPLKN2_None',
'V00WPLKN2_Severe', 'V00KSXLKN1_Never', 'V00KSXLKN1_Often',
'V00KSXLKN1_Sometimes', 'V00DILKN1_Moderate', 'V00DILKN1_Severe',
'V00DILKN2_Mild', 'V00DILKN2_Moderate', 'V00DILKN2_None', 'V00DILKN2_Severe',
'V00DILKN14_Mild', 'V00DILKN14_Moderate', 'V00DILKN14_None',
'V00DILKN14_Severe', 'V00KOOSFX5_Moderate', 'V00KOOSFX5_None', 'P02KPNRCV_Yes',
'P01KPR30CV_Yes', 'P01KPL30CV_Yes', 'P01HPL12CV_Yes',
'V00P7RKRCV_10__Pain_as_bad_as_you_can_imagine', 'V00P7RKRCV_4', 'V00P7RKRCV_6',
'V00P7RKRCV_7', 'V00P7RKRCV_9', 'V00P7LKRCV_10__Pain_as_bad_as_you_can_imagine',
'V00P7LKRCV_2', 'V00P7LKRCV_3', 'V00P7LKRCV_5', 'V00P7LKRCV_6', 'V00P7LKRCV_7',
'V00P7LKRCV_8', 'V00P7LKRCV_9', 'V00P7LKRCV_No_pain',
'P01PMRKRCV_10__Pain_as_bad_as_you_can_imagine', 'P01PMRKRCV_2', 'P01PMRKRCV_4',
'P01PMRKRCV_5', 'P01PMRKRCV_6', 'P01PMRKRCV_7', 'P01PMRKRCV_8', 'P01PMRKRCV_9',
'P01PMRKRCV_No_pain', 'P01PMLKRCV_10__Pain_as_bad_as_you_can_imagine',
'P01PMLKRCV_2', 'P01PMLKRCV_5', 'P01PMLKRCV_7', 'P01BL12SXL_SV_only',
'P01BL12SXR_Neither', 'P01LKP30CV_Yes', 'P01RKP30CV_Yes', 'V00RA_Yes',
'V00POLYRH_Yes', 'V00CAM12_Yes', 'P02KPMEDCV_Yes', 'P01ARTDRCV_Yes',
'P01KPMEDCV_Yes', 'V00FALLCV_One', 'V00HYINJCV_Yes', 'V00STINJCV_Yes',
'V00ACUTCV_Yes', 'V00ACUSCV_Yes', 'V00HOMECV_Yes', 'V00MASSCV_Yes',
'V00DIETCV_Yes', 'V00VITMCV_Yes', 'V00CAPSNCV_Yes', 'V00BRACCV_Yes',
'V00HERBCV_Yes', 'V00RELACV_Yes',
'P01RASTASV_DK_to_RA_inflam_arth__no_dk_to_meds',
'P01RASTASV_Report_RA_inflam_arth__dk_to_all_meds',
'P01RASTASV_Report_RA_inflam_arth__no_dk_to_meds', 'P02KSURGCV_Yes',
'V00OTHCAMC_Yes', 'V00OTHCAM_Yes', 'V00LFXPN_Yes', 'V00EKRSL_Yes',
'V00EKRSR_Yes', 'V00VITDCV_4_6_days_per_week', 'V00VITDCV_A_few_days_per_month',
'V00VITDCV_Every_day', 'V00CALCMCV_4_6_days_per_week',
'V00CALCMCV_A_few_days_per_month', 'V00CALCMCV_Didn_t_take',
'V00CALCMCV_Every_day', 'V00VIT1_Yes', 'V00RKEFFB_Yes',
'V00RKEFFPT_Too_tender_to_examine', 'V00LKEFFPT_Too_tender_to_examine',
'V00RKRFXPN_Yes']
```

[ ]:
```
#And we remove those features from the data set

df2_cluster7 = df_cluster_7.copy()
for col in low_coef_col7:
```

```
  df2_cluster7 = df2_cluster7.drop(col, axis = 1)
df2_cluster7.shape
```

[ ]: (154, 107)

Training

```
[ ]: ## Logistic Regression with sklearn

x_train_cluster_7_2 = df2_cluster7.drop('cumulative_outcome', axis='columns')
y_train_cluster_7_2 = df2_cluster7['cumulative_outcome']

logreg7_2 = LogisticRegression()
logreg7_2.fit(x_train_cluster_7_2, y_train_cluster_7_2.ravel())
logreg7_2.score(x_train_cluster_7_2, y_train_cluster_7_2.ravel())
```

[ ]: 0.7402597402597403

Testing

```
[ ]: #We remove the features from the testing data set, as we did for the training␣
 ↪data set

df2_test_cluster7 = df_test_cluster_7.copy()
for col in low_coef_col7:
  df2_test_cluster7 = df2_test_cluster7.drop(col, axis = 1)
df2_test_cluster7.shape
```

[ ]: (71, 107)

```
[ ]: #Predictions of the outcomes

x_test_cluster_7_2 = df2_test_cluster7.drop('cumulative_outcome',␣
 ↪axis='columns')
y_test_cluster_7_2 = df2_test_cluster7['cumulative_outcome']

logreg7_2.predict(x_test_cluster_7_2)
logreg7_2.score(x_test_cluster_7_2, y_test_cluster_7_2)
```

[ ]: 0.7464788732394366

```
[ ]: ## Coefficients of the logistic regression
logreg7_2.coef_
```

```
[ ]: array([[ 0.14328317,  0.19418191,  0.21186539,  0.26621889,  0.24404437,
          0.67326358, -0.33685068, -0.27489143,  0.14361648,  0.4967084 ,
         -0.54223877,  0.19716557,  0.39470086, -0.16078762,  0.21675511,
          0.84550977, -0.19198174,  0.51492131,  0.32782231, -0.74520099,
          0.48293486, -0.1437836 ,  0.46507872, -0.42606484, -0.53781106,
```

```
 0.65060245, -0.3054261 , -0.10814312,  0.28185197, -0.27342135,
-0.25990678,  0.23962267,  0.24343559,  0.24098433, -0.35388969,
 0.11541668, -0.59078377,  0.58892917, -0.23815384,  0.14755088,
 0.21335504, -0.21176802, -0.13513663,  0.21512702,  0.39794305,
 0.21675511, -0.27300524,  0.26266482,  0.27300524,  0.27591538,
 0.14943867,  0.15266586, -0.1875919 ,  0.1484697 , -0.10207526,
-0.32270241,  0.39821283, -0.33018413,  0.19699977,  0.29377084,
 0.33581123, -0.3012381 ,  0.52439993, -0.42598585,  0.22213838,
-0.20264122,  0.43993107, -0.44946842,  0.27464109,  0.97468415,
 0.91175137,  0.27961504, -0.24028799, -0.13480263, -0.24849664,
 0.12268177,  0.64590942,  0.17681616,  0.94788446,  0.16746642,
 0.27116238,  0.31757721,  0.5215343 ,  0.42373259,  0.4053504 ,
-0.60637519, -0.2089088 ,  0.28051927,  0.14833754, -0.10943271,
-0.3140632 ,  0.2777418 ,  0.26853048,  0.16943229,  0.13040493,
 0.16678743,  0.10947866, -0.21349041, -0.21349041,  0.53424343,
-0.15547174,  0.32225757, -0.18365801, -0.25848268,  0.16978475,
-0.11601871]])
```

5.8.2 Features selection: P-values

```
[ ]: #We look for the feature which have a p-value too high

high_pvalue_col7 = []
for i in range(len(p_values7)):
  if p_values7[i] > 0.5:
    high_pvalue_col7.append(p_values7.index[i])
print(len(high_pvalue_col7))
print(high_pvalue_col7)
```

```
99
['V00WOMTSR', 'V00WT25KG', 'V00BPDIAS', 'V00DTCHOL', 'P01BMI', 'V00PASE',
'P01KPNREV_Yes', 'P01KPNLEV_Yes', 'P01HPNR12_Yes', 'P01HPNL12_Yes',
'V00SF2_Yes__limited_a_little', 'V00SF8_Moderately', 'V00WPRKN2_Mild',
'V00WPRKN2_Moderate', 'V00WPRKN2_None', 'V00P7RKFR_Monthly', 'V00P7RKFR_Never',
'V00KSXRKN1_Never', 'V00KSXRKN1_Rarely', 'V00DIRKN2_None', 'V00WPLKN1_Mild',
'V00WPLKN1_Moderate', 'V00WPLKN2_Mild', 'V00WPLKN2_Moderate', 'V00WPLKN2_None',
'V00P7LKFR_Daily', 'V00P7LKFR_Monthly', 'V00P7LKFR_Never', 'V00P7LKFR_Weekly',
'V00KSXLKN1_Often', 'V00KSXLKN1_Sometimes', 'V00DILKN1_Mild',
'V00DILKN1_Moderate', 'V00DILKN1_None', 'V00DILKN2_Moderate', 'V00DILKN14_Mild',
'V00DILKN14_None', 'V00KOOSFX5_Mild', 'V00KOOSFX5_Moderate', 'V00KOOSFX5_None',
'P02KPNRCV_Yes', 'P02KPNLCV_Yes', 'P01KPR30CV_Yes',
'P01KPACTCV_No_Limits_or_avoidance', 'P01HPR12CV_Yes', 'P01HPL12CV_Yes',
'P01KPA30CV_Yes', 'V00P7LKRCV_2', 'V00P7LKRCV_5', 'V00P7LKRCV_7',
'V00P7LKRCV_No_pain', 'P01PMRKRCV_6', 'P01PMLKRCV_3', 'P01PMLKRCV_6',
'P01PMLKRCV_7', 'P01PMLKRCV_8', 'P01BL12SXL_IEI_only', 'P01BL12SXL_Neither',
'P01BL12SXL_SV_only', 'P01BL12SXR_IEI_only', 'P01BL12SXR_Neither',
'P01BL12SXR_SV_only', 'P01LKP30CV_Yes', 'P01RKP30CV_Yes', 'P02KSURG_Yes',
'P01RAIA_Yes', 'P01ARTHOTH_Yes', 'P01ARTDOC_Yes', 'P01INJR_Yes',
```

```
'P01KSURGR_Yes', 'P01KSURGL_Yes', 'V00BONEFX_Yes', 'P01OAHIPCV_Yes',
'P01OTARTCV_Yes', 'P01KPMEDCV_Yes', 'V00FALLCV_None', 'V00FALLCV_One',
'V00ACUSCV_Yes', 'V00VITMCV_Yes', 'V00RUBCV_Yes', 'V00CAPSNCV_Yes',
'V00BRACCV_Yes', 'V00RELACV_Yes', 'V00SPIRCV_Yes',
'P01RASTASV_Does_not_report_RA_inflam_arth',
'P01RASTASV_Report_RA_inflam_arth__no_to_all_meds', 'V00OTHCAMC_Yes',
'V00DISCOMF_Yes', 'V00REXPN_Yes',
'V00VITDCV_No_vitamins_minerals_taken_in_past_year',
'V00CALCMCV_4_6_days_per_week', 'V00CALCMCV_A_few_days_per_month',
'V00CALCMCV_No_vitamins_minerals_taken_in_past_year', 'P01FAMKR_Yes',
'V00RKEFFB_Yes', 'V00RKEFFPT_Yes', 'V00LKEFFB_Yes',
'V00LKEFFPT_Too_tender_to_examine', 'V00RKRFXPN_Yes']
```

[ ]:
```
# And we remove those features from the dataset

df3_cluster7 = df2_cluster7.copy()
for col in high_pvalue_col7:
  if col in df3_cluster7.columns:
    df3_cluster7 = df3_cluster7.drop(col, axis = 1)
df3_cluster7.shape
```

[ ]:
```
(154, 52)
```

Training

[ ]:
```
## Logistic Regression with sklearn

x_train_cluster_7_3 = df3_cluster7.drop('cumulative_outcome', axis='columns')
y_train_cluster_7_3 = df3_cluster7['cumulative_outcome']

logreg7_3 = LogisticRegression()
logreg7_3.fit(x_train_cluster_7_3, y_train_cluster_7_3.ravel())
logreg7_3.score(x_train_cluster_7_3, y_train_cluster_7_3.ravel())
```

[ ]:
```
0.7207792207792207
```

Testing

[ ]:
```
#We remove the features from the testing data set, as we did for the training␣
 ↪data set

df3_test_cluster7 = df2_test_cluster7.copy()
for col in high_pvalue_col7:
  if col in df3_test_cluster7.columns:
    df3_test_cluster7 = df3_test_cluster7.drop(col, axis = 1)
df3_test_cluster7.shape
```

[ ]:
```
(71, 52)
```

```
[ ]: #Predictions of the outcomes

x_test_cluster_7_3 = df3_test_cluster7.drop('cumulative_outcome',␣
 ↪axis='columns')
y_test_cluster_7_3 = df3_test_cluster7['cumulative_outcome']

logreg7_3.predict(x_test_cluster_7_3)
logreg7_3.score(x_test_cluster_7_3, y_test_cluster_7_3)
```

```
[ ]: 0.7464788732394366
```

```
[ ]: Accuracy_train2.append(logreg7_3.score(x_train_cluster_7_3, y_train_cluster_7_3.
 ↪ravel()))
Accuracy_test2.append(logreg7_3.score(x_test_cluster_7_3, y_test_cluster_7_3.
 ↪ravel()))
```

5.9 Summary

```
[ ]: summary2 = pd.DataFrame(data = [Accuracy_train, Accuracy_test, Accuracy_train2,␣
 ↪Accuracy_test2],
                           index = ['Accuracy_train', 'Accuracy_test',␣
 ↪'Accuracy_train_selection', 'Accuracy_test_selection'],
                           columns = ['Cluster 0','Cluster 1','Cluster 2','Cluster␣
 ↪3','Cluster 4','Cluster 5','Cluster 6','Cluster 7'])
summary2
```

```
[ ]:                          Cluster 0  Cluster 1  …  Cluster 6  Cluster 7
    Accuracy_train             0.754209   0.740061  …   0.704846   0.733766
    Accuracy_test              0.596639   0.625899  …   0.657143   0.746479
    Accuracy_train_selection   0.710438   0.718654  …   0.700441   0.720779
    Accuracy_test_selection    0.621849   0.633094  …   0.657143   0.746479

    [4 rows x 8 columns]
```

6 Final models: Performance assessment

For each cluster, we are going to choose the model with the higher accuracy between the model with or without features selection. Then, we are going to compute different performance metrics: the AUC, the TPR and the FPR.
For each cluster, the best model is the one with features selection.

```
[ ]: #Functions to compute the TPR and FPR thanks to the confusion matrix

from sklearn import metrics

def TPR(cm):
  return(round(cm[1][1]/(cm[1][0] + cm[1][1]), 4))
```

```
def FPR(cm):
  return(round(cm[0][1]/(cm[0][0] + cm[0][1]),4))
```

[ ]:
```
AUC = []
TPR_list = []
FPR_list = []
```

6.1 Cluster 0

[ ]:
```
y_pred_0 = logreg0_3.predict(x_test_cluster_0_3)

cm0 = metrics.confusion_matrix(y_test_cluster_0_3, y_pred_0)
cm0
```

[ ]:
```
array([[62,  6],
       [39, 12]])
```

[ ]:
```
print('TPR = ', TPR(cm0))
print('FPR = ', FPR(cm0))
TPR_list.append(TPR(cm0))
FPR_list.append(FPR(cm0))
```

```
TPR =  0.2353
FPR =  0.0882
```

[ ]:
```
# Computation of the AUC
y_pred_proba0 = logreg0_3.predict_proba(x_test_cluster_0_3)[::,1]
auc0 = metrics.roc_auc_score(y_test_cluster_0_3, y_pred_proba0)

AUC.append(round(auc0,4))
print(auc0)
```

```
0.6346597462514418
```

6.2 Cluster 1

[ ]:
```
y_pred_1 = logreg1_3.predict(x_test_cluster_1_3)

cm1 = metrics.confusion_matrix(y_test_cluster_1_3, y_pred_1)
cm1
```

[ ]:
```
array([[87,  1],
       [50,  1]])
```

[ ]:
```
print('TPR = ', TPR(cm1))
print('FPR = ', FPR(cm1))
TPR_list.append(TPR(cm1))
FPR_list.append(FPR(cm1))
```

```
TPR =  0.0196
FPR =  0.0114
```

[ ]:
```
# Computation of the AUC

y_pred_proba1 = logreg1_3.predict_proba(x_test_cluster_1_3)[::,1]
auc1 = metrics.roc_auc_score(y_test_cluster_1_3, y_pred_proba1)

AUC.append(round(auc1,4))
print(auc1)
```

```
0.5632798573975044
```

6.3 Cluster 2

[ ]:
```
y_pred_2 = logreg2_2.predict(x_test_cluster_2_2)

cm2 = metrics.confusion_matrix(y_test_cluster_2_2, y_pred_2)
cm2
```

[ ]:
```
array([[67, 10],
       [51, 17]])
```

[ ]:
```
print('TPR = ', TPR(cm2))
print('FPR = ', FPR(cm2))
TPR_list.append(TPR(cm2))
FPR_list.append(FPR(cm2))
```

```
TPR =  0.25
FPR =  0.1299
```

[ ]:
```
# Computation of the AUC

y_pred_proba2 = logreg2_2.predict_proba(x_test_cluster_2_2)[::,1]
auc2 = metrics.roc_auc_score(y_test_cluster_2_2, y_pred_proba2)

AUC.append(round(auc2,4))
print(auc2)
```

```
0.5909090909090909
```

6.4 Cluster 3

[ ]:
```
y_pred_3 = logreg3_3.predict(x_test_cluster_3_3)

cm3 = metrics.confusion_matrix(y_test_cluster_3_3, y_pred_3)
cm3
```

[ ]:
```
array([[59, 14],
       [36, 24]])
```

[ ]:
```
print('TPR = ', TPR(cm3))
print('FPR = ', FPR(cm3))
TPR_list.append(TPR(cm3))
FPR_list.append(FPR(cm3))
```

```
TPR =  0.4
FPR =  0.1918
```

[ ]:
```
# Computation of the AUC

y_pred_proba3 = logreg3_3.predict_proba(x_test_cluster_3_3)[::,1]
auc3 = metrics.roc_auc_score(y_test_cluster_3_3, y_pred_proba3)

AUC.append(round(auc3,4))
print(auc3)
```

```
0.7029680365296803
```

6.5 Cluster 4

[ ]:
```
y_pred_4 = logreg4_2.predict(x_test_cluster_4_2)

cm4 = metrics.confusion_matrix(y_test_cluster_4_2, y_pred_4)
cm4
```

[ ]:
```
array([[77, 15],
       [58, 14]])
```

[ ]:
```
print('TPR = ', TPR(cm4))
print('FPR = ', FPR(cm4))
TPR_list.append(TPR(cm4))
FPR_list.append(FPR(cm4))
```

```
TPR =  0.1944
FPR =  0.163
```

[ ]:
```
# Computation of the AUC

y_pred_proba4 = logreg4_2.predict_proba(x_test_cluster_4_2)[::,1]
auc4 = metrics.roc_auc_score(y_test_cluster_4_2, y_pred_proba4)

AUC.append(round(auc4,4))
print(auc4)
```

```
0.5552536231884059
```

6.6 Cluster 5

[ ]:
```
y_pred_5 = logreg5_3.predict(x_test_cluster_5_3)

cm5 = metrics.confusion_matrix(y_test_cluster_5_3, y_pred_5)
cm5
```

[ ]:
```
array([[59, 39],
       [58, 79]])
```

[ ]:
```
print('TPR = ', TPR(cm5))
print('FPR = ', FPR(cm5))
TPR_list.append(TPR(cm5))
FPR_list.append(FPR(cm5))
```

```
TPR =  0.5766
FPR =  0.398
```

[ ]:
```
# Computation of the AUC

y_pred_proba5 = logreg5_3.predict_proba(x_test_cluster_5_3)[::,1]
auc5 = metrics.roc_auc_score(y_test_cluster_5_3, y_pred_proba5)

AUC.append(round(auc5,4))
print(auc5)
```

```
0.58878295843885
```

6.7 Cluster 6

[ ]:
```
y_pred_6 = logreg6_2.predict(x_test_cluster_6_2)

cm6 = metrics.confusion_matrix(y_test_cluster_6_2, y_pred_6)
cm6
```

[ ]:
```
array([[68,  0],
       [36,  1]])
```

[ ]:
```
print('TPR = ', TPR(cm6))
print('FPR = ', FPR(cm6))
TPR_list.append(TPR(cm6))
FPR_list.append(FPR(cm6))
```

```
TPR =  0.027
FPR =  0.0
```

[ ]:
```
# Computation of the AUC

y_pred_proba6 = logreg6_2.predict_proba(x_test_cluster_6_2)[::,1]
auc6 = metrics.roc_auc_score(y_test_cluster_6_2, y_pred_proba6)
```

```
AUC.append(round(auc6,4))
print(auc6)
```

```
0.5631955484896662
```

6.8 Cluster 7

[ ]:
```
y_pred_7 = logreg7_3.predict(x_test_cluster_7_3)

cm7 = metrics.confusion_matrix(y_test_cluster_7_3, y_pred_7)
cm7
```

[ ]:
```
array([[53,  0],
       [18,  0]])
```

[ ]:
```
print('TPR = ', TPR(cm0))
print('FPR = ', FPR(cm0))
TPR_list.append(TPR(cm0))
FPR_list.append(FPR(cm0))
```

```
TPR =  0.2353
FPR =  0.0882
```

[ ]:
```
# Computation of the AUC

y_pred_proba7 = logreg7_3.predict_proba(x_test_cluster_7_3)[::,1]
auc7 = metrics.roc_auc_score(y_test_cluster_7_3, y_pred_proba7)

AUC.append(round(auc7,4))
print(auc7)
```

```
0.549266247379455
```

6.9 Summary

[ ]:
```
summary3 = pd.DataFrame(data = [Accuracy_train2, Accuracy_test2, AUC, TPR_list,␣
↪FPR_list],
                        index = ['Accuracy_train', 'Accuracy_test', 'AUC',␣
↪'TPR', 'FPR'],
                        columns = ['Cluster 0','Cluster 1','Cluster 2','Cluster␣
↪3','Cluster 4','Cluster 5','Cluster 6','Cluster 7'])
summary3
```

[ ]:

	Cluster 0	Cluster 1	…	Cluster 6	Cluster 7
Accuracy_train	0.710438	0.718654	…	0.700441	0.720779
Accuracy_test	0.621849	0.633094	…	0.657143	0.746479
AUC	0.634700	0.563300	…	0.563200	0.549300
TPR	0.235300	0.019600	…	0.027000	0.235300

```
FPR              0.088200   0.011400  …   0.000000   0.088200

[5 rows x 8 columns]
```

Project_242_Logistic_Regression_Model_Sklearn

December 17, 2021

[1]:
```
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

df = pd.read_csv('/Users/areinaud/My Drive/Études/5 - Berkeley/Fall Semester/
 ↪IEOR 242/Project/Data cleaning/cleaned_data.csv')


df.head()
```

[1]:

	ID	P01KPNREV	P01KPNLEV	P01KPACT30	P01HPNR12	P01HPNL12	\
0	9000296	Yes	No	No	Yes	No	
1	9000622	Yes	No	Yes	Yes	No	
2	9001695	Yes	No	No	No	No	
3	9001897	No	No	No	No	No	
4	9002411	Yes	No	No	No	No	

	V00SF2	V00SF3	V00SF8	V00WPRKN1	…	\
0	Not limited at all	Not limited at all	Not at all	None	…	
1	Yes, limited a little	Yes, limited a little	Moderately	Mild	…	
2	Not limited at all	Not limited at all	Not at all	None	…	
3	Not limited at all	Not limited at all	Not at all	None	…	
4	Yes, limited a little	Yes, limited a little	A little bit	None	…	

	P02WTGA	V00RKEFFB	V00RKEFFPT	V00LKEFFB	V00LKEFFPT	V00RKRFXPN	V00LKRFXPN	\
0	No	No	No	No	No	No	No	
1	No	No	No	No	No	Yes	No	
2	No	Yes	Yes	No	No	No	No	
3	No	Yes	No	No	No	No	No	
4	Yes	No	No	No	No	No	No	

	V00LLWGT	V00RLWGT	cumulative_outcome
0	12.0	18.0	No
1	14.0	14.0	Yes
2	13.0	13.0	Yes
3	16.0	17.0	Yes

```
4     18.0     22.0                  No

[5 rows x 122 columns]
```

[2]:
```
## Replace 'No' and 'Yes' by 0 and 1 in the outcome column

df['cumulative_outcome'] = df['cumulative_outcome'].apply(lambda x: 1 if x ==␣
 ↪'Yes' else 0)
df.head()
```

[2]:
```
        ID P01KPNREV P01KPNLEV P01KPACT30 P01HPNR12 P01HPNL12  \
0  9000296       Yes        No          No       Yes        No
1  9000622       Yes        No         Yes       Yes        No
2  9001695       Yes        No          No        No        No
3  9001897        No        No          No        No        No
4  9002411       Yes        No          No        No        No

                  V00SF2                  V00SF3        V00SF8 V00WPRKN1  …  \
0     Not limited at all     Not limited at all    Not at all      None  …
1  Yes, limited a little  Yes, limited a little    Moderately      Mild  …
2     Not limited at all     Not limited at all    Not at all      None  …
3     Not limited at all     Not limited at all    Not at all      None  …
4  Yes, limited a little  Yes, limited a little  A little bit      None  …

  P02WTGA V00RKEFFB V00RKEFFPT V00LKEFFB V00LKEFFPT V00RKRFXPN V00LKRFXPN  \
0      No        No         No        No         No         No         No
1      No        No         No        No         No        Yes         No
2      No       Yes        Yes        No         No         No         No
3      No       Yes         No        No         No         No         No
4     Yes        No         No        No         No         No         No

  V00LLWGT V00RLWGT cumulative_outcome
0     12.0     18.0                  0
1     14.0     14.0                  1
2     13.0     13.0                  1
3     16.0     17.0                  1
4     18.0     22.0                  0

[5 rows x 122 columns]
```

[4]:
```
## Drop first column (ID of the patient)

df = df.drop('ID', axis='columns')


## Check columns to convert to dummies
```

```
columns_dummies = []

for col in df.columns:
    if df[col].dtype != 'float64' and df[col].dtype != 'int64':
        columns_dummies.append(col)

print(f'Number of columns to dummy encode = {len(columns_dummies)} out of␣
 ↪{len(df.columns)}')


df_enc = pd.get_dummies(df, columns = columns_dummies, drop_first = True)
```

```
Number of columns to dummy encode = 103 out of 121
```

[5]:
```
df_enc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3702 entries, 0 to 3701
Columns: 228 entries, V00WOMTSL to V00LKRFXPN_Yes
dtypes: float64(17), int64(1), uint8(210)
memory usage: 1.2 MB
```

[6]:
```
df_enc.head()
```

[6]:

	V00WOMTSL	V00WOMTSR	P01KPACDCV	V00COMORB	V00HT25MM	V00WT25KG	\
0	0.0	0.0	0.0	0.0	1727.2	75.0	
1	0.0	20.9	15.0	1.0	1625.6	54.5	
2	0.0	NaN	0.0	0.0	1625.6	59.1	
3	0.0	0.0	0.0	0.0	1778.0	79.5	
4	1.1	2.1	0.0	0.0	1879.6	81.8	

	V00WTMAXKG	V00WTMINKG	V00BPDIAS	V00BPSYS	…	P01FAMKR_Yes	\
0	84.1	72.7	84.0	152.0	…	0	
1	62.3	50.9	60.0	136.0	…	0	
2	79.5	52.3	70.0	115.0	…	1	
3	84.1	70.5	82.0	150.0	…	0	
4	106.8	45.5	60.0	110.0	…	0	

	P02WTGA_Yes	V00RKEFFB_Yes	V00RKEFFPT_Too tender to examine	\
0	0	0	0	
1	0	0	0	
2	0	1	0	
3	0	1	0	
4	1	0	0	

	V00RKEFFPT_Yes	V00LKEFFB_Yes	V00LKEFFPT_Too tender to examine	\
0	0	0	0	
1	0	0	0	

```
2                1                0                                     0
3                0                0                                     0
4                0                0                                     0

   V00LKEFFPT_Yes  V00RKRFXPN_Yes  V00LKRFXPN_Yes
0               0               0               0
1               0               1               0
2               0               0               0
3               0               0               0
4               0               0               0

[5 rows x 228 columns]
```

[8]:
```
columns_with_nan_values = []
for col in df_enc.columns:
    if df_enc[col].isnull().values.any() == True:
        columns_with_nan_values.append(col)

print(columns_with_nan_values)
```

```
['V00WOMTSL', 'V00WOMTSR', 'P01KPACDCV', 'V00COMORB', 'V00HT25MM', 'V00WT25KG',
'V00WTMAXKG', 'V00WTMINKG', 'V00BPDIAS', 'V00BPSYS', 'V00DTVITD', 'V00DTCALC',
'V00DTCHOL', 'P01BMI', 'V00PASE', 'V00LLWGT', 'V00RLWGT']
```

[9]:
```
for col in columns_with_nan_values:
    col_mean = df[col].describe()['mean']
    df_enc[col] = df[col].fillna(col_mean)

df_enc.head()
```

[9]:
```
   V00WOMTSL  V00WOMTSR  P01KPACDCV  V00COMORB  V00HT25MM  V00WT25KG  \
0        0.0   0.000000         0.0        0.0     1727.2       75.0
1        0.0  20.900000        15.0        1.0     1625.6       54.5
2        0.0  10.076808         0.0        0.0     1625.6       59.1
3        0.0   0.000000         0.0        0.0     1778.0       79.5
4        1.1   2.100000         0.0        0.0     1879.6       81.8

   V00WTMAXKG  V00WTMINKG  V00BPDIAS  V00BPSYS  …  P01FAMKR_Yes  \
0        84.1        72.7       84.0     152.0  …             0
1        62.3        50.9       60.0     136.0  …             0
2        79.5        52.3       70.0     115.0  …             1
3        84.1        70.5       82.0     150.0  …             0
4       106.8        45.5       60.0     110.0  …             0

   P02WTGA_Yes  V00RKEFFB_Yes  V00RKEFFPT_Too tender to examine  \
0            0              0                                 0
1            0              0                                 0
2            0              1                                 0
```

```
3            0              1                                    0
4            1              0                                    0

   V00RKEFFPT_Yes  V00LKEFFB_Yes  V00LKEFFPT_Too tender to examine  \
0               0              0                                  0
1               0              0                                  0
2               1              0                                  0
3               0              0                                  0
4               0              0                                  0

   V00LKEFFPT_Yes  V00RKRFXPN_Yes  V00LKRFXPN_Yes
0               0               0               0
1               0               1               0
2               0               0               0
3               0               0               0
4               0               0               0

[5 rows x 228 columns]
```

[10]:
```
## Check if there are still NaN values in data frame
df_enc.isnull().values.any()
```

[10]: False

[11]:
```
## Divide data set in training and testing set

df_train, df_test = train_test_split(df_enc, test_size=0.3, random_state=88)
df_train.shape, df_test.shape
```

[11]: ((2591, 228), (1111, 228))

[15]:
```
X_train = df_train.drop(['cumulative_outcome'], axis=1)
y_train = df_train['cumulative_outcome']


X_test = df_test.drop(['cumulative_outcome'], axis=1)
y_test = df_test['cumulative_outcome']


logreg = LogisticRegression()
logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)

score = logreg.score(X_test, y_test)
print(score)
```

```
0.5742574257425742
```

```
/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[18]:
```
## Confusion matrix and accuracy
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score


y_test = df_test['cumulative_outcome']

cm = confusion_matrix(y_test, y_pred)
print ("Confusion Matrix : \n", cm)
print("Accuracy:", round(accuracy_score(y_test, y_pred),4))
print('TPR = ', cm[1,1]/(cm[1,1]+cm[0,1]))
print('FPR = ', cm[1,0]/(cm[1,0]+cm[0,0]))
```

```
Confusion Matrix :
 [[552  65]
 [408  86]]
Accuracy: 0.5743
TPR =  0.5695364238410596
FPR =  0.425
```

[44]:
```
logreg.coef_
```

[44]:
```
array([[-1.14219563e-02, -7.10408889e-03, -4.12329633e-03,
        -3.27051208e-03, -3.17514908e-03, -1.95325985e-03,
        -1.86105715e-03, -1.35870638e-03, -1.32238187e-03,
        -1.25053439e-03, -1.20369462e-03, -1.18035742e-03,
        -1.16200831e-03, -1.10170405e-03, -1.10170405e-03,
        -1.07446364e-03, -9.98325505e-04, -9.95786606e-04,
        -9.45349566e-04, -8.90553378e-04, -8.77223703e-04,
        -8.06138754e-04, -7.21270392e-04, -7.15197806e-04,
        -6.22735766e-04, -6.16474537e-04, -6.14071055e-04,
        -5.78481406e-04, -5.75572361e-04, -5.60175099e-04,
        -5.48544619e-04, -5.31013474e-04, -5.30363259e-04,
        -5.22513421e-04, -4.34637544e-04, -4.19281618e-04,
        -4.10346815e-04, -3.94857052e-04, -3.86518719e-04,
        -3.84390102e-04, -3.83839803e-04, -3.59701433e-04,
```

```
-3.51380122e-04, -2.99322416e-04, -2.99314238e-04,
-2.95334685e-04, -2.95130711e-04, -2.65358998e-04,
-2.64032674e-04, -2.60488832e-04, -2.53755455e-04,
-2.42697587e-04, -2.21358532e-04, -1.90365880e-04,
-1.85995429e-04, -1.83029295e-04, -1.68992022e-04,
-1.55783989e-04, -1.51360648e-04, -1.43454595e-04,
-1.14042528e-04, -1.11330605e-04, -1.10988829e-04,
-1.01774872e-04, -1.00553660e-04, -9.74183549e-05,
-8.89057739e-05, -8.38977877e-05, -8.24158682e-05,
-5.93865635e-05, -5.76184589e-05, -4.75448679e-05,
-3.95340537e-05, -3.28310001e-05, -1.79149060e-05,
-1.62052316e-05, -1.37324814e-05,  0.00000000e+00,
 0.00000000e+00,  1.78429149e-09,  1.27552649e-06,
 7.07710767e-06,  7.72017004e-06,  1.49770307e-05,
 1.87047139e-05,  2.49158990e-05,  2.68901618e-05,
 4.30437818e-05,  5.10902165e-05,  5.79464984e-05,
 6.00087892e-05,  6.10478668e-05,  6.57123392e-05,
 6.79270684e-05,  6.85963911e-05,  6.97566481e-05,
 7.54105445e-05,  7.71220241e-05,  7.72678234e-05,
 8.04903226e-05,  8.34650145e-05,  8.35515321e-05,
 9.38330005e-05,  9.97693011e-05,  1.04597914e-04,
 1.11834051e-04,  1.12497543e-04,  1.13848366e-04,
 1.15091698e-04,  1.18951281e-04,  1.26016992e-04,
 1.26876983e-04,  1.35131717e-04,  1.46055589e-04,
 1.53590324e-04,  1.56280500e-04,  1.61790383e-04,
 1.70305065e-04,  1.72676881e-04,  1.74710844e-04,
 1.80786929e-04,  1.84693406e-04,  1.87404055e-04,
 1.89310009e-04,  1.99184589e-04,  1.99184589e-04,
 2.04236631e-04,  2.05339886e-04,  2.08082659e-04,
 2.16124784e-04,  2.23490291e-04,  2.37057943e-04,
 2.47380811e-04,  2.71108896e-04,  2.71792531e-04,
 2.97296939e-04,  3.03795405e-04,  3.11169104e-04,
 3.11655934e-04,  3.14479278e-04,  3.16316287e-04,
 3.19851639e-04,  3.22573690e-04,  3.25673137e-04,
 3.39256596e-04,  3.50593485e-04,  3.62814380e-04,
 3.84521006e-04,  3.85446068e-04,  3.89908123e-04,
 4.08613091e-04,  4.13718875e-04,  4.16570568e-04,
 4.28258753e-04,  4.41863668e-04,  4.44334305e-04,
 4.54693004e-04,  4.83257423e-04,  4.98689983e-04,
 5.08205926e-04,  5.12561711e-04,  5.37144695e-04,
 5.53118234e-04,  5.62745885e-04,  5.68587713e-04,
 5.75857978e-04,  5.92569179e-04,  6.16548634e-04,
 6.39072089e-04,  6.63667583e-04,  6.76327862e-04,
 6.99531961e-04,  7.09102205e-04,  7.26835731e-04,
 7.51061084e-04,  7.85727967e-04,  7.86066566e-04,
 7.90465690e-04,  7.96877222e-04,  8.01990624e-04,
 8.10358395e-04,  8.15936153e-04,  8.36283601e-04,
```

```
       8.69601583e-04,  9.15457038e-04,  9.24283045e-04,
       9.89984603e-04,  1.02192374e-03,  1.04094963e-03,
       1.05806152e-03,  1.05840143e-03,  1.06812812e-03,
       1.07810884e-03,  1.09439672e-03,  1.11022796e-03,
       1.17775470e-03,  1.22269439e-03,  1.24011333e-03,
       1.32064762e-03,  1.35850037e-03,  1.45014947e-03,
       1.47673024e-03,  1.55009744e-03,  1.56074300e-03,
       1.56782010e-03,  1.97505915e-03,  1.98353238e-03,
       2.07556366e-03,  2.08726865e-03,  2.10571240e-03,
       2.16884423e-03,  2.34994588e-03,  2.42710601e-03,
       2.46309613e-03,  2.48290146e-03,  2.56175042e-03,
       2.57224500e-03,  2.82252246e-03,  2.89876378e-03,
       3.25668845e-03,  3.28107146e-03,  3.69811424e-03,
       6.37582372e-03,  6.82599807e-03,  7.08210911e-03,
       1.31559383e-02,  1.99116757e-02]])
```

[ ]: