Content:

Load Python Pakages

In [119…

```
#basics
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import missingno as msno
import warnings
warnings.filterwarnings("ignore")

#preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer


#feature selection
from sklearn.feature_selection import mutual_info_classif
from sklearn.feature_selection import SelectFromModel

#transformers and pipeline
from sklearn.pipeline import Pipeline
from sklearn import set_config


#algorithms
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.naive_bayes import GaussianNB


#model evaluation
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_auc_score,  f1_score, recall_score, prec
from sklearn.metrics import make_scorer
```

```
# Optuna and visualization tools
import optuna
from optuna.samplers import TPESampler

# Dealing with imbalanced target
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as imbpipeline

random_state = 10
```

First look to data

In [126…
```
# Read the data
original_train_df = pd.read_excel("C:\\Users\\Rıza\\Desktop\\evrim\\midus_data_evrim.xls

# reserved
pipe_data = original_train_df.copy()

# use for analysis
train_df = original_train_df.copy()
train_df.head()
```

Out[1268]:

	HgbA1c	Total cholesterol	triglycerides	HDL	Total cholesterol/HDL	LDL	Creatinine	DHEA-S	DHEA	Glucose	...	Ly
0	5.8622	197.0	71.0	111.0	1.774775	72.0	0.7	90.0	5.0	86.0	...	
1	7.4000	202.0	290.0	38.0	5.315789	106.0	0.9	145.0	4.3	94.0	...	
2	6.3000	192.0	108.0	52.0	3.692308	118.0	0.9	76.0	3.1	72.0	...	
3	5.4824	149.0	270.0	28.0	5.321429	67.0	0.7	150.0	12.9	76.0	...	
4	5.6000	226.0	213.0	35.0	6.457143	148.0	1.1	181.0	6.9	152.0	...	

5 rows × 36 columns

In [126…
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1255 entries, 0 to 1254
Data columns (total 36 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   HgbA1c                                  1235 non-null   float64
 1   Total cholesterol                       1244 non-null   float64
 2   triglycerides                           1243 non-null   float64
 3   HDL                                     1242 non-null   float64
 4   Total cholesterol/HDL                   1242 non-null   float64
 5   LDL                                     1242 non-null   float64
 6   Creatinine                              1244 non-null   float64
 7   DHEA-S                                  1239 non-null   float64
 8   DHEA                                    1240 non-null   float64
 9   Glucose                                 1236 non-null   float64
 10  Insulin                                 1236 non-null   float64
 11  IGF-1                                   1236 non-null   float64
 12  IL-6                                    1241 non-null   float64
```

```
 13  IL-8                                              1241 non-null   float64
 14  IL-10                                             1241 non-null   float64
 15  TNF alpha                                         1241 non-null   float64
 16  Fibrinogen                                        1235 non-null   float64
 17  CRP                                               1235 non-null   float64
 18  E-selectin                                        1241 non-null   float64
 19  Intercellular adhesion molecule 1                 1241 non-null   float64
 20  Lutein                                            1232 non-null   float64
 21  Zeaxanthin                                        1232 non-null   float64
 22  Beta-cryptoxanthin                                1232 non-null   float64
 23  13-cis-beta-carotene                              1224 non-null   float64
 24  Alpha carotene                                    1228 non-null   float64
 25  Trans beta carotene                               1220 non-null   float64
 26  Lycopene                                          1231 non-null   float64
 27  Gamma tocopherol                                  1232 non-null   float64
 28  Alpha tocoperol                                   1232 non-null   float64
 29  Retinol                                           1232 non-null   float64
 30  n-Telopeptide type 1 collagen                     1243 non-null   float64
 31  Bone specific alkaline phosphatase                1243 non-null   float64
 32  Amino terminal propeptide type 1 procollagen      1243 non-null   float64
 33  Age                                               1255 non-null   int64
 34  Sex                                               1255 non-null   int64
 35  Gait speed                                        1238 non-null   float64
dtypes: float64(34), int64(2)
memory usage: 353.1 KB
```

In [127…

```
#An important decision: Include only Age and Sex along with blood sample measurements.
#Other sample chracteristics and redundant features are exculuded
#train_df = train_df.drop(['BMI','Education','PA level','WHR','Serum IL6 (pg/mL)','Serum

#correct column name
train_df.rename(columns={'HgbA1c': 'HgA1c'}, inplace=True)

#Important decision 2: Analysis include only elderly people
train_df = train_df[train_df['Age']>= 60]
train_df.shape
```

Out[1270]:

```
(405, 36)
```

Data Cleaning (Missing value handling)

Lets check and visualize missing values.

In [127…

```
missing = pd.DataFrame(train_df.isnull().sum().sort_values(ascending=False))
missing.columns = ["missing_count"]
missing = missing.loc[(missing!=0).any(axis=1)]
missing["missing_percent"] = missing[0:] / len(train_df) *100
missing.style.background_gradient('viridis')
```

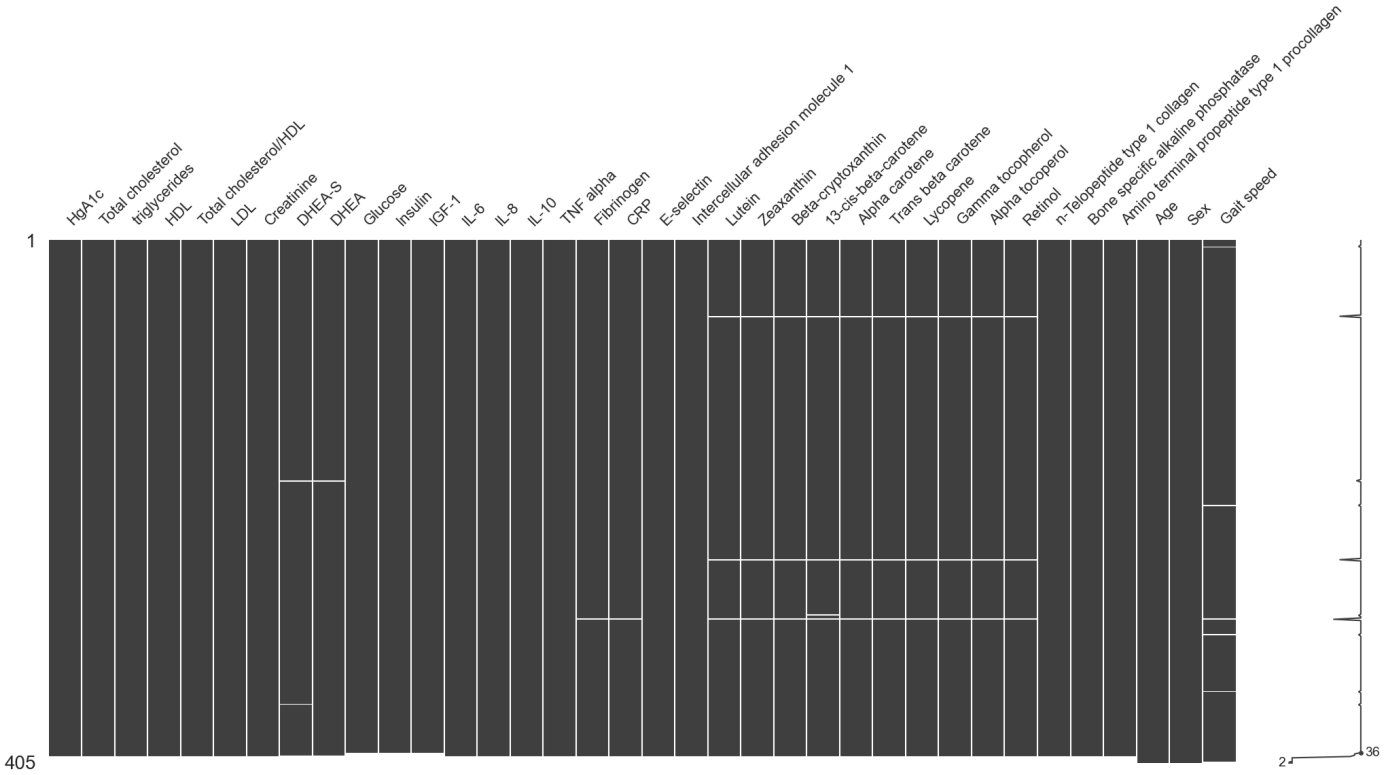
Out[1271]:

	missing_count	missing_percent
13-cis-beta-carotene	9	2.222222
Insulin	8	1.975309
Zeaxanthin	8	1.975309
Beta-cryptoxanthin	8	1.975309
Alpha carotene	8	1.975309
Trans beta carotene	8	1.975309

Lycopene	8	1.975309
Gamma tocopherol	8	1.975309
IGF-1	8	1.975309
Lutein	8	1.975309
Glucose	8	1.975309
DHEA-S	8	1.975309
Alpha tocoperol	8	1.975309
Retinol	8	1.975309
DHEA	7	1.728395
Fibrinogen	6	1.481481
CRP	6	1.481481
Gait speed	6	1.481481
n-Telopeptide type 1 collagen	5	1.234568
Bone specific alkaline phosphatase	5	1.234568
Amino terminal propeptide type 1 procollagen	5	1.234568
HgA1c	5	1.234568
Intercellular adhesion molecule 1	5	1.234568
Total cholesterol	5	1.234568
TNF alpha	5	1.234568
IL-10	5	1.234568
IL-8	5	1.234568
IL-6	5	1.234568
Creatinine	5	1.234568
LDL	5	1.234568
Total cholesterol/HDL	5	1.234568
HDL	5	1.234568
triglycerides	5	1.234568
E-selectin	5	1.234568

In [127…

```
%matplotlib inline
msno.matrix(train_df)
plt.show()
```



I will drop missing values on Gait speed since it will be our target.

In [127…
```
train_df = train_df.dropna(subset=['Gait speed'])
```

Let's observe missing values again.

In [127…
```
missing = pd.DataFrame(train_df.isnull().sum().sort_values(ascending=False))
missing.columns = ["count"]
missing = missing.loc[(missing!=0).any(axis=1)]
missing["percent"] = missing[0:] / len(train_df) *100
missing.style.background_gradient('viridis')
```

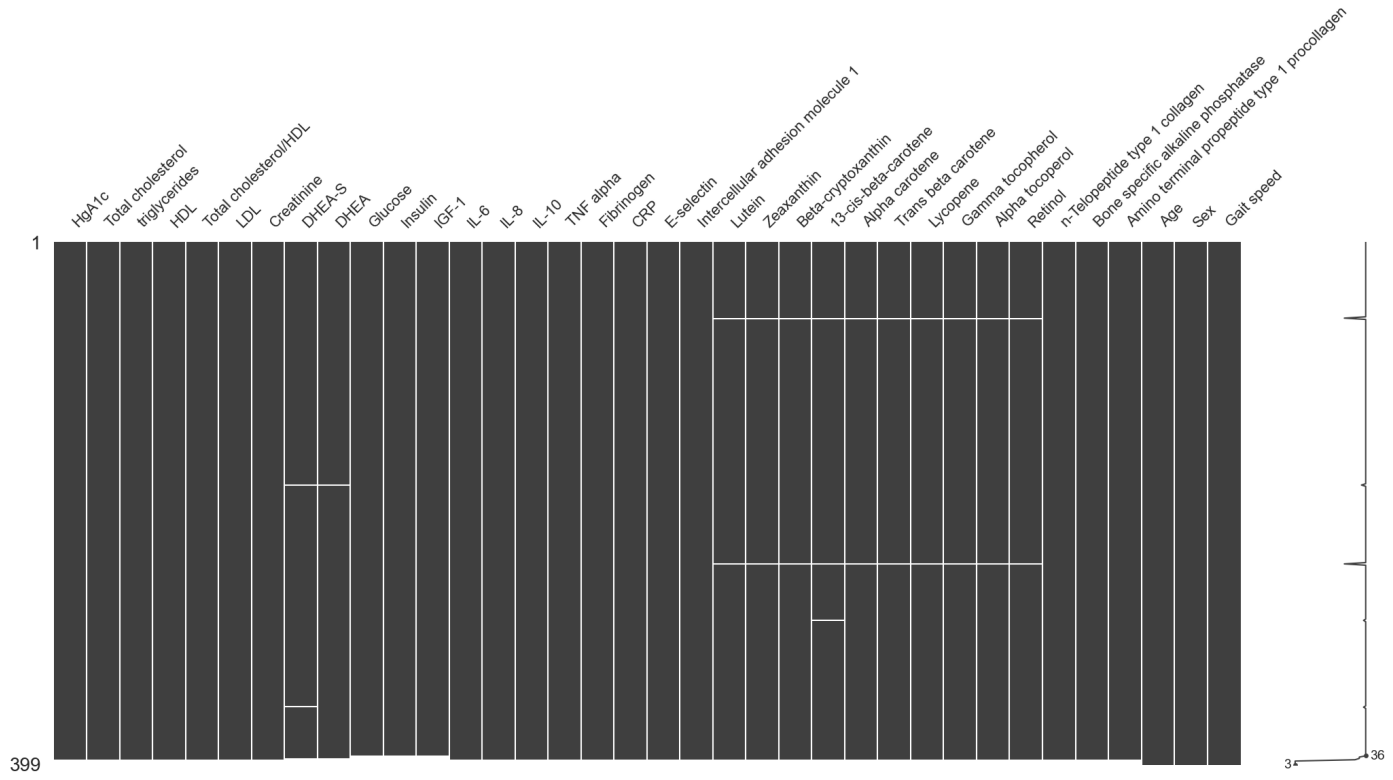
Out[1274]:

	count	percent
Glucose	7	1.754386
Insulin	7	1.754386
13-cis-beta-carotene	7	1.754386
DHEA-S	7	1.754386
IGF-1	7	1.754386
Beta-cryptoxanthin	6	1.503759
Alpha carotene	6	1.503759
Trans beta carotene	6	1.503759
Lycopene	6	1.503759
Gamma tocopherol	6	1.503759
Zeaxanthin	6	1.503759
Lutein	6	1.503759
DHEA	6	1.503759
Alpha tocoperol	6	1.503759

Retinol	6	1.503759
n-Telopeptide type 1 collagen	4	1.002506
Bone specific alkaline phosphatase	4	1.002506
Amino terminal propeptide type 1 procollagen	4	1.002506
HgA1c	4	1.002506
E-selectin	4	1.002506
Intercellular adhesion molecule 1	4	1.002506
Total cholesterol	4	1.002506
CRP	4	1.002506
Fibrinogen	4	1.002506
TNF alpha	4	1.002506
IL-10	4	1.002506
IL-8	4	1.002506
IL-6	4	1.002506
Creatinine	4	1.002506
LDL	4	1.002506
Total cholesterol/HDL	4	1.002506
HDL	4	1.002506
triglycerides	4	1.002506

In [127…

```
msno.matrix(train_df)
plt.show()
```



In [127…

```
missing_rows_count = train_df.isnull().any(axis=1).sum()

print("Total rows with missing values:", missing_rows_count)
```

```
Total rows with missing values: 12
```

We have 399 samples that has gait speed information. This will be the final data that we will use for analysis & modelling.

There are 12 observations that include missing values. I will apply mean imputation for the missing values in modelling phase.

Descriptive statistics

Let's report basic descriptive statistics for data.

In [127…
```
#descriptive statistics
train_df.describe().T
```

Out[1277]:

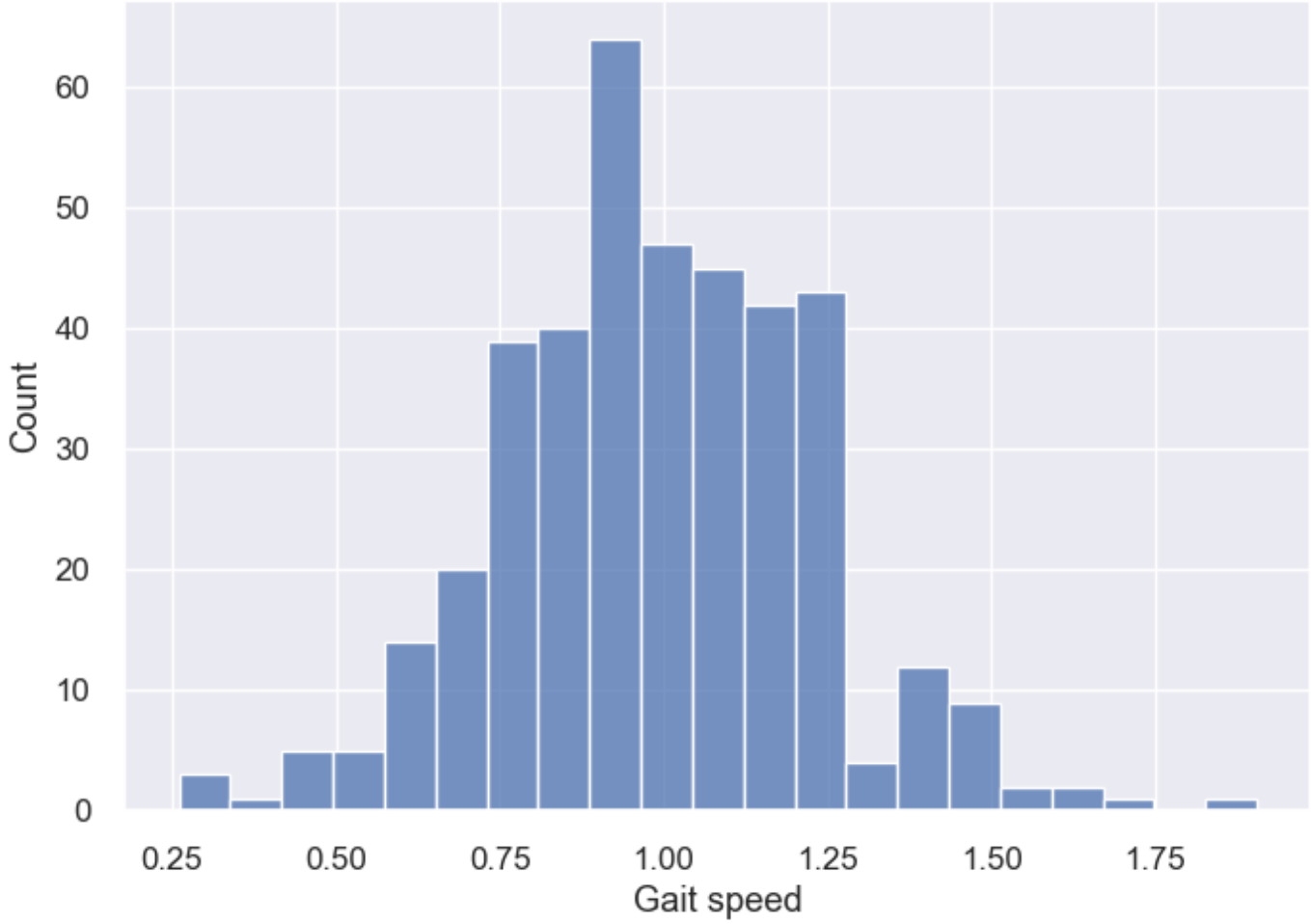
	count	mean	std	min	25%	50%	75%	max
HgA1c	395.0	6.228691	0.957446	4.000000	5.735600	5.988800	6.372700	12.0184
Total cholesterol	395.0	179.964557	40.590736	91.000000	150.000000	176.000000	209.000000	308.0000
triglycerides	395.0	124.992405	69.865007	37.000000	80.000000	105.000000	151.000000	444.0000
HDL	395.0	55.600000	17.204710	19.000000	43.500000	53.000000	67.000000	115.0000
Total cholesterol/HDL	395.0	3.490188	1.206184	1.338235	2.648309	3.260000	4.046129	10.0000
LDL	395.0	99.400000	35.083034	16.000000	74.000000	94.000000	123.500000	231.0000
Creatinine	395.0	0.834430	0.202971	0.500000	0.700000	0.800000	1.000000	2.1000
DHEA-S	392.0	106.272959	72.843275	2.000000	54.000000	91.000000	146.000000	495.0000
DHEA	393.0	6.061089	4.089507	0.400000	3.100000	4.900000	8.200000	24.0000
Glucose	392.0	101.696429	26.586099	5.000000	90.000000	96.500000	104.000000	377.0000
Insulin	392.0	12.589286	10.062322	1.000000	6.000000	10.000000	16.000000	89.0000
IGF-1	392.0	125.454082	49.518479	29.000000	90.000000	121.000000	155.000000	291.0000
IL-6	395.0	1.267139	1.214476	0.190000	0.700000	0.950000	1.380000	13.9300
IL-8	395.0	15.938987	13.573989	4.960000	10.590000	14.040000	17.340000	235.1400
IL-10	395.0	0.474886	2.279680	0.060000	0.180000	0.250000	0.345000	43.6600
TNF alpha	395.0	2.443747	0.946566	0.510000	1.910000	2.240000	2.775000	9.5200
Fibrinogen	395.0	361.050633	91.543629	123.000000	303.000000	349.000000	409.500000	857.0000
CRP	395.0	2.796807	4.272309	0.100186	0.770000	1.450000	3.360000	32.1000
E-selectin	395.0	39.901089	18.820551	0.090000	27.650000	37.170000	47.590000	149.2300
Intercellular adhesion molecule 1	395.0	303.284456	106.478817	44.000000	240.470000	286.460000	346.100000	896.4700
Lutein	393.0	0.320529	0.217711	0.037400	0.178300	0.268300	0.401500	1.6350
Zeaxanthin	393.0	0.068958	0.046197	0.012400	0.041200	0.058100	0.080100	0.3713
Beta-cryptoxanthin	393.0	0.218787	0.194538	0.029700	0.106000	0.159000	0.253400	1.4406

13-cis-beta-carotene	392.0	0.080794	0.077477	0.000000	0.039000	0.061100	0.094175	0.9081
Alpha carotene	393.0	0.095572	0.120648	0.000800	0.037200	0.067100	0.109100	1.7418
Trans beta carotene	393.0	0.713740	0.889518	0.023500	0.255500	0.429200	0.810000	7.0869
Lycopene	393.0	0.422764	0.235637	0.022600	0.266400	0.380600	0.519100	1.8364
Gamma tocopherol	393.0	3.282850	2.559660	0.440000	1.730000	2.470000	3.960000	20.8200
Alpha tocoperol	393.0	31.394555	12.925956	5.610000	22.800000	29.200000	36.710000	93.1600
Retinol	393.0	1.916107	0.705643	0.710000	1.480000	1.810000	2.170000	6.9000
n-Telopeptide type 1 collagen	395.0	14.390506	7.028894	2.040000	9.910000	13.070000	16.805000	54.6800
Bone specific alkaline phosphatase	395.0	28.168000	12.022478	7.300000	21.410000	25.740000	32.170000	163.9100
Amino terminal propeptide type 1 procollagen	395.0	54.471013	28.149825	7.510000	33.105000	51.000000	69.615000	182.7900
Age	399.0	68.260652	6.431098	60.000000	63.000000	67.000000	73.000000	84.0000
Sex	399.0	1.533835	0.499480	1.000000	1.000000	2.000000	2.000000	2.0000
Gait speed	399.0	0.988025	0.235064	0.260513	0.835225	0.983226	1.128889	1.9050

Let's also observe target variable (Gate Speed)

In [127…
```
sns.histplot(x='Gait speed',  data=train_df )
```

Out[1278]:
```
<Axes: xlabel='Gait speed', ylabel='Count'>
```



We will discretize the target (Gait speed) as below 0.8 and above 0.8 and follow a classification approach. This is an important decision. The justification of this decision comes from the fact that a Gait speed 0.8 is a critical threshold for fall risk. And it has a huge meaning from a clinical perspective.

- Class 1 indicates below-0.8 (observations that has a fall risk)
- Class 0 indicated above-0.8 (observations that has a normal Gait speed)

In [127…
```
# Gait speed is mapped to 0 and 1. 1 indicates samples has a Gait speed below 0.8 m / s
train_df['Gait speed'] = (train_df['Gait speed'] < 0.8 ).astype(int)
```

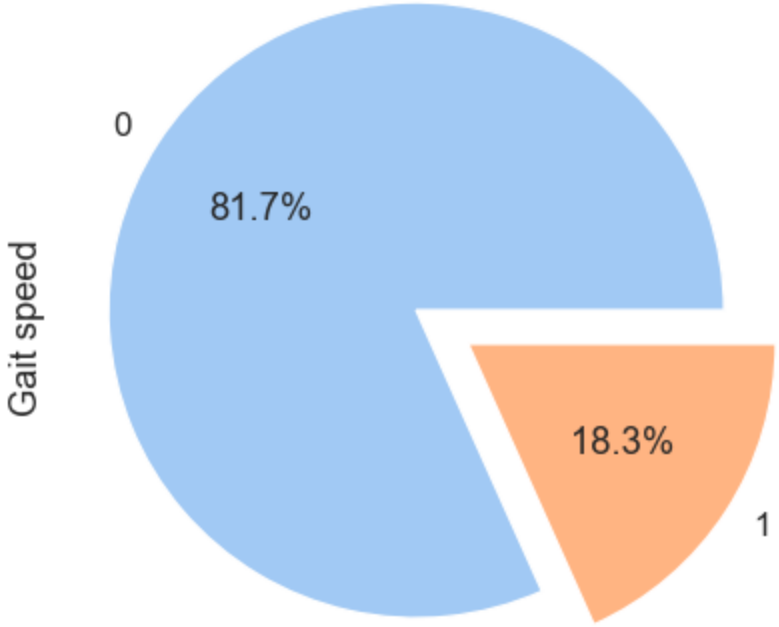
In [128…
```
train_df['Gait speed'].value_counts()
```

Out[1280]:
```
0    326
1     73
Name: Gait speed, dtype: int64
```

We have 73 samples belonging class 1 and 326 samples class 0

Let's make a visualization for class distribution.

In [128…
```
#Target Distribution plot
plt.figure(figsize=(5,5))
palette_color = sns.color_palette('pastel')
explode = [0.1, 0.1]
train_df.groupby('Gait speed')['Gait speed'].count().plot.pie(colors=palette_color,explo
```



81.7% of samples are above 0.8. 18.3% of samples are below 0.8.

Target variable distribution is quite imbalanced. At the modeling stage, the Synthetic Minority Over-sampling Technique (SMOTE) and class_weight adjustment are used to deal with this imbalance distribution. SMOTE performed better, this notebook includes only SMOTE version of the work.

Due to imbalanced nature of the data F1 score is used as an optimization metric.

F1 score is defined as the harmonic mean of precision and recall.

Exploratory Data Analysis

On this part we will explore relationship beetween target and features. This part includes:

- Some visualization
- Reporting mutual information scores for features with respect to target
- And a correlation matrix between features

In [128…
```
train_df.nunique().sort_values()
```

Out[1282]:
```
Gait speed                2
Sex                       2
Creatinine               14
Age                      25
Insulin                  44
HDL                      74
IL-10                    79
Glucose                  80
HgA1c                    86
LDL                     132
Total cholesterol       155
```

```
DHEA                                          162
IGF-1                                         167
triglycerides                                 170
IL-6                                          182
Retinol                                       183
DHEA-S                                        189
TNF alpha                                     199
Fibrinogen                                    221
Gamma tocopherol                              280
CRP                                           281
Zeaxanthin                                    326
13-cis-beta-carotene                          333
n-Telopeptide type 1 collagen                 341
IL-8                                          347
Alpha carotene                                351
Bone specific alkaline phosphatase            366
Alpha tocoperol                               371
Lutein                                        375
Beta-cryptoxanthin                            376
E-selectin                                    379
Lycopene                                      380
Total cholesterol/HDL                         380
Trans beta carotene                           384
Amino terminal propeptide type 1 procollagen  389
Intercellular adhesion molecule 1             392
dtype: int64
```

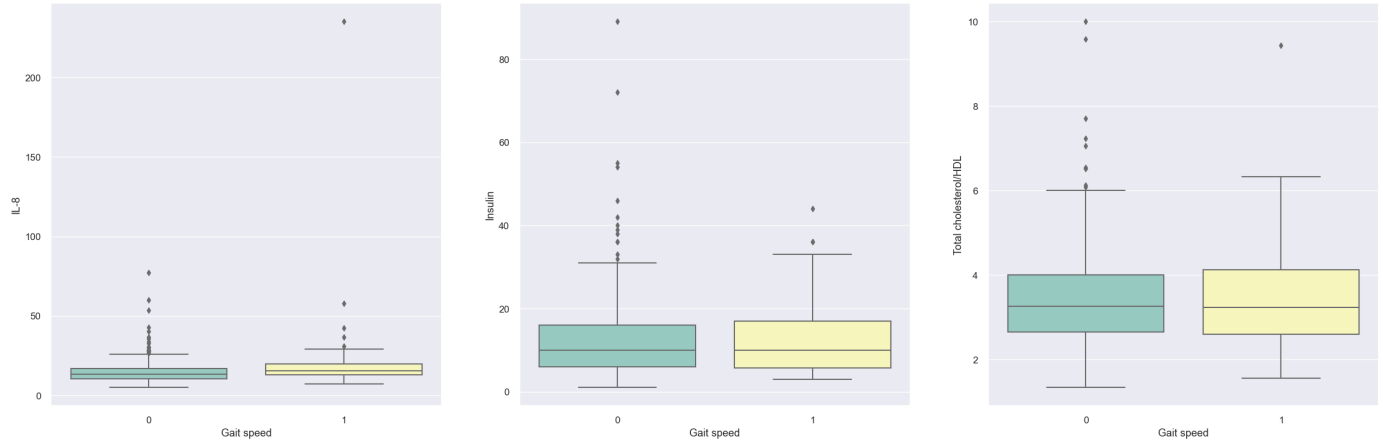
In [128…

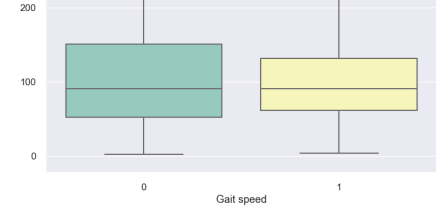
```python
# Store continuous and discrete features to different lists for visualization purposes
feature_list = [feature for feature in train_df.columns if not feature  == "Gait speed"]

discrete_features = [ 'Sex']

continuous_features = list(set(feature_list) - set(discrete_features))

assert feature_list.sort() == (discrete_features + continuous_features).sort()
```

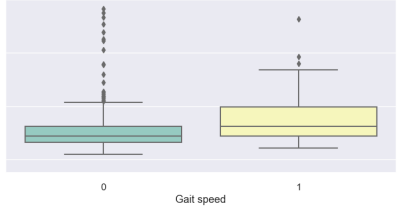
Let's observe features with respect to target.

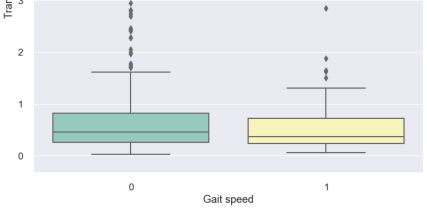
In [128…

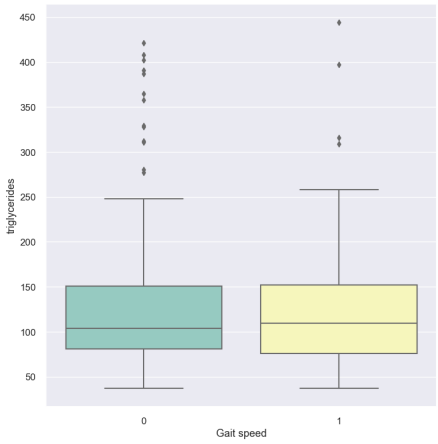
```python
#Box-plots
fig, ax = plt.subplots(12, 3, figsize=(30, 130))
for var, subplot in zip(continuous_features, ax.flatten()):
    sns.boxplot(x='Gait speed', y=var, data=train_df, ax=subplot, palette='Set3')
```

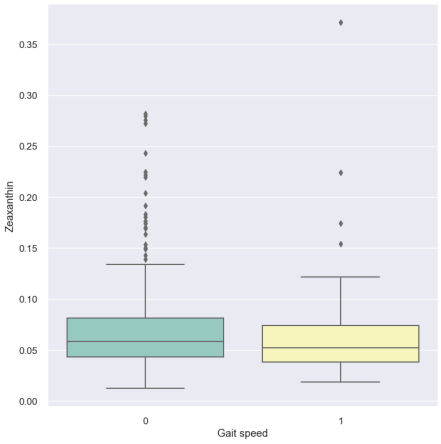


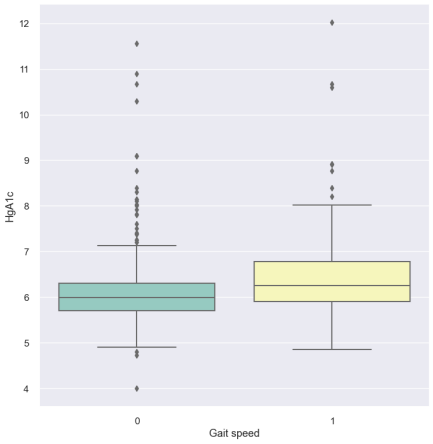

200
100
0
0
1
Gait speed

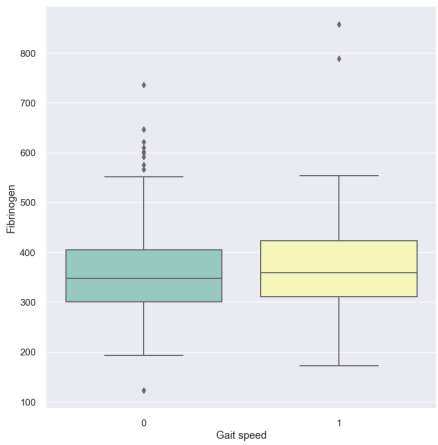

6
4
2
0
0
1
Gait speed

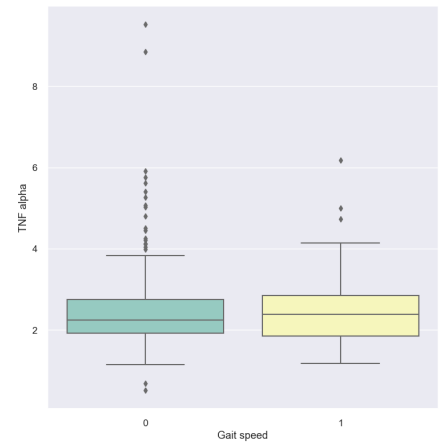

Tran
3
2
1
0
0
1
Gait speed

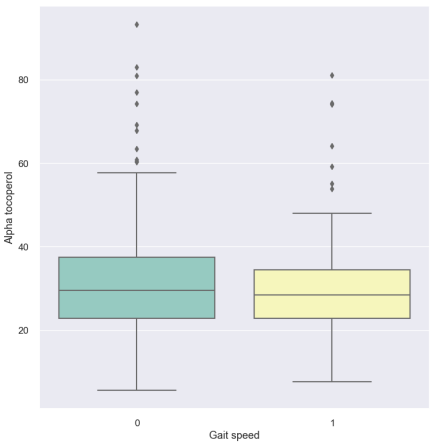

triglycerides
450
400
350
300
250
200
150
100
50
0
1
Gait speed

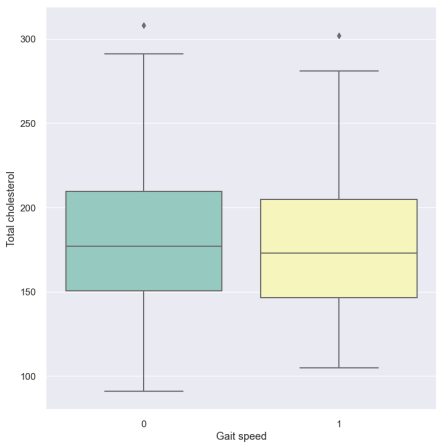

Zeaxanthin
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
1
Gait speed

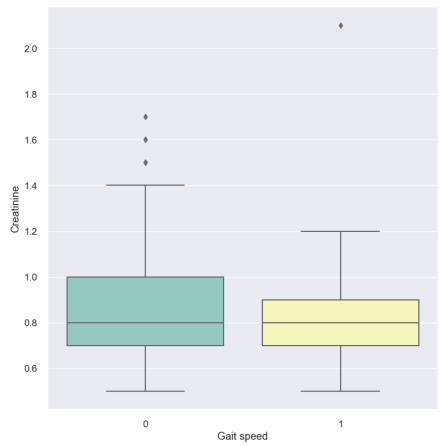

HgA1c
12
11
10
9
8
7
6
5
4
0
1
Gait speed

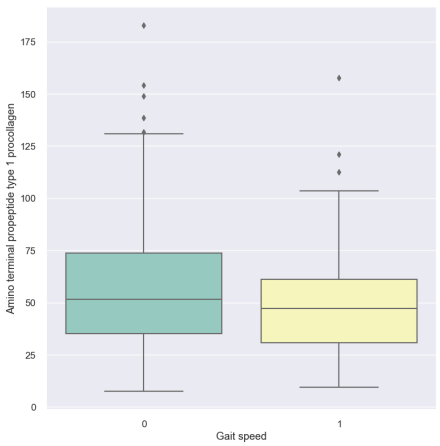

Fibrinogen
800
700
600
500
400
300
200
100
0
1
Gait speed

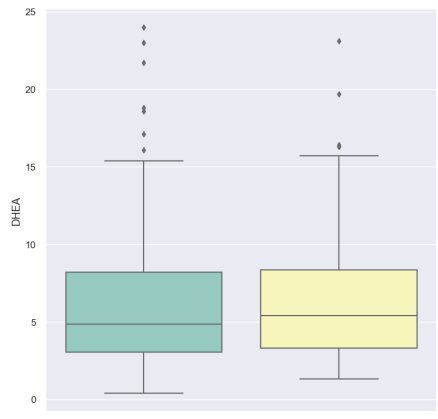

TNF alpha
8
6
4
2
0
1
Gait speed

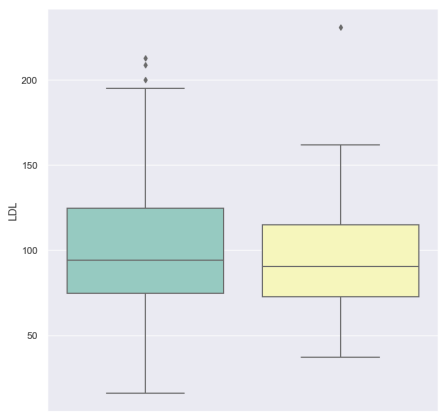

Alpha tocoperol
80
60
40
20
0
1
Gait speed

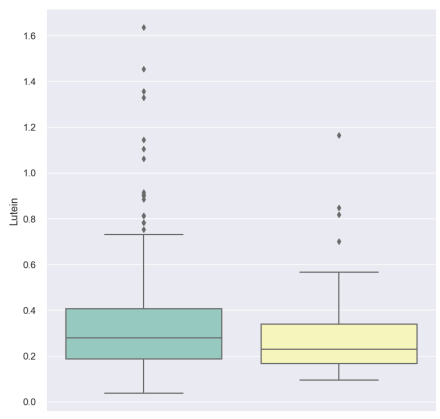

Total cholesterol
300
250
200
150
100
0
1
Gait speed

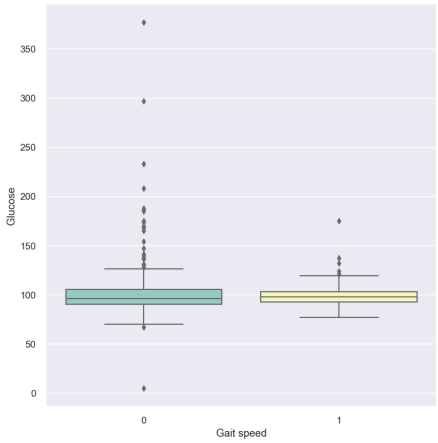

Creatinine
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0
1
Gait speed

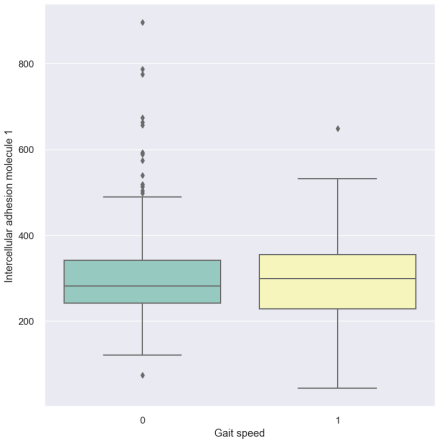

Amino terminal propeptide type 1 procollagen
175
150
125
100
75
50
25
0
0
1
Gait speed

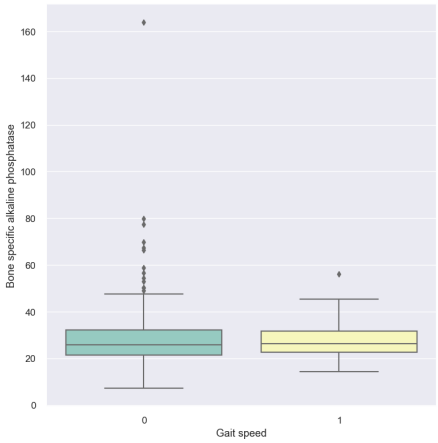

DHEA
25
20
15
10
5
0
0
1
Gait speed

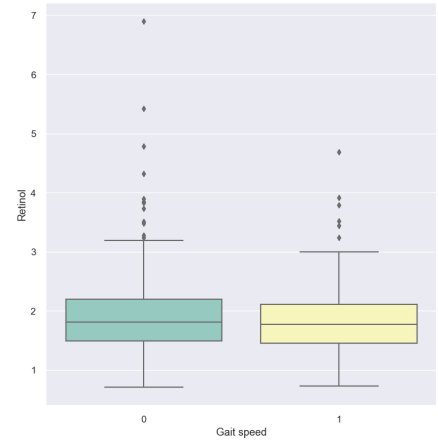

LDL
200
150
100
50
0
1
Gait speed

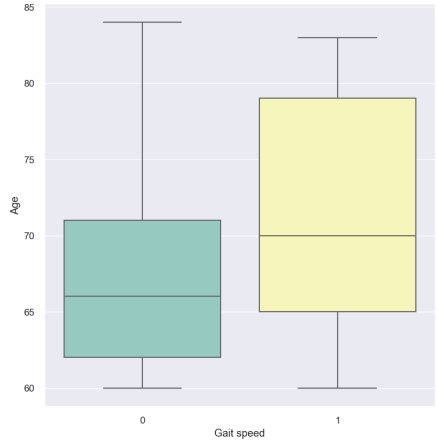

Lutein
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1
Gait speed

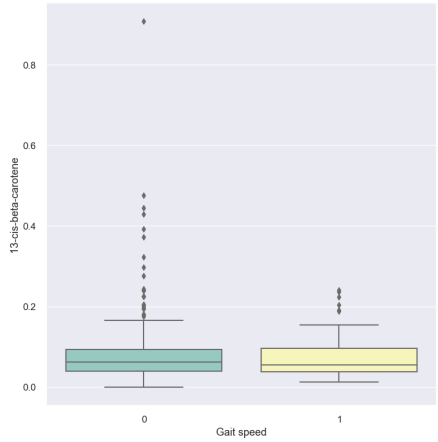

Glucose
Gait speed

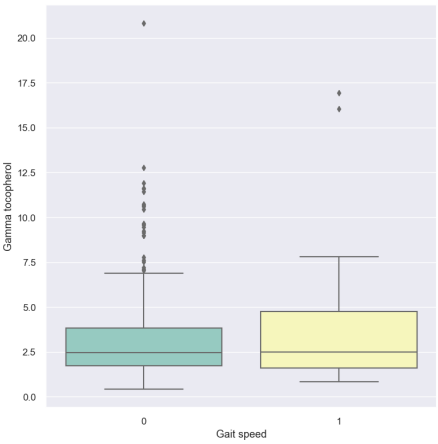

Intercellular adhesion molecule 1
Gait speed

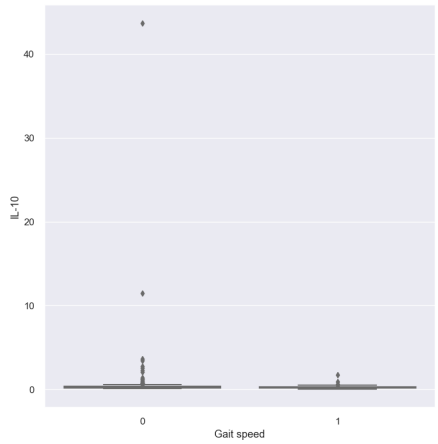

Bone specific alkaline phosphatase
Gait speed

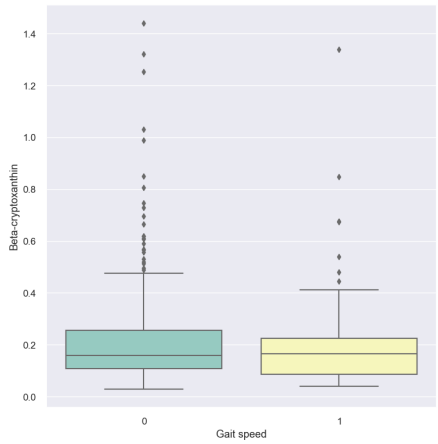

Retinol
Gait speed

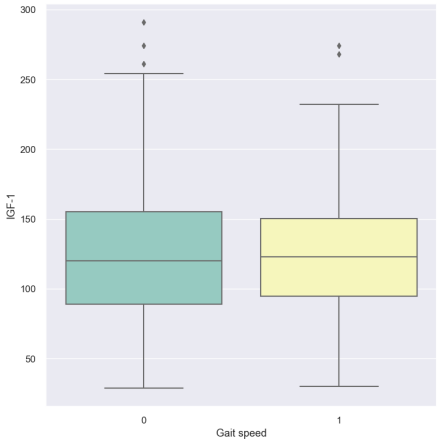

Age
Gait speed

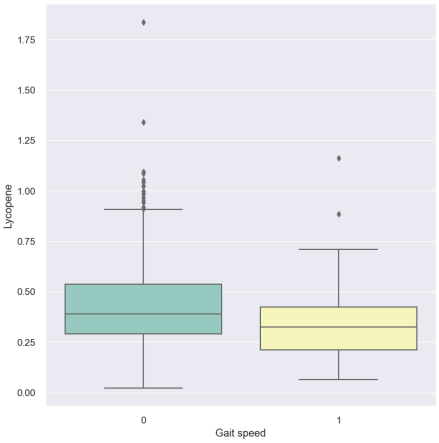

13-cis-beta-carotene
Gait speed

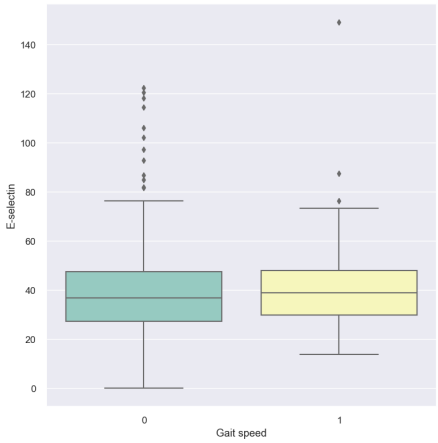

Gamma tocopherol
Gait speed

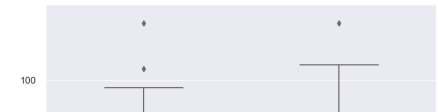

IL-10
Gait speed

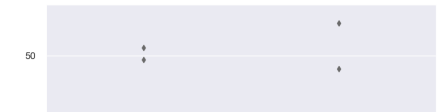

Beta-cryptoxanthin
Gait speed

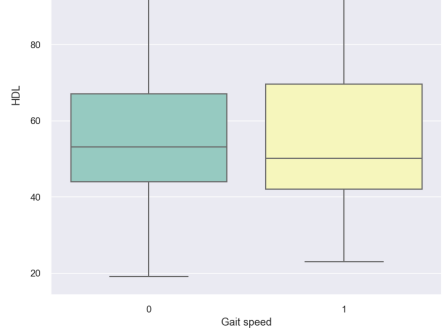

IGF-1
Gait speed

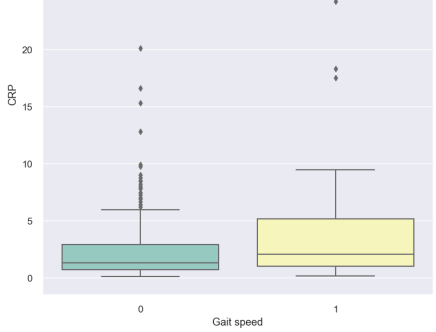

Lycopene
Gait speed

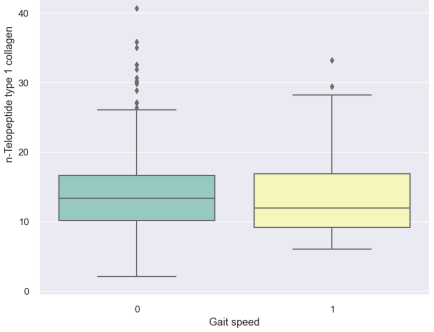

E-selectin
Gait speed

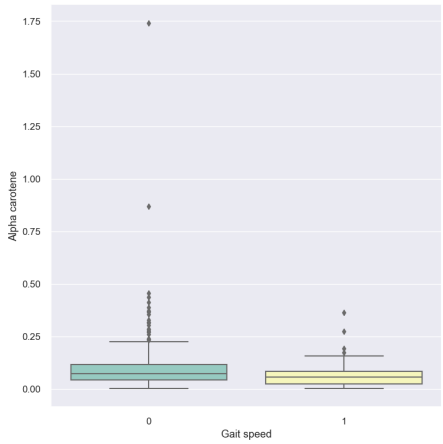


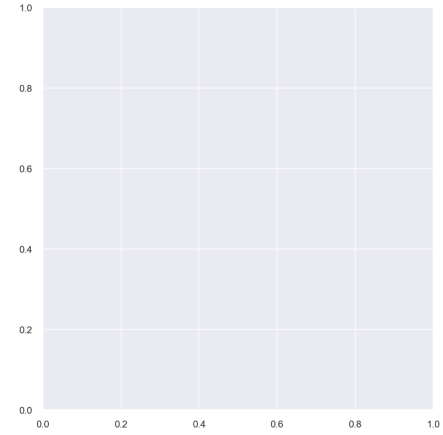


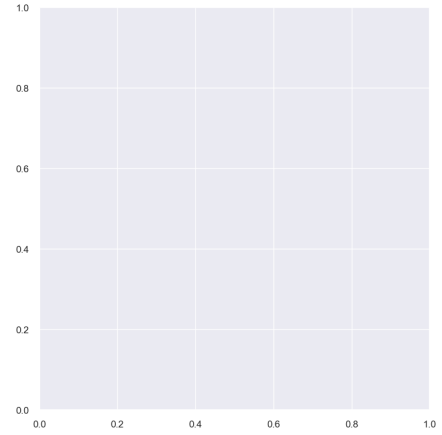






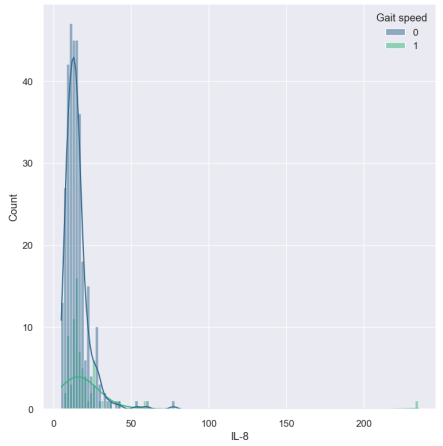


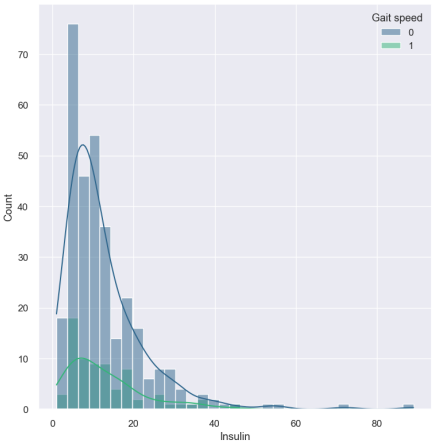


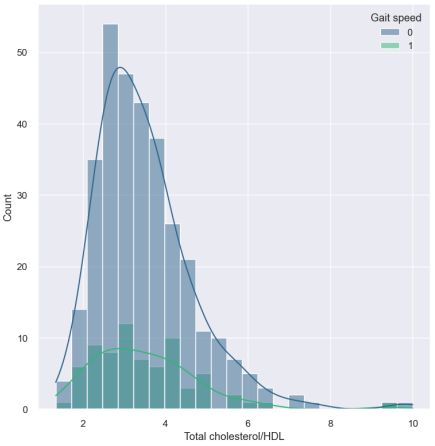


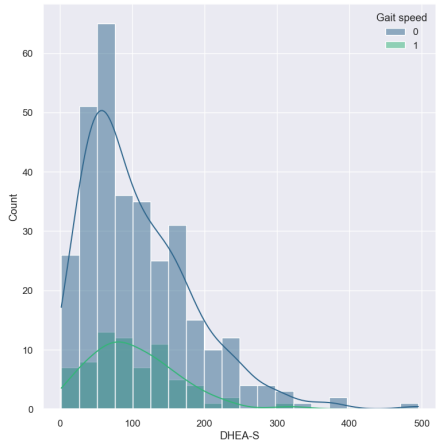
In [128…

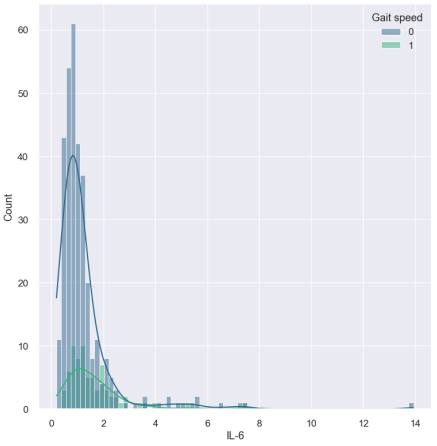
```
#Histograms
fig, ax = plt.subplots(12, 3, figsize=(30, 130))
for var, subplot in zip(continuous_features, ax.flatten()):
    sns.histplot(x=var,  data=train_df, ax=subplot, hue='Gait speed', kde=True ,  palett
```

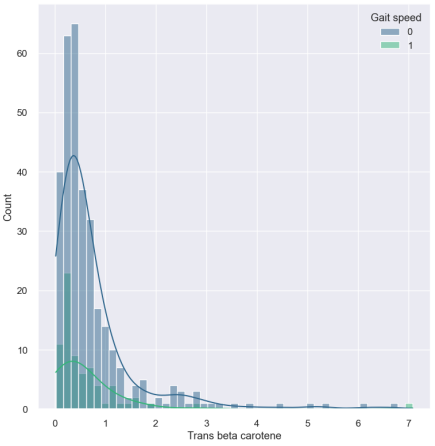


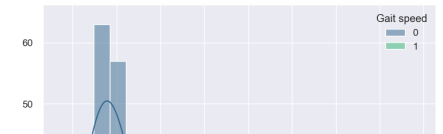


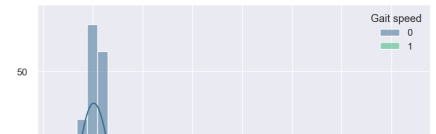


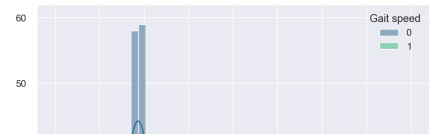


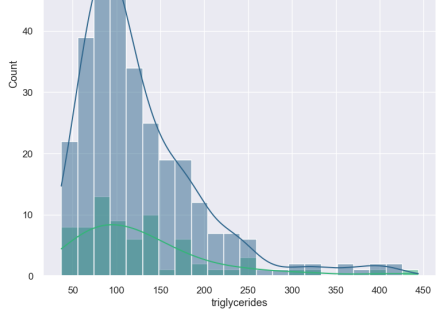


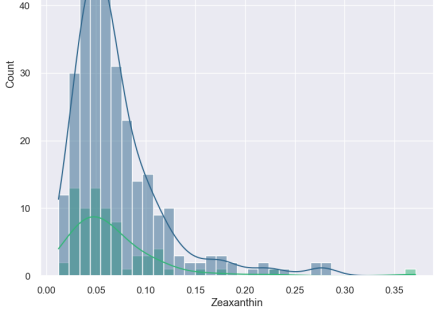


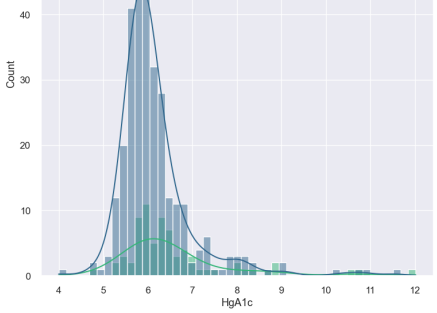


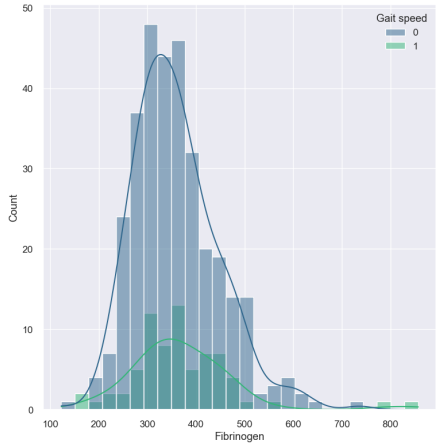


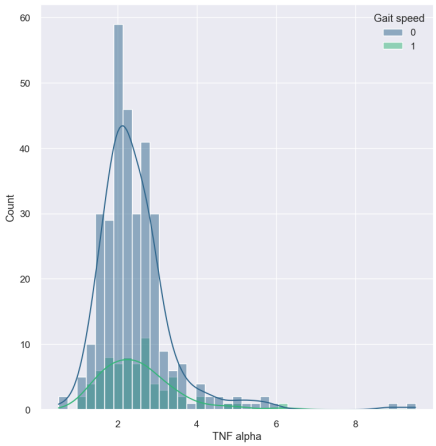


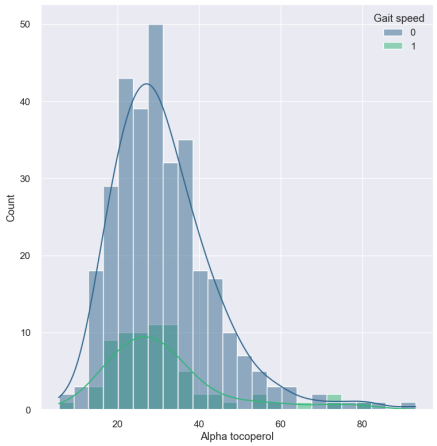

Count
triglycerides

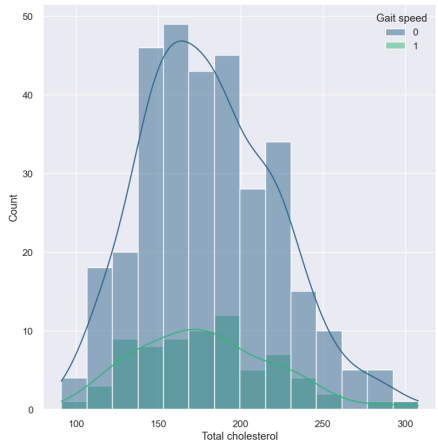

Count
Zeaxanthin

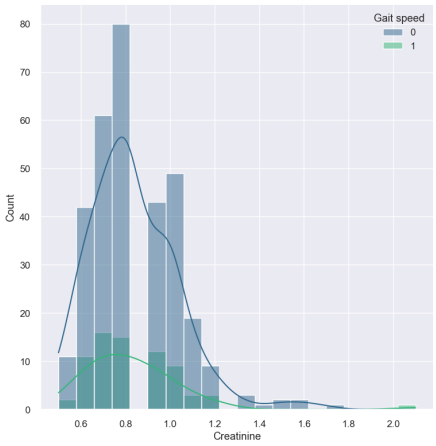

Count
HgA1c

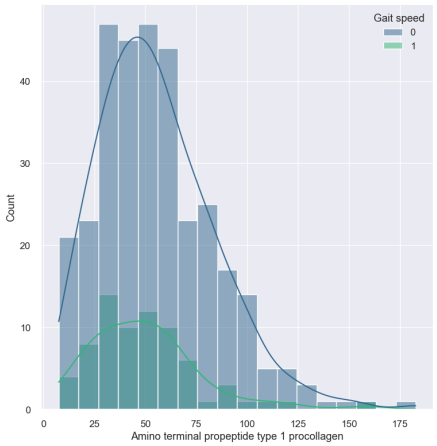

Gait speed
0
1
Count
Fibrinogen

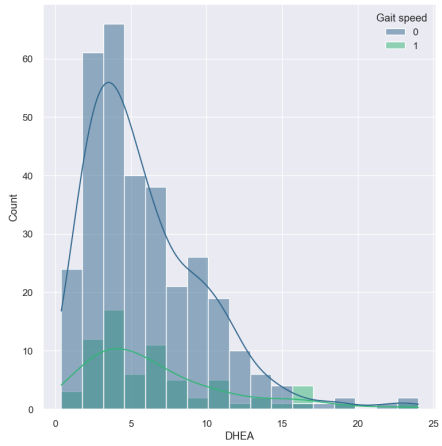

Gait speed
0
1
Count
TNF alpha

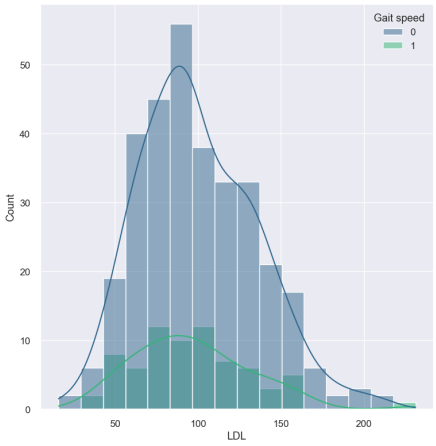

Gait speed
0
1
Count
Alpha tocoperol

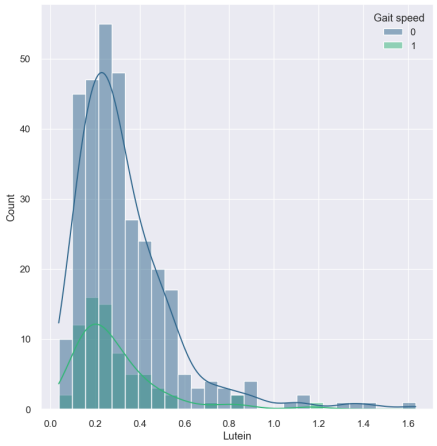

Gait speed
0
1
Count
Total cholesterol

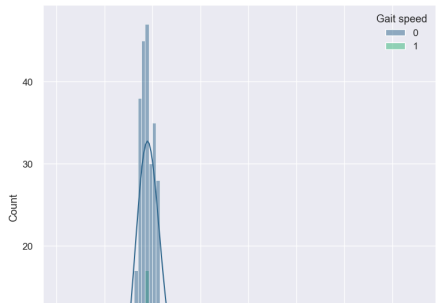

Gait speed
0
1
Count
Creatinine

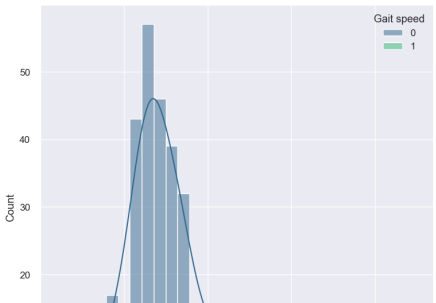

Gait speed
0
1
Count
Amino terminal propeptide type 1 procollagen

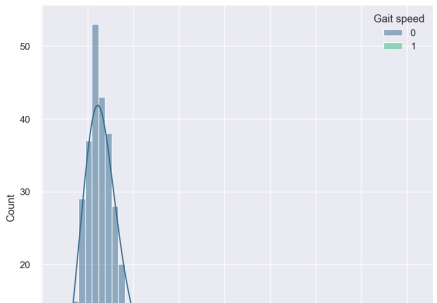

Gait speed
0
1
Count
DHEA

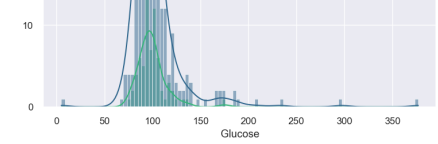

Gait speed
0
1
Count
LDL

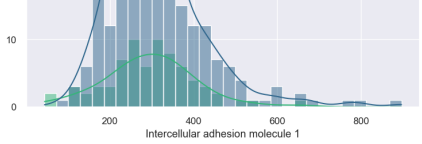

Gait speed
0
1
Count
Lutein

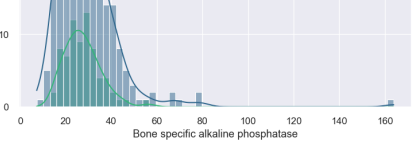

Gait speed
0
1
Count

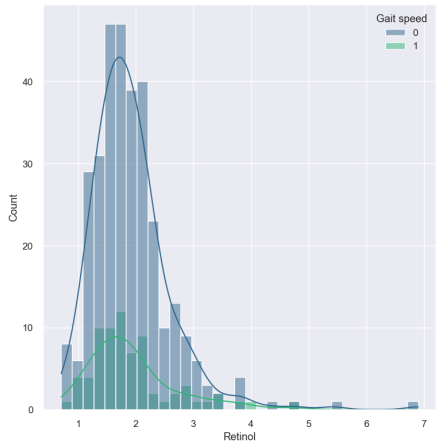

Gait speed
0
1
Count

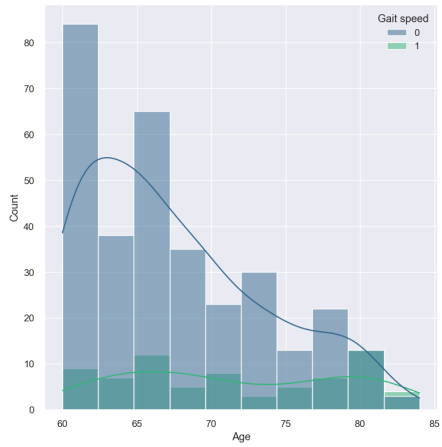

Gait speed
0
1
Count

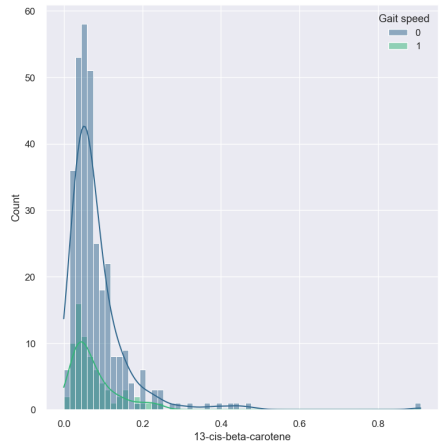

Glucose

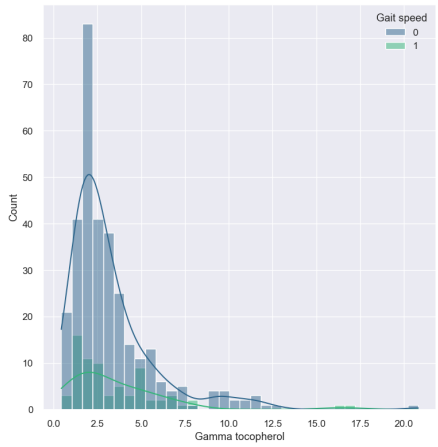

Intercellular adhesion molecule 1

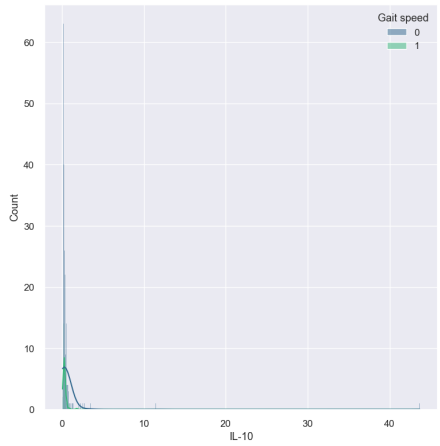

Bone specific alkaline phosphatase

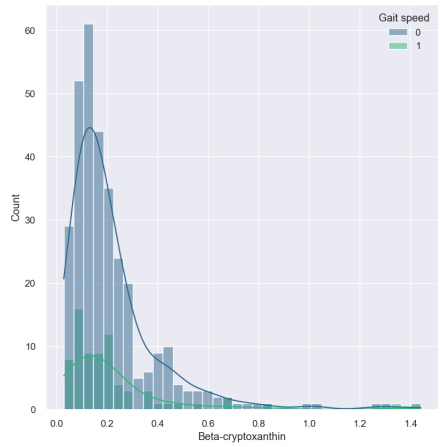

Gait speed
0
1
Count
Retinol

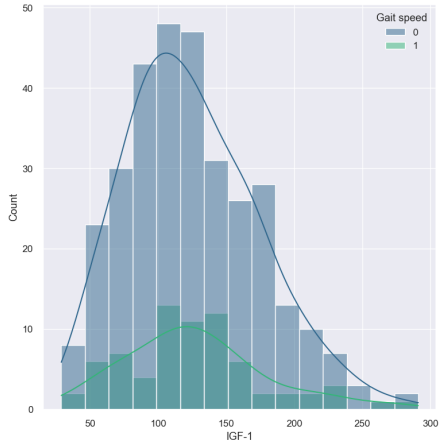

Gait speed
0
1
Count
Age

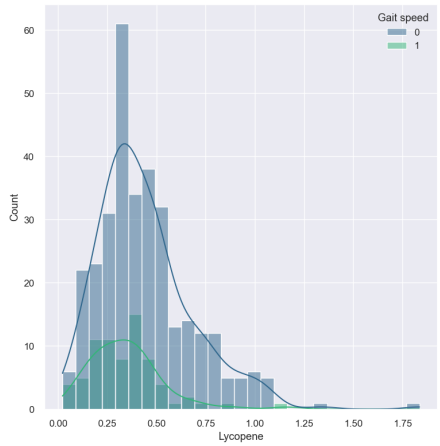

Gait speed
0
1
Count
13-cis-beta-carotene

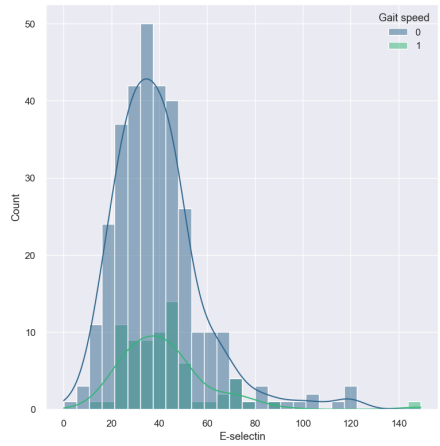

Gait speed
0
1
Count
Gamma tocopherol

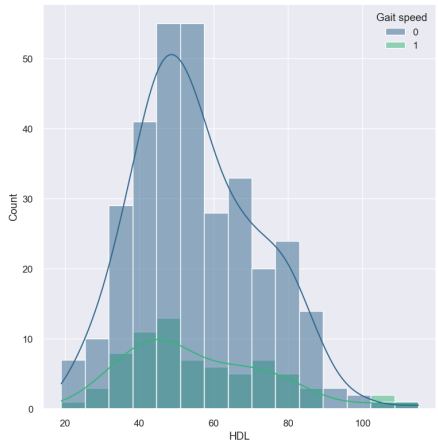

Gait speed
0
1
Count
IL-10

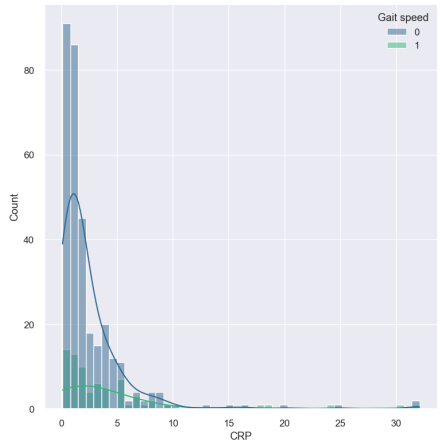

Gait speed
0
1
Count
Beta-cryptoxanthin

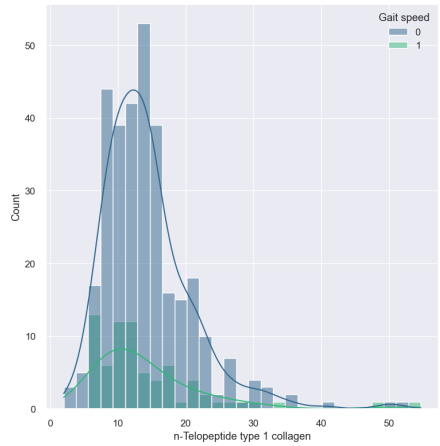

Gait speed
0
1
Count
IGF-1

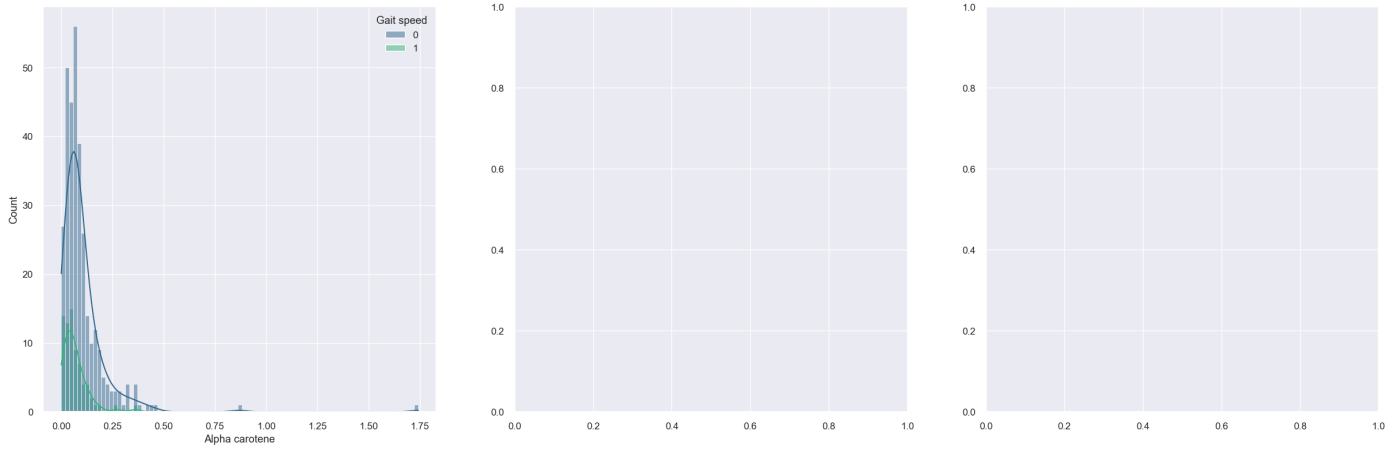

Gait speed
0
1
Count
Lycopene


Gait speed
0
1
Count
E-selectin


Gait speed
0
1
Count
HDL

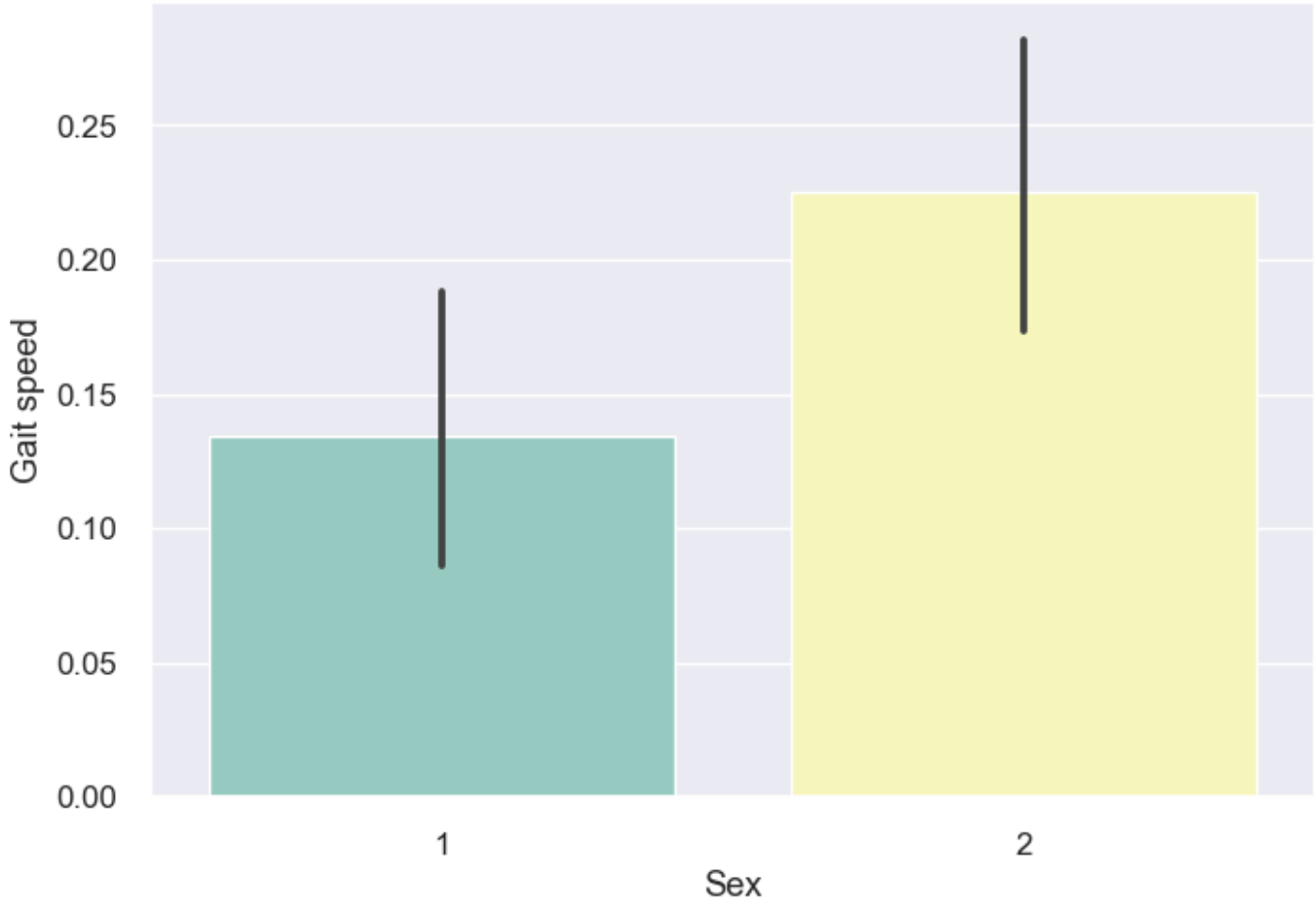

Gait speed
0
1
Count
CRP


Gait speed
0
1
Count
n-Telopeptide type 1 collagen



In [128…
```
sns.barplot(x='Sex', y= 'Gait speed', data=train_df, palette='Set3')
```

Out[1286]:
```
<Axes: xlabel='Sex', ylabel='Gait speed'>
```



Let's look at the relationship between features and the target more quantitatively...

Correlation is not a good measure while measuring relations between features and target for classification problems.

(It is good at measuring linear relations between continuous variables, but our target is discrete)

Let's check mutual information, which measures this kind of relationship well.

In [128…
```
y = train_df['Gait speed']
```

In [128…
```
# determine the mutual information for features
```

```
#we need to fill missing values to get results from mutual_info_classif function
mutual_df = train_df[feature_list ]

mutual_info = mutual_info_classif(mutual_df.fillna(mutual_df.mean()), y, random_state=ra

mutual_info = pd.Series(mutual_info)
mutual_info.index = mutual_df.columns
mutual_info = pd.DataFrame(mutual_info.sort_values(ascending=False), columns = ["Numeric
mutual_info.style.background_gradient("cool")
```

Out[1288]:

	Numerical_Feature_MI
n-Telopeptide type 1 collagen	0.039396
Sex	0.034662
Age	0.031769
IL-6	0.026613
Alpha carotene	0.020803
IL-8	0.017848
E-selectin	0.012973
DHEA	0.012207
IL-10	0.007493
LDL	0.003699
triglycerides	0.003275
CRP	0.001197
HDL	0.000240
Bone specific alkaline phosphatase	0.000217
Gamma tocopherol	0.000210
Total cholesterol	0.000000
TNF alpha	0.000000
Total cholesterol/HDL	0.000000
Retinol	0.000000
Trans beta carotene	0.000000
Zeaxanthin	0.000000
Lycopene	0.000000
Lutein	0.000000
13-cis-beta-carotene	0.000000
Intercellular adhesion molecule 1	0.000000
Insulin	0.000000
HgA1c	0.000000
Glucose	0.000000
Fibrinogen	0.000000
DHEA-S	0.000000
Creatinine	0.000000

Beta-cryptoxanthin	0.000000
Amino terminal propeptide type 1 procollagen	0.000000
Alpha tocoperol	0.000000
IGF-1	0.000000

In [128…
```
most_info_cols = mutual_info[mutual_info['Numerical_Feature_MI'] > 0.012].index.values.t
```

Let's make one more visualization which shows a pair plot that includes only features with highest mutual information score.

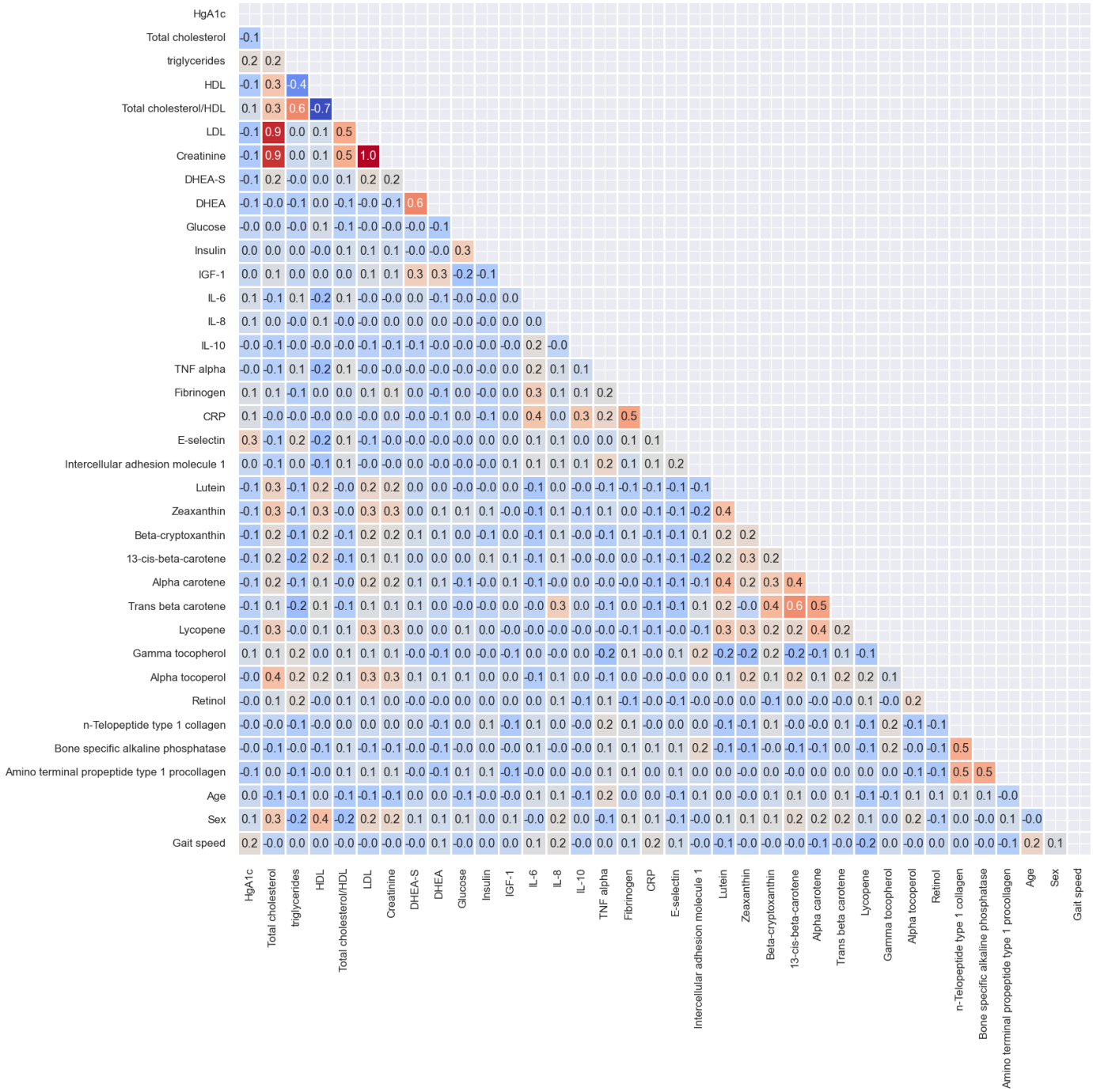
In [129…
```
#pair-plot for most important features
sns.pairplot(train_df[most_info_cols + ["Gait speed"]], hue="Gait speed")
```

Out[1290]:
```
<seaborn.axisgrid.PairGrid at 0x2003d15ed10>
```

Lastly I will report correlations between features. Note that correlation is not a good measure for target.I will only report it for reference.

In [129…
```
# Display correlations between numerical features
```

```
sns.set(font_scale=1.1)
correlation_train = train_df.corr()
mask = np.triu(correlation_train.corr())
plt.figure(figsize=(17, 17))
sns.heatmap(correlation_train,
            annot=True,
            fmt='.1f',
            cmap='coolwarm',
            square=True,
            mask=mask,
            linewidths=1,
            cbar=False);
```



Preprocessing

I will use tree-based boosting methods(LightGBM and XGBoost). This medhods do not require extensive

preprocessing unlike linear and distance based algorithms or neural networks.

We will need only a simple imputer object for missing value imputation. Another thing we will need is SMOTE to straggle unbalance class distribution.

In [129…
```
#for imputing missing values
preprocessor = SimpleImputer(strategy='mean')

#SMOTE for oversampling of minority class(class 1)
sm = SMOTE(random_state=random_state)
```

Splitting data to train and test sets (Perform adversarial validation)

Before moving to the modeling phase, I will create train and test splits.

Getting stable models on small datasets is often challenging. Ensuring that the training and test data distributions are similar is one of the key issues to obtain stable models.

To ensure that the distributions of the training and test data are similar, we will employ adversarial validation.

Adversarial validation is a simple technique that assesses how similar the distributions of the training and test datasets are. It involves creating a combined dataset and training a classifier to distinguish between the training and test examples based on their features. If the classifier struggles to differentiate between the two datasets, it indicates that their distributions are similar, which is crucial for building reliable models.

Typically, an AUC score around 0.5 indicates that the model struggles to differentiate between the train and test samples. This suggests that their distributions are similar, making it challenging for the model to distinguish between them.

To verify this, I will employ a Gaussian Naive Bayes (GaussianNB) model.

In [129…
```
#seperate target
y = train_df['Gait speed']
train_df = train_df.drop(['Gait speed'], axis=1)
```

In [129…
```
#create train-test split.
X_train, X_test, y_train, y_test = train_test_split(train_df, y,
                                                    test_size=0.2,
                                                    random_state=random_state,
                                                    shuffle=True,
                                                    stratify=y)


print(X_train.shape, X_test.shape)
```

```
(319, 35) (80, 35)
```

In [129…
```
adver_clf = Pipeline([

    ('preprocessor', preprocessor),
    ('scaler', StandardScaler()),
    ("GaussianNB", GaussianNB())
])
```

In [129…

```python
def adversarial_validation(X_train, X_test):
    # Label train samples as 0 and test samples as 1
    y_train = np.zeros(X_train.shape[0])
    y_test = np.ones(X_test.shape[0])

    # Concatenate train and test data
    X_adv = np.concatenate([X_train, X_test], axis=0)
    y_adv = np.concatenate([y_train, y_test], axis=0)

    # Initialize cross-validation splitter
    cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state)

    auc_scores = []
    for train_index, val_index in cv.split(X_adv, y_adv):
        X_adv_train, X_adv_val = X_adv[train_index], X_adv[val_index]
        y_adv_train, y_adv_val = y_adv[train_index], y_adv[val_index]

        # Train a classifier
        clf =  adver_clf

        clf.fit(X_adv_train, y_adv_train)

        # Predict probabilities for the validation set
        y_adv_pred = clf.predict_proba(X_adv_val)[:, 1]

        # Calculate AUC-ROC score
        auc_score = roc_auc_score(y_adv_val, y_adv_pred)
        auc_scores.append(auc_score)

    avg_auc_score = np.mean(auc_scores)

    return avg_auc_score

avg_auc_score = adversarial_validation(X_train, X_test)
print("Average AUC-ROC Score for Adversarial Validation with Cross-Validation:", avg_auc
```

```
Average AUC-ROC Score for Adversarial Validation with Cross-Validation: 0.49309740823412
695
```

With an AUC score of 0.49, it suggests that our model struggles to effectively differentiate between the train and test samples, indicating similarity in the distributions of the training and test datasets. Based on this observation, we can proceed to the modeling phase.

Modeling and hyperparameter tuning

%80 of the data is used for training. %20 of the data is reserved as a test set for final model evaluation.

Target variable distribution is quite imbalanced. At the modeling stage, the Synthetic Minority Over-sampling Technique (SMOTE) and class_weight adjustment are used to deal with this imbalance distribution. SMOTE performed better. This notebook includes only results with SMOTE.

Boosting methods are generally performs better on tabular data. We tried XGBoost and LightGBM. XGBoost was the better performing one. This notebook only includes results of XGBoost algorithm.

Although accuracy is a default metric for most classification problems, it's generally misleading for imbalanced data sets. That is why the F1 score is used as an optimization metric.

F1 score is defined as the harmonic mean of precision and recall.

For model training a hyperparameter optimization with cross validation is performed.

Given the limited size of the dataset, a rigorous cross-validation approach was adopted to ensure reliable model evaluation. The selected method, RepeatedStratifiedKFold, was configured with parameters (n_splits=5, n_repeats=10, random_state=random_state). This approach divides the training data into 5 folds, ensuring that each class is represented proportionally within each fold, and repeats this process 10 times to enhance robustness.

During each iteration, the model is trained on 4 folds while the performance is evaluated on the remaining fold, known as the hold-out data. This process is repeated, with the hold-out fold changing in each iteration to ensure comprehensive evaluation. Furthermore, to address variability and ensure thorough evaluation, this entire process is repeated 10 times for each combination of hyperparameters before reporting the final evaluation metric. The final performance metric is calculated as the average score across all iterations.

For hyperparameter optimization OPTUNA package is used. OPTUNA is an open-source hyperparameter optimization framework based on Bayesian optimization, aimed at automating the tuning process of machine learning models.

Modelling phase has 3 consecutive steps:

- Step 1: Perform hyperparameter optimization with cross validation with using all features
- Step 2: Perform a feature selection based on models feature importance score with trained model in step 1
- Step 3: Repeat hyperparameter optimization with cross validation only with selected features

Step 1: Train a base model with all features (XGBoost + SMOTE)

The following code block performs hyperparameter optimization with cross-validation.

It utilizes the Optuna package for hyperparameter optimization.

Cross-validation strategy: RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state)

Note: The optimization process takes some time, so I have commented it out.

We can uncomment and run it.

In [129…

```
#xgboost



#def objective(trial):

#    n_estimators = trial.suggest_int("n_estimators", 10,500)
    # L1 regularization weight.
#    alpha = trial.suggest_float("alpha", 1e-8, 1.0, log=True)
    # sampling ratio for training data.
```

```
#     subsample = trial.suggest_float("subsample", 0.2, 1.0)
    # sampling according to each tree.
#     colsample_bytree = trial.suggest_float("colsample_bytree", 0.2, 0.8)
    # maximum depth of the tree, signifies complexity of the tree.
#     max_depth = trial.suggest_int("max_depth", 3, 12)
    # minimum child weight, larger the term more conservative the tree.
#     min_child_weight = trial.suggest_int("min_child_weight", 1, 10)
    # learning rate
#     learning_rate =  trial.suggest_float("learning_rate", 1e-4, 0.1, log=True)
    # defines how selective algorithm is.
#     gamma = trial.suggest_float("gamma", 1e-8, 1.0, log=True)




#     xgb_clf = XGBClassifier(n_estimators = n_estimators,alpha=alpha,subsample=subsample
#                             max_depth=max_depth,min_child_weight =min_child_weight,learn
#                             random_state =random_state)


     # -- Make a pipeline
#     xgb_pipeline = imbpipeline([

#     ('preprocessor', preprocessor),
#     ('smote', sm),
#     ("xgb_clf", xgb_clf)
#     ])

#     ss = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state) #
#     score = cross_val_score(xgb_pipeline, X_train, y_train, scoring= make_scorer(f1_sco
#     score = score.mean()
#     return score


#sampler = TPESampler(seed=random_state) # create a seed for the sampler for reproducibi
#study = optuna.create_study(direction="maximize", sampler=sampler)
#study.optimize(objective, n_trials=300)
```

The above code suggest best hyperparameters as following with a cross validation f1 score 0.399.

{'n_estimators': 376, 'alpha': 0.000402612165325408, 'subsample': 0.26337477699169953,'colsample_bytree': 0.6230940284871702, 'max_depth': 12, 'min_child_weight': 8, 'learning_rate': 0.005497165288965543, 'gamma': 0.0009854270359337684}

I will repeat cross validation with the found hyperparameters for demostration of findings.

In [129…
```
#Parameters found in tuning process by Optuna
xgb_optuna_params1 = {'n_estimators': 376, 'alpha': 0.000402612165325408, 'subsample': 0
                      'colsample_bytree': 0.6230940284871702, 'max_depth': 12, 'min_chil
                      'learning_rate': 0.005497165288965543, 'gamma': 0.0009854270359337

# Model pipeline with found hyperparameters
xgb_tunned1 = XGBClassifier(**xgb_optuna_params1, random_state=random_state)

pipe_xgb1 =imbpipeline([

    ('preprocessor', preprocessor),
     ('smote', sm),
    ("xgb_tunned", xgb_tunned1)
])
```

In [129…
```
# Cross-validate and report CV and test results
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state)
```

```
cv_scores = cross_val_score(pipe_xgb1, X_train, y_train, cv=cv, scoring='f1')  #cross-va
print("Mean CV Score: {:.3f}".format(np.mean(cv_scores)))

#Test score
pipe_xgb1 = pipe_xgb1.fit(X_train, y_train)
preds_test = pipe_xgb1.predict(X_test)
print("Test Score: {:.3f}".format(f1_score(y_test, preds_test)))
```

```
Mean CV Score: 0.399
Test Score: 0.462
```

We have a Mean F1 CV Score of 0.399 and a Test Score of 0.462 on testing data. Let's also report other classification metrics for evluation.

In [130…

```
# Accuracy Cross-validation score
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state)

cv_scores = cross_val_score(pipe_xgb1, X_train, y_train, cv=cv, scoring='accuracy')  # c
print("Mean CV Score: {:.3f}".format(np.mean(cv_scores)))

#Test score
pipe_xgb1 = pipe_xgb1.fit(X_train, y_train)
preds_test = pipe_xgb1.predict(X_test)
print("Test Score: {:.3f}".format(accuracy_score(y_test, preds_test)))
```

```
Mean CV Score: 0.709
Test Score: 0.738
```

In [130…

```
# AUC Cross-validation score
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state)

cv_scores = cross_val_score(pipe_xgb1, X_train, y_train, cv=cv, scoring='roc_auc')  #cro
print("Mean CV Score: {:.3f}".format(np.mean(cv_scores)))

#Test score
pipe_xgb1 = pipe_xgb1.fit(X_train, y_train)
preds_test = pipe_xgb1.predict_proba(X_test)[:, 1]
print("Test Score: {:.3f}".format(roc_auc_score(y_test, preds_test)))
```

```
Mean CV Score: 0.707
Test Score: 0.735
```

In [130…

```
# Recall Cross-validation score
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state)

cv_scores = cross_val_score(pipe_xgb1, X_train, y_train, cv=cv, scoring='recall')  # cro
print("Mean CV Score: {:.3f}".format(np.mean(cv_scores)))

#Test score
pipe_xgb1 = pipe_xgb1.fit(X_train, y_train)
preds_test = pipe_xgb1.predict(X_test)
print("Test Score: {:.3f}".format(recall_score(y_test, preds_test)))
```

```
Mean CV Score: 0.532
Test Score: 0.600
```

In [130…

```
# Specificity Cross-validation score (please notice pos_label=0 on recall score)
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state)

specificity = make_scorer(recall_score, pos_label=0)

cv_scores = cross_val_score(pipe_xgb1, X_train, y_train, cv=cv, scoring=specificity)  #
print("Mean CV Score: {:.3f}".format(np.mean(cv_scores)))

#Test score
```

```
pipe_xgb1 = pipe_xgb1.fit(X_train, y_train)
preds_test = pipe_xgb1.predict(X_test)
print("Test Score: {:.3f}".format(recall_score(y_test, preds_test, pos_label=0)))
```

```
Mean CV Score: 0.749
Test Score: 0.769
```

In [130…

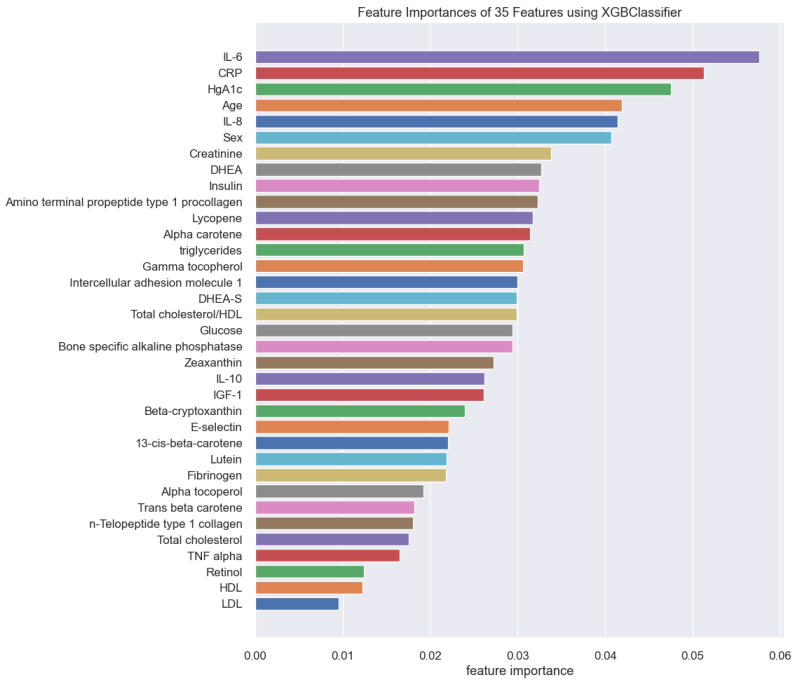
```
# Precision cross-validation scores
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state)

cv_scores = cross_val_score(pipe_xgb1, X_train, y_train, cv=cv, scoring='precision')  #
print("Mean CV Score: {:.3f}".format(np.mean(cv_scores)))

#Test score
pipe_xgb1 = pipe_xgb1.fit(X_train, y_train)
preds_test = pipe_xgb1.predict(X_test)
print("Test Score: {:.3f}".format(precision_score(y_test, preds_test)))
```

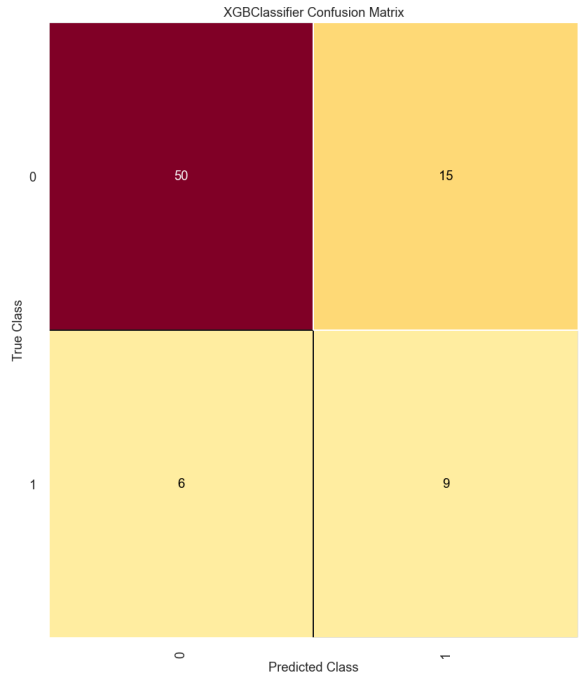
```
Mean CV Score: 0.326
Test Score: 0.375
```

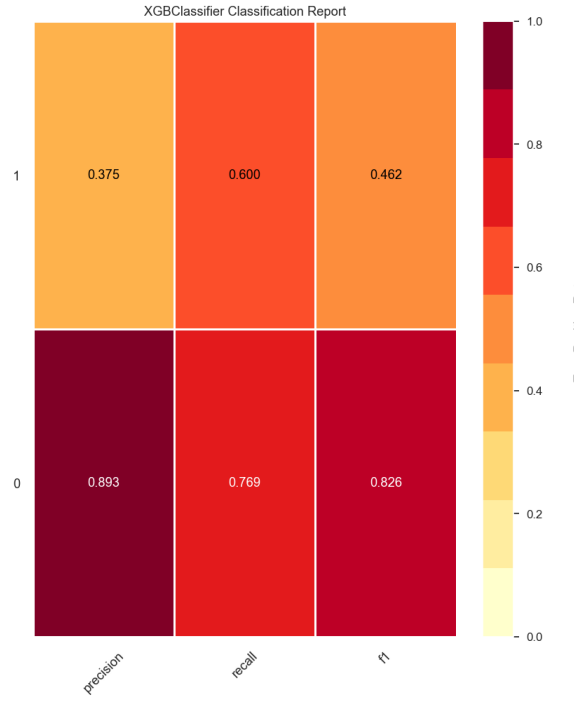
Let's also visualize base model performance summary on test set and features importances from the model.

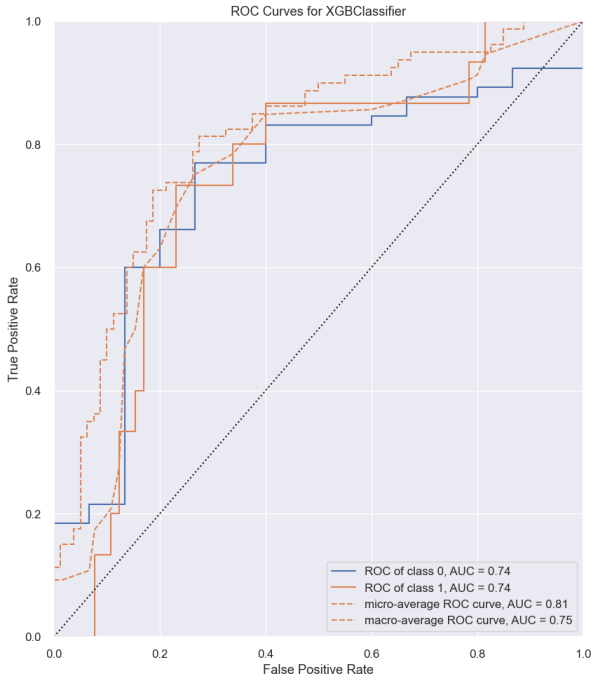
In [130…

```
#yellowbrick is a good library for model evaluation visualizations
from yellowbrick.features import FeatureImportances
from yellowbrick.classifier import ConfusionMatrix, ClassificationReport, ROCAUC



fig, axes = plt.subplots(2, 2, figsize=(20, 20))

model = pipe_xgb1

pipe_xgb1.named_steps['xgb_tunned'].importance_type = 'gain'

visualgrid = [
    FeatureImportances(pipe_xgb1.named_steps['xgb_tunned'], absolute = True, relative=Fa
    ConfusionMatrix(model, ax=axes[0][1]),
    ClassificationReport(model, ax=axes[1][0]),
    ROCAUC(model, ax=axes[1][1]),
]

for viz in visualgrid:
    viz.fit(X_train, y_train)
    viz.score(X_test, y_test)
    viz.finalize()

plt.show()
```









Step 2: Feature Selection Based on Model Importance Scores

The above feature importance plot shows importance levels of features based on gain. Gain is defined as the improvement in accuracy brought by a feature to the branches it is on[https://xgboost.readthedocs.io/en/stable/R-package/discoverYourData.html].

We can see that some of the features have higher contribution on decision process while some of them less. We have a small data set with relatively high number of columns (399 samples with 35 columns in train set). This problem known as Curse of Dimensionality. It is very likely that less informative features are resulting overfitting. To asses this fenomena we will apply feature selection. There are many feature selection techniques. SelectFromModel() method from scikit-learn's feature selection module is used for feature selection. This method selects features based on model's importance scores (above importance plot). During feature selection proses, starting from most important feature in every iteration the next important feature is

added and cross validation is repeated with the based model hyperparameters. The highest cross-validation score is optained with six highest important features.

The following code block demostrates this work.

In [130…

```
%%time


for i in range(1,36):


    # Create a pipeline for feature selection
    pipeline = imbpipeline([
        ('preprocessor', preprocessor),
         ('smote', sm),
        ('selector', SelectFromModel(xgb_tunned1, threshold=-np.inf, max_features= i)),
        ('classification',xgb_tunned1)
    ])


    # Cross-validation scores with different number of features
    cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state)
    cv_scores = cross_val_score(pipeline, X_train, y_train, cv=cv, scoring='f1')  #cross
    print("Number of selected features:", i)
    print("Mean CV Score: {:.4f}".format(np.mean(cv_scores)))
```

```
Number of selected features: 1
Mean CV Score: 0.3139
Number of selected features: 2
Mean CV Score: 0.3100
Number of selected features: 3
Mean CV Score: 0.3493
Number of selected features: 4
Mean CV Score: 0.3838
Number of selected features: 5
Mean CV Score: 0.3959
Number of selected features: 6
Mean CV Score: 0.4051
Number of selected features: 7
Mean CV Score: 0.4050
Number of selected features: 8
Mean CV Score: 0.3997
Number of selected features: 9
Mean CV Score: 0.3990
Number of selected features: 10
Mean CV Score: 0.3986
Number of selected features: 11
Mean CV Score: 0.3989
Number of selected features: 12
Mean CV Score: 0.3905
Number of selected features: 13
Mean CV Score: 0.3913
Number of selected features: 14
Mean CV Score: 0.3927
Number of selected features: 15
Mean CV Score: 0.3866
Number of selected features: 16
Mean CV Score: 0.3948
Number of selected features: 17
Mean CV Score: 0.3932
Number of selected features: 18
Mean CV Score: 0.3901
Number of selected features: 19
Mean CV Score: 0.3884
```

```
Number of selected features: 20
Mean CV Score: 0.3912
Number of selected features: 21
Mean CV Score: 0.3900
Number of selected features: 22
Mean CV Score: 0.3909
Number of selected features: 23
Mean CV Score: 0.3886
Number of selected features: 24
Mean CV Score: 0.3894
Number of selected features: 25
Mean CV Score: 0.3915
Number of selected features: 26
Mean CV Score: 0.3934
Number of selected features: 27
Mean CV Score: 0.3862
Number of selected features: 28
Mean CV Score: 0.3877
Number of selected features: 29
Mean CV Score: 0.3884
Number of selected features: 30
Mean CV Score: 0.3945
Number of selected features: 31
Mean CV Score: 0.3882
Number of selected features: 32
Mean CV Score: 0.3961
Number of selected features: 33
Mean CV Score: 0.3892
Number of selected features: 34
Mean CV Score: 0.3880
Number of selected features: 35
Mean CV Score: 0.3989
CPU times: total: 2h 21min 39s
Wall time: 11min 34s
```

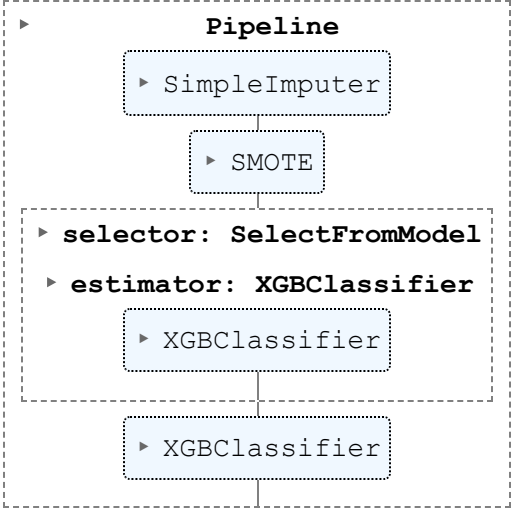
Selecting 6 features yiels best CV score.

In [130…

```
%%time


# Cv method

cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state)

# Create a pipeline for feature selection
pipeline = imbpipeline([
    ('preprocessor', preprocessor),
     ('smote', sm),
    ('selector', SelectFromModel(xgb_tunned1, threshold=-np.inf, max_features= 6)),
    ('classification',xgb_tunned1)
])

# Fit the pipeline on the training data to select features
pipeline.fit(X_train, y_train)
```

```
CPU times: total: 5.45 s
Wall time: 447 ms
```

Out[1307]:



In [130…
```
#Prints out selected and eleminated features
selected_features = np.array( preprocessor.get_feature_names_out())[pipeline.named_steps

print("Number of selected features:", len(selected_features))

print("_____________________________________")

print("Selected features:", selected_features)


print("_____________________________________")

print("Dropped features:", list(set(X_train.columns) - set(selected_features)))
```

```
Number of selected features: 6

_____________________________________
Selected features: ['HgA1c' 'IL-6' 'IL-8' 'CRP' 'Age' 'Sex']

_____________________________________
Dropped features: ['Insulin', 'Total cholesterol/HDL', 'DHEA-S', 'Trans beta carotene',
'triglycerides', 'Zeaxanthin', 'Fibrinogen', 'TNF alpha', 'Alpha tocoperol', 'Total chol
esterol', 'Creatinine', 'Amino terminal propeptide type 1 procollagen', 'DHEA', 'LDL',
'Lutein', 'Glucose', 'Intercellular adhesion molecule 1', 'Bone specific alkaline phosph
atase', 'Retinol', '13-cis-beta-carotene', 'Gamma tocopherol', 'IL-10', 'Beta-cryptoxant
hin', 'IGF-1', 'Lycopene', 'E-selectin', 'HDL', 'n-Telopeptide type 1 collagen', 'Alpha
carotene']
```

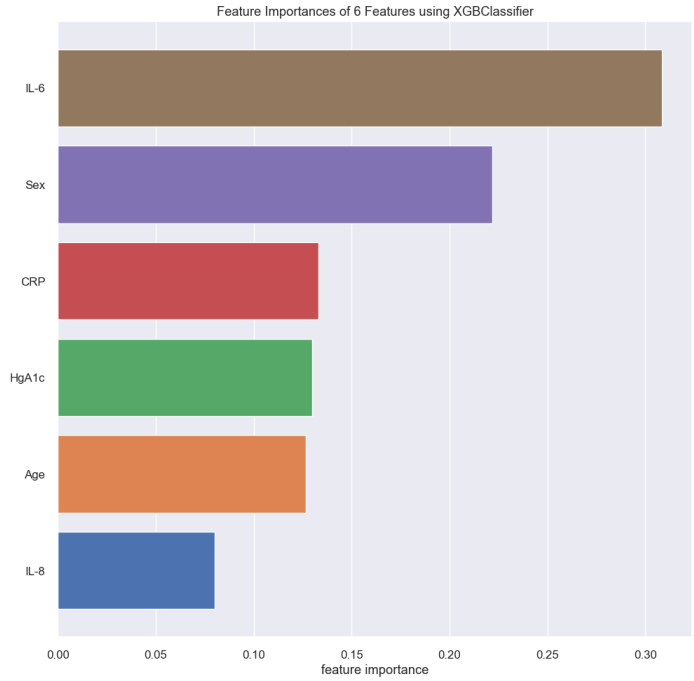
In [130…
```
# Reduce feature set to selected ones
X_train_selected = X_train[selected_features]
X_test_selected = X_test[selected_features]
```

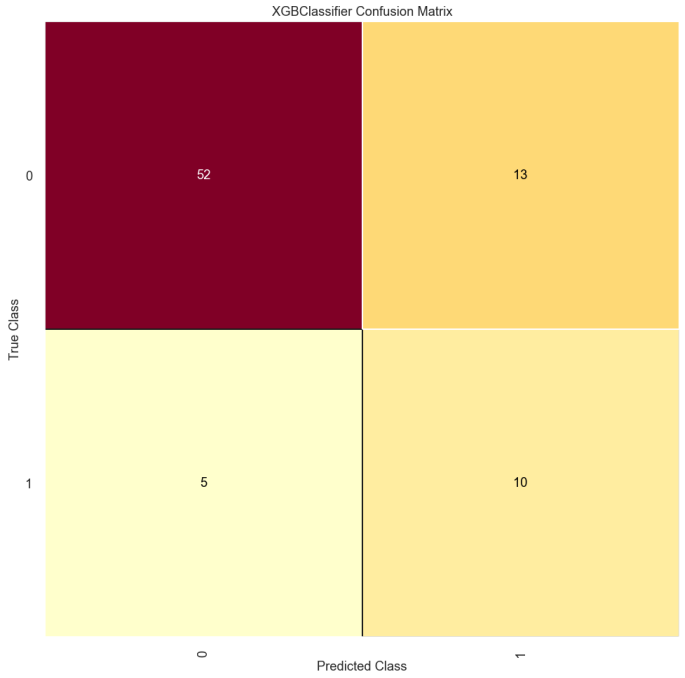
In [131…
```
# Report Cross-validation score with selected features
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state)

cv_scores = cross_val_score(pipe_xgb1, X_train_selected, y_train, cv=cv, scoring='f1')
print("Mean CV Score: {:.3f}".format(np.mean(cv_scores)))

#Test score
pipe_xgb1 = pipe_xgb1.fit(X_train_selected, y_train)
preds_test = pipe_xgb1.predict(X_test_selected)
print("Test Score: {:.3f}".format(f1_score(y_test, preds_test)))
```

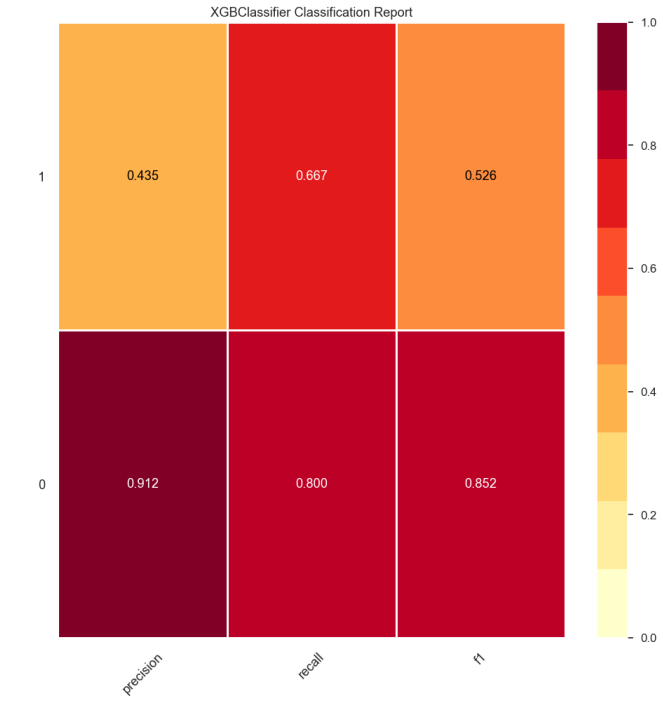
```
Mean CV Score: 0.427
Test Score: 0.526
```

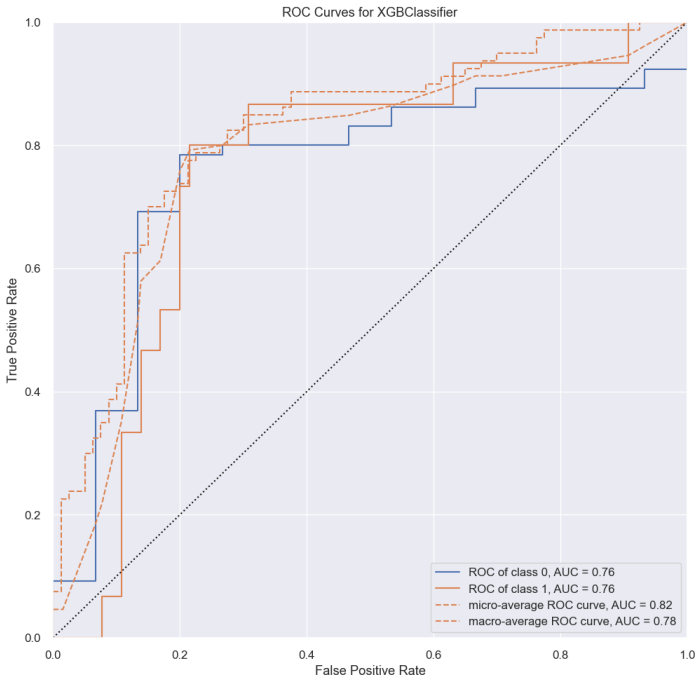
In [131…
```
from yellowbrick.features import FeatureImportances
from yellowbrick.classifier import ConfusionMatrix, ClassificationReport, ROCAUC
```

```
fig, axes = plt.subplots(2, 2, figsize=(20, 20))

model = pipe_xgb1

pipe_xgb1.named_steps['xgb_tunned'].importance_type = 'total_gain'

visualgrid = [
    FeatureImportances(pipe_xgb1.named_steps['xgb_tunned'], absolute = True, relative=Fa
    ConfusionMatrix(model, ax=axes[0][1]),
    ClassificationReport(model, ax=axes[1][0]),
    ROCAUC(model, ax=axes[1][1]),
]

for viz in visualgrid:
    viz.fit(X_train_selected, y_train)
    viz.score(X_test_selected, y_test)
    viz.finalize()

plt.show()
```









- With the selected features we have a better CV score of 0.427. Before feature selection it was 0.399.
- We get a significant performance increase just eleminating unnnecesary features.
- Remember that this is still our base model It's hyperparameters are selected based on all features

Let's repeat hyperparameter optimization with selected features.

Step 3: Hyperparameter tunning with selected features

Following code block performs hyperparameter tunning only with selected features.

In [131…

```
#xgboost



#def objective(trial):

#    n_estimators = trial.suggest_int("n_estimators", 10,500)
    # L1 regularization weight.
#    alpha = trial.suggest_float("alpha", 1e-8, 1.0, log=True)
    # sampling ratio for training data.
#    subsample = trial.suggest_float("subsample", 0.2, 1.0)
    # sampling according to each tree.
#    colsample_bytree = trial.suggest_float("colsample_bytree", 0.2, 0.8)
    # maximum depth of the tree, signifies complexity of the tree.
#    max_depth = trial.suggest_int("max_depth", 3, 12)
    # minimum child weight, larger the term more conservative the tree.
#    min_child_weight = trial.suggest_int("min_child_weight", 1, 10)
    # learning rate
#    learning_rate =  trial.suggest_float("learning_rate", 1e-4, 0.1, log=True)
    # defines how selective algorithm is.
#    gamma = trial.suggest_float("gamma", 1e-8, 1.0, log=True)




#    xgb_clf = XGBClassifier(n_estimators = n_estimators,alpha=alpha,subsample=subsample
#                            max_depth=max_depth,min_child_weight =min_child_weight,learn
#                            random_state =random_state)


     # -- Make a pipeline
#    xgb_pipeline = imbpipeline([

#    ('preprocessor', preprocessor),
#    ('smote', sm),
#    ("xgb_clf", xgb_clf)
#    ])

#    ss = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state) #
#    score = cross_val_score(xgb_pipeline, X_train_selected, y_train, scoring= make_scor
#    score = score.mean()
#    return score


#sampler = TPESampler(seed=random_state) # create a seed for the sampler for reproducibi
#study = optuna.create_study(direction="maximize", sampler=sampler)
#study.optimize(objective, n_trials=300)
```

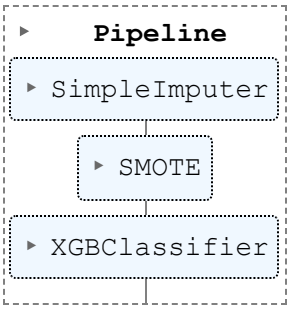
The above code suggest best hyperparameters as following with a cross validation f1 score 0.461.

{'n_estimators': 422, 'alpha': 0.00025316727636985045, 'subsample': 0.2625265855043195,'colsample_bytree': 0.22066847706109496, 'max_depth': 6, 'min_child_weight': 9, 'learning_rate': 0.0003133276734497596, 'gamma': 1.468816194758717e-06}

I will repeat cross validation with the found hyperparameters for demostration of findings.

In [131…
```
xgb_optuna_params1 =  {'n_estimators': 422, 'alpha': 0.00025316727636985045, 'subsample'
                       'colsample_bytree': 0.22066847706109496, 'max_depth': 6, 'min_chi
                       'learning_rate': 0.0003133276734497596, 'gamma': 1.46881619475871

# Model pipeline with found hyperparameters
xgb_tunned1 = XGBClassifier(**xgb_optuna_params1, random_state=random_state)

pipe_xgb1 =imbpipeline([

    ('preprocessor', preprocessor),
     ('smote', sm),
    ("xgb_tunned", xgb_tunned1)
])

pipe_xgb1
```

Out[1313]:



In [131…
```
# F1 Cross-validation score
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state)

cv_scores = cross_val_score(pipe_xgb1, X_train_selected, y_train, cv=cv, scoring='f1')
print("Mean CV Score: {:.3f}".format(np.mean(cv_scores)))

#Test score
pipe_xgb1 = pipe_xgb1.fit(X_train_selected, y_train)
preds_test = pipe_xgb1.predict(X_test_selected)
print("Test Score: {:.3f}".format(f1_score(y_test, preds_test)))
```

```
Mean CV Score: 0.461
Test Score: 0.537
```

The CV score for base model with selected features was 0.427.

CV score is increased to 0.461 after hyperparameter tunning with selected features. This will be our final model. Let's also report CV scores of other metrics for this final model.

In [131…
```
# Accuracy Cross-validation score
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state)

cv_scores = cross_val_score(pipe_xgb1, X_train_selected, y_train, cv=cv, scoring='accura
print("Mean CV Score: {:.3f}".format(np.mean(cv_scores)))

#Test score
pipe_xgb1 = pipe_xgb1.fit(X_train_selected, y_train)
```

```
preds_test = pipe_xgb1.predict(X_test_selected)
print("Test Score: {:.3f}".format(accuracy_score(y_test, preds_test)))
```

```
Mean CV Score: 0.708
Test Score: 0.762
```

In [131…

```
# AUC Cross-validation score
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state)

cv_scores = cross_val_score(pipe_xgb1, X_train_selected, y_train, cv=cv, scoring='roc_au
print("Mean CV Score: {:.3f}".format(np.mean(cv_scores)))

#Test score
pipe_xgb1 = pipe_xgb1.fit(X_train_selected, y_train)
preds_test = pipe_xgb1.predict_proba(X_test_selected)[:, 1]
print("Test Score: {:.3f}".format(roc_auc_score(y_test, preds_test)))
```

```
Mean CV Score: 0.739
Test Score: 0.754
```

In [131…

```
# Recall Cross-validation score
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state)

cv_scores = cross_val_score(pipe_xgb1, X_train_selected, y_train, cv=cv, scoring='recall
print("Mean CV Score: {:.3f}".format(np.mean(cv_scores)))

#Test score
pipe_xgb1 = pipe_xgb1.fit(X_train_selected, y_train)
preds_test = pipe_xgb1.predict(X_test_selected)
print("Test Score: {:.3f}".format(recall_score(y_test, preds_test)))
```

```
Mean CV Score: 0.683
Test Score: 0.733
```

In [131…

```
# Specificity Cross-validation score (please notice pos_label=0 on recall score)
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state)

cv_scores = cross_val_score(pipe_xgb1, X_train_selected, y_train, cv=cv, scoring=specifi
print("Mean CV Score: {:.3f}".format(np.mean(cv_scores)))

#Test score
pipe_xgb1 = pipe_xgb1.fit(X_train_selected, y_train)
preds_test = pipe_xgb1.predict(X_test_selected)
print("Test Score: {:.3f}".format(recall_score(y_test, preds_test, pos_label=0)))
```

```
Mean CV Score: 0.714
Test Score: 0.769
```

In [131…

```
# Cross-validation scores
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=random_state)

cv_scores = cross_val_score(pipe_xgb1, X_train_selected, y_train, cv=cv, scoring='precis
print("Mean CV Score: {:.3f}".format(np.mean(cv_scores)))

#Test score
pipe_xgb1 = pipe_xgb1.fit(X_train_selected, y_train)
preds_test = pipe_xgb1.predict(X_test_selected)
print("Test Score: {:.3f}".format(precision_score(y_test, preds_test)))
```

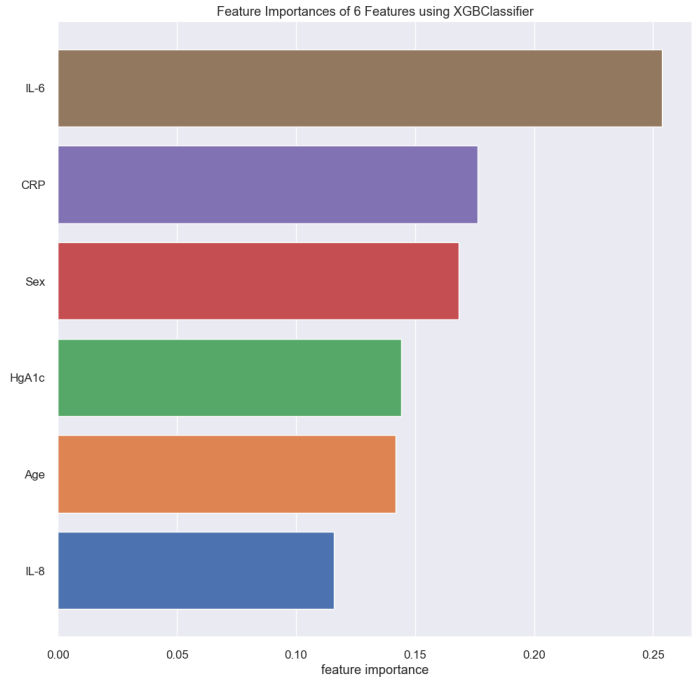
```
Mean CV Score: 0.351
Test Score: 0.423
```

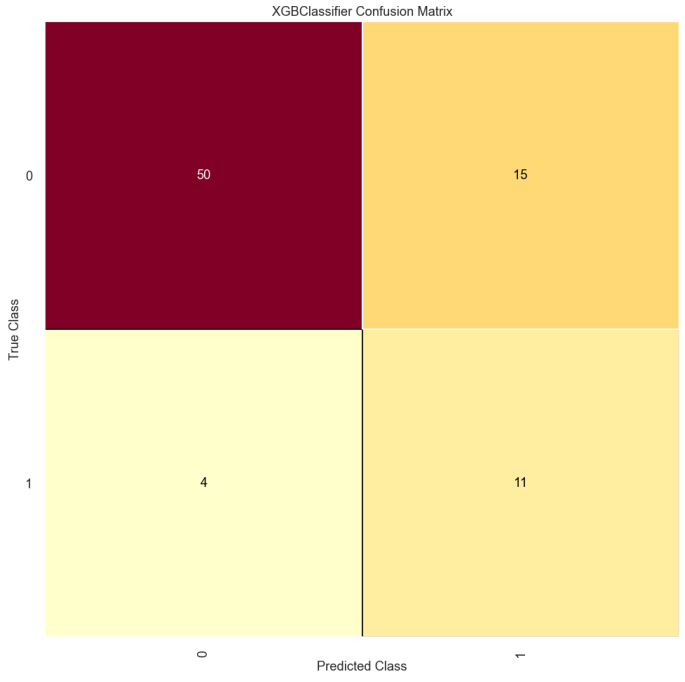
Let's plot model performance summary on test set and feature importances one last time.

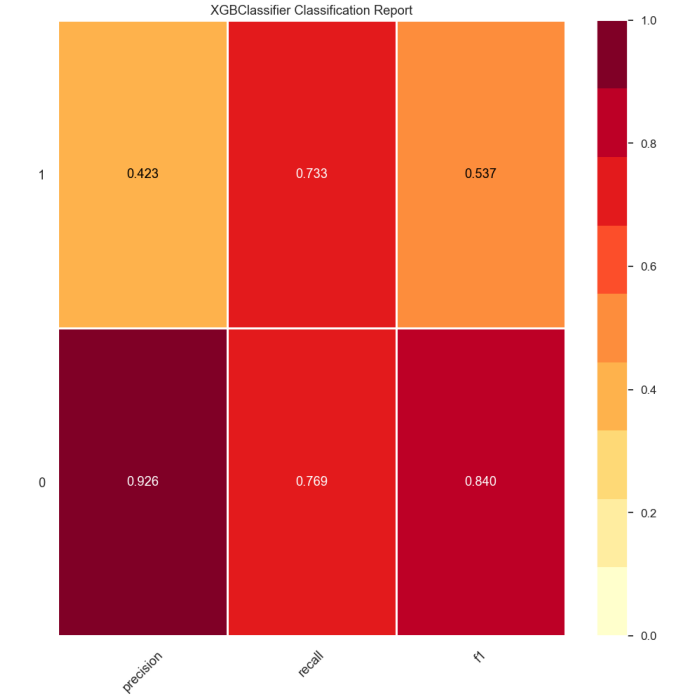
In [132…

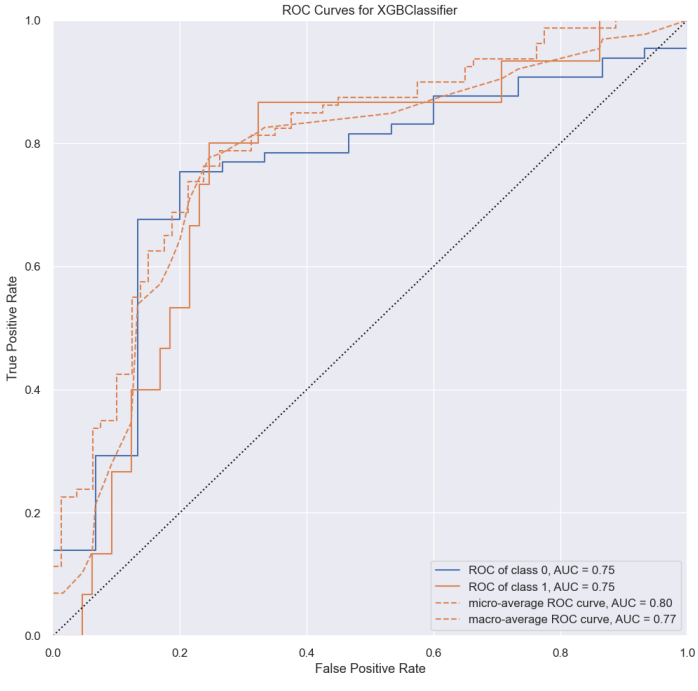
```
from yellowbrick.features import FeatureImportances
from yellowbrick.classifier import ConfusionMatrix, ClassificationReport, ROCAUC
```

```
fig, axes = plt.subplots(2, 2, figsize=(20, 20))

model = pipe_xgb1

pipe_xgb1.named_steps['xgb_tunned'].importance_type = 'total_gain'

visualgrid = [
    FeatureImportances(pipe_xgb1.named_steps['xgb_tunned'], absolute = True, relative=Fa
    ConfusionMatrix(model, ax=axes[0][1]),
    ClassificationReport(model, ax=axes[1][0]),
    ROCAUC(model, ax=axes[1][1]),
]

for viz in visualgrid:
    viz.fit(X_train_selected, y_train)
    viz.score(X_test_selected, y_test)
    viz.finalize()

plt.show()
```









Explainable ML with SHAP

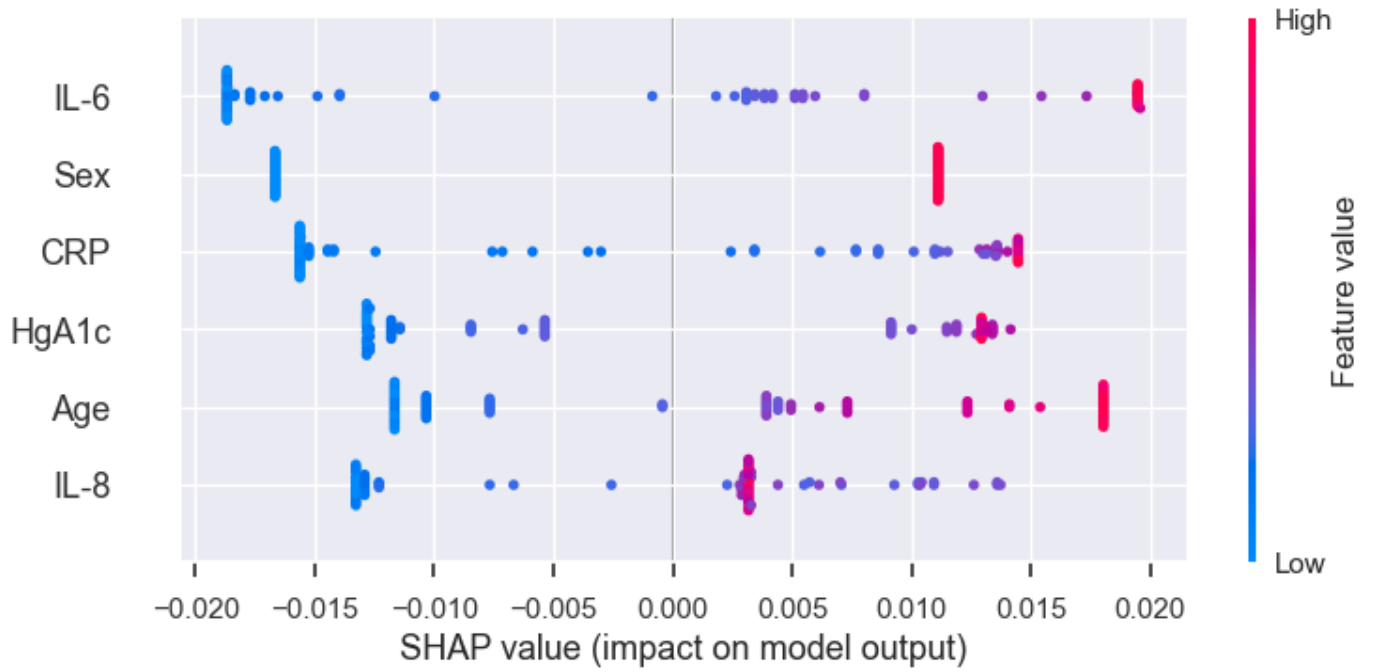
Another usefull tool for explaining machine learning models is Shapley values. Shapley values are a widely used approach from game theory. They help Shapley-based explanations of machine learning models.

For more information you can check:
https://shap.readthedocs.io/en/latest/example_notebooks/overviews/An%20introduction%20to%20explainable%

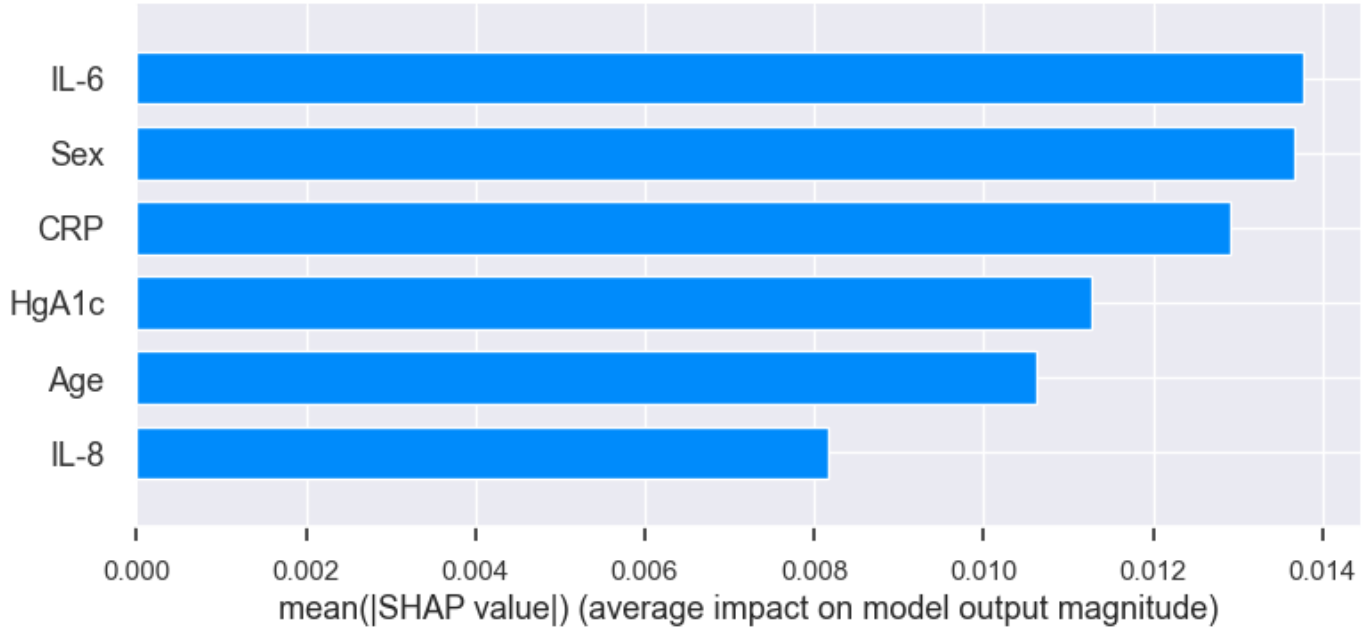
Lastly I want to report SHAP based feature importances.

In [132…
```
#pip install shap
```

In [133…
```
import shap

explainer = shap.TreeExplainer(pipe_xgb1.named_steps['xgb_tunned'], data=X_train_selecte


shap_values = explainer.shap_values(X_test_selected)

# Summary plot of feature importance
shap.summary_plot(shap_values, X_test_selected)
```



In [135…
```
shap.summary_plot(shap_values, X_test_selected, plot_type = 'bar')
```


IL-6
Sex
CRP
HgA1c
Age
IL-8
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
mean(|SHAP value|) (average impact on model output magnitude)