Course:

Machine Learning Lab

Half-term Activities Report

Methodology and Tools for Developing Machine Learning models

Name: Evgenii Vylegzhanin

Date: 29.10.2021

CONTENTS

CHAPTER 1. METHODOLOGY TO DEVELOP MACHINE LEARNING MODELS

According to the book "Hands-On Machine Learning with Scikit-Learn&TensorFlow" by Aurelien Geron, the main steps we have to go through while working on a machine learning project:

1. Look at the big picture
2. Get the data
3. Discover and visualize the data to gain insights
4. Prepare the data for Machine Learning algorithms
5. Select a model and train it
6. Fine-tune your model
7. Present your solution
8. Launch, monitor, and maintain your system

These steps can be divided into 4 groups: the first group contains only step number 1; the second one is needed for data processing and contains steps from number 2 to number 4; the third group is all about machine learning models, it consists of steps number 5 and number 6; in the final step data scientist has to present their solution, fix some problems, launch, monitor, and maintain the developed system.

During the first stage a data scientist besides other activities such as defining objectives, checking current solutions and making assumptions is doing some research about the main topic, going deeper into it for better understanding and finding some insights. For example, it would be very useful for data preparation and feature engineering. From my perspective, it is one of the most important part of the whole process which simplifies the data preparation stage and makes it more effective to estimate the solutions.

The second group is considered to be the most time-consuming and boring part. Data preparation accounts for about 80% of the work of data scientists: building training sets: 3%; cleaning and organizing data: 60%; Collecting data sets: 19%; Mining data for patterns: 9%, Refining algorithms: 4%; Other: 5%.

The third group is about choosing, optimizing, and tuning different machine learning models. It is always better to automate the process as much as possible. During this stage it is also important to measure performance and make some changes in data and used models by using the appropriate metrics, cross-validation (not always a good choice, it depends on the considered project), analyzing errors and so on.

The last group is needed to present your results, launch, monitor, and maintain your system. During previous stages it is recommended to document, visualize, and describe all steps so it would be easier to prepare the presentation of the final solution.

All tools (libraries, functions, metrics) are described in the main parts of the paper.

CHAPTER 2. THE PROBLEM. OBSTRUCTIVE SLEEP APNEA (OSA) CASE STUDY

2.1. Describing the problem

2.1.1. Definition of the OSA

Obstructive Sleep Apnea (OSA) is a disorder in which breathing is repetitively interrupted during sleep due to collapse of the upper airway. It is the most common sleep-disordered breathing.

Important term: AHI (Apnea-Hypopnea Index) – measures the severity of the disease. It represents the number of times the patient stops breathing per hour.

2.1.2. Typical OSA symptoms

- Snores loudly
- Excessive sleepiness
- Morning headaches
- Not rested after sleeping
- Dry mouth

2.1.3. OSA risk factors

- Excess weight
- Over 40 years old
- Gender (men are twice as likely to have it)
- African-Americans are more susceptible to OSA
- Use of alcohol or sedatives
- Family history

2.2. Expected outcome

The expected outcome of this project consists of two parts: prediction of the Apnea-Hypopnea Index (regression models) and prediction of the class of Apnea-Hypopnea Index (classification models).

CHAPTER 3. DATA PREPARATION

3.1. Read raw data

First, we have to import ‘pandas’ library to be able to work with spreadsheets, tables and process them. Pandas – a software library written for the Python programming language for data manipulation and analysis.

I imported it with the following command using the short name for easier access to it:

```
import pandas as pd
```

After that we have to read excel file:

```
data = pd.read_excel("Info_BDApnea_QuironMalaga.xlsx", sheet_name='Hoja1')
```

Using the method ‘head(n)’ we can return top n rows of a DataFrame or Series where n is a user input value. So, we returned 5 first rows of the DataFrame:

```
data.head(5)
```

3.2. Choosing features from the initial dataset

In order to choose features we can glance at the initial excel file and remove useless features such as ‘patient’ and ‘comentaries’ (unique for each row), ‘audios tumbado’, ‘fotos’, ‘audio’ (now we don’t work with multimedia data), ‘ERWORTH’, ‘IAH Supino’, ‘IAH Lateral’, ‘IMC’, ‘Roncandor’ (these five columns have small number of data, the most records are empty), ‘Enfermedades’ (has many unique columns, but could be useful since family history and sedatives are the risk factors), features after ‘Enfermedades’ are difficult to process as well (little information about the meaning of these features, many unique and empty records).

It is also useful to keep in mind side information (risk factors, typical symptoms and so on).

So, we have only 7 features: ‘Gender’, ‘IAH’, ‘Peso’, ‘Talla’, ‘Edad’, PerCervical’, ‘Fumador’. Let’s create the DataFrame with such features.

```
OSA_df = data[['Gender','IAH','Peso',
                    'Talla','Edad', 'PerCervical', 'Fumador']]
```

Renaming columns to English names:

```
OSA_df.columns = ['Gender','IAH','Weight',
                    'Height','Age','Cervical', 'Smoker']
```

At the end of the table there are many empty records. So, let's fix it. If some records have 'NaN' in 'Gender' column, then we delete them.

```
OSA_df_reduced = OSA_df.dropna(subset = ["Gender"])
```

3.3. Checking missing values and object columns.

It is good to check if our new DataFrame has some missing values. We can use method info().

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 683 entries, 0 to 682
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Gender    683 non-null    object
 1   IAH       649 non-null    float64
 2   Weight    678 non-null    object
 3   Height    677 non-null    float64
 4   Age       678 non-null    float64
 5   Cervical  678 non-null    float64
 6   Smoker    680 non-null    object
dtypes: float64(4), object(3)
memory usage: 42.7+ KB
```

Figure 1. Info() of the initial DataFrame

This dataset has many missing values and 3 different columns with the type 'object'. First, let's work with object columns.

3.4. Categorical features and the 'object' type of column

There are three columns with 'object' type (two categorical features and one column with some wrong records). If we look at the column 'Weight' then we will see that it is supposed to be numeric. So, let's fix it.

```
OSA_df_reduced['Weight'] = OSA_df_reduced['Weight'].apply(pd.to_numeric, errors='coerce')
```

We have changed all non-numeric values to 'NaN'.

As for the object column 'Gender', we have to figure out how many values it has. According to figure 10, column 'Gender' has only two genders: hombre and mujer.

```
OSA_df_reduced['Gender'].value_counts()
```

```
OSA_df_reduced['Gender'].value_counts()
```

```
hombre    488
mujer     195
Name: Gender, dtype: int64
```

Figure 2. List of values of the column 'Gender'

Let’s check the same information for the column ‘Smoker’.

List of values: ‘no’, ‘si’, ‘antiguo’, ‘ns’, ‘poco’, ‘si (poco)’.

```
OSA_df_reduced['Smoker'].value_counts()
```

```
no           373
si           165
antiguo      119
ns            16
poco           6
si (poco)      1
Name: Smoker, dtype: int64
```

Figure 3. List of values of the column 'Smoker'.

As we can see, there are three categories which mean almost the same (si, poco, si(poco)). I want to combine them into one category ‘si’.

```
OSA_df_reduced.loc[OSA_df_reduced['Smoker'] ==
                   'poco', 'Smoker'] = 'si'
OSA_df_reduced.loc[OSA_df_reduced['Smoker'] ==
                   'si (poco)', 'Smoker'] = 'si'
```

If we do not know this information (‘Smoker’ = ns), then we replace it with NaN.

```
OSA_df_reduced.loc[OSA_df_reduced['Smoker'] ==
                   'ns', 'Smoker'] = np.NaN
```

3.5. Dealing with NaN values

In order to apply the label encoding we have to deal with NaN values and -1 values (which mean also NaN in this dataset).

Now we use pandas method ‘describe’ to look at some basic characteristics.

```
OSA_df_reduced.describe()
```

The result has interesting values.

	IAH	Weight	Height	Age	Cervical
count	649.000000	676.000000	677.000000	678.000000	678.000000
mean	20.364653	87.665680	171.144756	49.280236	40.188053
std	18.692784	18.542861	11.661385	12.851542	5.746654
min	0.000000	-1.000000	-1.000000	-1.000000	-1.000000
25%	6.300000	75.000000	165.000000	40.000000	38.000000
50%	14.200000	86.000000	171.000000	49.000000	41.000000
75%	30.000000	98.000000	178.000000	59.000000	43.000000
max	108.600000	165.000000	199.000000	88.000000	53.000000

Figure 4. The result of 'describe' method

As we can see minimum of some columns has number -1 (it was a replacement of missing values). So, let's replace -1 with NaN in order to analyze them.

For this purpose, we have to import numpy library and use its method 'replace'.

```
import numpy as np
OSA_df_reduced = OSA_df_reduced.replace(-1.0,np.NaN)
```

After that we import two other libraries to analyzi NaNs in the DataFrame. Missingno – provides a series of visualisations to understand the presence and distribution of missing data within a pandas dataframe. Matplotlib – a comprehensive library for creating static, animated, and interactive visualizations in Python.

```
import missingno as msno
import matplotlib.pyplot as plt
```

Using these libraries we plot two figures.

```
msno.matrix(OSA_df)
```

```
msno.matrix(OSA_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5070a89f90>
```

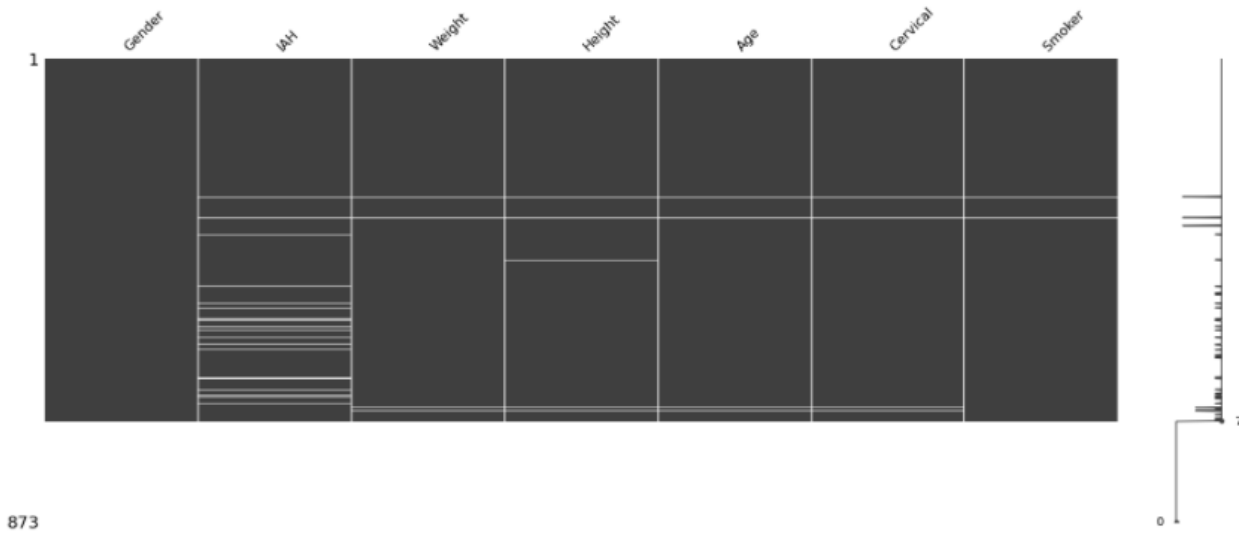

Figure 5. Plot of missing values in the DataFrame

```
msno.bar(OSA_df)
```

```
msno.bar(OSA_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f506301a3d0>
```

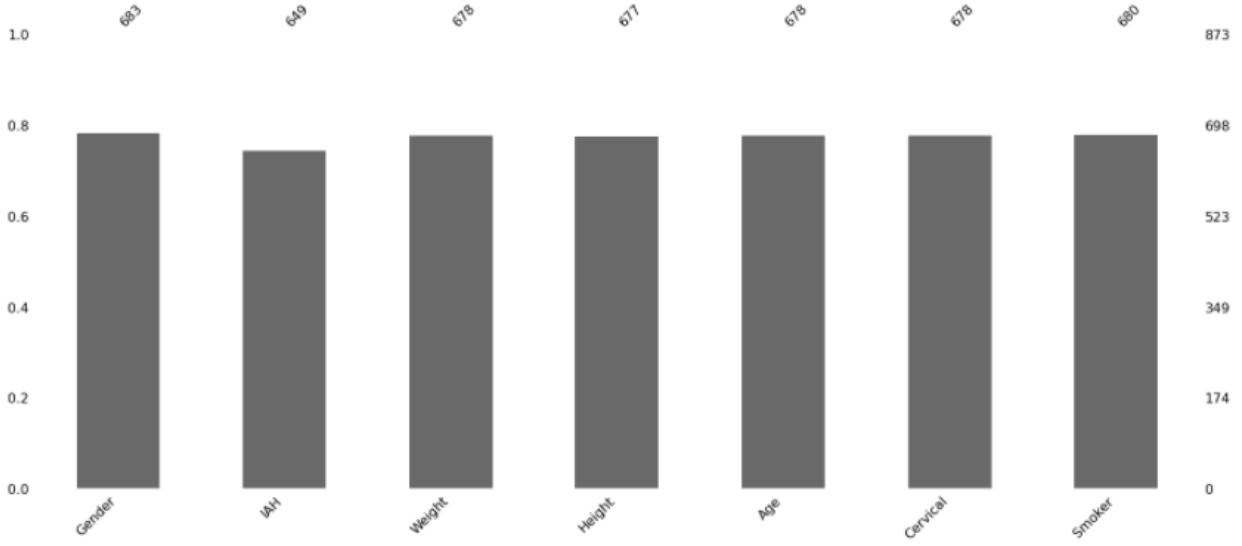

Figure 6. Barplot of missing values

The last step in data preparation is to drop all NaN values.

```
OSA_df_cleaned = OSA_df_reduced.dropna()
```

Now we have a smaller number of records in the DataFrame (625 instead of 683). So, we can update indexes to simplify the managing of the data.

```
OSA_df_cleaned
```

	Gender	IAH	Weight	Height	Age	Cervical	Smoker
1	hombre	29.6	119.0	174.0	56.0	48.0	si
3	hombre	19.7	78.0	168.0	39.0	42.0	no
4	hombre	9.0	80.0	173.0	32.0	40.0	no
5	hombre	2.0	109.0	190.0	32.0	42.0	no
6	hombre	34.0	86.0	169.0	39.0	42.0	no
...	...	...	...	...	...	...	...
676	mujer	36.3	82.0	165.0	64.0	39.0	antiguo
678	hombre	9.2	105.0	180.0	35.0	45.0	no
679	hombre	52.2	90.0	180.0	50.0	42.0	no
681	mujer	5.4	66.0	164.0	57.0	35.0	si
682	mujer	31.9	70.0	163.0	60.0	35.0	antiguo

625 rows × 7 columns

Figure 7. DataFrame after NaN deleting

Updating indexes.

```
OSA_df_cleaned.reset_index(drop=True)
```

Let’s check the result of the preparation.

```
OSA_df_cleaned.describe()
```

	IAH	Weight	Height	Age	Cervical
count	625.000000	625.000000	625.00000	625.000000	625.00000
mean	20.379296	87.692800	171.36800	49.443200	40.62800
std	18.642982	18.271941	9.49809	12.362723	3.97705
min	0.000000	48.000000	144.00000	20.000000	30.00000
25%	6.300000	75.000000	165.00000	40.000000	38.00000
50%	14.400000	86.000000	171.00000	49.000000	41.00000
75%	30.000000	98.000000	178.00000	59.000000	43.00000
max	108.600000	165.000000	197.00000	88.000000	53.00000

Figure 8. 'Describe' method for the result

The last thing to do is the label encoding.

3.6. Label Encoding

First, we import the following library:

```
from sklearn.preprocessing import LabelEncoder
```

And apply LabelEncoder for both categorical features:

```
labelencoder = LabelEncoder()
OSA_df_cleaned['Gender'] = labelencoder.fit_transform(OSA_df_cleaned['Gender'])
OSA_df_cleaned['Smoker'] = labelencoder.fit_transform(OSA_df_cleaned['Smoker'])
```

The result can be seen in the figure 9.

	Gender	IAH	Weight	Height	Age	Cervical	Smoker
0	0	29.6	119.0	174.0	56.0	48.0	2
1	0	19.7	78.0	168.0	39.0	42.0	1
2	0	9.0	80.0	173.0	32.0	40.0	1
3	0	2.0	109.0	190.0	32.0	42.0	1
4	0	34.0	86.0	169.0	39.0	42.0	1
...	...	...	...	...	...	...	...
620	1	36.3	82.0	165.0	64.0	39.0	0
621	0	9.2	105.0	180.0	35.0	45.0	1
622	0	52.2	90.0	180.0	50.0	42.0	1
623	1	5.4	66.0	164.0	57.0	35.0	2
624	1	31.9	70.0	163.0	60.0	35.0	0

625 rows × 7 columns

Figure 9. Features after the label encoding

3.7. Feature engineering

3.7.1. Body mass index

I would like to add the feature that indicates the body mass index. The formula:

$$BMI = \frac{weight_{kg}}{height_{m}^{2}}$$

```
OSA_df_cleaned['BMI'] = OSA_df_cleaned['Weight']/((OSA_df_cleaned['Height']/100)**2)
```

3.7.2. Weight status

Another feature to create is ‘Weight status’ based on body mass index (BMI).

Table 1. Correspondence between 'BMI' and "Weight status'

BMI	Weight class	Label
Below 18.5	Underweight	0
18.5-24.9	Normal	1
25-29.9	Overweight	2
30 and up	Very overweight	3

Code:

```
OSA_df_cleaned['Weight status'] = pd.cut(OSA_df_cleaned['BMI'], bins=[0, 18.5, 24.9, 29.9, float('Inf')], labels=['Underweight', 'Normal', 'Overweight', 'Very overweight'])
```

After creating categories, we have to apply LabelEncoder:

```
OSA_df_cleaned['Weight status'] = labelencoder.fit_transform(OSA_df_cleaned['Weight status'])
```

3.7.3. Age group

As we know people over 40 years old are in the risk of having the OSA. So, let’s create two age group for people under and over 40 years old.

```
OSA_df_cleaned['Age group'] = pd.cut(OSA_df_cleaned['Age'], bins=[0, 40, float('Inf')], labels=['Low Risk', 'High Risk'])
OSA_df_cleaned['Age group'] = labelencoder.fit_transform(OSA_df_cleaned['Age group'])
```

3.7.4 Result of the feature engineering

The result can be seen on the figure 10.

	Gender	IAH	Weight	Height	Age	Cervical	Smoker	BMI	Weight status	Age group
0	0	29.6	119.0	174.0	56.0	48.0	2	39.305060	3	0
1	0	19.7	78.0	168.0	39.0	42.0	1	27.636054	1	1
2	0	9.0	80.0	173.0	32.0	40.0	1	26.729927	1	1
3	0	2.0	109.0	190.0	32.0	42.0	1	30.193906	3	1
4	0	34.0	86.0	169.0	39.0	42.0	1	30.110991	3	1
...	...	...	...	...	...	...	...	...	...	...
620	1	36.3	82.0	165.0	64.0	39.0	0	30.119376	3	0
621	0	9.2	105.0	180.0	35.0	45.0	1	32.407407	3	1
622	0	52.2	90.0	180.0	50.0	42.0	1	27.777778	1	0
623	1	5.4	66.0	164.0	57.0	35.0	2	24.538965	0	0
624	1	31.9	70.0	163.0	60.0	35.0	0	26.346494	1	0

625 rows × 10 columns

Figure 10. Result of feature engineering

CHAPTER 4. REGRESSION PROBLEM

Now, let's summarize the content of the dataset for the regression problem, including average values, maximum and minimum value, and std for each variable. In order to do that, it would be better to provide tables separately for continuous and categorical values.

Table 2. Categorical value. Review.

	Gender	Smoker	Weight status	Age group
Number of each value	Male (450)	No (351)	Overweight (261)	Low Risk (177)
	Female (175)	Yes (161)	Very overweight (260)	High Risk (448)
	-	Used to (113)	Normal (104)	-

Table 3. Continuous values. Review.

	IAH	Weight	Height	Age	Cervical	BMI
Mean	20.38	87.69	171.37	49.44	40.63	29.82
Std	18.64	18.27	9.49	12.36	3.98	5.58
Min	0	48	144	20	30	18.29
Max	108.6	165	197	88	53	63.65

4.1. EDA for regression model

4.1.1. Correlation between features

It is good practice to check correlated features and remove them. We can use library 'seaborn' to do it.

```
import seaborn as sns
OSA_df_reduced.corr()
heatmap = sns.heatmap(OSA_df_cleaned.corr(),
                      vmin=-1, vmax=1, annot=True)
```

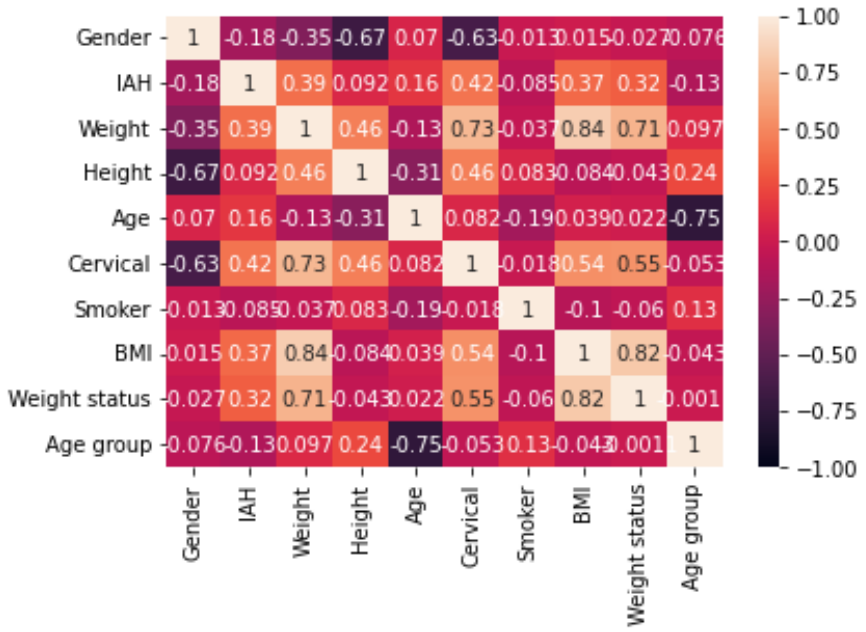

Figure 11. Correlation between features for regression problem

Obviously features 'Age'/'Age Group', 'BMI'/'Weight status' are highly correlated. During feature selection we will choose only one feature in each pair.

4.1.2. Scatter plots

Scatter plots are very useful for EDA in Regression. So, let's use scatter_matrix to plot all scatter plots between variables.

```
from pandas.plotting import scatter_matrix
axes = scatter_matrix(OSA_df_reduced, alpha=0.6,
                      figsize=(12, 8), diagonal='kde')
```

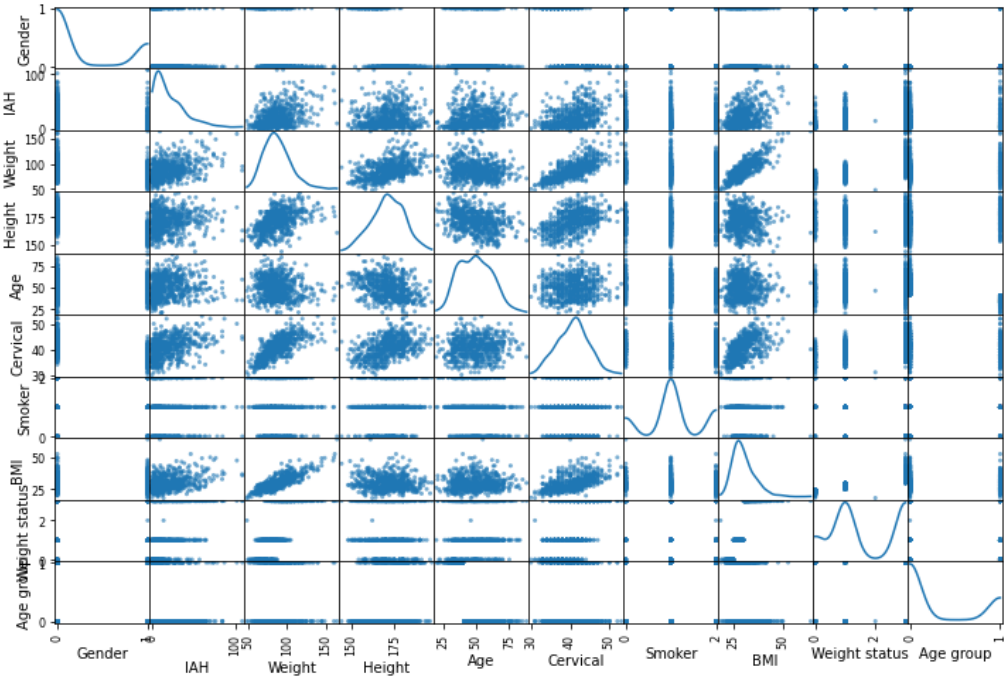

Figure 12. Scatter matrix for the regression problem

4.1.3. ‘Violinplot’ from ‘Seaborn’ library

Another useful tool is the ‘violinplot’ from ‘seaborn’ library. It helps to visualize the distribution of different numeric values. We can visualize data distributions by the ranges, medians and distributions of the data. Let’s have a look at two examples for our case study.

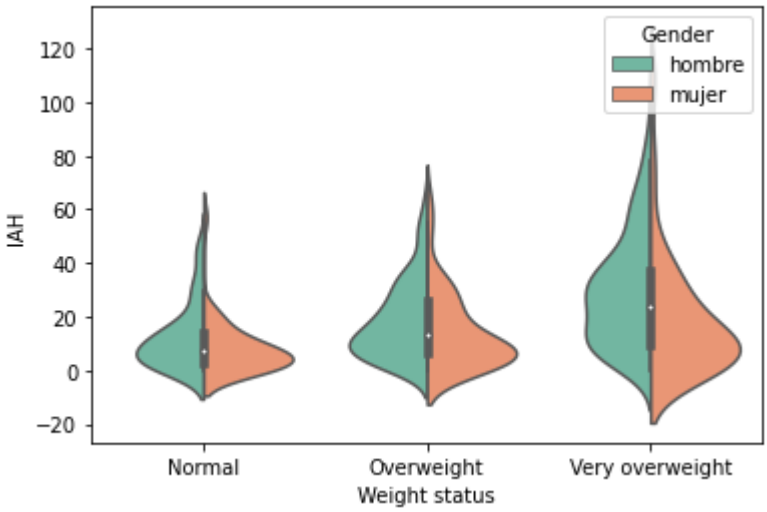

Figure 13. Violinplot for ‘Weight status’, ‘Gender’, and ‘IAH’

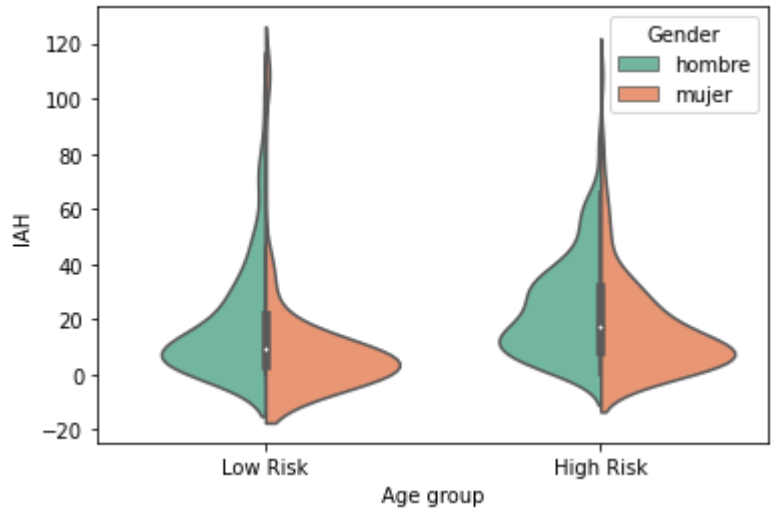

Figure 14. Violinplot for ‘Age group, ‘Gender’, and ‘IAH’

This gives us good intuition about the data. I plotted ‘violinplot’ for other features as well.

4.2. Feature selection

First, we need to split our data into train and test subsets.

```
from sklearn.model_selection import train_test_split
X = OSA_df_cleaned.drop(['IAH'],axis=1)
y = OSA_df_cleaned.IAH
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
0.20, random_state=42)
```

After that we will use two popular feature selection techniques:

- Correlation Statistics
- Mutual Information Statistics

4.2.1. Correlation Statistics

The scikit-learn machine learning library provides an implementation of the correlation statistic in the f_regression() function. This function can be used in a feature selection strategy, such as selecting the top k most relevant features (largest values) via the SelectKBest class.

For example, we can define the SelectKBest class to use f_regression() function and select all features, then transform the train and test sets.

```
fs = SelectKBest(score_func=f_regression, k='all')
fs.fit(X_train, y_train)
X_train_fs = fs.transform(X_train)
X_test_fs = fs.transform(X_test)
for i in range(len(fs.scores_)):
	print('Feature %s: %f' % (X.columns[i], fs.scores_[i]))
plt.bar([i for i in range(len(fs.scores_))], fs.scores_)
plt.show()
```

The result can be seen below.

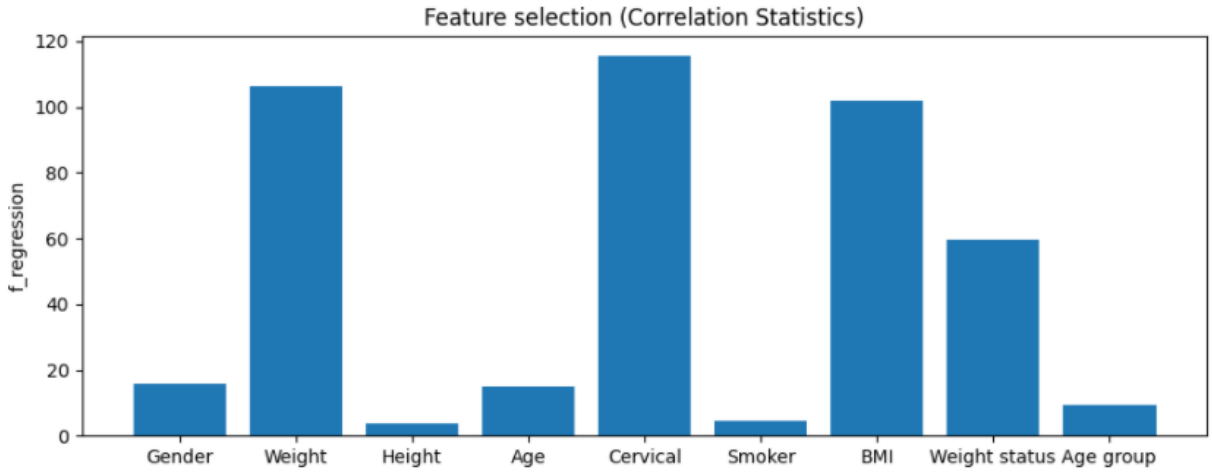

Figure 15. Feature selection via Correlation Statistics for the regression

4.2.2. Mutual Information Statistics

Mutual information is calculated between two variables and measures the reduction in uncertainty for one variable given a known value of the other variable.

The scikit-learn machine learning library provides an implementation of mutual information for feature selection with numeric input and output variables via the mutual_info_regresion() function.

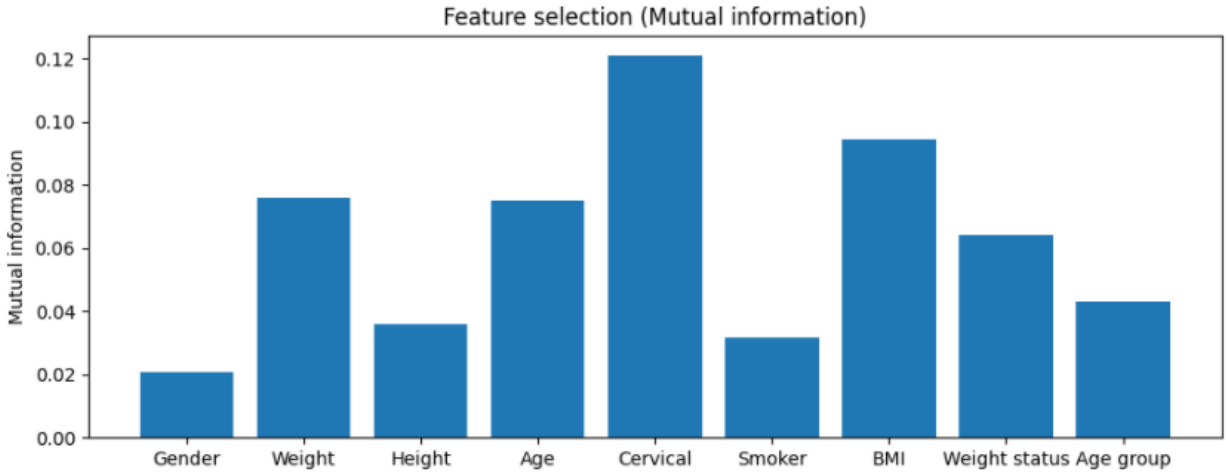

Figure 16. Feature selection via Mutual Information Statistics for the regression

Combining two techniques we can conclude that the most valuable features are ‘Weight’, ‘Age’, ‘Cervical’ and ‘BMI’.

4.3. Regression models and results

4.3.1. Methodology and tools

For regression problem I used several models:

- Multiple linear regression
- Decision Tree
- Support Vector Machines
- Ridge regression
- Random forest
- XGBoost
- LightGBM
- CatBoost

I tried to analyze the results using two metrics: mean squared error and mean absolute error.

As for features, I used two sets of features: the first set (‘Age’, ‘Cervical’, ‘BMI’, ‘Gender’ and ‘Smoker’) is a set of features that I consider the most powerful from the predictive point of view, the second set (‘Age’, ‘Cervical’, ‘BMI’, and

‘Weight’) is a set of features which is the most important considering feature selection methods.

To evaluate the models and choose parameters I used grid search and cross-validation with 5 folds. Since we have relatively small dataset, we can use the grid search instead of the random search. For the hyperparameter tuning I used the ‘neg_mean_squared_error’ metric.

4.3.2. Comparison of the performance of all models

1. The first set of features

Table 4. Performance of all models for the first set of features.

	RMSE	MAE
LightGBM	14.775296	11.902945
Random Forest	15.037943	12.060496
XGBoost	15.167377	11.699624
Support Vector Machines	15.261742	11.580226
Ridge Regression	15.416183	12.385416
Linear Regression	15.484188	12.433457
CatBoost	15.973515	12.675998
DecisionTree	16.713646	12.989686

Let’s plot all results for better understanding of the performance of all models.

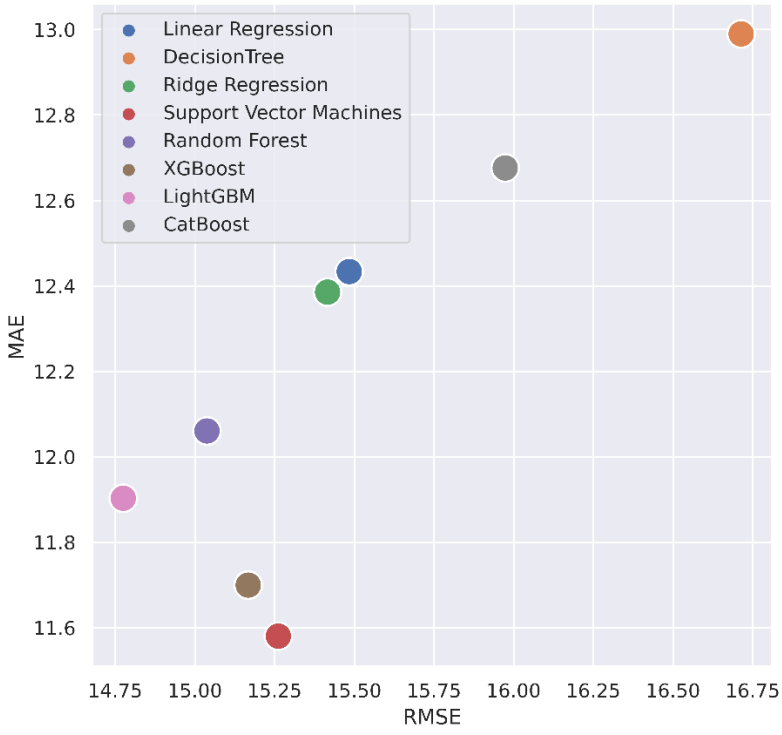

Figure 17. Performance of all models for the first set of features.

2. The second set of features

Table 5.Performance of all models for the second set of features.

	RMSE	MAE
Random Forest	14.472711	11.498644
DecisionTree	14.803555	11.700865
LightGBM	14.822141	11.961986
XGBoost	14.974634	11.633279
CatBoost	15.195056	12.125387
Ridge Regression	15.501052	12.616005
Linear Regression	15.598855	12.651881
Support Vector Machines	15.724625	12.132597

Let’s plot all results for better understanding of the performance of all models.

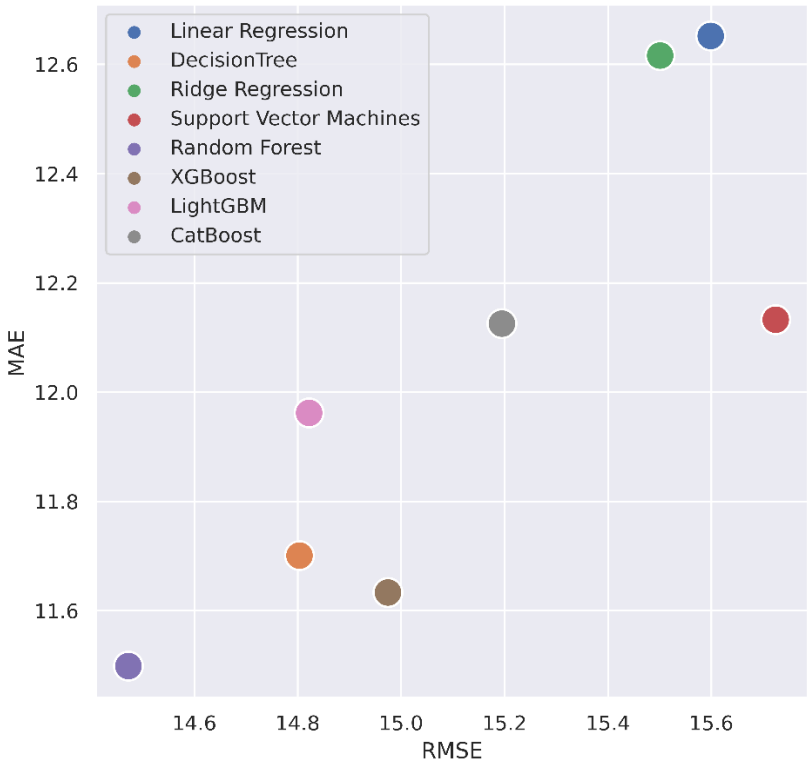

Figure 18.Performance of all models for the second set of features.

As we can see in the figures 17 and 18, the results depend significantly on the features that we choose and models that we use. However, three machine learning models such as LightGBM, XGBoost, and Random Forest work well for both sets of features. The rest of the models work a bit worse or work well for a particular set of features.

CHAPTER 5. CLASSIFICATION PROBLEM

Let's prepare the same summary of the dataset for classification problem that we did for the regression problem. Here I also consider both genders and use it as a feature.

Table 6. Categorical value. Review.

	Gender	Smoker	Weight status	Age group	OSA_Class
Number of each value	Male (450)	No (351)	Overweight (261)	Low Risk (177)	Healthy (237)
	Female (175)	Yes (161)	Very overweight (260)	High Risk (448)	Mild (237)
	-	Used to (113)	Normal (104)	-	Severe (151)

Table 7. Continuous values. Review.

	IAH	Weight	Height	Age	Cervical	BMI
Mean	20.38	87.69	171.37	49.44	40.63	29.82
Std	18.64	18.27	9.49	12.36	3.98	5.58
Min	0	48	144	20	30	18.29
Max	108.6	165	197	88	53	63.65

5.1. EDA for classification model

I would like to consider both genders 'Male' and 'Female' and divide the 'IAH' values into three classes 'Healthy', 'Mild', and 'Severe'.

We can do it in the same way we did it for 'Age group' and 'Weight status'.

```
OSA_class['OSA Class'] = pd.cut(OSA_class['IAH'], bins=[0,  10, 30, float('Inf')], labels=['Healthy', 'Mild', 'Severe'])
```

We can check the number of values for each class.

```
OSA_clmodel.OSA_Class.value_counts()
```

```
Mild       237
Healthy    237
Severe     151
Name: OSA_Class, dtype: int64
```

Figure 19. List of values of the column 'OSA_Class'

Then let's apply LabelEncoder to turn strings into numeric values.

```
OSA_clmodel['OSA_Class'] = labelencoder.fit_transform(OSA_clmodel['OS
                                                      A_Class'])
```

	Gender	IAH	Weight	Height	Age	Cervical	Smoker	BMI	Weight status	Age group	OSA_Class
0	0	29.6	119.0	174.0	56.0	48.0	2	39.305060	3	0	1
1	0	19.7	78.0	168.0	39.0	42.0	1	27.636054	1	1	1
2	0	9.0	80.0	173.0	32.0	40.0	1	26.729927	1	1	0
3	0	2.0	109.0	190.0	32.0	42.0	1	30.193906	3	1	0
4	0	34.0	86.0	169.0	39.0	42.0	1	30.110991	3	1	2
...	...	...	...	...	...	...	...	...	...	...	...
620	1	36.3	82.0	165.0	64.0	39.0	0	30.119376	3	0	2
621	0	9.2	105.0	180.0	35.0	45.0	1	32.407407	3	1	0
622	0	52.2	90.0	180.0	50.0	42.0	1	27.777778	1	0	2
623	1	5.4	66.0	164.0	57.0	35.0	2	24.538965	0	0	0
624	1	31.9	70.0	163.0	60.0	35.0	0	26.346494	1	0	2

625 rows × 11 columns

Figure 20. The result after applying LabelEncoder

Now, ‘OSA_Class’ is our target column.

5.2. Different plots for better intuition

5.2.1. Histograms per class

Let’s plot histograms for main features and identify some dependencies.

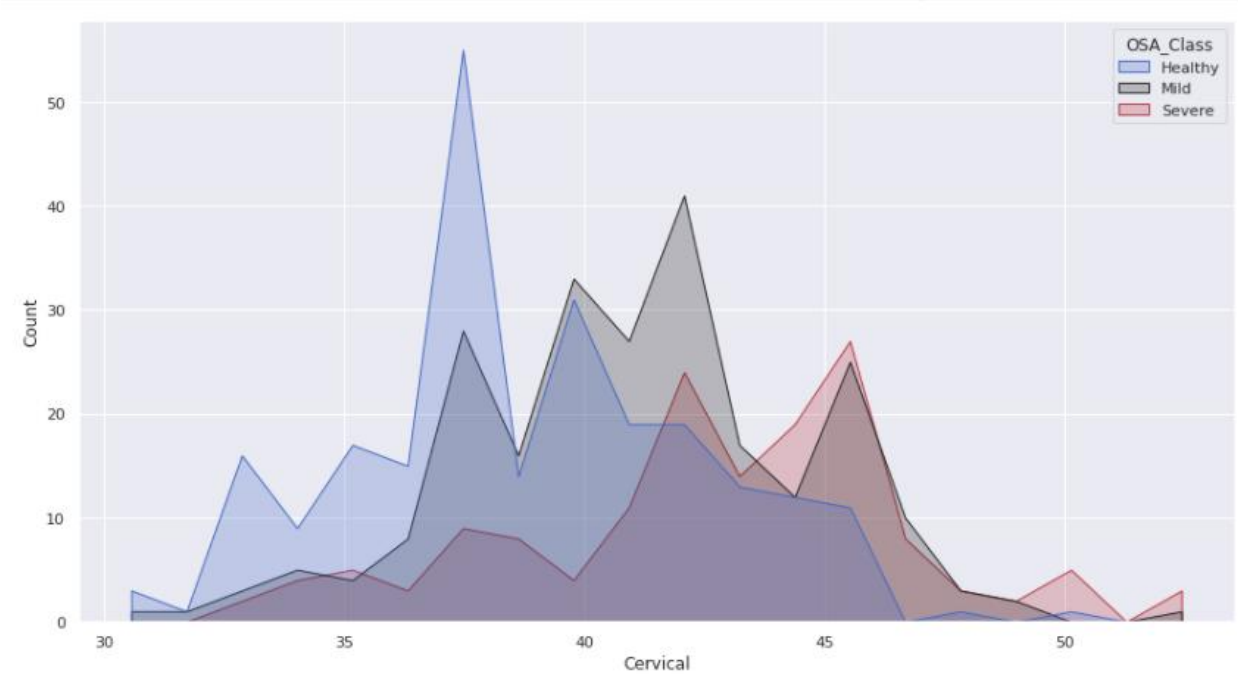

Figure 21. Histogram for 'OSA_class' and 'Cervical'

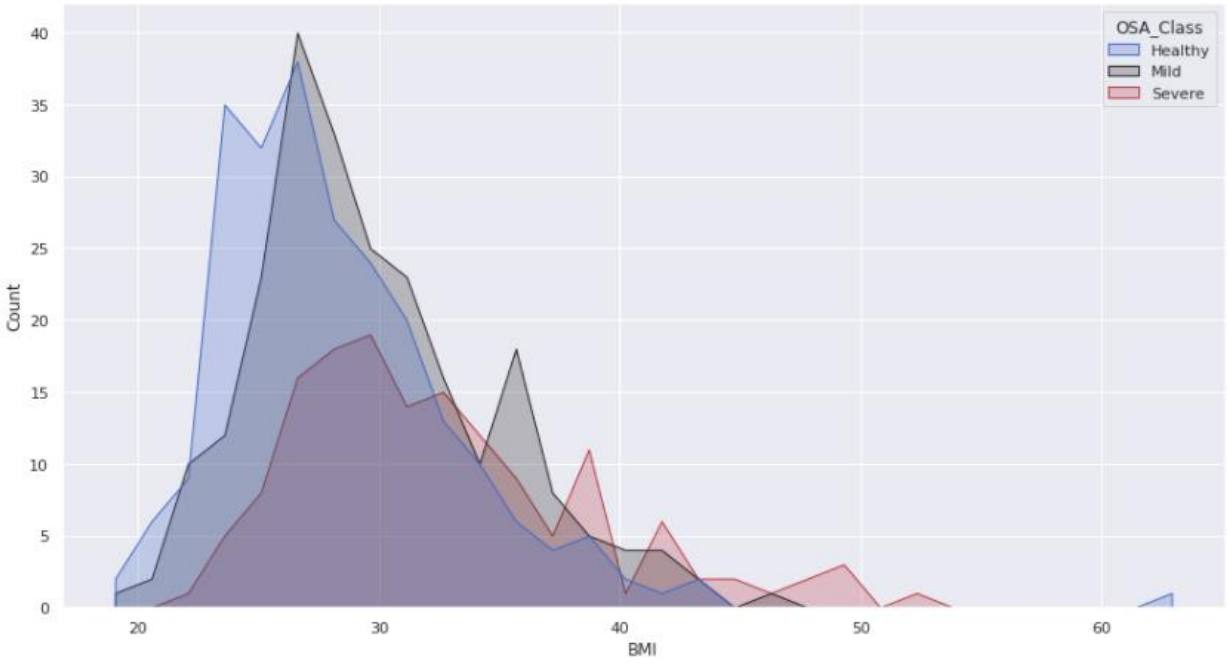

Figure 22. Histogram for 'OSA_Class' and 'BMI'

5.2.2. Scatter plots

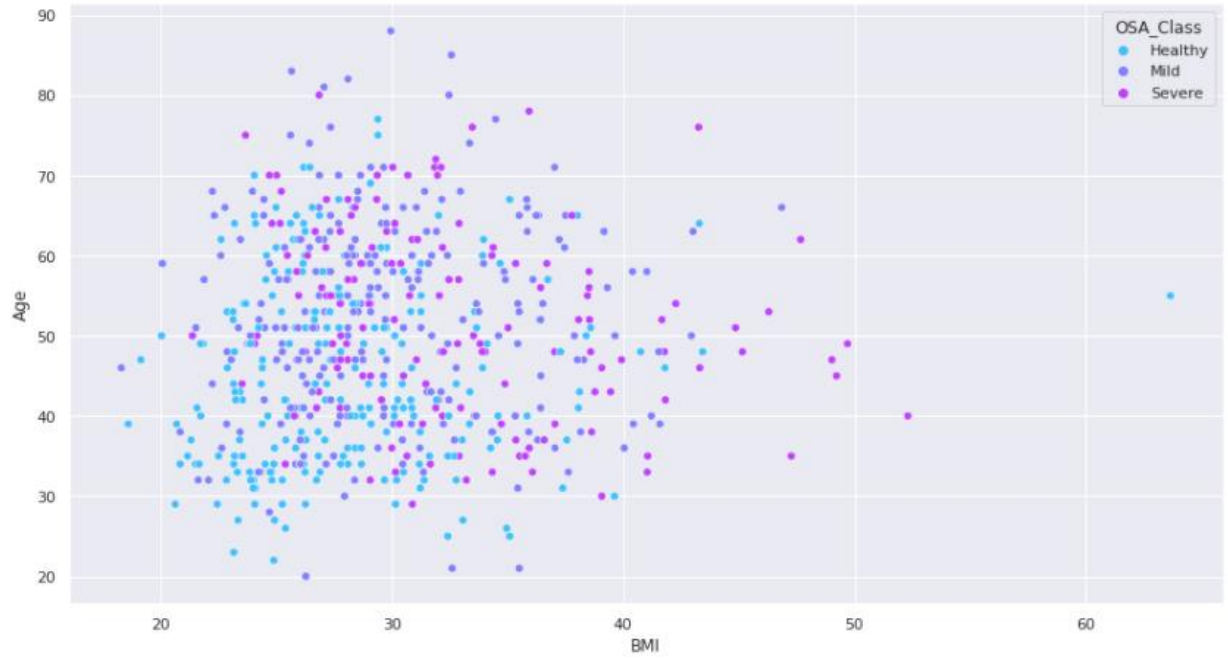

Figure 23. Scatter plot for 'BMI', 'Age', and 'OSA_Class'

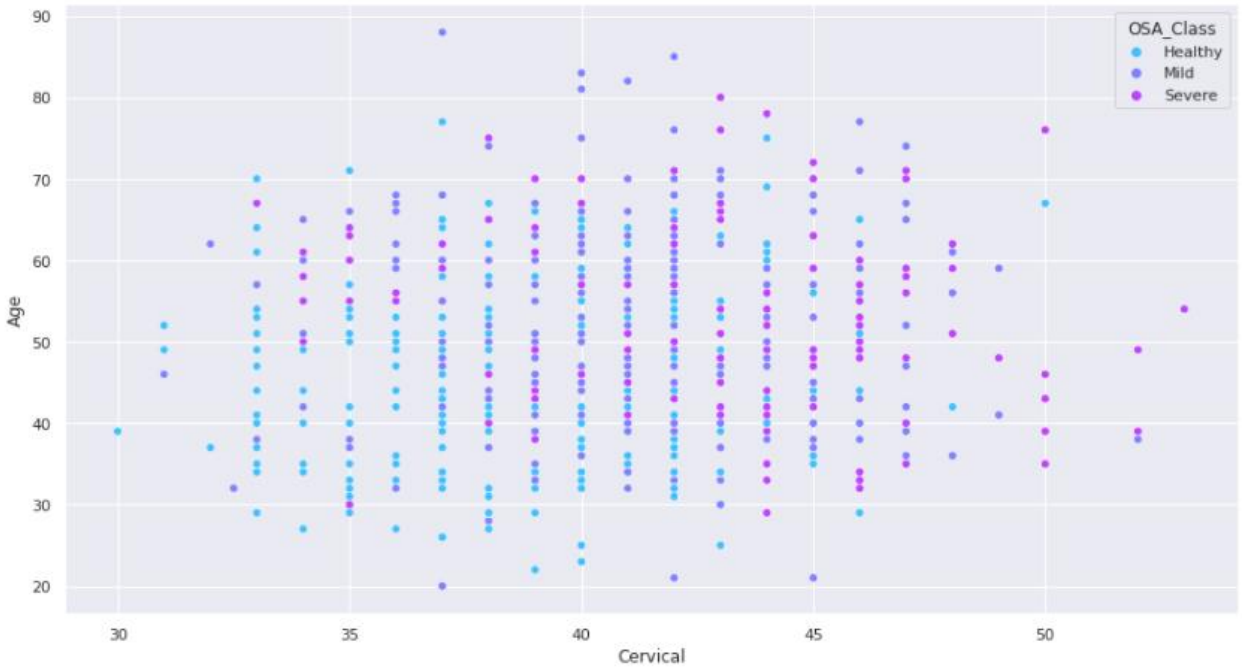

Figure 24. Scatter plot for 'Cervical', 'Age', and 'OSA_Class'

5.3. Feature selection

For the feature selection in the classification problem, we will go through the same process that we did for the regression problem.

Again, we need to split the dataset into two parts: train and test subsets.

```
X = OSA_clmodel.drop(['IAH'],axis=1)
y = OSA_clmodel.IAH
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

5.3.1. Correlation Statistics

Let’s first plot a correlation matrix.

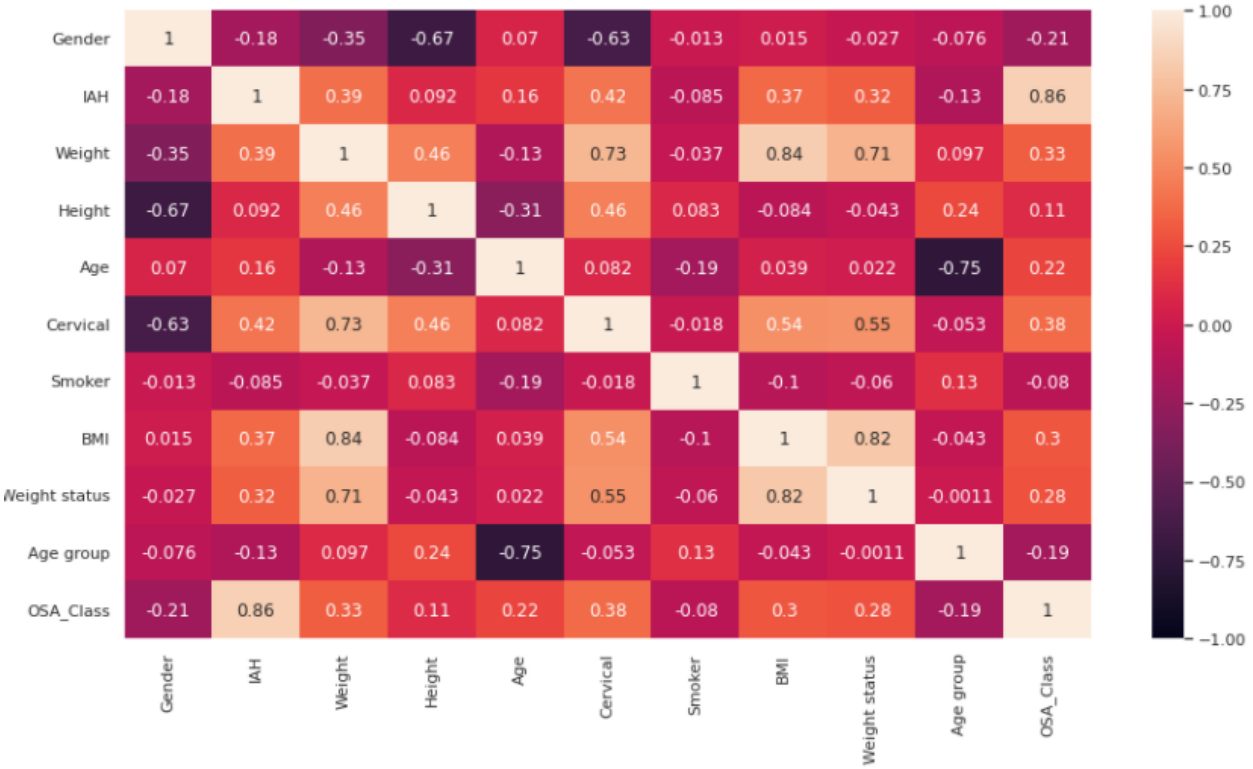

Figure 25. Correlation matrix of features for the classification problem

Also, we I used the ANOVA (analysis of variance) method. It is a type of F-statistic. Importantly, ANOVA is used when one variable is numeric and one is categorical, such as numerical variables and a classification target variable in a classification task.

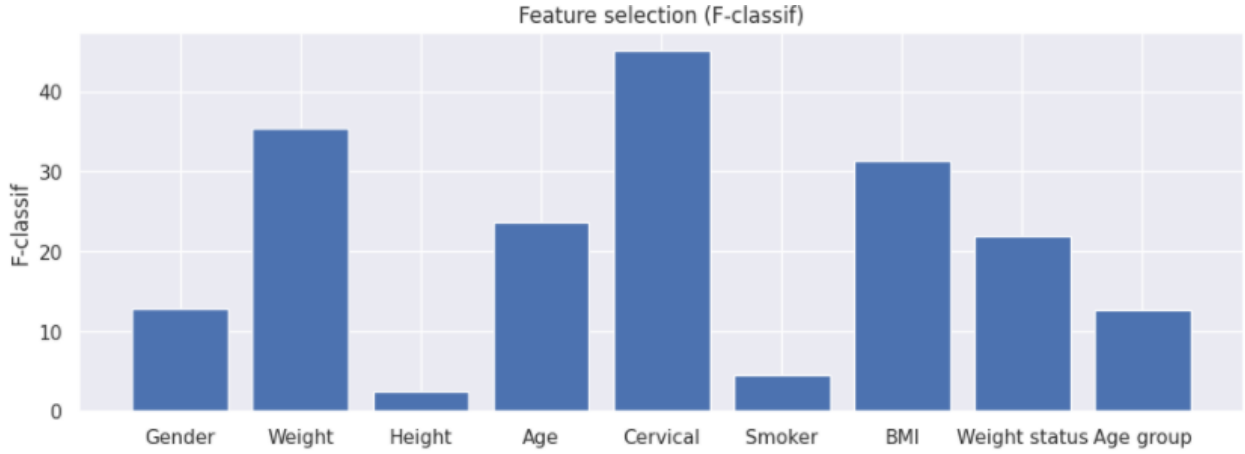

Figure 26. Feature selection via Correlation Statistics for the classification

5.3.2. Mutual information Statistics

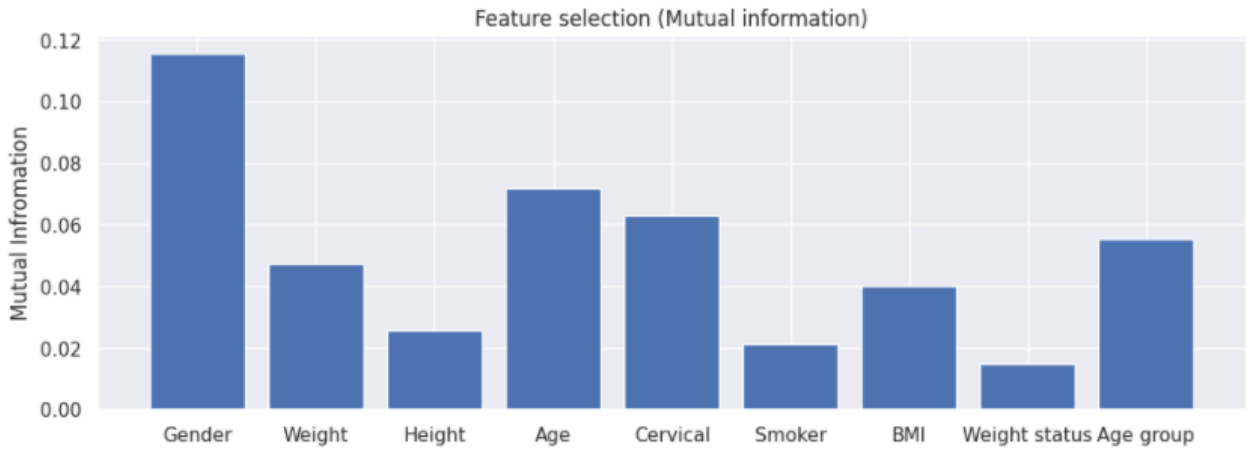

Figure 27. Feature selection via Mutual Information for the classification

Combining two techniques we can conclude that the most valuable features are ‘Gender’, ‘Weight’, ‘Age’, and ‘Cervical’.

5.4. Classification models and results

5.4.1. Methodology and tools

For classification problem I used following models:

- Logistic regression
- Decision Tree
- Support Vector Machines
- Ridge classifier
- Random forest classifier
- XGBoost
- LightGBM
- CatBoost

I analyzed the results using two metrics: f1_micro and f1_macro.

As for features, I used two sets of features: the first set (‘Age’, ‘Cervical’, ‘BMI’, ‘Gender’ and ‘Smoker’) is a set of features that I consider the most powerful from the predictive point of view, the second set (‘Gender’, ‘Weight’, ‘Cervical’, and ‘Age’) is a set of features which is the most important considering feature selection methods.

To evaluate the models and choose parameters I used the grid search and the cross-validation with 5 folds. For hyperparameter tuning I used ‘roc_auc_ovr’ metric.

5.4.2. Comparison of the performance of all models

1. The first set of features

Table 8. Performance of all models for the first set of features.

	F1_macro	F1_micro
XGBoost	0.509823	0.536
LightGBM	0.505743	0.536
Random Forest	0.498101	0.536
CatBoost	0.480900	0.520
Logistic Regression	0.479502	0.536
Ridge	0.479156	0.536
Support Vector Machines	0.398128	0.504
Decision Tree	0.375350	0.496

Let’s plot all results for better understanding of the performance of all models.

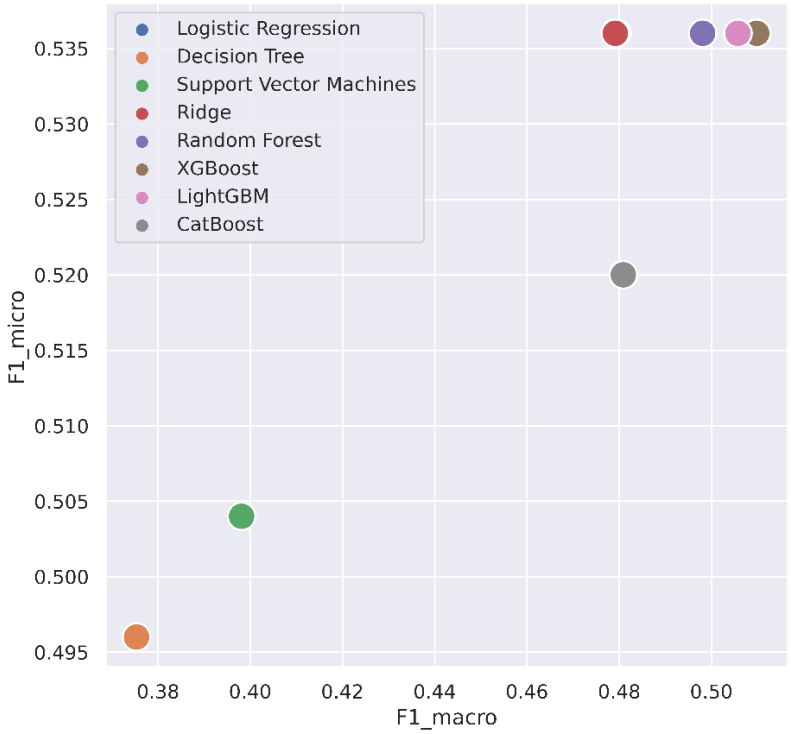

Figure 28. Performance of all models for the first set of features.

2. The second set of features

Table 9. Performance of all models for the second set of features.

	F1_macro	F1_micro
XGBoost	0.489948	0.512
Ridge	0.476006	0.512
CatBoost	0.467515	0.488
Support Vector Machines	0.465304	0.520
Logistic Regression	0.462950	0.496
Random Forest	0.456799	0.496
Decision Tree	0.434467	0.464
LightGBM	0.422447	0.464

Let’s plot all results for better understanding of the performance of all models.

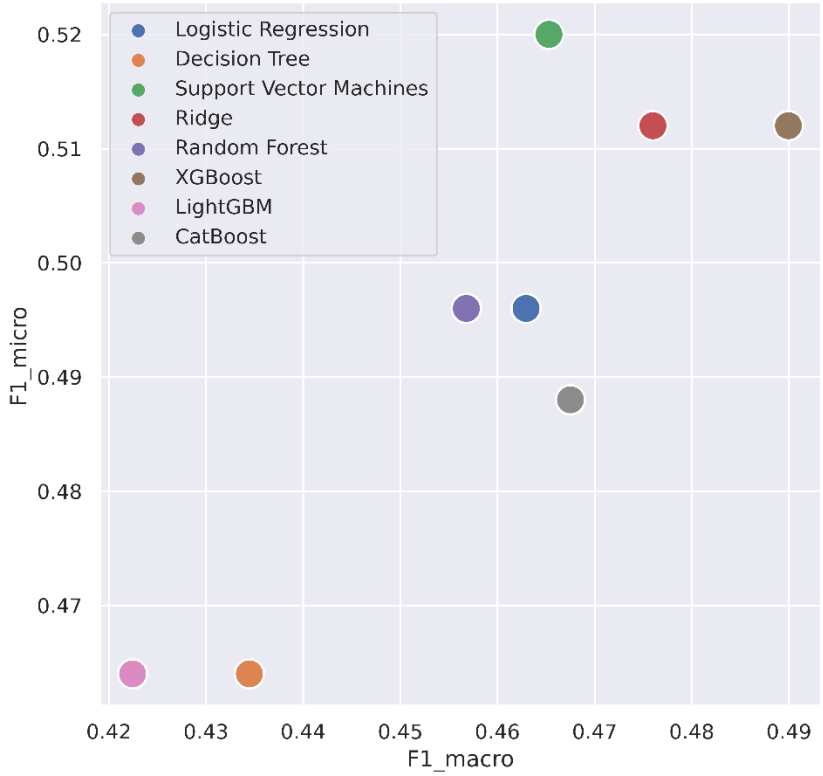

Figure 29. Performance of all models for the second set of features.

The situation is similar to the regression problem, but there are different models which work better than others. For example, here we can see that RidgeClassifier, XGBoost, RandomForest, and LogisticRegression work better than other models. The dependency on the set of features still stands.

The code with model evaluations could be found in the attached .ipynb files.

6. CONCLUSIONS

In this work, I did the whole machine learning project considering two main problems: regression and classification. During the project, I used different libraries and tools to preprocess, visualize and analyze data. They significantly simplify the working process and make it more clear for understanding.

The project can be divided into several parts: data reading, preprocessing, exploratory data analysis, feature selection, models evaluation, and other smaller parts. In this paper, I presented all technical details of each part including the code and comments for it.

The main library of the project is sklearn. It is a very powerful tool for machine learning projects. Two other important libraries are pandas and numpy. Pandas provides great tools for data analysis and data manipulation. Numpy provides some mathematical functions, routines, and tools. For data visualization, I used seaborn and matplotlib. Both libraries are great, helpful, and easy to understand.

Finally, I evaluated several models to predict the necessary values. Several models such as XGBoost, LightGBM, and RandomForest worked better than others for this particular problem.

Results could be further improved by trying other parameters of the models, different models, features (consider other techniques for feature engineering and so on). Also, I could try to improve the results using unsupervised learning including clustering and principal component analysis, but I decided not to do it since the number of features is relatively small and there is a good understanding of the dataset.

REFERENCES

[1] “Hands-On Machine Learning with Scikit-Learn & Tensorflow”, Aurelien Geron, O’Reilly, 2017.