Statistics for Data Analytics Final Project

Introduction

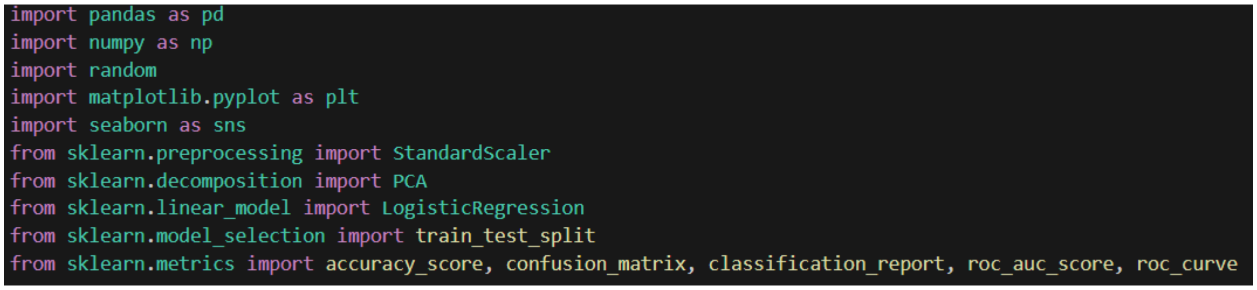
“The MIMIC II (Multiparameter Intelligent Monitoring in Intensive Care) Databases contain physiologic signals and vital signs time series captured from patient monitors, and comprehensive clinical data obtained from hospital medical information systems of Intensive Care Unit (ICU) patients.” - PhysioNet

According to the resource the data were collected between 2001 and 2008 from a variety of ICUs (Intensive Care Unit) (medical, surgical, coronary care, and neonatal) in a single tertiary teaching hospital. The MIMIC II Clinical Database contains clinical data from bedside workstations and hospital archives.

Summary

The objective of this project is applying all the concepts we learned during the semester related to Data Mining and Statistics beginning from the data analysis until implementing a logistic model which permit us to observe the results of our methodology.

First Steps – Importing Libraries

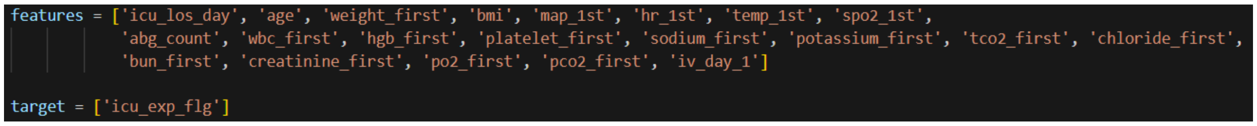


```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score, roc_curve
```

For this project I used a vast library to clean, process, analyse, and visualise the data. Libraries such as Pandas and NumPy are commonly used to preprocess, clean, and transform data; Matplotlib, Seaborn, and Random I used them for visualisation; and from the Scikit-Learn package, which is related to Machine Learning, I picked the preprocessing and decomposition modules to standardise and normalise the data, the Logistic Regression for creating a model and the metrics to obtain the results.

Exploring the data

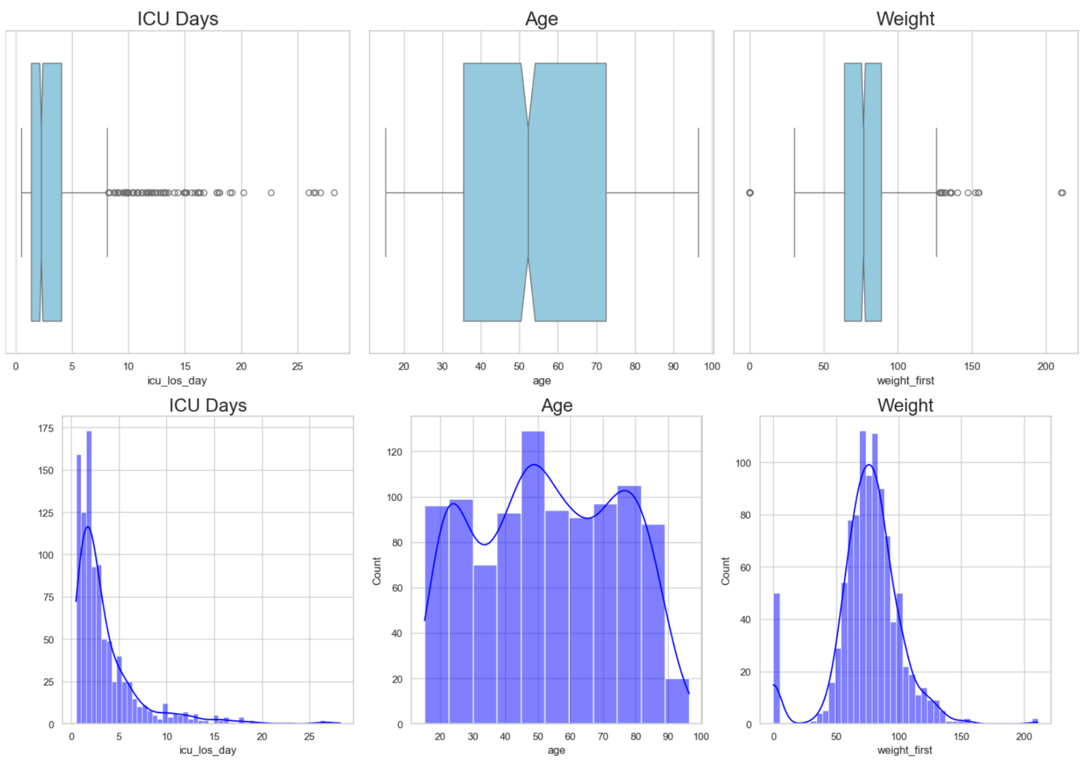
After loading the data, the next step is exploring the data. Analysing the variables descriptions I separated an array of my features which will be pre-processed and kept my specific target aside.

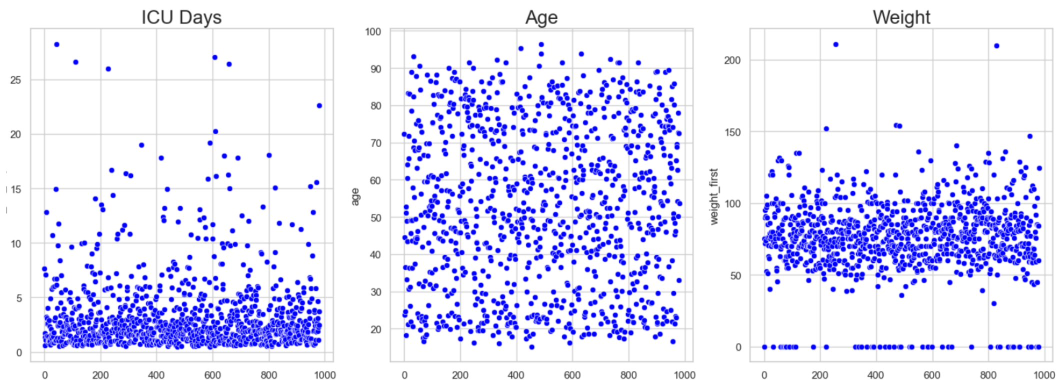


```
features = ['icu_los_day', 'age', 'weight_first', 'bmi', 'map_1st', 'hr_1st', 'temp_1st', 'spo2_1st',
            'abg_count', 'wbc_first', 'hgb_first', 'platelet_first', 'sodium_first', 'potassium_first', 'tco2_first', 'chloride_first',
            'bun_first', 'creatinine_first', 'po2_first', 'pco2_first', 'iv_day_1']

target = ['icu_exp_flg']
```

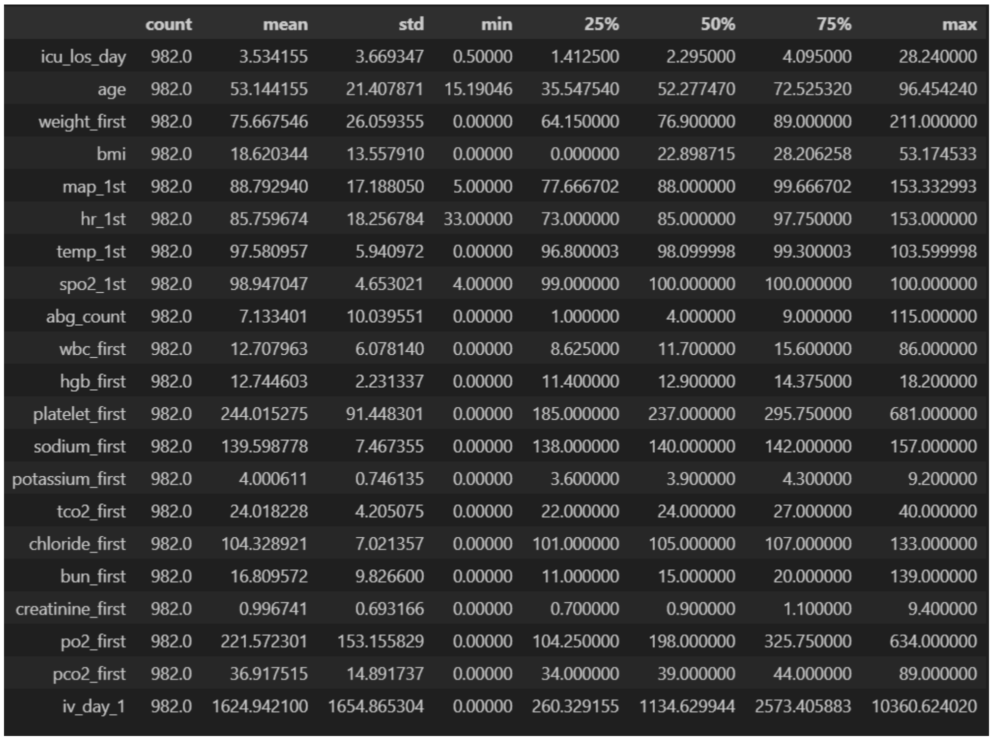
Based on that I set my ***x*** variable with the features ***x = UnitCareData.loc[:,features],*** getting a shape of ***(982, 21)***

Continuing with the exploratory data, another import point in the project was analysing the behaviour of the variables in terms of knowing their distributions, outliers, and how clean the data is. In this step all the features were plotted using boxplot, histograms, and scatterplots to identify outliers and distributions, due to the quantity of the variables I decided to divide in four group just to make easy the interpretation and analysis. The next pictures show the plots related to the analysis.





Being important the statistical descriptions of the features, it is presented via ***x.describe().T.***

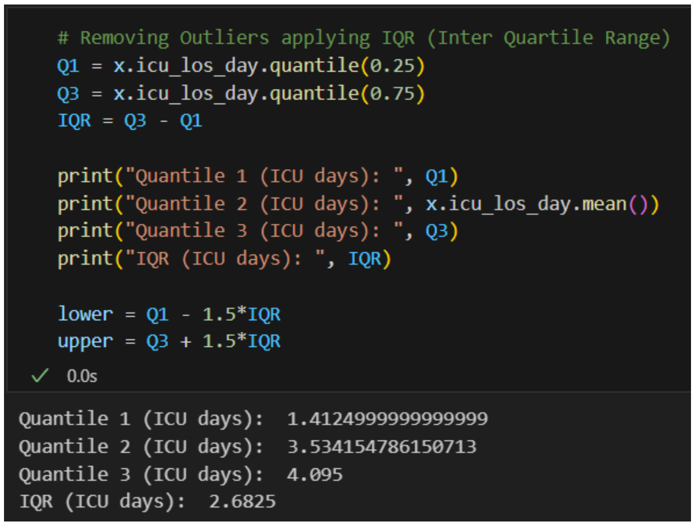


	count	mean	std	min	25%	50%	75%	max
icu_los_day	982.0	3.534155	3.669347	0.50000	1.412500	2.295000	4.095000	28.240000
age	982.0	53.144155	21.407871	15.19046	35.547540	52.277470	72.525320	96.454240
weight_first	982.0	75.667546	26.059355	0.00000	64.150000	76.900000	89.000000	211.000000
bmi	982.0	18.620344	13.557910	0.00000	0.000000	22.898715	28.206258	53.174533
map_1st	982.0	88.792940	17.188050	5.00000	77.666702	88.000000	99.666702	153.332993
hr_1st	982.0	85.759674	18.256784	33.00000	73.000000	85.000000	97.750000	153.000000
temp_1st	982.0	97.580957	5.940972	0.00000	96.800003	98.099998	99.300003	103.599998
spo2_1st	982.0	98.947047	4.653021	4.00000	99.000000	100.000000	100.000000	100.000000
abg_count	982.0	7.133401	10.039551	0.00000	1.000000	4.000000	9.000000	115.000000
wbc_first	982.0	12.707963	6.078140	0.00000	8.625000	11.700000	15.600000	86.000000
hgb_first	982.0	12.744603	2.231337	0.00000	11.400000	12.900000	14.375000	18.200000
platelet_first	982.0	244.015275	91.448301	0.00000	185.000000	237.000000	295.750000	681.000000
sodium_first	982.0	139.598778	7.467355	0.00000	138.000000	140.000000	142.000000	157.000000
potassium_first	982.0	4.000611	0.746135	0.00000	3.600000	3.900000	4.300000	9.200000
tco2_first	982.0	24.018228	4.205075	0.00000	22.000000	24.000000	27.000000	40.000000
chloride_first	982.0	104.328921	7.021357	0.00000	101.000000	105.000000	107.000000	133.000000
bun_first	982.0	16.809572	9.826600	0.00000	11.000000	15.000000	20.000000	139.000000
creatinine_first	982.0	0.996741	0.693166	0.00000	0.700000	0.900000	1.100000	9.400000
po2_first	982.0	221.572301	153.155829	0.00000	104.250000	198.000000	325.750000	634.000000
pco2_first	982.0	36.917515	14.891737	0.00000	34.000000	39.000000	44.000000	89.000000
iv_day_1	982.0	1624.942100	1654.865304	0.00000	260.329155	1134.629944	2573.405883	10360.624020

Apart from the process to observe outliers and distributions, I needed to observe whether the data contains null or duplicated values. For fulfilling this step, I used the ***notnull()*** and ***duplicated()*** functions which are included in Panda's library.

Cleaning and Preprocessing data

In this section I showed the procedure to identify and remove outliers in the data, I did it per each variable depending on the plot that I got in the previous section. To identify the outliers, I applied the IQR concept (Inter Quartile Range) to get the lower and upper bound for setting the data limits.

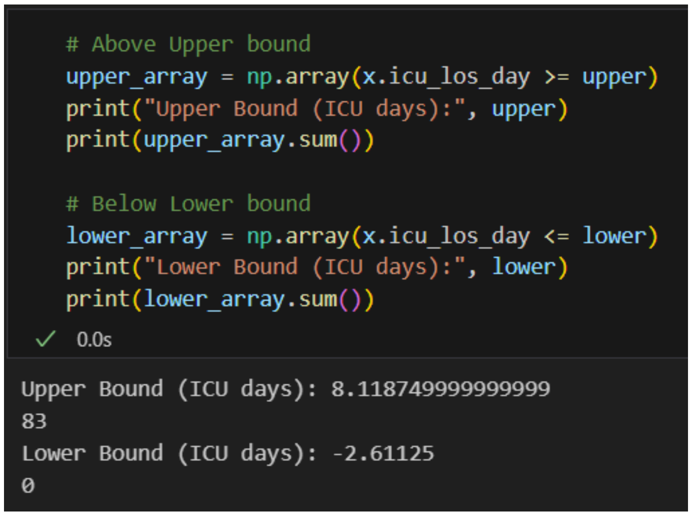


```python
# Removing Outliers applying IQR (Inter Quartile Range)
Q1 = x.icu_los_day.quantile(0.25)
Q3 = x.icu_los_day.quantile(0.75)
IQR = Q3 - Q1

print("Quantile 1 (ICU days): ", Q1)
print("Quantile 2 (ICU days): ", x.icu_los_day.mean())
print("Quantile 3 (ICU days): ", Q3)
print("IQR (ICU days): ", IQR)

lower = Q1 - 1.5*IQR
upper = Q3 + 1.5*IQR
```

```
✓ 0.0s
```

```
Quantile 1 (ICU days):  1.4124999999999999
Quantile 2 (ICU days):  3.534154786150713
Quantile 3 (ICU days):  4.095
IQR (ICU days):  2.6825
```

I presented the lower and upper bounds to obtain the quantity of outliers that I needed to remove on the top and bottom.

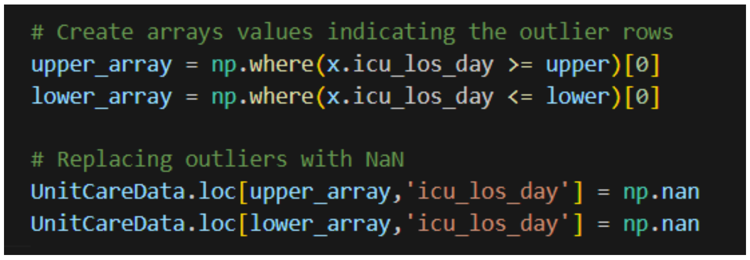


```python
# Above Upper bound
upper_array = np.array(x.icu_los_day >= upper)
print("Upper Bound (ICU days):", upper)
print(upper_array.sum())

# Below Lower bound
lower_array = np.array(x.icu_los_day <= lower)
print("Lower Bound (ICU days):", lower)
print(lower_array.sum())
```

```
✓ 0.0s
```

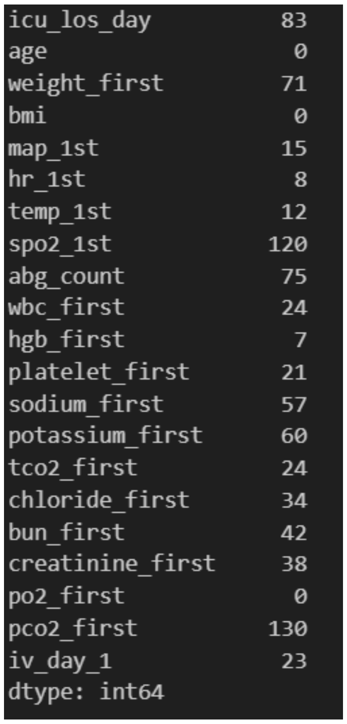
```
Upper Bound (ICU days): 8.118749999999999
83
Lower Bound (ICU days): -2.61125
0
```

Once I knew the quantity of data that I needed to remove, the next step was finding the outliers to replace them by null values. I decided to replace the outliers by null value because of practicity.



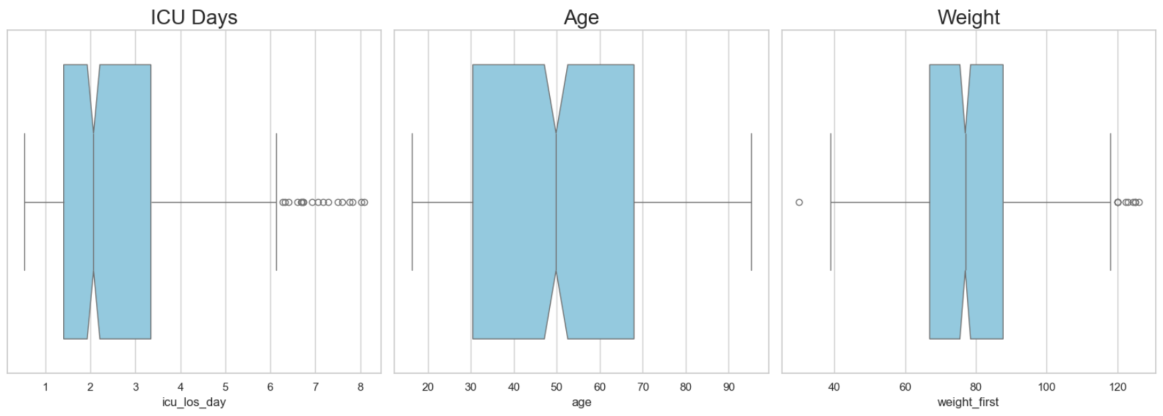
```python
# Create arrays values indicating the outlier rows
upper_array = np.where(x.icu_los_day >= upper)[0]
lower_array = np.where(x.icu_los_day <= lower)[0]

# Replacing outliers with NaN
UnitCareData.loc[upper_array,'icu_los_day'] = np.nan
UnitCareData.loc[lower_array,'icu_los_day'] = np.nan
```

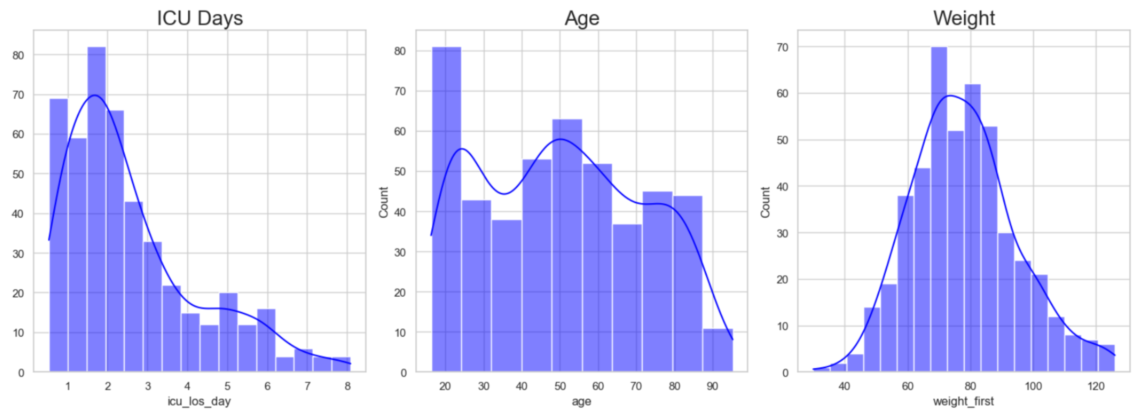
After done the IQR process for all the variables, I checked the result through the ***isnull()*** and ***sum()*** functions to realise how many data I needed to remove from the dataset.



```
icu_los_day          83
age                   0
weight_first         71
bmi                   0
map_1st              15
hr_1st                8
temp_1st             12
spo2_1st            120
abg_count            75
wbc_first            24
hgb_first             7
platelet_first       21
sodium_first         57
potassium_first      60
tco2_first           24
chloride_first       34
bun_first            42
creatinine_first     38
po2_first             0
pco2_first          130
iv_day_1             23
dtype: int64
```

For removing the outliers, I dropped the null value from the dataset and started to plot the new features with non-outliers. The visualisation was with boxplot and histograms.

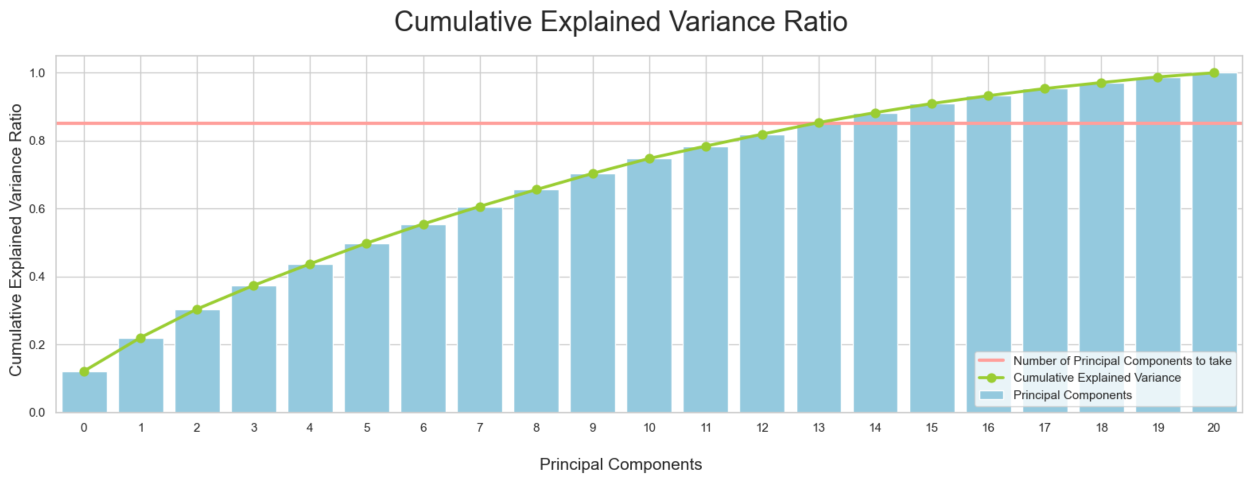




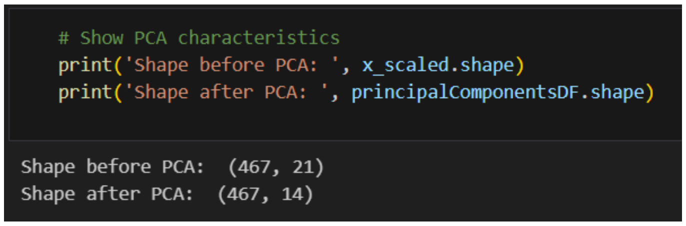
Principal Component Analysis

Principal Component Analysis is affected by scale, so it is vital to scale the features in the dataset before applying PCA. I applied ***StandardScaler()*** function from Scikit-Learn package, this function helps me to scale the dataset's features onto unit scale (mean = 0 and variance = 1), which is a requirement for the best performance of many machine learning algorithms.

The scaled process was given by ***x_scaled = StandardScaler().fit_transform(x).*** Once I had the scaled data, the next step was examined the quantity of principal components that I needed to pick for the analysis; setting my PCA target in 85% to preserve the information, the cumulative summation of the explained variance ratio showed me how many PCA I needed to take in the project.



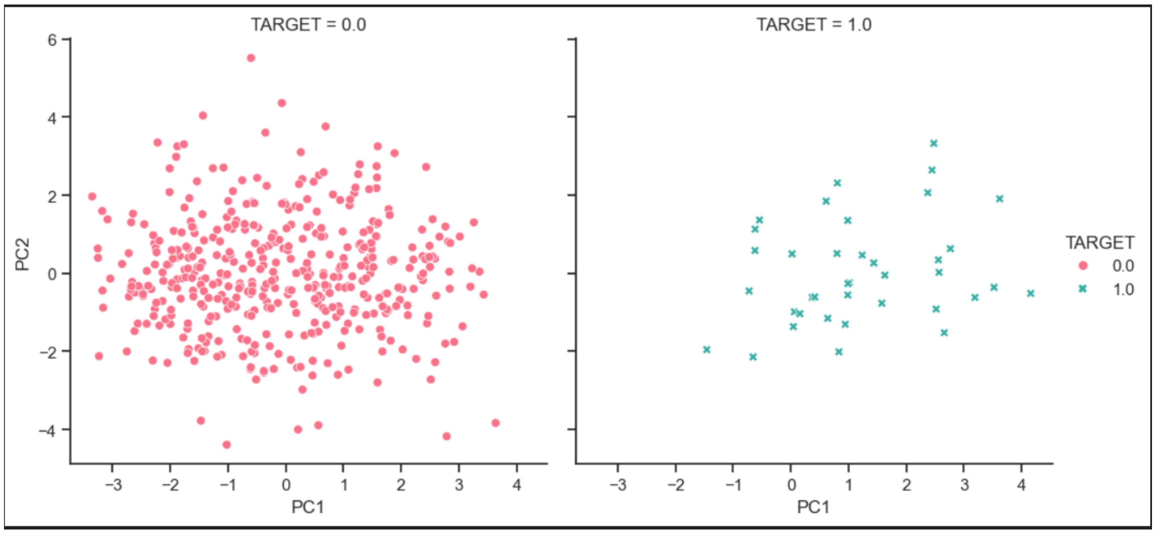
The explained variance ratio graph presents 14 principal components which retain the 85% of the variance in my dataset. Applying PCA technique with 14 principal components, I presented the new shape of the dataset.



```
# Show PCA characteristics
print('Shape before PCA: ', x_scaled.shape)
print('Shape after PCA: ', principalComponentsDF.shape)
```

```
Shape before PCA:  (467, 21)
Shape after PCA:  (467, 14)
```

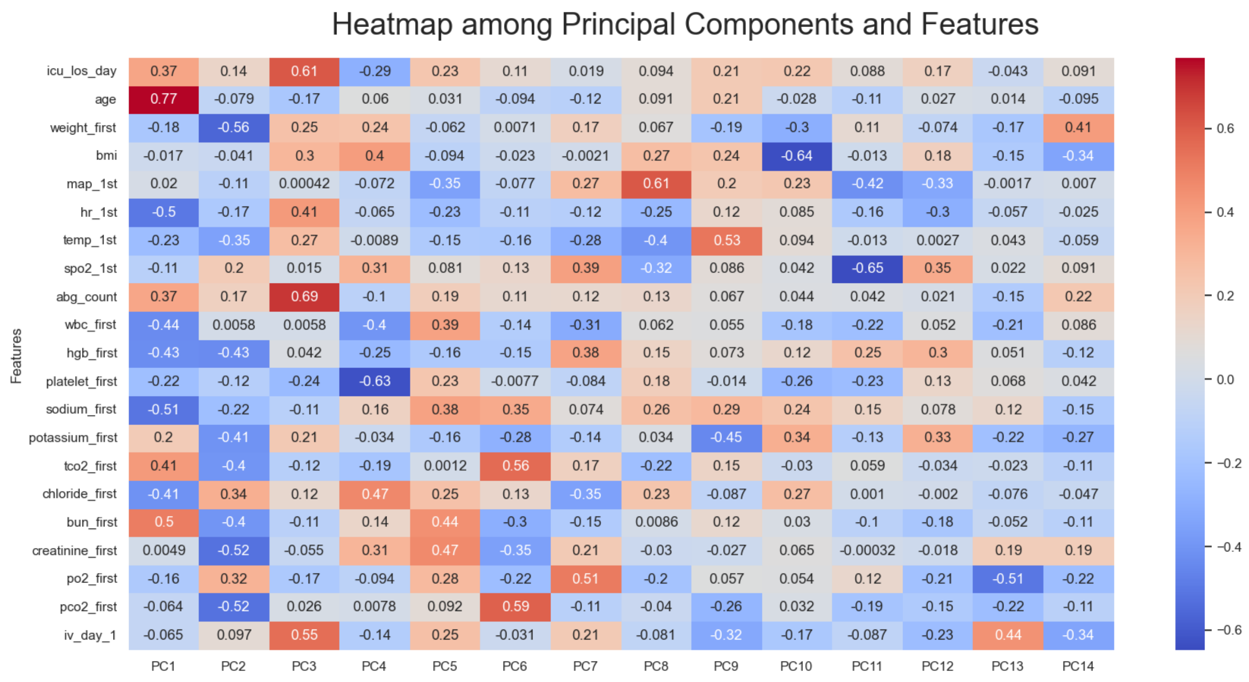
Due to the dimensionality, I showed the first and second principal components just to have an idea how the behaviour is.



Nevertheless, one of the most important graphs after implementing PCA technique is knowing which is the contribution of the original variables to the principal components. By concept, *“the PCA Loadings are the coefficients of the linear combination of the original variables from which the principal components are constructed.”*

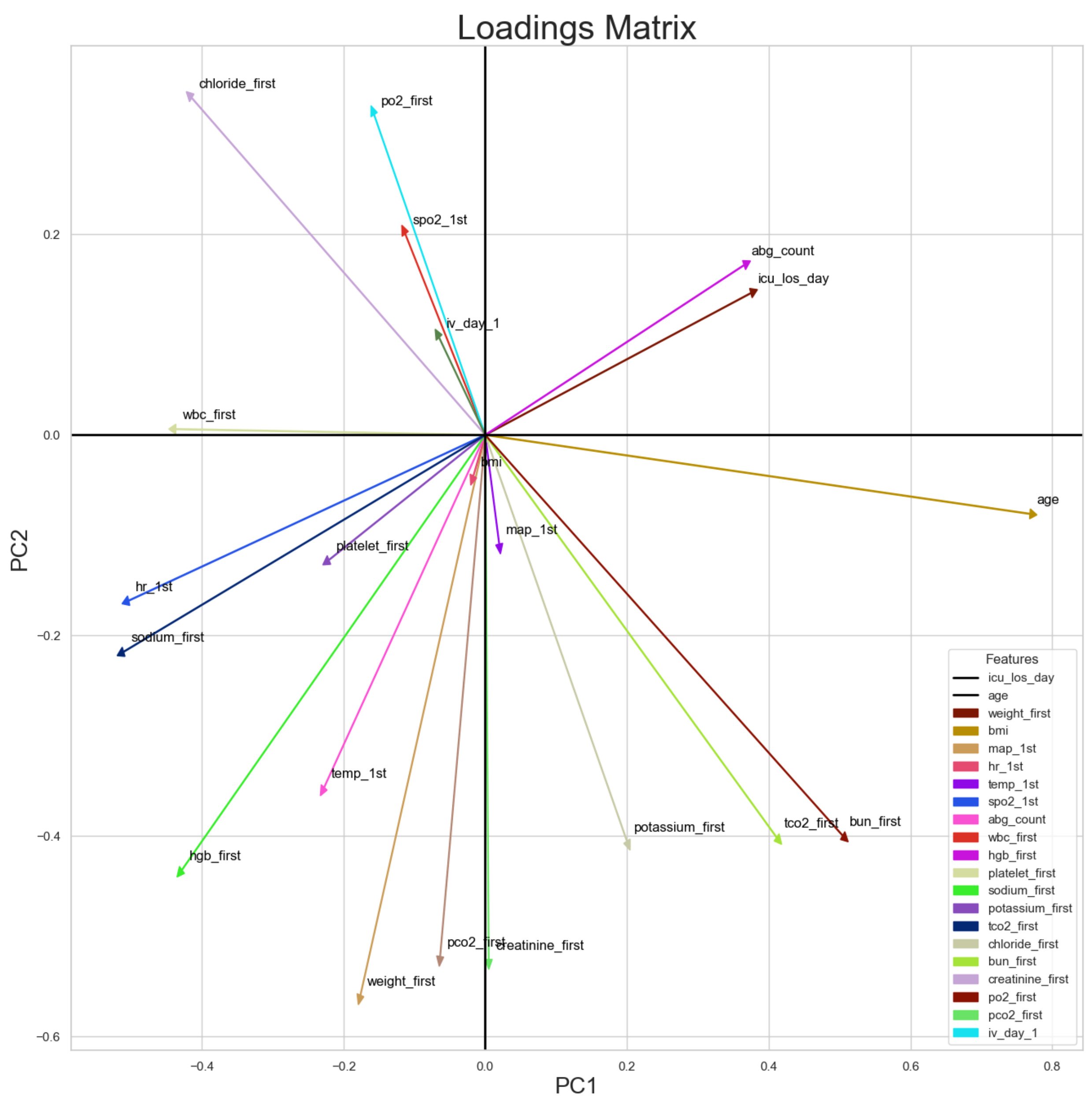
Constructing the Loading matrix, we can determine what the contribution is per each principal components in relation to the features. The positive or negative value determine the direction of the relationship, on the other hand the higher absolute values determine the contribution to the principal component. This matrix is important due to observe the useful variable to pick for the logistic regression model that the project required to build.

Via heatmap graph, we can see the highest and lowest correlation among the PCs and features.



Features	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13	PC14
icu_los_day	0.37	0.14	0.61	-0.29	0.23	0.11	0.019	0.094	0.21	0.22	0.088	0.17	-0.043	0.091
age	0.77	-0.079	-0.17	0.06	0.031	-0.094	-0.12	0.091	0.21	-0.028	-0.11	0.027	0.014	-0.095
weight_first	-0.18	-0.56	0.25	0.24	-0.062	0.0071	0.17	0.067	-0.19	-0.3	0.11	-0.074	-0.17	0.41
bmi	-0.017	-0.041	0.3	0.4	-0.094	-0.023	-0.0021	0.27	0.24	-0.64	-0.013	0.18	-0.15	-0.34
map_1st	0.02	-0.11	0.00042	-0.072	-0.35	-0.077	0.27	0.61	0.2	0.23	-0.42	-0.33	-0.0017	0.007
hr_1st	-0.5	-0.17	0.41	-0.065	-0.23	-0.11	-0.12	-0.25	0.12	0.085	-0.16	-0.3	-0.057	-0.025
temp_1st	-0.23	-0.35	0.27	-0.0089	-0.15	-0.16	-0.28	-0.4	0.53	0.094	-0.013	0.0027	0.043	-0.059
spo2_1st	-0.11	0.2	0.015	0.31	0.081	0.13	0.39	-0.32	0.086	0.042	-0.65	0.35	0.022	0.091
abg_count	0.37	0.17	0.69	-0.1	0.19	0.11	0.12	0.13	0.067	0.044	0.042	0.021	-0.15	0.22
wbc_first	-0.44	0.0058	0.0058	-0.4	0.39	-0.14	-0.31	0.062	0.055	-0.18	-0.22	0.052	-0.21	0.086
hgb_first	-0.43	-0.43	0.042	-0.25	-0.16	-0.15	0.38	0.15	0.073	0.12	0.25	0.3	0.051	-0.12
platelet_first	-0.22	-0.12	-0.24	-0.63	0.23	-0.0077	-0.084	0.18	-0.014	-0.26	-0.23	0.13	0.068	0.042
sodium_first	-0.51	-0.22	-0.11	0.16	0.38	0.35	0.074	0.26	0.29	0.24	0.15	0.078	0.12	-0.15
potassium_first	0.2	-0.41	0.21	-0.034	-0.16	-0.28	-0.14	0.034	-0.45	0.34	-0.13	0.33	-0.22	-0.27
tco2_first	0.41	-0.4	-0.12	-0.19	0.0012	0.56	0.17	-0.22	0.15	-0.03	0.059	-0.034	-0.023	-0.11
chloride_first	-0.41	0.34	0.12	0.47	0.25	0.13	-0.35	0.23	-0.087	0.27	0.001	-0.002	-0.076	-0.047
bun_first	0.5	-0.4	-0.11	0.14	0.44	-0.3	-0.15	0.0086	0.12	0.03	-0.1	-0.18	-0.052	-0.11
creatinine_first	0.0049	-0.52	-0.055	0.31	0.47	-0.35	0.21	-0.03	-0.027	0.065	-0.00032	-0.018	0.19	0.19
po2_first	-0.16	0.32	-0.17	-0.094	0.28	-0.22	0.51	-0.2	0.057	0.054	0.12	-0.21	-0.51	-0.22
pco2_first	-0.064	-0.52	0.026	0.0078	0.092	0.59	-0.11	-0.04	-0.26	0.032	-0.19	-0.15	-0.22	-0.11
iv_day_1	-0.065	0.097	0.55	-0.14	0.25	-0.031	0.21	-0.081	-0.32	-0.17	-0.087	-0.23	0.44	-0.34

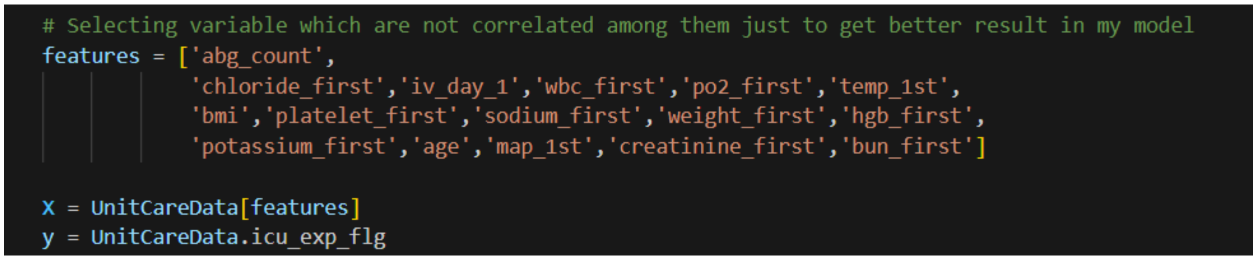
Plotting the Loading Matrix in Cartesian coordinate system, permits to observe the magnitudes and directions of each variable which can analyse and decide the best group of variables to take for model to train.



For instance, we observe the variable ***icu_los_day*** and ***abg_count*** with similar magnitude and direction, so in this case the best option is taking one of them to not skew the model. For training models in Machine Learning always we need to keep an eye in the quality of the variables to take in the same way the quantity and how much value these variables will provide to the model.

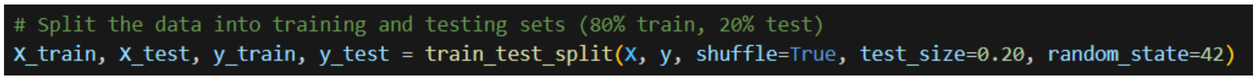
Logistic Regression

Once analysed the variable to pick for the model, I separated them from the dataset and from the target. For the project, the target was the ***icu_exp_flg*** variable which represents in binary values whether a patient died or not in ICU. I performed the logistic regression model due to the target contains binary values which this model is suitable for the case.



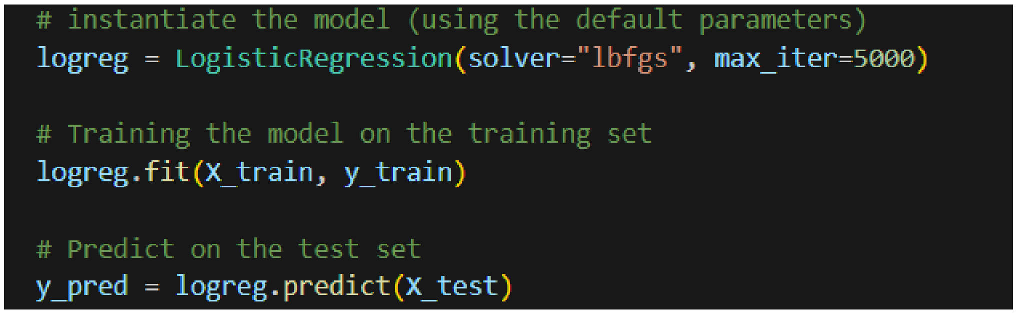
```
# Selecting variable which are not correlated among them just to get better result in my model
features = ['abg_count',
            'chloride_first','iv_day_1','wbc_first','po2_first','temp_1st',
            'bmi','platelet_first','sodium_first','weight_first','hgb_first',
            'potassium_first','age','map_1st','creatinine_first','bun_first']

X = UnitCareData[features]
y = UnitCareData.icu_exp_flg
```

After clarifying the features, the next phase is set the train and test data which I split it in 20% from the data dedicated for model testing and 80% for model training.



```
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True, test_size=0.20, random_state=42)
```

As part of the process, it is vital to normalise the data. To avoid the different range of value in the features I normalised them using the ***StandardScaler()*** function from Scikit-Learn. With the scaled data the model was launched to train and predict.

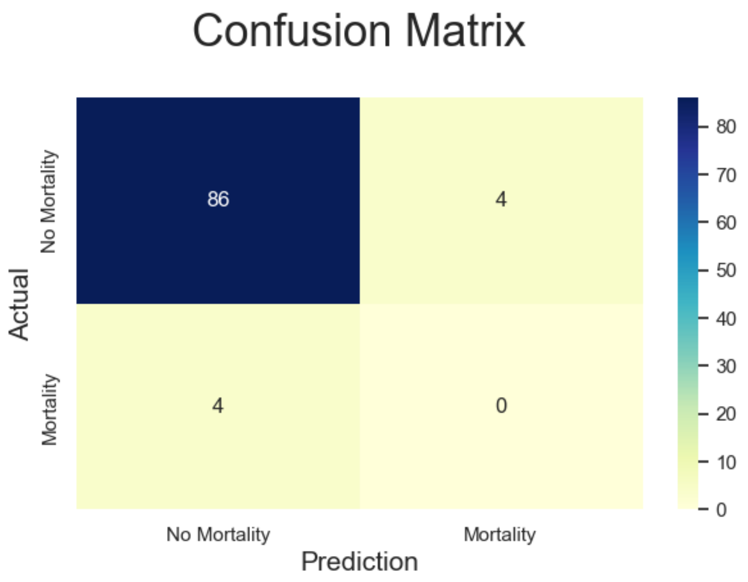


```
# instantiate the model (using the default parameters)
logreg = LogisticRegression(solver="lbfgs", max_iter=5000)

# Training the model on the training set
logreg.fit(X_train, y_train)

# Predict on the test set
y_pred = logreg.predict(X_test)
```

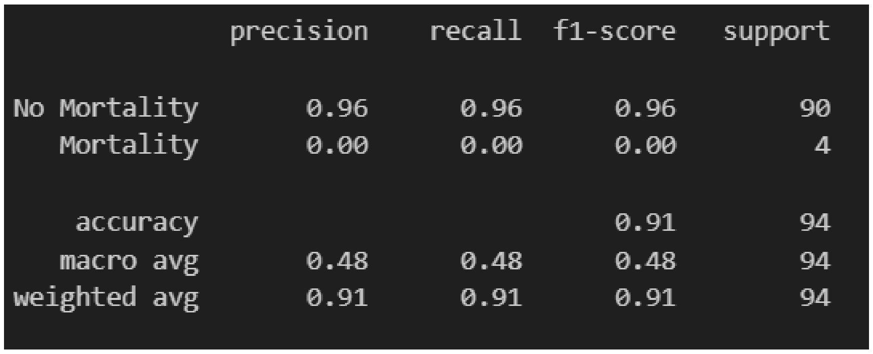
I defined the ***max_iter*** (max iterations) parameter in 5000 to improve the model in order to get more accuracy.

Evaluating the model

To evaluate the model, I presented the accuracy score on the test given by 91.489% and the accuracy score on train set given by 92.493%, following the confusion matrix.



The confusion matrix plot gave me insights to understand how the model performed, for my case I noticed the model was good predicting the no mortality rate however was weak predicting the mortality rate. This data is supported by the classification report where can obtain more details related to the model where I focused on the precision and recall value.



```
              precision    recall  f1-score   support

No Mortality       0.96      0.96      0.96        90
   Mortality       0.00      0.00      0.00         4

    accuracy                           0.91        94
   macro avg       0.48      0.48      0.48        94
weighted avg       0.91      0.91      0.91        94
```

This matrix showed 96% precision for predicting ***no mortality*** in the test model which concerns with the quality of positive predictions, on the other hand with the same value we found the recall which concerns with the quantity of the relevant instances captured by the model. In simple words, we can say the higher precision and recall value, the lowest rate of false positives the model will have. It was different for the mortality variable which was not accurate for the model.

Conclusion

To sum up this project permitted me to apply and connect all the statistics and data mining concepts in a practical way which it gave me a big picture of the lifecycle of Data Mining, to elaborate questions, do evaluations, work in the data, and get insights. I took this project as a part of the analytical skills development, the importance of understanding every phase is key to improve our criteria in the moment to do an analysis or examination case.

References

MIMIC II Databases (physionet.org)

How to compute PCA loadings and the loading matrix with scikit-learn - Simone Centellegher, PhD - Data scientist and Researcher (scentellegher.github.io)

Python Machine Learning - Logistic Regression (w3schools.com)

Precision-Recall Curve | ML - GeeksforGeeks

Author: Manuel Alonso Bernabé Baldano