# Web programming project report

**LAI Khang Duy**
**Zilu YANG**
**KLIMINA Mariia**
**Nadia KACEM CHAOUCHE**

*Université de Paris*
*UFR des Sciences Fondamentales et Biomédicales*

Université de Paris

28-03-2022

## Contents

## 1 Introduction

With the development of the data era, data science plays an increasingly important role in our daily life, so a good data exploration tool is especially important. As the final project for this course, we created a web application for visualizing existing datasets, exploratory data analysis, and training different models. It was done using the Streamlit framework. Streamlit is an open-source app framework for Machine Learning and Data Science teams and is a very powerful python library.

In this report, there are five parts: Introduction, Application Architecture, Implementation, Problems encountered, Demonstrations, and Conclusion.

## 2 Application Architecture

Overall, the web application is divided into five parts: Side Bar, Data Visualization, Data pre-processing, Models, and Evaluation. They are distinct from each other but at the same time interconnected. To give you a clearer picture of our applications, we will show you the architecture of the web application here:

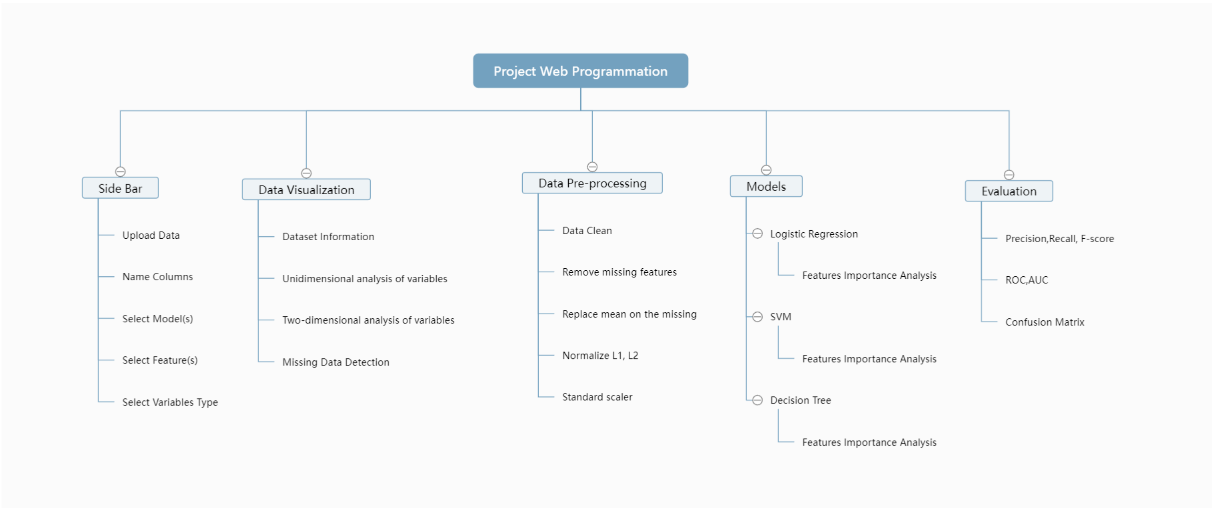


**Figure 1:** System architechture.

## 3 Implementation

In this section, **the implementation of the application will be explained** in terms of the five different parts mentioned in the previous section.

### 3.1 Sidebar

The sidebar is a very crucial part because it collects all kinds of information from the user and it is connected to every other part of the website. In general, the Streamlit library was used for this project, as it contains almost all the necessary functions for this part.

The collected information from the user are:

- The Data file uploader.
- The column's names input.
- Training models (Logistic regression, SVM, and Decision tree).
- The features (with a multiselection option).
- The quantitative variables.
- The qualitative variables.

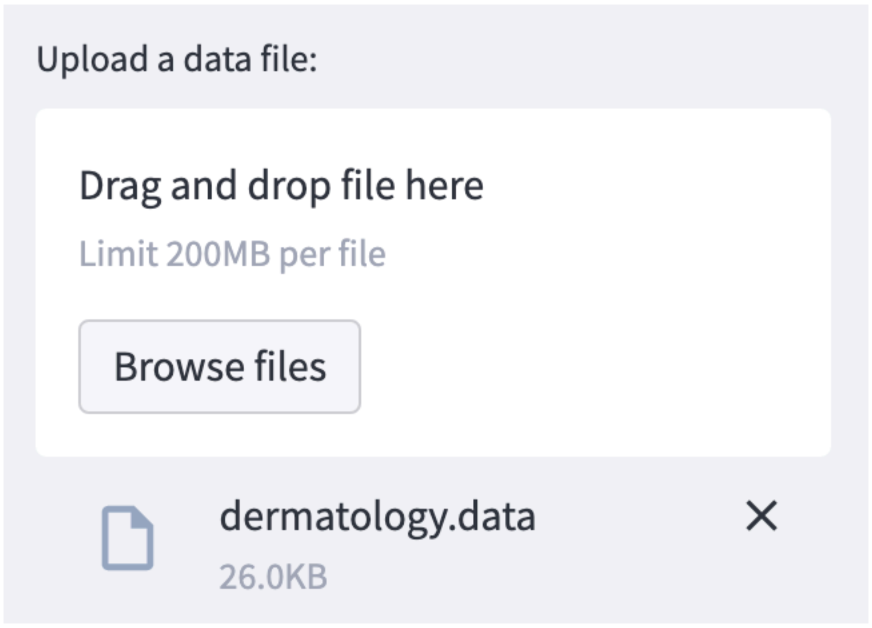
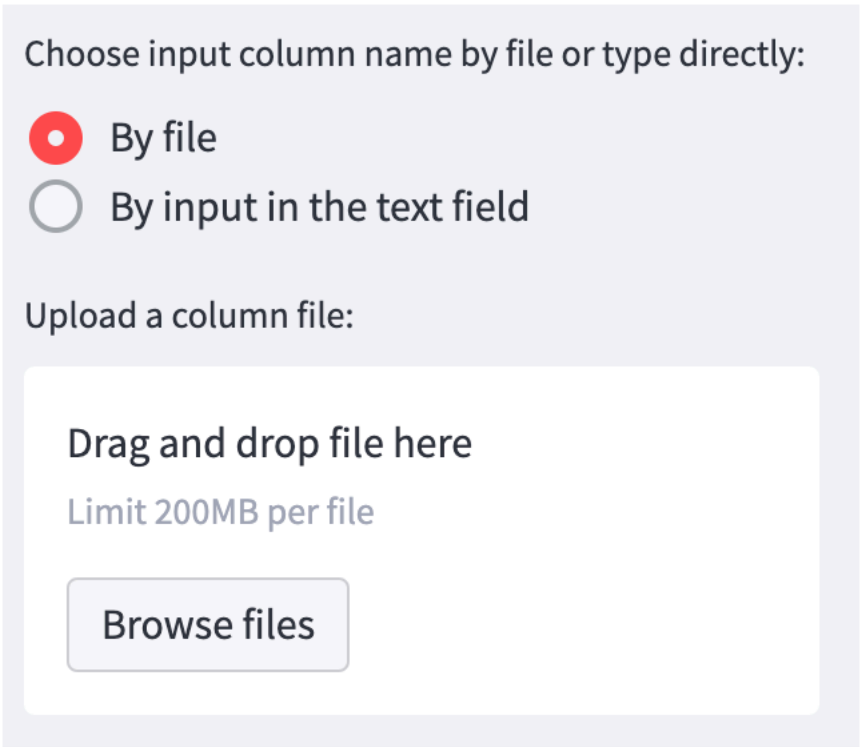


**Figure 2:** The Data file uploader.

At the end of this function, a dictionary is generated, and it contains all kinds of inputs that were received from the user. Then this dictionary is used by all the other functionalities of the site, as

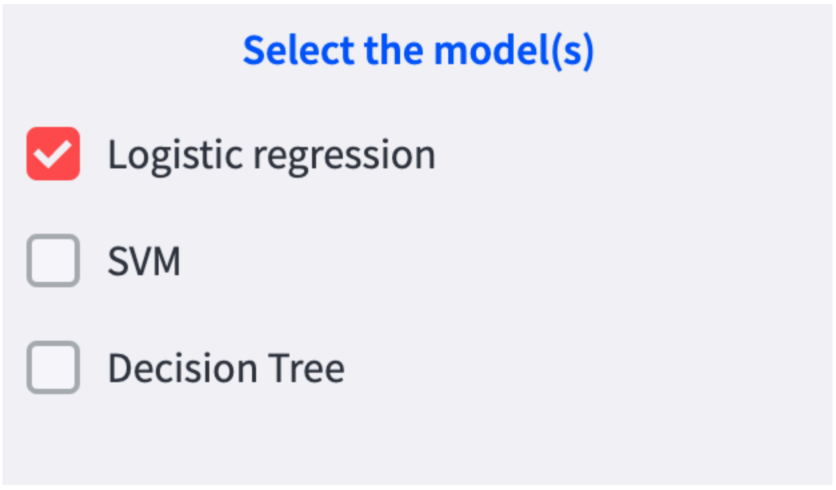

Choose input column name by file or type directly:

[✓ By file]
By input in the text field

Upload a column file:

Drag and drop file here
Limit 200MB per file
Browse files

**Figure 3:** The column's names input.


Select the model(s)

[✓ Logistic regression]
SVM
Decision Tree

**Figure 4:** Training models input.

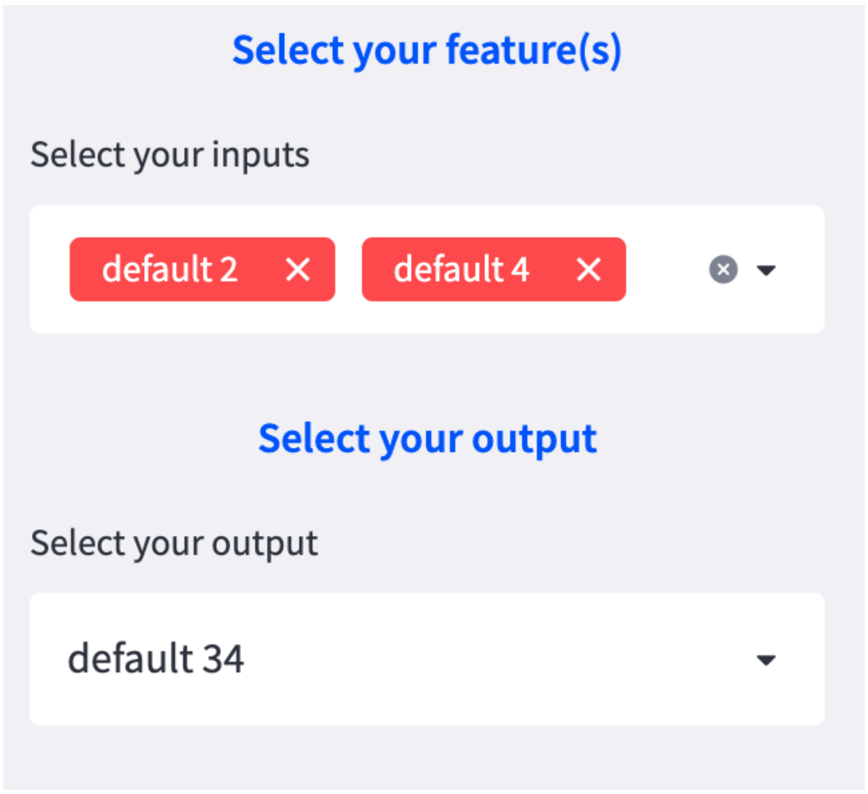


**Figure 5:** The features.

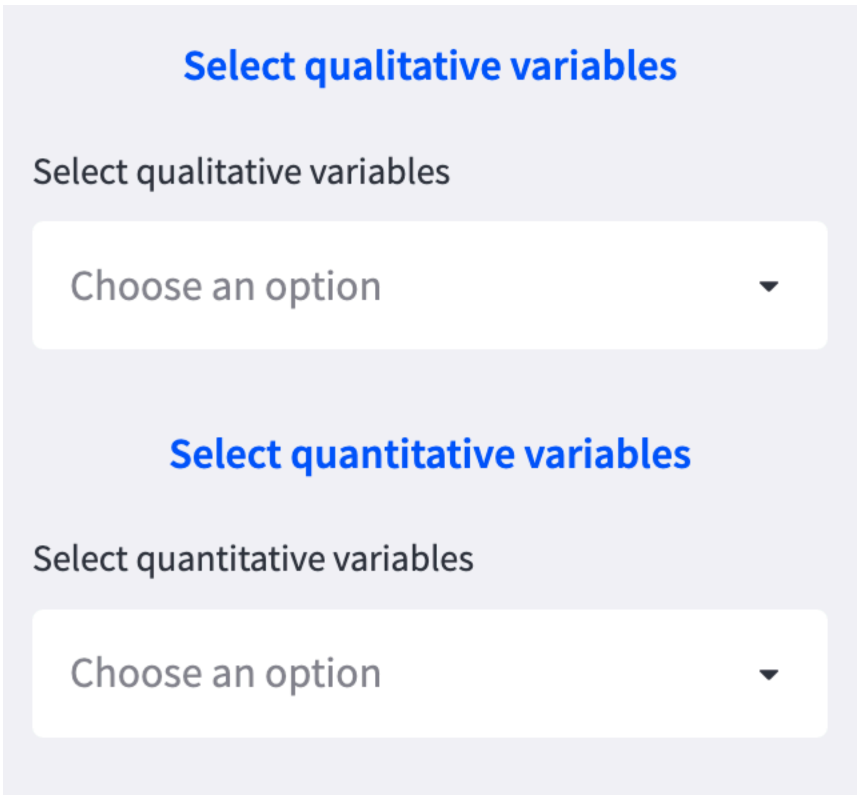

Select qualitative variables
Select qualitative variables
Choose an option
Select quantitative variables
Select quantitative variables
Choose an option

**Figure 6:** Quantitative variables and qualitative variables.

it's considered a very important tool for the continuity of the execution.

## 3.2 Data visualization

After obtaining the dataset from the sidebar, we can start visualizing the data and performing basic EDA analysis. As shown in the architecture, we have four parts, and first, we would like to explain their functions.

- Dataset Information

This part presents the dataset simply and straightforwardly by giving the data frame and the user has three ways to view the dataset: the head of the dataset, the tail of the dataset, and the entire dataset. In addition, the user can view the dimensions of the dataset.

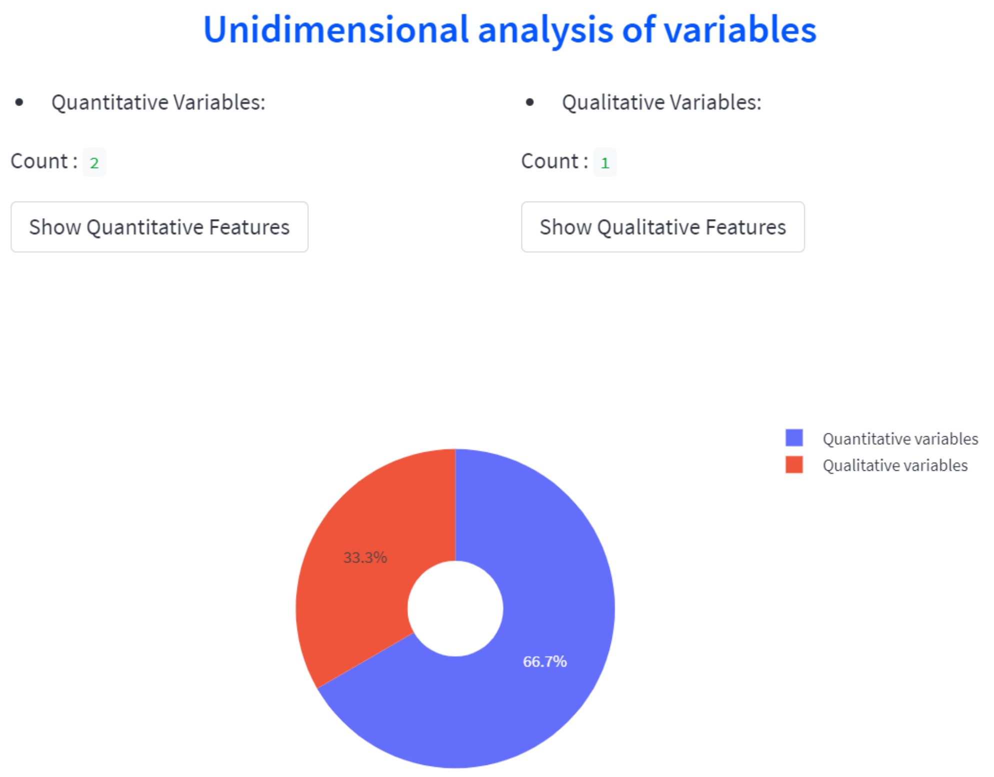
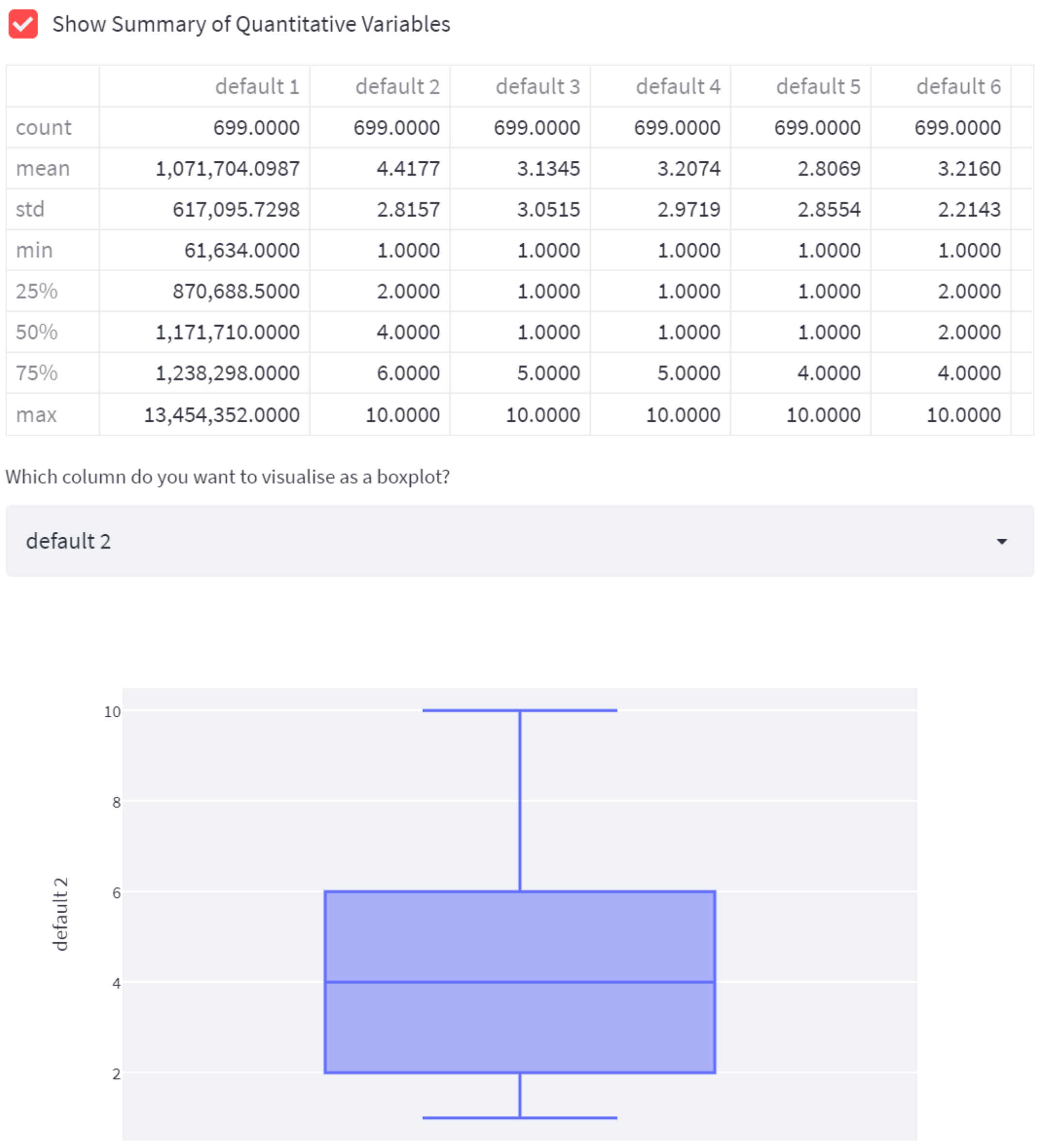


**Figure 7:** Unidimensional analysis 1.

- Unidimensional analysis of variables

In this section, we analyze the dataset in terms of both qualitative and quantitative variables. We first give the percentages of each type of variable so that the user can have a more intuitive view



[✓] Show Summary of Quantitative Variables

| | default 1 | default 2 | default 3 | default 4 | default 5 | default 6 |
|---|---|---|---|---|---|---|
| count | 699.0000 | 699.0000 | 699.0000 | 699.0000 | 699.0000 | 699.0000 |
| mean | 1,071,704.0987 | 4.4177 | 3.1345 | 3.2074 | 2.8069 | 3.2160 |
| std | 617,095.7298 | 2.8157 | 3.0515 | 2.9719 | 2.8554 | 2.2143 |
| min | 61,634.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| 25% | 870,688.5000 | 2.0000 | 1.0000 | 1.0000 | 1.0000 | 2.0000 |
| 50% | 1,171,710.0000 | 4.0000 | 1.0000 | 1.0000 | 1.0000 | 2.0000 |
| 75% | 1,238,298.0000 | 6.0000 | 5.0000 | 5.0000 | 4.0000 | 4.0000 |
| max | 13,454,352.0000 | 10.0000 | 10.0000 | 10.0000 | 10.0000 | 10.0000 |

Which column do you want to visualise as a boxplot?

default 2

**Figure 8:** Unidimensional analysis 2.

of them, and then we summarize the quantitative variables, in particular, giving basic information such as counts, means, standard deviations with a data frame, and the boxplots, etc. As you can see in the following figure.

- Two-dimensional analysis of variables

A scatter plot showing the relationship between any two selected variables is presented here. In this scatter plot, the user can view multiple X variables in relation to the same Y variable at the same time.

- Missing Data Detection

After exploring the nature and relationships of the different variables, we are now interested in the problems that exist in the dataset. In this application, we detect the missing values and display the columns of missing values by heatmap.

In terms of the tools used, in addition to the basic tools of Streamlit, we also introduced the library Plotly to visualize this data.

## 3.3 Data preprocessing

After getting the data from the sidebar, the data preprocessing module will process the data before feeding it into the model's module.

The preprocessing module contains:

- NaN handler module.

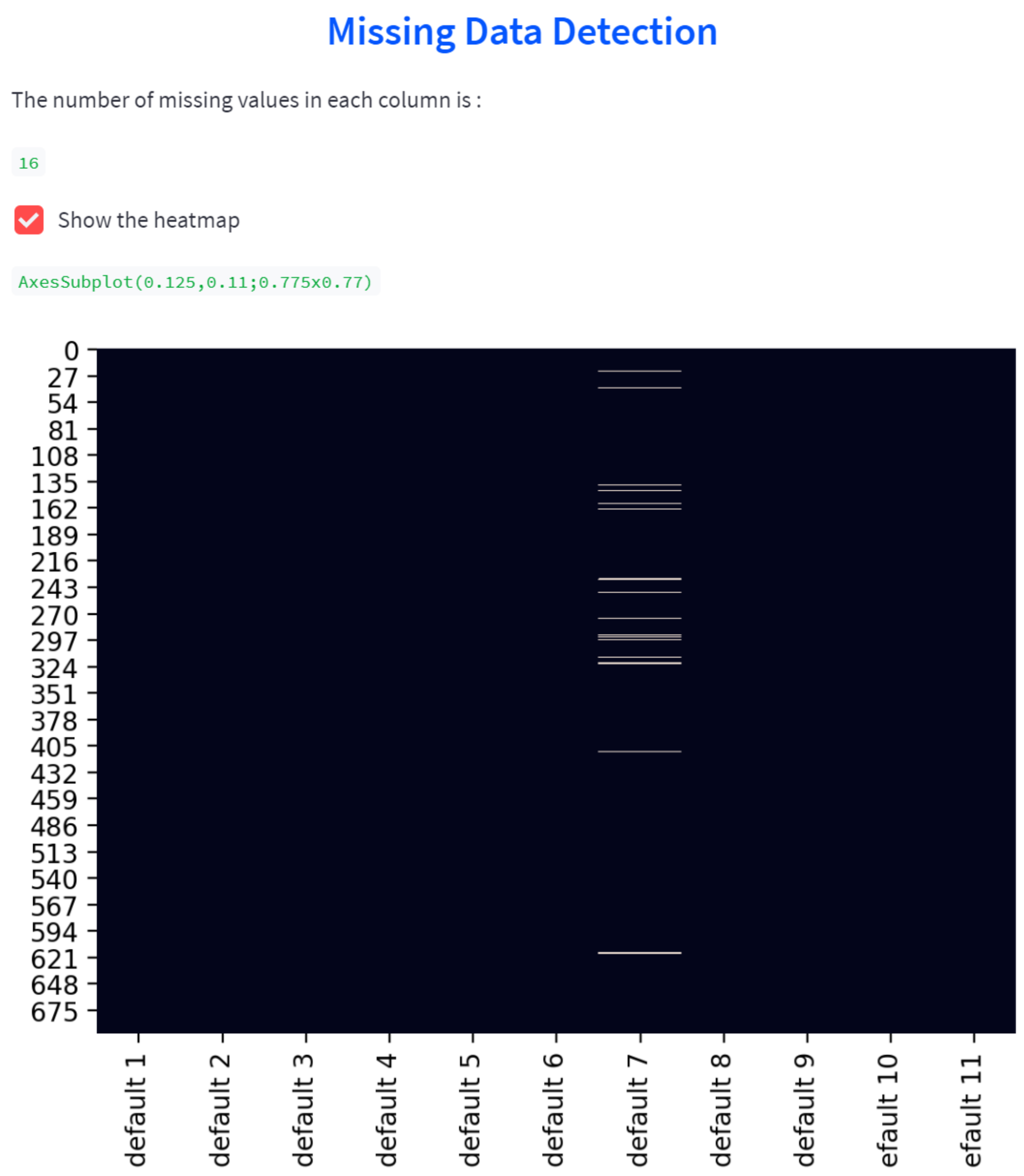
NaN handler is an important part of preprocessing. We can't feed the model with a dataframe containing NaN value. In this application, we included 2 methods to eliminate the NaN value before the next step.

Users can choose between removing the NaN rows or replacing NaN values with the mean of the column.

- Normalize:

In general, in most of the datasets, there will be a variety of unbalance between features. In order to solve it, we created a check box for fixing this problem. For example, we have 2 features, 1 have the range of 0 and 10 and one have the range of 10000 and 900000000. It makes the dataset unbalanced and could affect the outcome. We use normalization to eliminate that problem.

Users can choose several types of normalization and balance their data.



Missing Data Detection

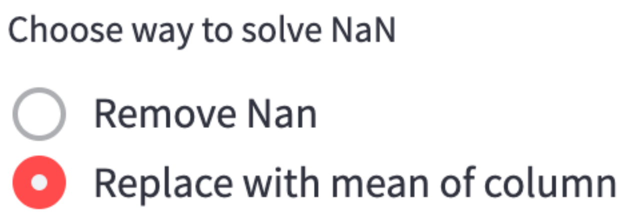
The number of missing values in each column is :

```
16
```

[✓ Show the heatmap]

```
AxesSubplot(0.125,0.11;0.775x0.77)
```

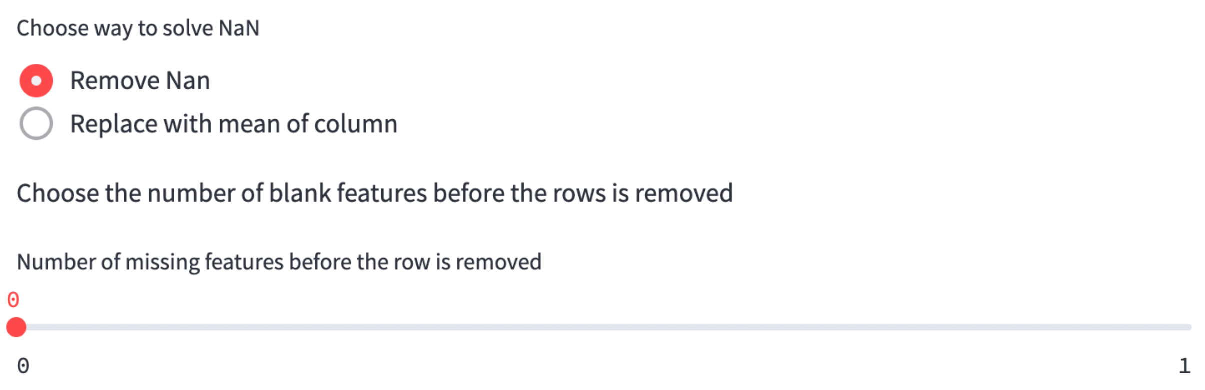
**Figure 9:** Missing data heatmap.



Choose way to solve NaN

Remove Nan [✓ Replace with mean of column]

**Figure 10:** Remove NaN 1.


Choose way to solve NaN

[✓ Remove Nan]
Replace with mean of column

Choose the number of blank features before the rows is removed

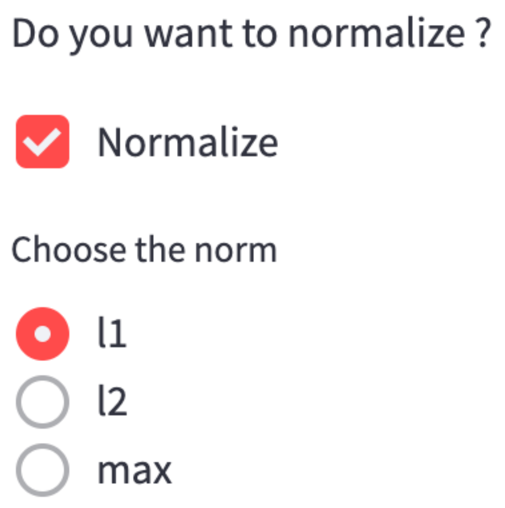
Number of missing features before the row is removed

0

0 1

**Figure 11:** Remove NaN 2.

- L1
- L2

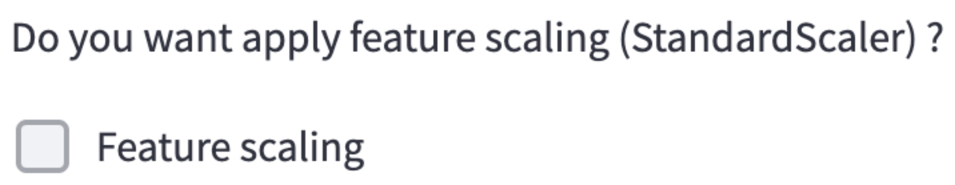

Do you want to normalize ?

[✓ Normalize]

Choose the norm

[✓ l1]
l2
max

- Max
- Feature scaling


Do you want apply feature scaling (StandardScaler) ?

Feature scaling

**Figure 12:** Remove NaN 2.

This module uses the StandardScaler function on the sklearn library.

### 3.4 Models

For the model training part three models were used:

- Logistic Regression
- Supporting Vector Machine
- Decision Tree

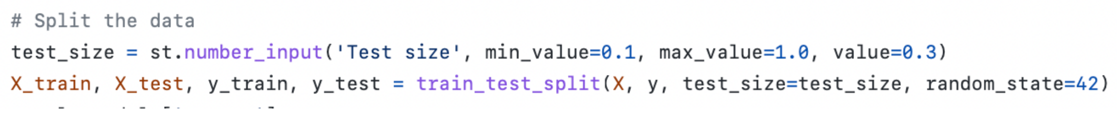
The libraries that were imported:

- sklearn – for training models
- streamlit – for printing the outputs

The model part is divided into 3 parts: creating a function, fitting the models, and calculating predictions.

First of all, the function model_select was created. At this point, we are using the cleaned data from the file clean.py. Then, we split the data.

1. Test,train,split

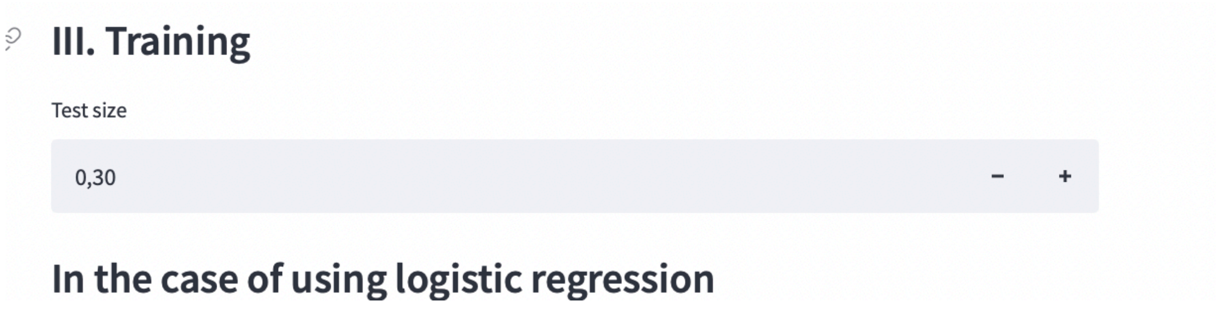


```
# Split the data
test_size = st.number_input('Test size', min_value=0.1, max_value=1.0, value=0.3)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=42)
```

**Figure 13:** Test, train, split.

When it comes to splitting the data, the user will have an opportunity to choose the test size sample. The default will be 30%.

From the user's side it looks like this:



III. Training

Test size

0,30

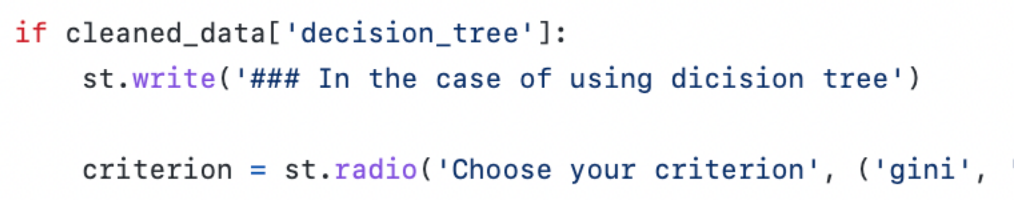
In the case of using logistic regression

**Figure 14:** Input testset size.

2. At the beginning of the code, we set up the dictionary reslut_models. It will be returned at the end of the function. What is the inside of a dictionary? We added all prediction scores,

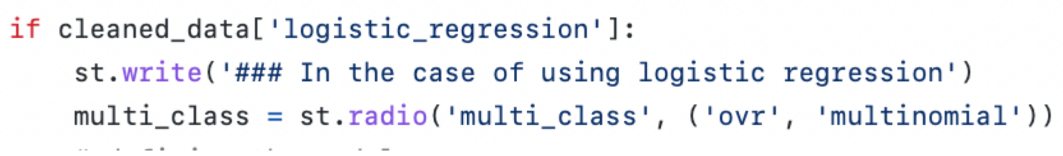
y sample, y_test sample, X_train,y_train sample, and X_test sample. This dictionary will be used for evaluation purposes.

3. After the data is defined and split, we have created 3 conditions for each model: Decision Tree, SVM, and Logistic Regression. With each model, the user can precise the type and the criteria. It was done with st.radio option from the streamlit library.



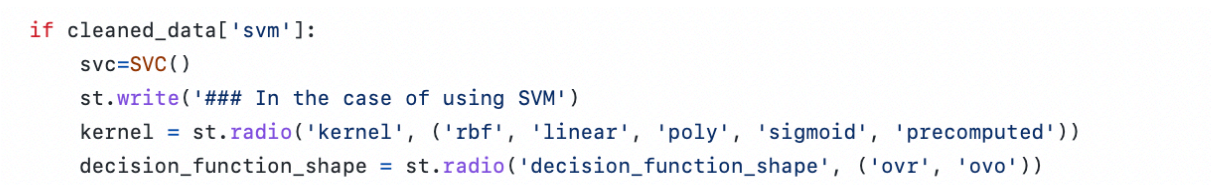
```
if cleaned_data['decision_tree']:
    st.write('### In the case of using dicision tree')

    criterion = st.radio('Choose your criterion', ('gini',
```

**Figure 15:** Input testset size.



```
if cleaned_data['logistic_regression']:
    st.write('### In the case of using logistic regression')
    multi_class = st.radio('multi_class', ('ovr', 'multinomial'))
```

**Figure 16:** Input testset size.



```
if cleaned_data['svm']:
    svc=SVC()
    st.write('### In the case of using SVM')
    kernel = st.radio('kernel', ('rbf', 'linear', 'poly', 'sigmoid', 'precomputed'))
    decision_function_shape = st.radio('decision_function_shape', ('ovr', 'ovo'))
```

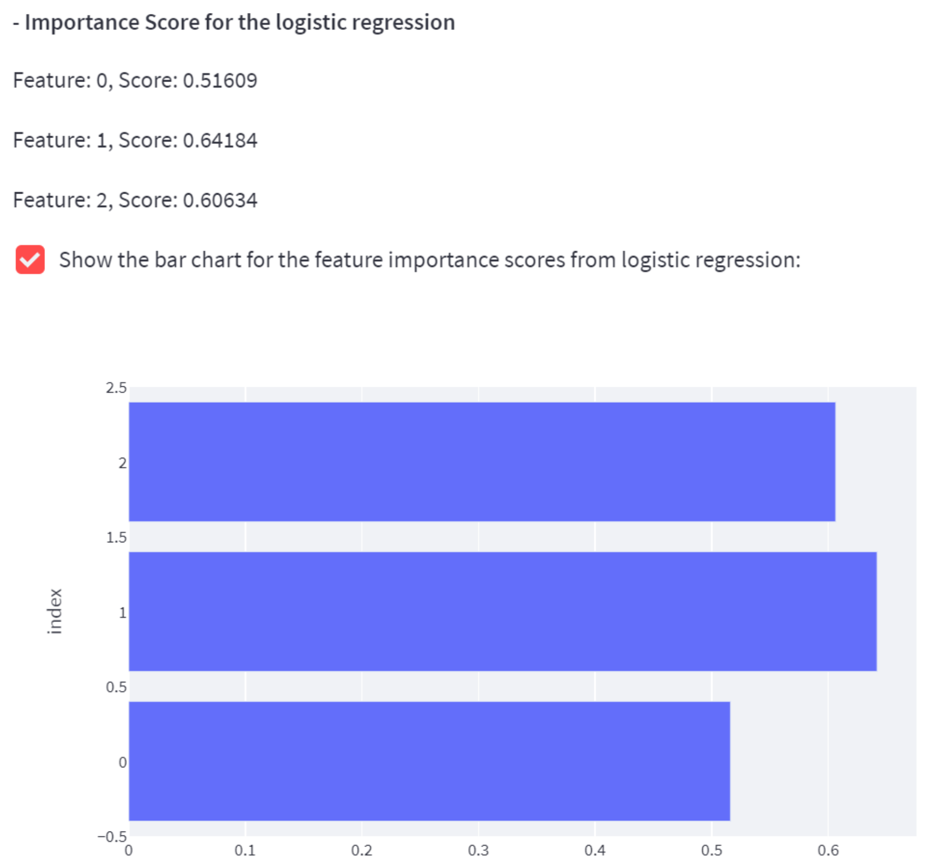
**Figure 17:** Input testset size.

4. When it comes to training models the algorithm is the same for each case. First, we define the model with the tools from the sklearn library and precise all the criteria and model options that we have already chosen. For example, in the case of Decision Tree we did it like that:

Then we train models, using the **fit** feature from the sklearn library. After models are trained, we call for the prediction score using the **predict** feature and the X_test sample. Overall, the whole training process is displayed in the image.

When we have the scores, it is crucial to upload them to the dictionary that we created at the beginning **“result_models”**. As was mentioned earlier, we will pass this dictionary to the evaluation process in order to get the scores such as recall, precision, f1 score, roc_auc_score, etc.

In addition, for each model, we also give the importance of the selected features for training and present them to the user by a bar chart. As shown in the figure below, here is an example of how we measure the important features in the logistic regression model:



- **Importance Score for the logistic regression**

```
Feature: 0, Score: 0.51609

Feature: 1, Score: 0.64184

Feature: 2, Score: 0.60634
```

✓ Show the bar chart for the feature importance scores from logistic regression:

**Figure 18:** Input testset size.

We can see that of the three features measured: feature 0, feature 1, and feature 2, feature 1 has the highest feature importance value of 0.64184, so this variable has a greater impact on model training than the other two. We can also visualize this result on the histogram.

To conclude, it is worth mentioning that we found that features importance is only discussed in the context of the linear kernel, so we do not perform features importance on SVM.

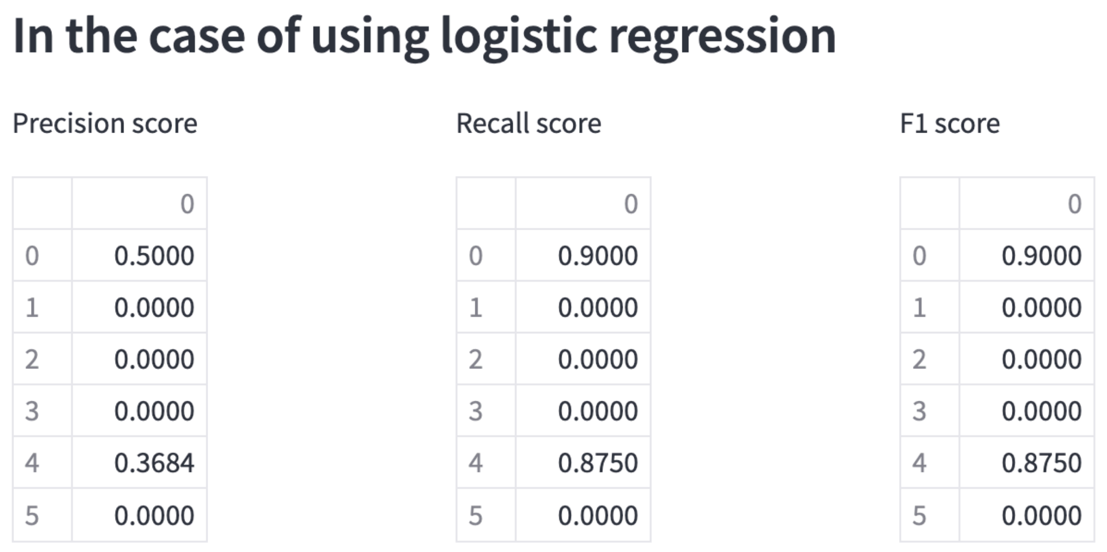
### 3.5 Evaluation

This module will handle the outcome of the model's module. It contains:

- Different types of scores:

We have

- Precision
- Recall
- F1 score



## In the case of using logistic regression

Precision score

| | 0 |
|---|---|
| 0 | 0.5000 |
| 1 | 0.0000 |
| 2 | 0.0000 |
| 3 | 0.0000 |
| 4 | 0.3684 |
| 5 | 0.0000 |

Recall score

| | 0 |
|---|---|
| 0 | 0.9000 |
| 1 | 0.0000 |
| 2 | 0.0000 |
| 3 | 0.0000 |
| 4 | 0.8750 |
| 5 | 0.0000 |

F1 score

| | 0 |
|---|---|
| 0 | 0.9000 |
| 1 | 0.0000 |
| 2 | 0.0000 |
| 3 | 0.0000 |
| 4 | 0.8750 |
| 5 | 0.0000 |

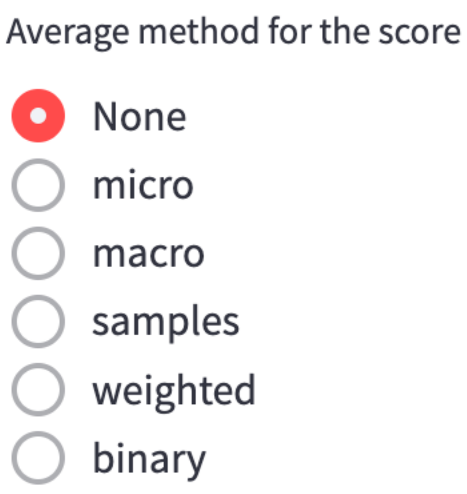
**Figure 19:** Score

Each model will have a different set of scores. If the user attempts to choose all the models, the function will calculate all the scores of all models.

Users can also choose which type of average method they want to use.

- Confusion matrix
- ROC AUC score
- ROC curve

These features for now only work with multi classes.


Average method for the score

[✓ None] micro macro samples weighted binary

**Figure 20:** Average method chooser

## 4 Problems encountered

### 4.1 Determining the nature of variables:

We found that in some datasets the original authors used numbers for qualitative variables, for example, 2 for benign and 4 for malignant in the cancer breast dataset, so it was difficult to determine whether the data was a qualitative or quantitative variable from the type of data itself.

*Solution*: We added an area in the sidebar where the user can select the type of variable so that the determination of the type of variable is more in line with the real use case.

### 4.2 The collection of sidebar information.

*Solution*: We use a dictionary to store the information selected by the user in the sidebar. Furthermore, we put the contents of the sidebar into a function that can be used in another section such as Data Visualization.

### 4.3 The Button ‘Done’ in the sidebar:

We faced a problem with the button being reloaded each time we added input and deleted the choices (initialization).

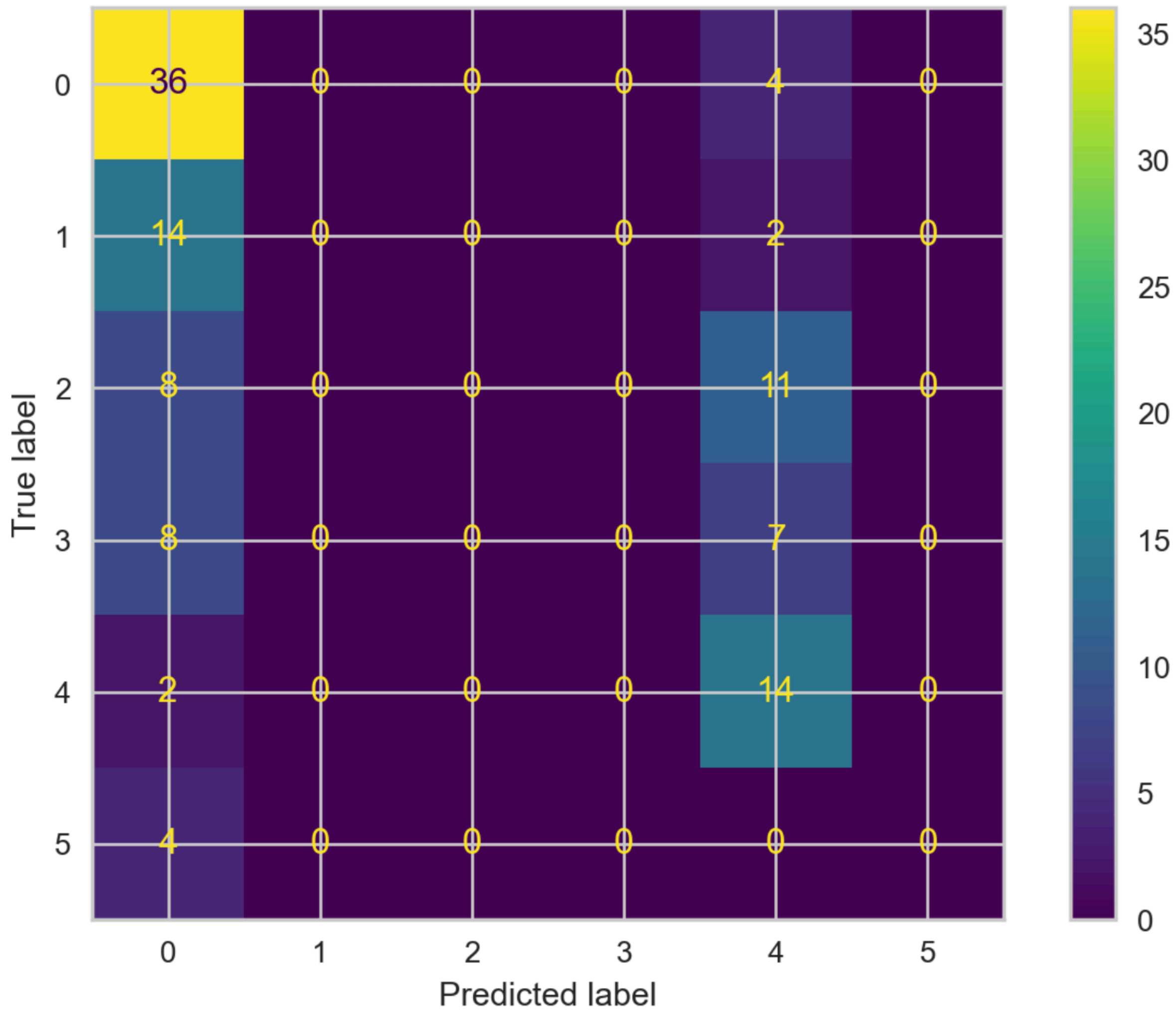


**Figure 21:** Confusion matrix

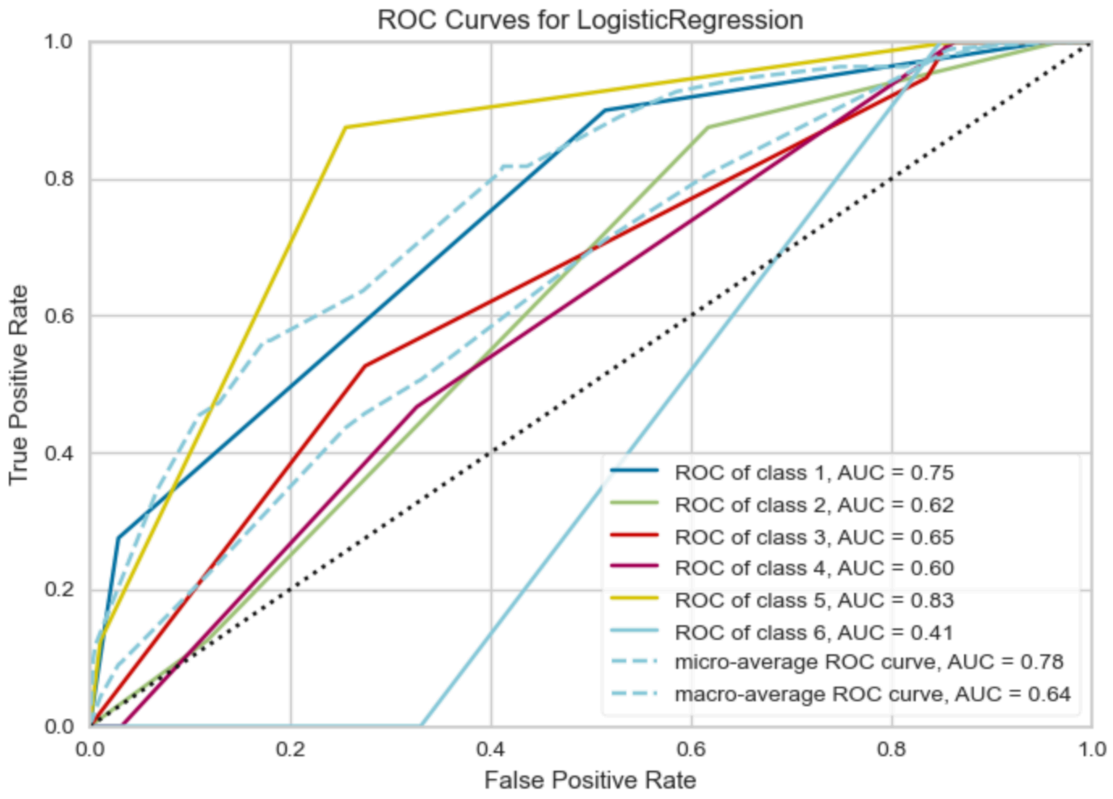


**Figure 22:** ROC Curve

*Solution*: After a long search, it turned out it was a streamlit problem, and a lot of programmers had it, so the solution was to replace it with a ‘checkbox’.

## 5 Conclusion. What could be done in the future?

This project focuses on constructing a web application using Streamlit in python. The web application made it easier and more intuitive for users to explore the dataset. In addition, by completing this project, we learned more about visualizing datasets, exploratory analysis, and using models for prediction and evaluation. In terms of group collaboration, we had good communication from deciding plans to assigning tasks to weekly meetings to discuss progress. Finally, in terms of improvement, we hope to add more algorithmic models in the future as well as give the user more freedom to personalize his settings.