Comments

Group : Marvine Porquet, François Lagadec

Organisation :

- Artificial dataset / Marvine Porquet
- Analisys / François Lagadec
- Visualization 1 and 2 / François Lagadec
- Supervised learning / Marvine Porquet

Source of the dataset:

https://www.kaggle.com/yasserh/wine-quality-dataset

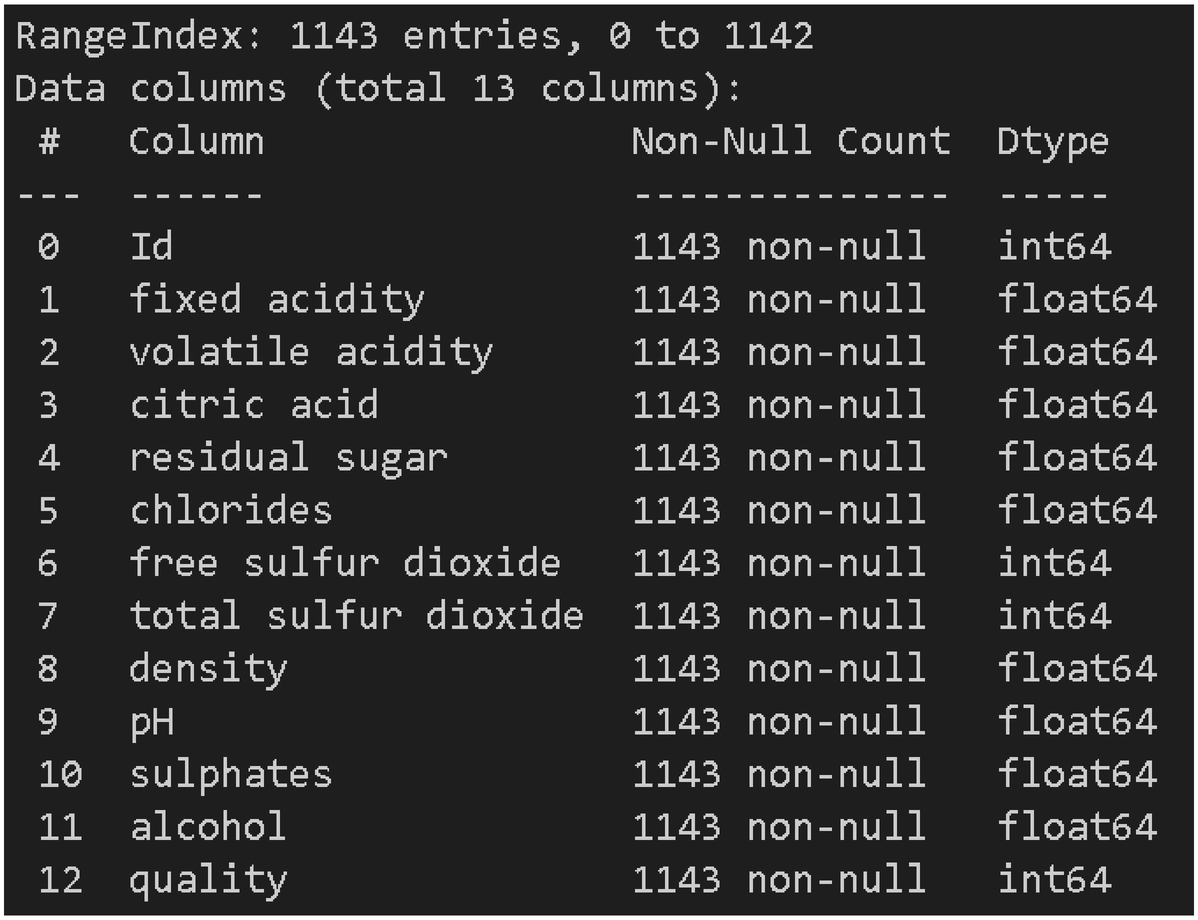
Description of the dataset:

This dataset is related to red variants of the Portuguese "Vinho Verde" wine. The dataset describes the amount of various chemicals present in wine and their effect on its quality.

Problematic:

Can we determine the quality of a red wine from its composition?

Overlook of the dataset:



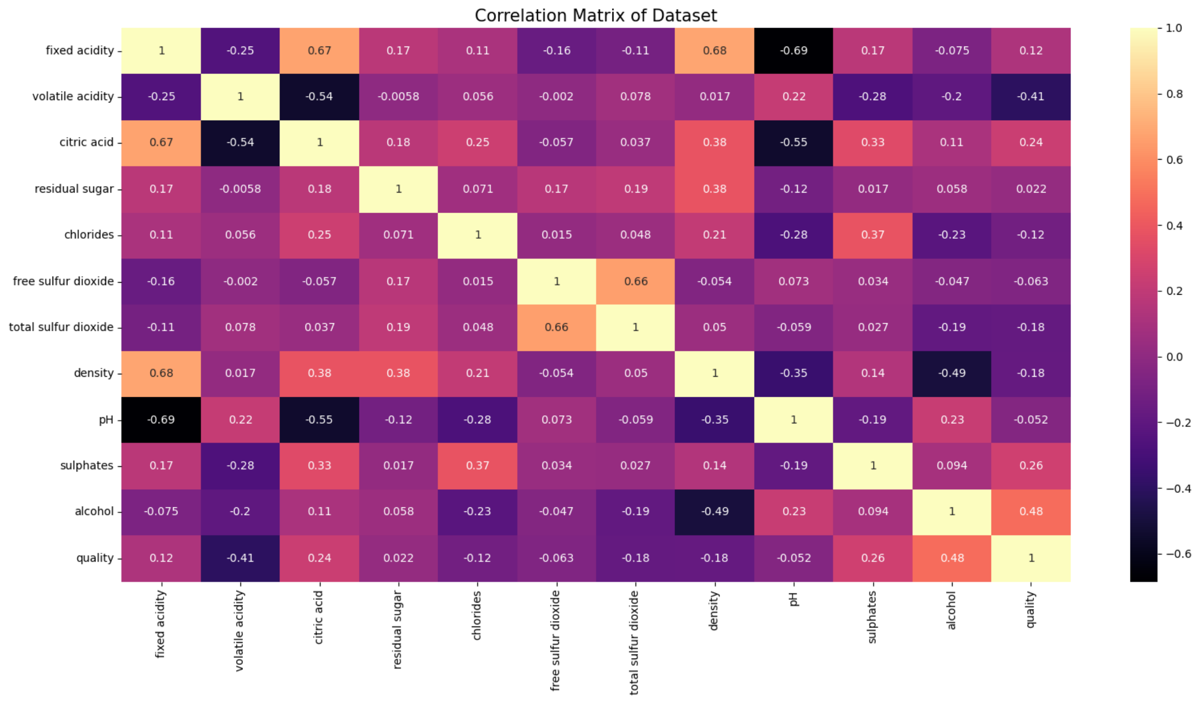
```
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Id                    1143 non-null   int64  
 1   fixed acidity         1143 non-null   float64
 2   volatile acidity      1143 non-null   float64
 3   citric acid           1143 non-null   float64
 4   residual sugar        1143 non-null   float64
 5   chlorides             1143 non-null   float64
 6   free sulfur dioxide   1143 non-null   int64  
 7   total sulfur dioxide  1143 non-null   int64  
 8   density               1143 non-null   float64
 9   pH                    1143 non-null   float64
 10  sulphates             1143 non-null   float64
 11  alcohol               1143 non-null   float64
 12  quality               1143 non-null   int64  
```

Our dataset is composed of 13 columns, including 9 containing floats and 4 containing integers.

We have 1143 entries with no null value.

Potential correlation:

We consider that there is a correlation if the coefficient is between 0.48 and 1. Or -1 and -0.48 for an inverse correlation.



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1	-0.25	0.67	0.17	0.11	-0.16	-0.11	0.68	-0.69	0.17	-0.075	0.12
volatile acidity	-0.25	1	-0.54	-0.0058	0.056	-0.002	0.078	0.017	0.22	-0.28	-0.2	-0.41
citric acid	0.67	-0.54	1	0.18	0.25	-0.057	0.037	0.38	-0.55	0.33	0.11	0.24
residual sugar	0.17	-0.0058	0.18	1	0.071	0.17	0.19	0.38	-0.12	0.017	0.058	0.022
chlorides	0.11	0.056	0.25	0.071	1	0.015	0.048	0.21	-0.28	0.37	-0.23	-0.12
free sulfur dioxide	-0.16	-0.002	-0.057	0.17	0.015	1	0.66	-0.054	0.073	0.034	-0.047	-0.063
total sulfur dioxide	-0.11	0.078	0.037	0.19	0.048	0.66	1	0.05	-0.059	0.027	-0.19	-0.18
density	0.68	0.017	0.38	0.38	0.21	-0.054	0.05	1	-0.35	0.14	-0.49	-0.18
pH	-0.69	0.22	-0.55	-0.12	-0.28	0.073	-0.059	-0.35	1	-0.19	0.23	-0.052
sulphates	0.17	-0.28	0.33	0.017	0.37	0.034	0.027	0.14	-0.19	1	0.094	0.26
alcohol	-0.075	-0.2	0.11	0.058	-0.23	-0.047	-0.19	-0.49	0.23	0.094	1	0.48
quality	0.12	-0.41	0.24	0.022	-0.12	-0.063	-0.18	-0.18	-0.052	0.26	0.48	1

If we rely on the matrix created by the analasys.py file, we can identify several cases of correlation:

- Positive correlation:

  - “Fixed acidity” and “Citric acid” = 0.67

  - “Fixed acidity” and “Density” = 0.68

  - “Free sulfur dioxide” and “Total sulfur dioxide” = 0.66

  - “Alcohol” and “Quality” = 0.48

- Negative correlation:

  - “Fixed acidity” and “pH” = -0.69

  - “Volatile acidity” and “Citric acid” = -0.54

  - “Citric acid” and “pH” = -0.55

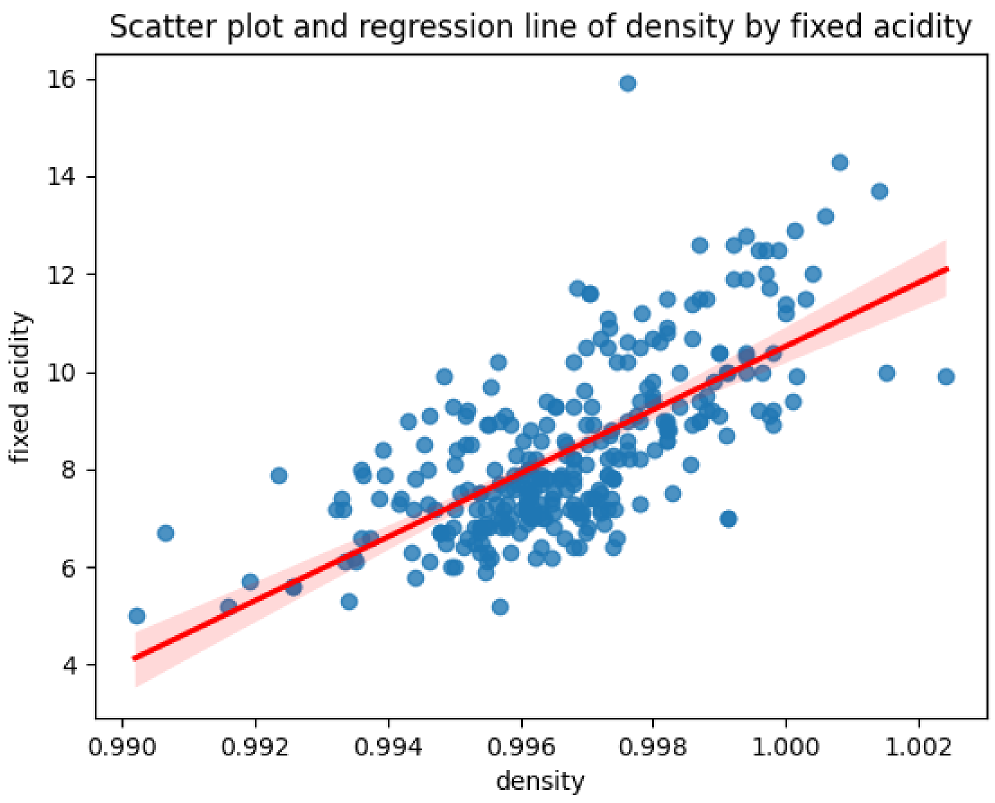
  - “Density” and “Alcohol” = -0.49

Visualization:

Visualization 1:

For the visualization part, we have chosen two representations. First one scatterplot (or pairplot if the user let the input blank). For this visualization, the user is invited to fill two fields for two columns of the dataset and one field to choose the range of point in the dataset. By default, we choose alcohol as first value, quality as second value and 1143 point in the dataset.

This type of visualization allows us to observe and understand the relations between our parameters, such as identifying correlations (see image below):

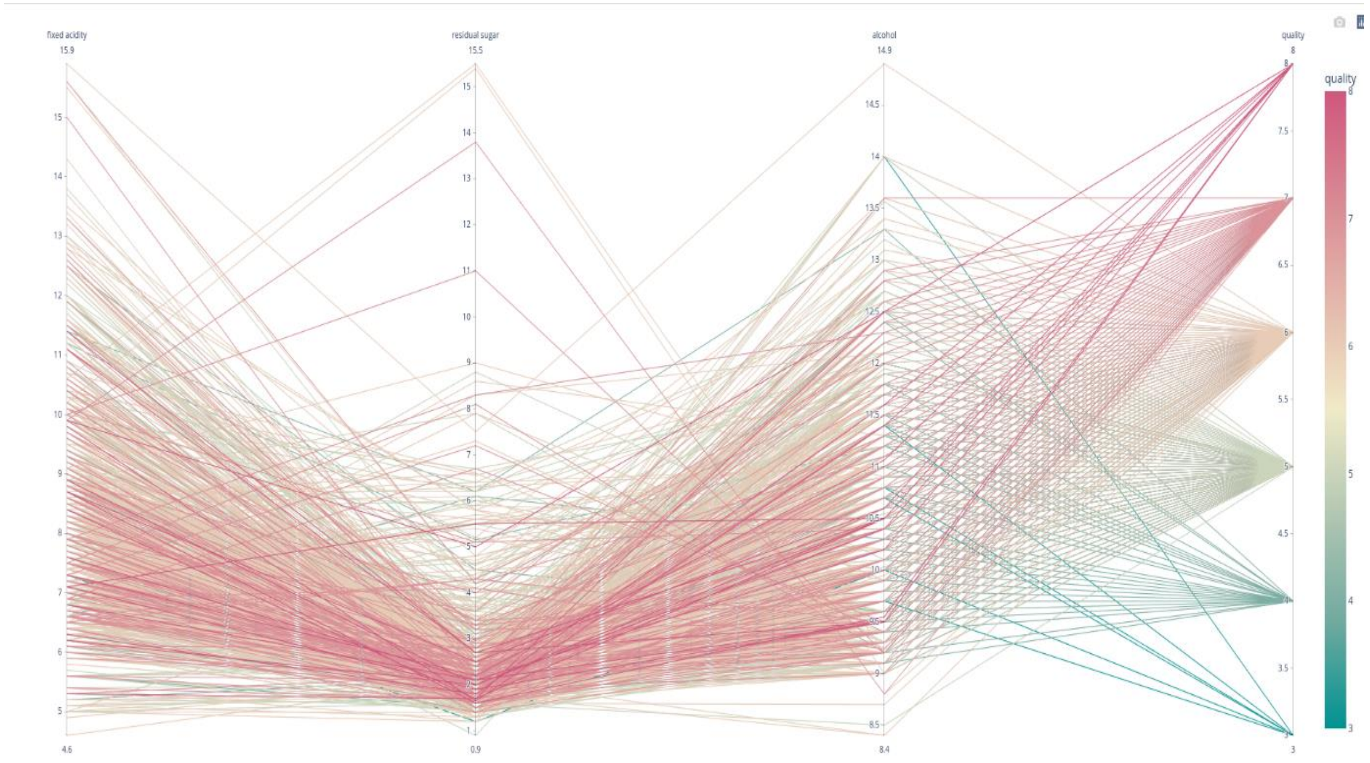


Visualization 2:

For the second visualization, we use parallel coordinates plot. We invited the user to fill three fields this time, to choose three columns in the dataset (to be represented according to the quality of wine) and one field to choose the range of point in the dataset. By default, we choose fixed acidity, residual sugar, alcohol and 1143 point in the dataset.

(Warning): This graphic only works on the google browser. You must use this browser to see this visualization.

With this visualization, this allows us to have a better visualization of the different aberrations of each column (see image below):



Surpervised learning, Logistic regression:

To answer our problematic "Can we determine the quality of a red wine from its composition?", we have decided to use a supervised learning method called logistic regression. We choose this model because we need a regression method that predict us ordinal outputs, it is an Ordinal logistic regression.

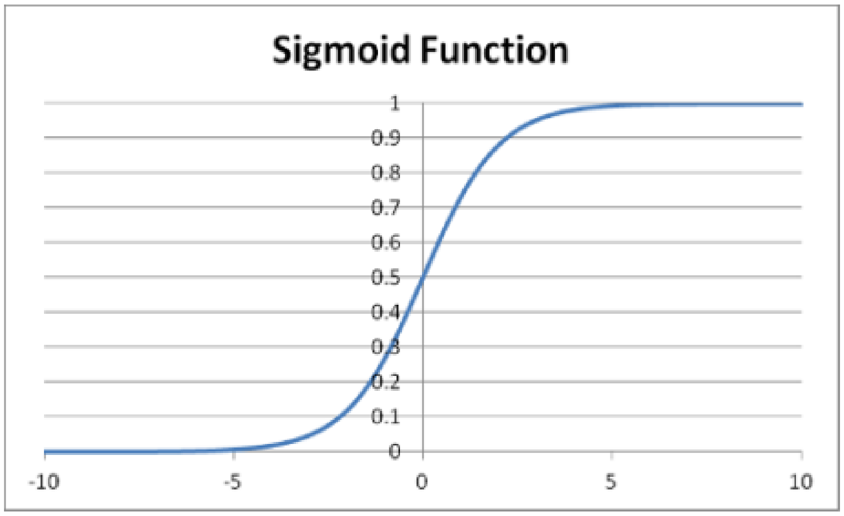
Logistic regression can be viewed as extending linear regression to handle binary output y by warping the output of a linear function to the range between 0 and 1.

Consider the linear regression function: $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} = w_0 + w_1x_1 + \cdots + w_mx_m$

Where 'w' is the model parameter or feature weights, 'x(i)' is the ith feature value and 'm' the number of features.

It allows us to introduce this function: $\sigma(\boldsymbol{w}^T\boldsymbol{x}) = \dfrac{1}{1+\exp(-\boldsymbol{w}^T\boldsymbol{x})}$

Referred to as the sigmoid function or logistic function:



Thanks to this logistic function, we can know the probability of an output:

$$P(y=1|\boldsymbol{x};\boldsymbol{w}) = \sigma(\boldsymbol{w}^T\boldsymbol{x}) = \frac{1}{1+\exp(-\boldsymbol{w}^T\boldsymbol{x})}$$

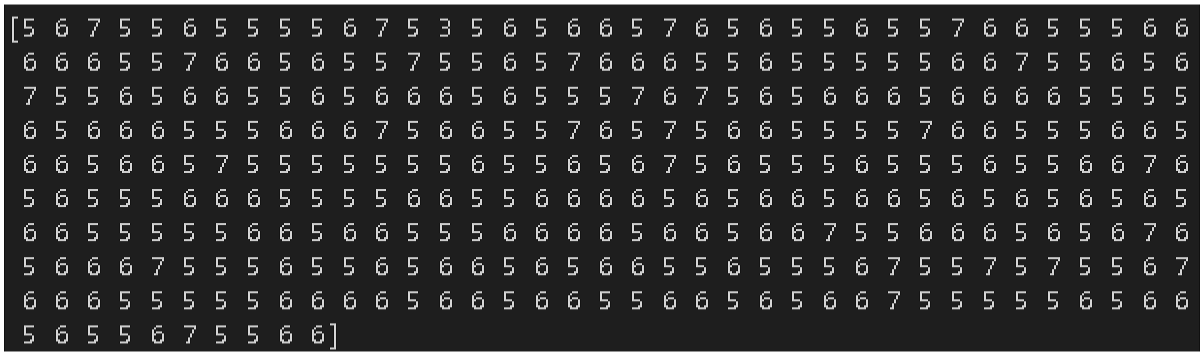
After computing the probability of each possible output, our model selects only the highest for our final prediction.

The whole training part of our model consists in adjusting these weights correctly. In order to obtain the best possible result.

Result:

Once learning is complete, we give our model a set of tests to predict. After comparing the expected results and the results of our model. We get an accuracy score of 67%, which is a rather low score.

Outliers not having much impact on logistic regression. We can explain this result by the lack of data. Indeed, although this method is efficient, it requires large training datasets to be accurate. We notice for example that our prediction does not contain any output equal to 4 or 8, since it has not been exposed to enough data of this type during its training.



```
[5 6 7 5 5 6 5 5 5 5 6 7 5 3 5 6 5 6 6 5 7 6 5 6 5 5 6 5 5 7 6 6 5 5 5 6 6
 6 6 6 5 5 7 6 6 5 6 5 5 7 5 5 6 5 7 6 6 6 5 5 6 5 5 5 5 5 6 6 7 5 5 6 5 6
 7 5 5 6 5 6 6 5 5 6 5 6 6 6 5 6 5 5 5 7 6 7 5 6 5 6 6 6 5 6 6 6 6 5 5 5 5
 6 5 6 6 6 5 5 5 6 6 6 7 5 6 6 5 5 7 6 5 7 5 6 6 5 5 5 5 7 6 6 5 5 5 6 6 5
 6 6 5 6 6 5 7 5 5 5 5 5 5 5 6 5 5 6 5 6 7 5 6 5 5 5 6 5 5 5 6 5 5 6 6 7 6
 5 6 5 5 5 6 6 6 5 5 5 5 6 6 5 5 6 6 6 6 5 6 5 6 6 5 6 6 5 6 5 6 5 6 5 6 5
 6 6 5 5 5 5 5 6 6 5 6 6 5 5 5 6 6 6 6 5 6 6 5 6 6 7 5 5 6 6 6 5 6 5 6 7 6
 5 6 6 6 7 5 5 5 6 5 5 6 5 6 6 5 6 5 6 6 5 5 6 5 5 5 6 7 5 5 7 5 7 5 5 6 7
 6 6 6 5 5 5 5 5 6 6 6 6 5 6 6 5 6 6 5 5 6 6 5 6 5 6 6 7 5 5 5 5 5 6 5 6 6
 5 6 5 5 6 7 5 5 6 6]
```

The next step:

It might be interesting to try to predict the quality of other types of wine using the same parameters. Like white wine or rosé wine.

And then, assemble the data and train a model to recognize which wine belongs to which type.

Third-party libraries:

Pandas:

pandas.read_csv("data.csv"): Read a comma-separated values (csv) file into DataFrame. We use one parameter (the name of our dataset name data.csv)

pandas.DataFrame.dropna(inplace=True): Remove missing values. We use "inplace=True" to keep the DataFrame with valid values in there.

pandas.DataFrame.drop('Id', axis=1, inplace=True): Drop specified labels from rows or columns. We define the column that we want to drop (here the column 'Id'), we drop all the column (axis=1) and we keep the DataFrame with valid values in there

pandas.DataFrame.head(): Return the first *n* rows., by default 5. We don't use any parameter in this function.

pandas.DataFrame.isna(): Detect missing values. We don't use any parameter in this function.

pandas.DataFrame.corr(): Compute pairwise correlation of columns, excluding NA/null values. We don't use any parameter in this function.

pandas.DataFrame.reset_index(): Reset the index, or a level of it. We don't use any parameter in this function.

pandas.DataFrame.iloc[:int(range),:]: Purely integer-location based indexing for selection by position. We don't use any parameter in this function.

pandas.DataFrame.copy(): Make a copy of this object's indices and data. We don't use any parameter in this function.

pandas.DataFrame.rename(columns={"fixed acidity": "fix acd", "volatile acidity": "vlti acd", "citric acid": "citr acd", "residual sugar": "resid sgr", "chlorides": "chlor", "free sulfur dioxide": "free sf dx", "total sulfur dioxide": "total sf dx", "density": "dens", "sulphates": "sulpha", "alcohol": "alcl", "quality": "qual"}): Alter axes labels. We rename the column with another values (for instance, fixed acidity became fix acd, volatile acidity became vlti acd, and so on)

pandas.plotting.scatter_matrix(df.sample(int(range)), figsize=(40,30)): Draw a matrix of scatter plots. We use our dataFrame df in which we take a sample of range value (corresponds to the value given by the user), and we define the width & height of our plot (here 40 for width & 30 for height).

Matplotlib:

matplotlib.pyplot.subplots(figsize=(15, 15)): Create a figure and a set of subplots. We define the width & height of our plot (here 15 for width & 15 for height).

matplotlib.pyplot.show(): Display a figure. We don't use any parameter in this function.

matplotlib.pyplot.subplot(1, 3, 1): Add a subplot to the current figure. We create a subplot which represents the top plot of a grid with 1 row and 3 columns.

matplotlib.pyplot.figure(figsize=(10, 4)): Create a new figure. We define the width & height of our plot (here 10 for width & 4 for height).

matplotlib.pyplot.plot(kind='hist'): Plot y versus x as lines and/or markers. We made a plot that corresponds to histogram.

matplotlib.pyplot.title(f'{i} histogram plot: $\mu = {round(mean, 2)}$ $\sigma = {round(standard_deviation, 2)}$ \n'): Set a title for the axes. The value corresponds to the value of the column selected in the loop (represent by i value). We also display the mean value with 2 digits after the comma, preceeded by the mu symbol and the standard deviation with 2 digits after the comma, preceded by the sigma symbol

matplotlib.pyplot.xlabel(f'{i}'): Set the label for the x-axis. The value corresponds to the value of the column selected in the loop (represent by i value).

matplotlib.pyplot.savefig(f'analysis_plots/{i} histogram plot.pdf'): Save the current figure. We save our pdf in the folder analysis_plots in which the name corresponds to the value of the column selected in the loop (represent by i value).

matplotlib.pyplot.close(): Close a figure window. We don't use any parameter in this function.

Seaborn:

seaborn.heatmap(df.corr(), cmap='magma', annot=True): Plot rectangular data as a color-encoded matrix. We use our dataFrame df with corr() function, the cmap correspond to the color in the plot (we use magma to highlight the correlations and we display the value of correlation with annot=True.

seaborn.regplot(x=x_value, y=y_value, data=df.sample(int(range)), line_kws={"color": "red"}).set(title='Scatter plot and regression line of '+ x_value + ' by ' + y_value): Plot data and a linear regression model fit. We define x and y value that corresponds to the value given by the user (these values are the name of columns in the dataset like pH, alcohol and so on), data correspond to the dataset we use (here it's df) in which we take a sample of range value (corresponds to the value given by the user). We finally draw a regression line in red and we set the title of the plot (with x_value and y_value).

Numpy :

numpy.std(df[i]): Compute the standard deviation along the specified axis. We use a column in our dataset df (in this case it's i because we use it in a loop in which i is the value of the column of the dataset).

Plotly:

plotly.express.parallel_coordinates(df, color='quality', color_continuous_scale=px.colors.diverging.Tealrose, color_continuous_midpoint=5.5, title="Parallel coordinates for the wine data"): Create a parallel coordinates plot. We define our dataFrame df, the color corresponds to the column 'quality'. The color_continuous_midpoint correpond to the size of the line dranw between 2 columns. Finally, we set the title of the plot.

Sklearn:

sklearn.linear_model.LogisticRegression(random_state=0, solver='lbfgs', max_iter=8000): Logistic Regression (aka logit, MaxEnt) classifier. We use random_state to shuffle the data, we use the solver='lbfgs' to optimize problem solving, and we set the maximum number of iterations (max_iter) to 8000.

sklearn.model_selection.train_test_split(x, y, test_size=0.3, randon_state=101): Split arrays or matrices into random train and test subsets. We use x that correspond to the dataFrame without the quality column, and we use y that correspond to the dataFrame with only quality column. We also define the proportion of the dataset to include on the test split (here 0.30 so 30%). Finally, we control the shuffling applied to the data before applying the split with random_state=101.

sklearn.fit(x_train, y_train): x_train correspond to the training data and y_train correspond to the target values.

sklearn.predict(x_test). Predict using the linear model. X_test correspond to sample of value we use.

sklearn.metrics.accuracy_score(y_test, predictions): Accuracy classification score. Y_test correspond to correct target values and prediction to the estimated target values

CSV:

csv.writer(file_name): Return a writer object responsible for converting the user's data into delimited strings on the given file-like object. File_name correspond to our csv file.

csvwriter.writerow(array): Write the *row* parameter to the writer's file object, formatted according to the current Dialect. Array correspond to our data