Andrew D #datascience Follow

Jul 7 · 12 min read ★ · Listen

Exploratory Data Analysis in Python — A Step-by-Step Process

What is exploratory analysis, how it is structured and how to apply it in Python with the help of Pandas and other data analysis and visualization libraries

Exploratory data analysis (*EDA*) is an especially important activity in the routine of a data analyst or scientist.

It enables an in depth understanding of the dataset, define or discard hypotheses and create predictive models on a solid basis.

It uses data manipulation techniques and several statistical tools to describe and understand the relationship between variables and how these can impact business.

In fact, **it's thanks to EDA that we can ask ourselves meaningful questions that can impact business.**

In this article, I will share with you a template for exploratory analysis that I have used over the years and that has proven to be solid for many projects and domains. This is implemented through the use of the Pandas library — an essential tool for any analyst working with Python.

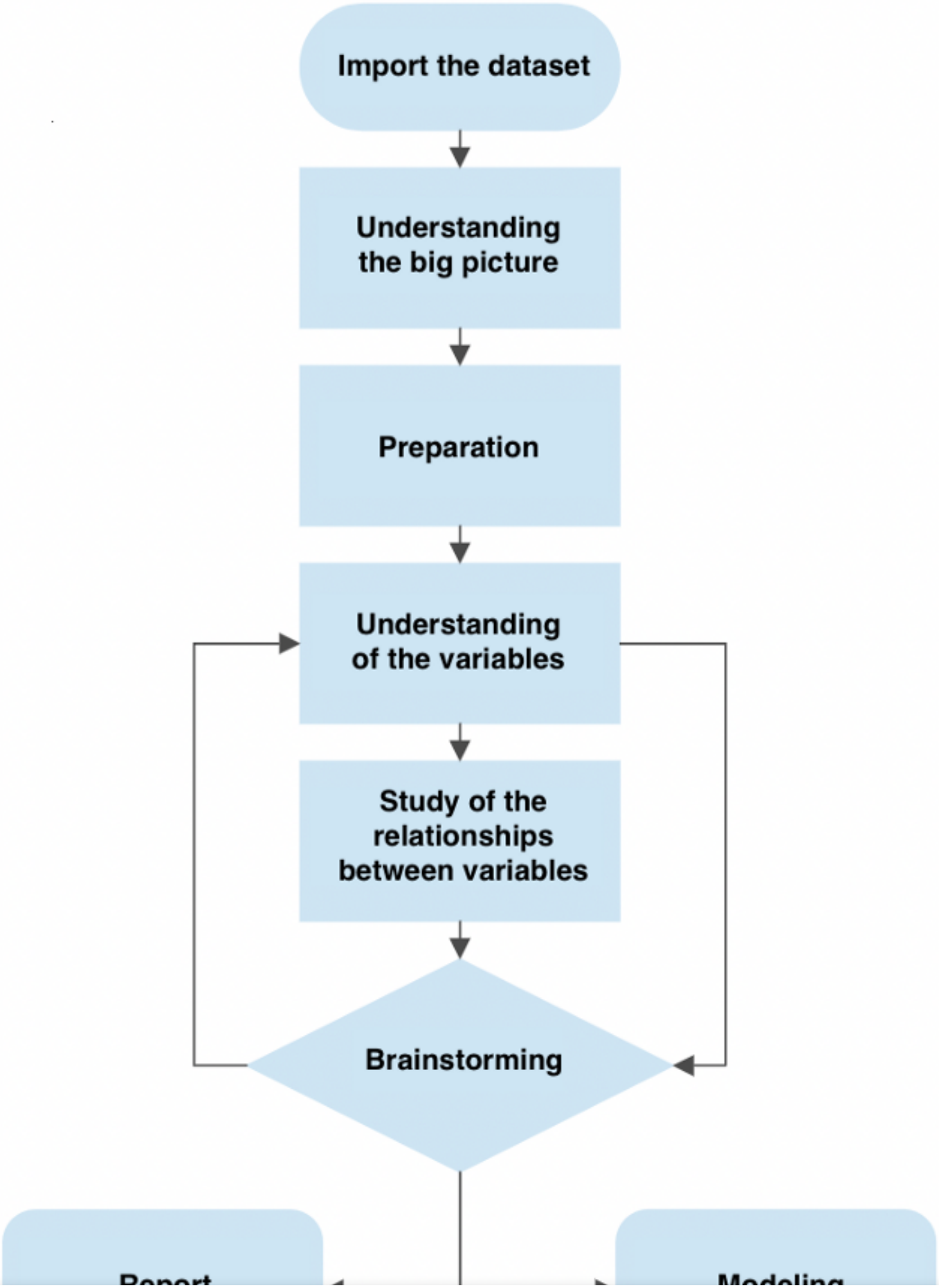
The process consists of several steps:

1. Importing a dataset
2. Understanding the big picture
3. Preparation
4. Understanding of variables
5. Study of the relationships between variables
6. Brainstorming

This template is the result of many iterations and allows me to ask myself meaningful questions about the data in front of me. At the end of the process, **we will be able to consolidate a business report or continue with the data modeling phase**.

The image below shows how the brainstorming phase is connected with that of understanding the variables and how this in turn is connected again with the brainstorming phase.

This process describes how we can move to ask new questions until we are satisfied.



Import the dataset
Understanding the big picture
Preparation
Understanding of the variables
Study of the relationships between variables
Brainstorming

We will see some of the most common and important features of Pandas and also some techniques to manipulate the data in order to understand it thoroughly.

Motivation for the exploratory analysis process

I have discovered with time and experience that a large number of companies are looking for insights and value that **come from fundamentally descriptive activities.**

This means that companies are often willing to allocate resources to acquire the necessary awareness of the phenomenon that we analysts are going to study.

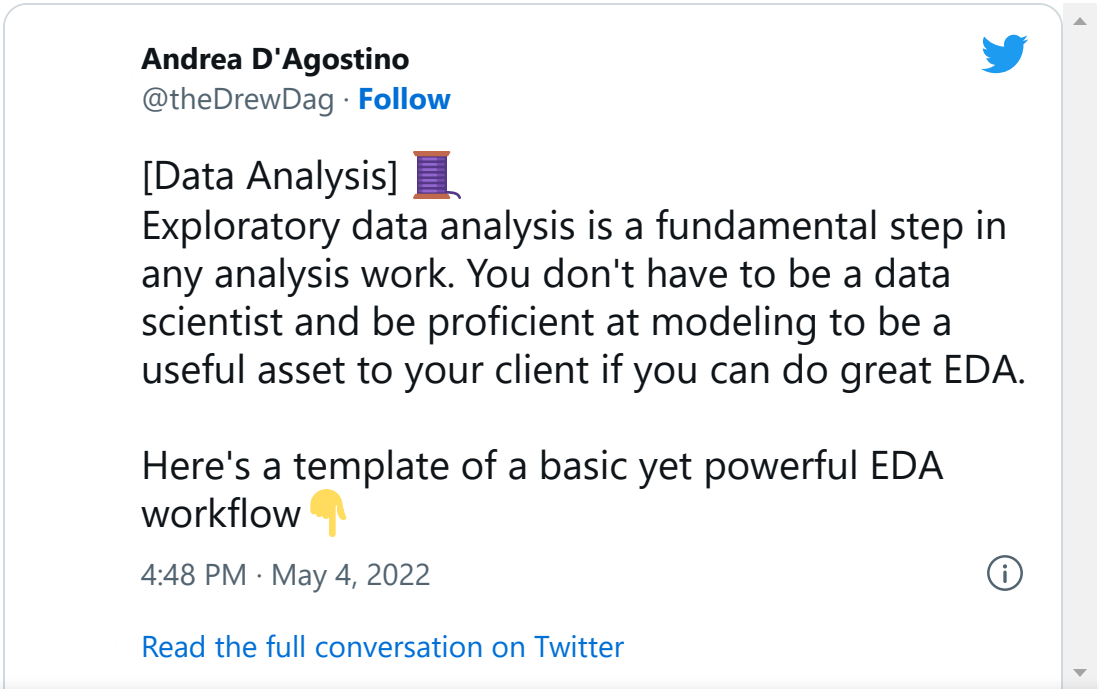
The knowledge of something.

> *If we are able to investigate the data and ask the right questions, the EDA process becomes extremely powerful. By combining data visualization skills, a skilled analyst is able to build a career only by leveraging these skills. You don't even have to go into modeling.*

A good approach to EDA therefore allows us to provide added value to many business contexts, especially where our client / boss finds difficulties in the interpretation or access to data.

This is the basic idea that led me to put down such a template.

I wrote a Twitter thread that puts my thoughts on the matter on paper



Required libraries for EDA

Before starting, let's see what are the fundamental libraries required to carry out the EDA. There are many useful libraries but here we will only see the ones that this template leverages

```
# data manipulation
import pandas as pd
import numpy as np

# data viz
import matplotlib.pyplot as plt
from matplotlib import rcParams
import seaborn as sns

# apply some cool styling
plt.style.use("ggplot")
rcParams['figure.figsize'] = (12,  6)

# use sklearn to import a dataset
from sklearn.datasets import load_wine
```

eda_libs1.py hosted with ❤ by GitHub view raw

1. Importing a working dataset

The data analysis pipeline begins with the import or creation of a working dataset. The exploratory analysis phase begins immediately after.

Importing a dataset is simple with Pandas through functions dedicated to reading the data. If our dataset is a .csv file, we can just use

```
df = pd.read_csv("path/to/my/file.csv")
```

df stands for dataframe, which is Pandas's object similar to an Excel sheet. This nomenclature is often used in the field. The *read_csv* function takes as input the path of the file we want to read. There are many other arguments that we can specify.


For ease, in this example we will use Sklearn to import the *wine dataset*. This dataset is widely used in the industry for educational purposes and contains information on the chemical composition of wines for a classification task. We will not use a .csv but a dataset present in Sklearn to create the dataframe

```
# carichiamo il dataset
wine = load_wine()

# convertiamo il dataset in un dataframe Pandas
df = pd.DataFrame(data=wine.data, columns=wine.feature_names)
# creiamo la colonna per il target
df["target"] = wine.target
```

eda_dataset.py hosted with ❤ by GitHub view raw

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	target
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	13.71	5.65	2.45	20.5	95.0	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740.0	2
174	13.40	3.91	2.48	23.0	102.0	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750.0	2
175	13.27	4.28	2.26	20.0	120.0	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835.0	2
176	13.17	2.59	2.37	20.0	120.0	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840.0	2
177	14.13	4.10	2.74	24.5	96.0	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560.0	2

178 rows × 14 columns

Example of the dataset. Image by author.

Now that we've imported a usable dataset, let's move on to applying the EDA pipeline.

2. Understanding the big picture

In this first phase, our goal is to understand what we are looking at, but without going into detail. **We try to understand the problem we want to solve, thinking about the entire dataset and the meaning of the variables.**

This phase can be slow and sometimes even boring, but it will give us the opportunity to make an opinion of our dataset.

- **Type**: the type or format of the variable. This can be categorical, numeric, Boolean, and so on
- **Context**: useful information to understand the semantic space of the variable. In the case of our dataset, the context is always the chemical-physical one, so it's easy. In another context, for example that of real estate, a variable could belong to a particular segment, such as the anatomy of the material or the social one (how many neighbors are there?)
- **Expectation**: how relevant is this variable with respect to our task? We can use a scale "High, Medium, Low".
- **Comments**: whether or not we have any comments to make on the variable

Of all these, *Expectation* is one of the most important because it helps us develop the analyst's "sixth sense" — as we accumulate experience in the field we will be able to mentally map which variables are relevant and which are not.

In any case, the point of carrying out this activity is that it enables us to do some preliminary reflections on our data, which helps us to start the analysis process.

Useful properties and functions in Pandas

We will leverage several Pandas features and properties to understand the big picture. Let's see some of them.

`.head()` and `.tail()`

Two of the most commonly used functions in Pandas are .head() and .tail(). These two allow us to view an arbitrary number of rows (by default 5) from the beginning or end of the dataset. Very useful for accessing a small part of the dataframe quickly.

```
df.head()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	target
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	0

df.head() example. Image by author.

`.shape`

and the length of the dataset.

```
df.shape
```

```
(178, 14)
```

df.shape example. Image by author.

`.describe()`

The *describe* function does exactly this: it provides purely descriptive information about the dataset. This information includes statistics that summarize the central tendency of the variable, their dispersion, the presence of empty values and their shape.

```
df.describe()
```

acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	target
CCO	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	1.590899	5.058090	0.957449	2.611685	746.893258	0.938202
146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453	0.572359	2.318286	0.228572	0.709990	314.907474	0.775035
CCO	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.410000	1.280000	0.480000	1.270000	278.000000	0.000000
500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000	1.250000	3.220000	0.782500	1.937500	500.500000	0.000000
CCO	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000	1.555000	4.690000	0.965000	2.780000	673.500000	1.000000
500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	1.950000	6.200000	1.120000	3.170000	985.000000	2.000000
CCO	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.580000	13.000000	1.710000	4.000000	1680.000000	2.000000

df.describe() example. Image by author.

`.info()`

Unlike .describe(), *.info()* gives us a shorter summary of our dataset. It returns us information about the data type, non-null values and memory usage.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   alcohol                       178 non-null    float64
 1   malic_acid                    178 non-null    float64
 2   ash                           178 non-null    float64
 3   alcalinity_of_ash             178 non-null    float64
 4   magnesium                     178 non-null    float64
 5   total_phenols                 178 non-null    float64
 6   flavanoids                    178 non-null    float64
 7   nonflavanoid_phenols          178 non-null    float64
 8   proanthocyanins               178 non-null    float64
 9   color_intensity               178 non-null    float64
 10  hue                           178 non-null    float64
 11  od280/od315_of_diluted_wines  178 non-null    float64
 12  proline                       178 non-null    float64
 13  target                        178 non-null    int64  
dtypes: float64(13), int64(1)
memory usage: 19.6 KB
```

df.info() example. Image by author.

There are also *.dtypes* and *.isna()* which respectively give us the data type info and whether the value is null or not. However, using .info() allows us to access this information with a single command.

What is our goal?

This is an important question that we must always ask ourselves. In our case, we see how the target is a numeric categorical variable that covers the values of 0, 1 and 2. These numbers identify the type of wine.

If we check the Sklearn documentation on this dataset we see that it was built precisely for classification tasks. If we wanted to do modeling, **the idea would then be to use the features of the wine to predict its type.** In a data analysis setting instead, we would want to study how the different types of wine have different features and how these are distributed.

At this stage we want to start cleaning our dataset in order to continue the analysis. Some of the questions we will ask ourselves are

- are there any useless or redundant variables?
- are there any duplicate columns?
- does the nomenclature make sense?
- are there any new variables we want to create?

Let's see how to apply these ideas to our dataset.

- All the variables appear to be physical-chemical measures. **This means they could all be useful and help define the segmentation of the type of wine**. We have no reason to remove columns
- To check for duplicate rows we can use `.isduplicated().sum()` — this will print us the number of duplicated rows in our dataset

```
df.duplicated().sum()
```

```
0
```

A simple check for row duplication. Image by author.

- The nomenclature can certainly be optimized. For example *od280 / od315_of_diluted_wines* is difficult to understand. Since it indicates a research methodology that measures protein concentration in the liquid, we will call it *protein_concentration*

```
df.rename(columns={"od280/od315_of_diluted_wines": "protein_concentration"}, inplace=True)
```

rename.py hosted with ❤ by GitHub view raw

- One of the most common *feature engineering* methods is to create new features that are **the linear / polynomial combination of the existing ones**. This becomes useful for providing more information to a predictive model to improve its performance. We will not do this in our

4. Understanding of the variables

While in the previous point we are describing the dataset in its entirety, now we try to accurately describe all the variables that interest us. For this reason, this step can also be called **univariate analysis**.

Categorical variables

In this context, `.value_counts()` is one of the most important functions to understand how many values of a given variable there are in our dataset. Let's take the target variable for example.

```
df.target.value_counts()
```

```
1    71
0    59
2    48
Name: target, dtype: int64
```

value_counts() example. Image by author.

You can also express the data as a percentage by passing *normalize = True*

```
df.target.value_counts(normalize=True)
```

```
1    0.398876
0    0.331461
2    0.269663
Name: target, dtype: float64
```

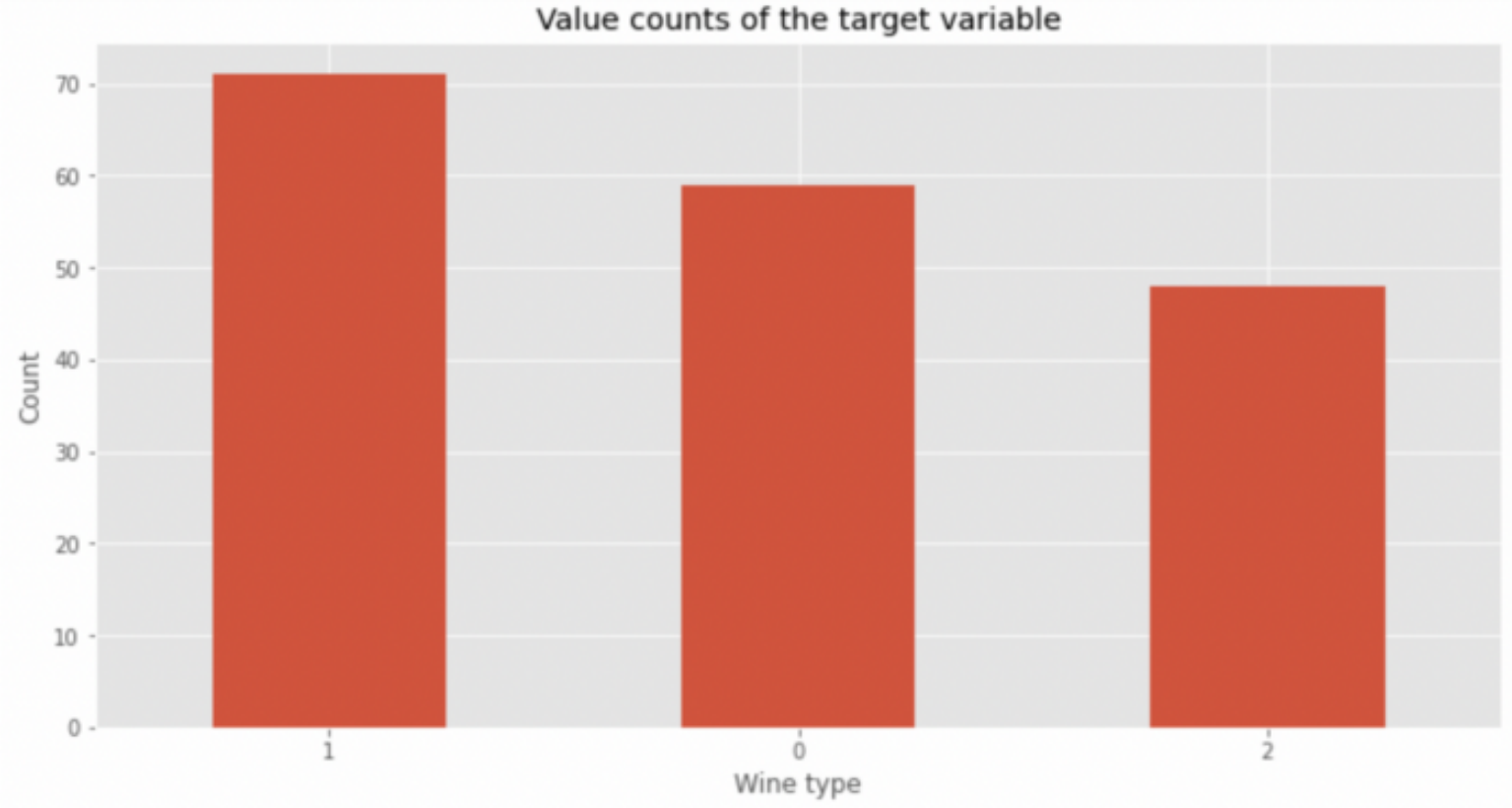
Percentages in value_counts(). Image by author.

We can also plot the data with

```
df.target.value_counts().plot(kind="bar")
plt.title("Value counts of the target variable")
plt.xlabel("Wine type")
plt.xticks(rotation=0)
plt.ylabel("Count")
plt.show()
```

vc_eng.py hosted with ❤ by GitHub view raw

Implementation of value_counts() and its plot.



Plot of the value counts of the target variable. Image by author.

value_counts() can be used with any variable, but works best with categorical variables such as our target. **This function also informs us of how balanced the classes are within the dataset**. In this case, class 2 appears less than the other two classes — in the modeling phase perhaps we can implement data balancing techniques to not confuse our model.

Numeric variables

If instead we want to analyze a numeric variable, we can describe its distribution with *describe()* as we have seen before and we can display it with *.hist()*.

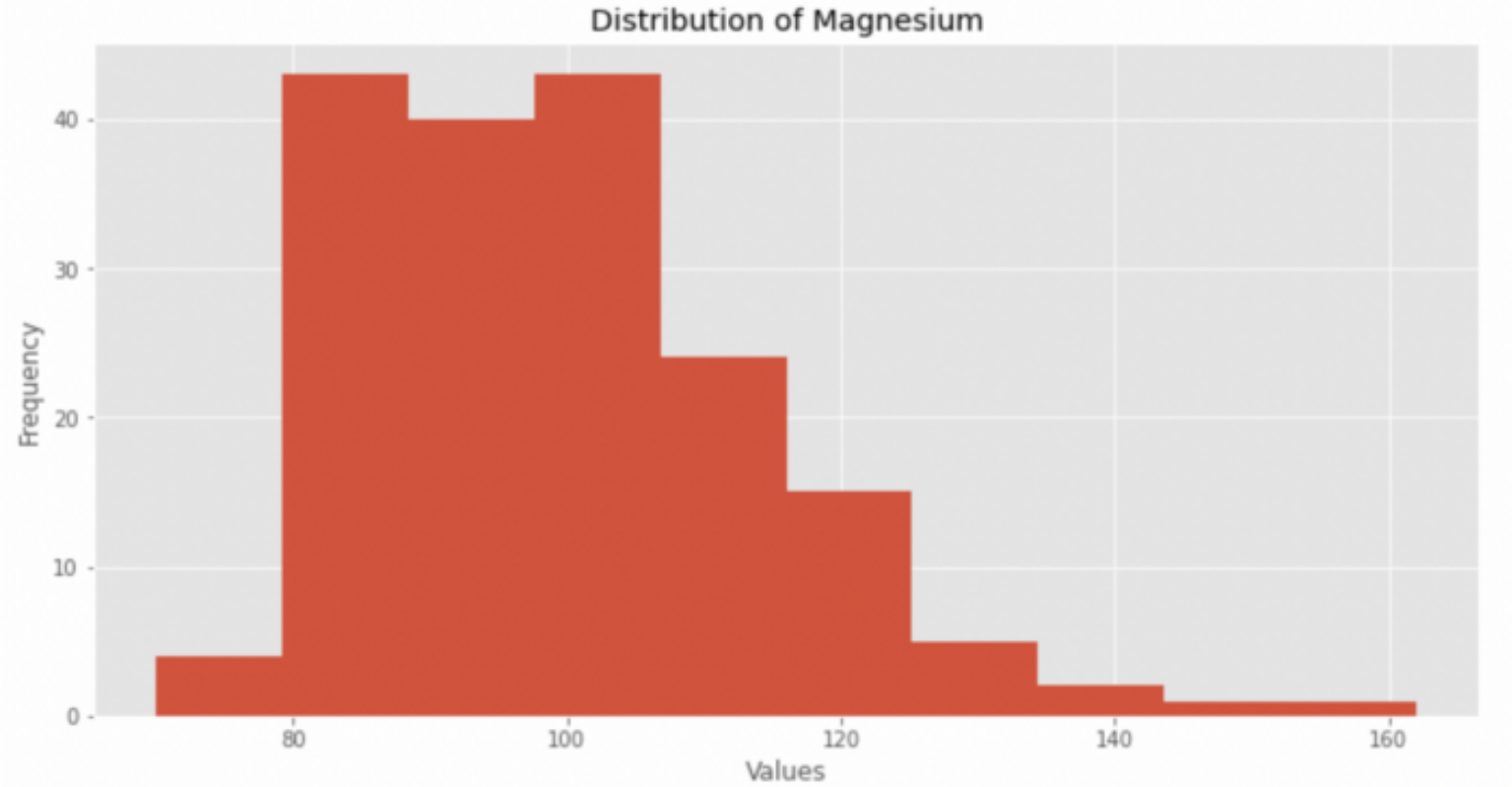
Take for example the variable *magnesium*

Let's use *.describe()* first

```
df.magnesium.describe()
```

```
count    178.000000
mean      99.741573
std       14.282484
min       70.000000
25%       88.000000
```

and then plot the histogram



The histogram of the magnesium variable. Image by author.

We also evaluate distribution kurtosis and asymmetry:

```
1  print(f"Skewness: {df['magnesium'].skew()}")
2  print(f"Kurtosis: {df['magnesium'].kurt()}")
```

kurtskew.py hosted with ❤ by GitHub view raw

```
print(f"Kurtosis: {df['magnesium'].kurt()}")
print(f"Skewness: {df['magnesium'].skew()}")
```

```
Kurtosis: 2.1049913235905557
Skewness: 1.098191054755161
```

Kurtosis and skewness values for Magnesium. Image by author.

- show spikes
- has kurtosis and asymmetry values greater than 1

We do this for each variable, and we will have a pseudo-complete descriptive picture of their behavior.

We need this work to fully understand each variable, **and unlocks the study of the relationship between variables.**

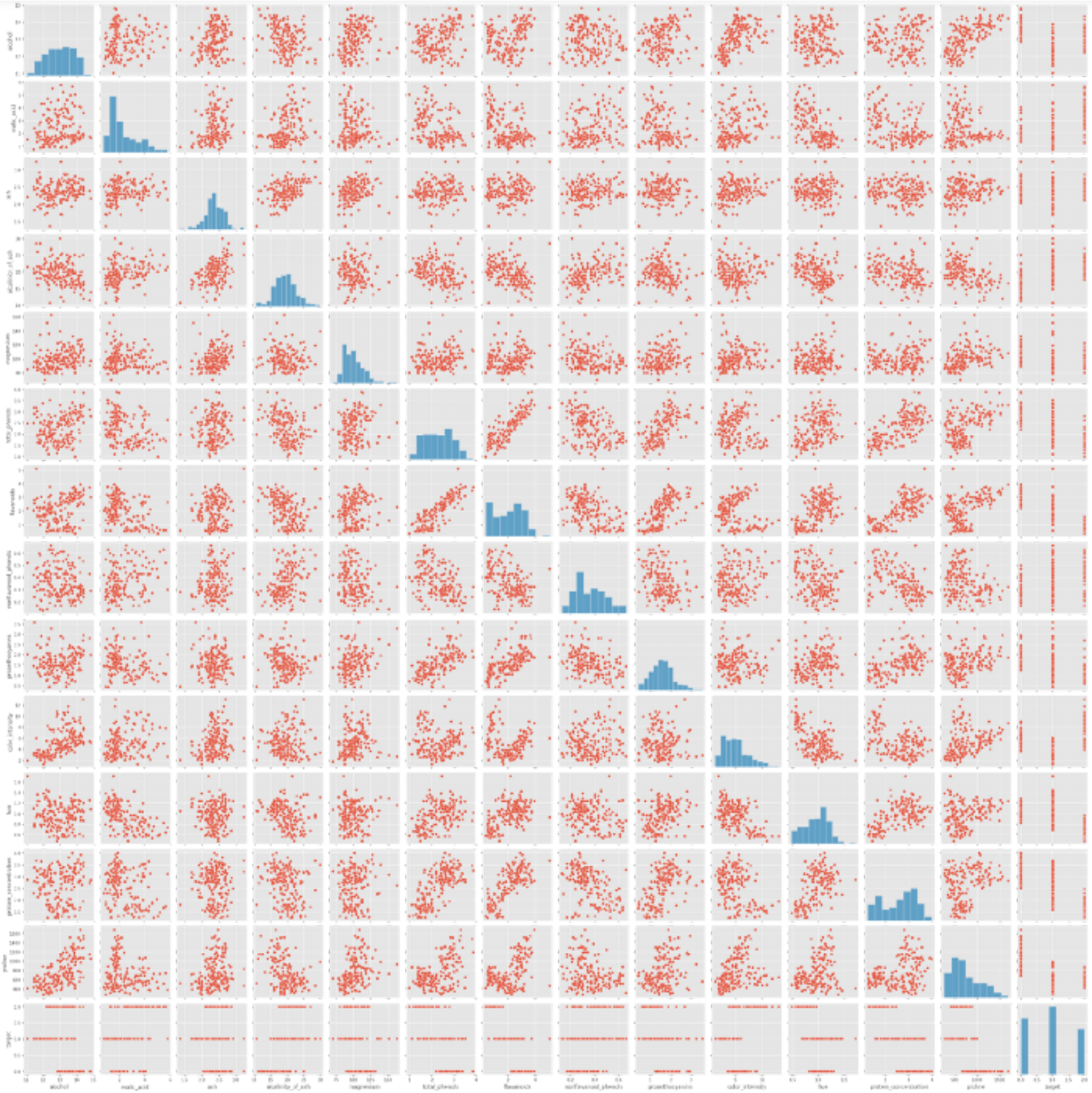
4. Study of relationships between variables

Now the idea is to find interesting relationships that show the influence of one variable on the other, preferably on the target.

This job unlocks the first *intelligence* options — in a business context such as digital marketing or online advertising, this information offers value and the ability to act strategically.

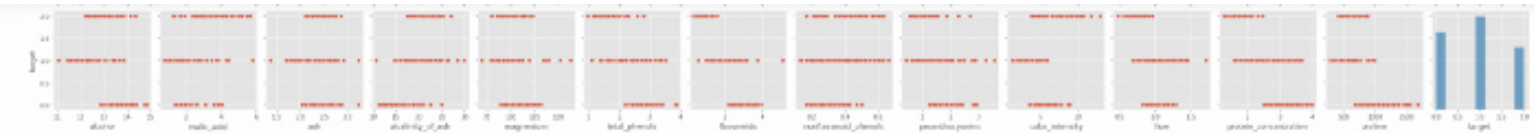
We can start exploring relationships with the help of Seaborn and *pairplot*.

```
sns.pairplot(df)
```

Application of sns.pairplot on the dataframe. Image by author.

As you can see, *pairplot* displays all the variables against each other in a scatterplot. **It is very useful for grasping the most important relationships without having to go through every single combination manually**. Be warned though — it is computationally expensive to compute, so it is best suited for datasets with relatively low number of variables like this one.



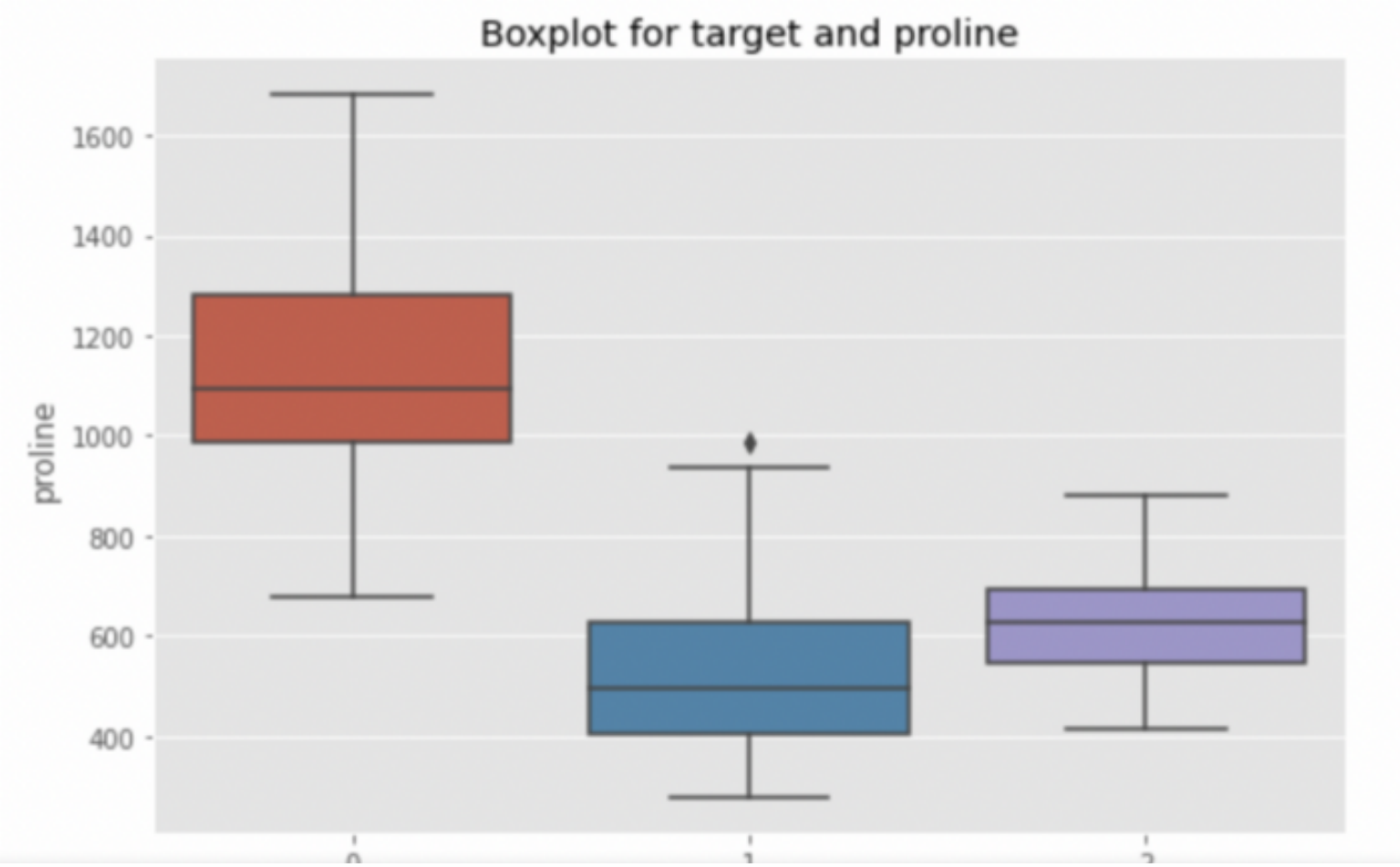
Pairplot for the target. Image by author.

The best way to understand the relationship between a numeric variable and a categorical variable is through a boxplot.

Let's create a boxplot for *alcohol*, *flavanoids*, *color_intensity* and *proline*. Why these variables? **Because visually they show slightly more marked segmentations for a given wine type**. For example, let's look at proline vs target

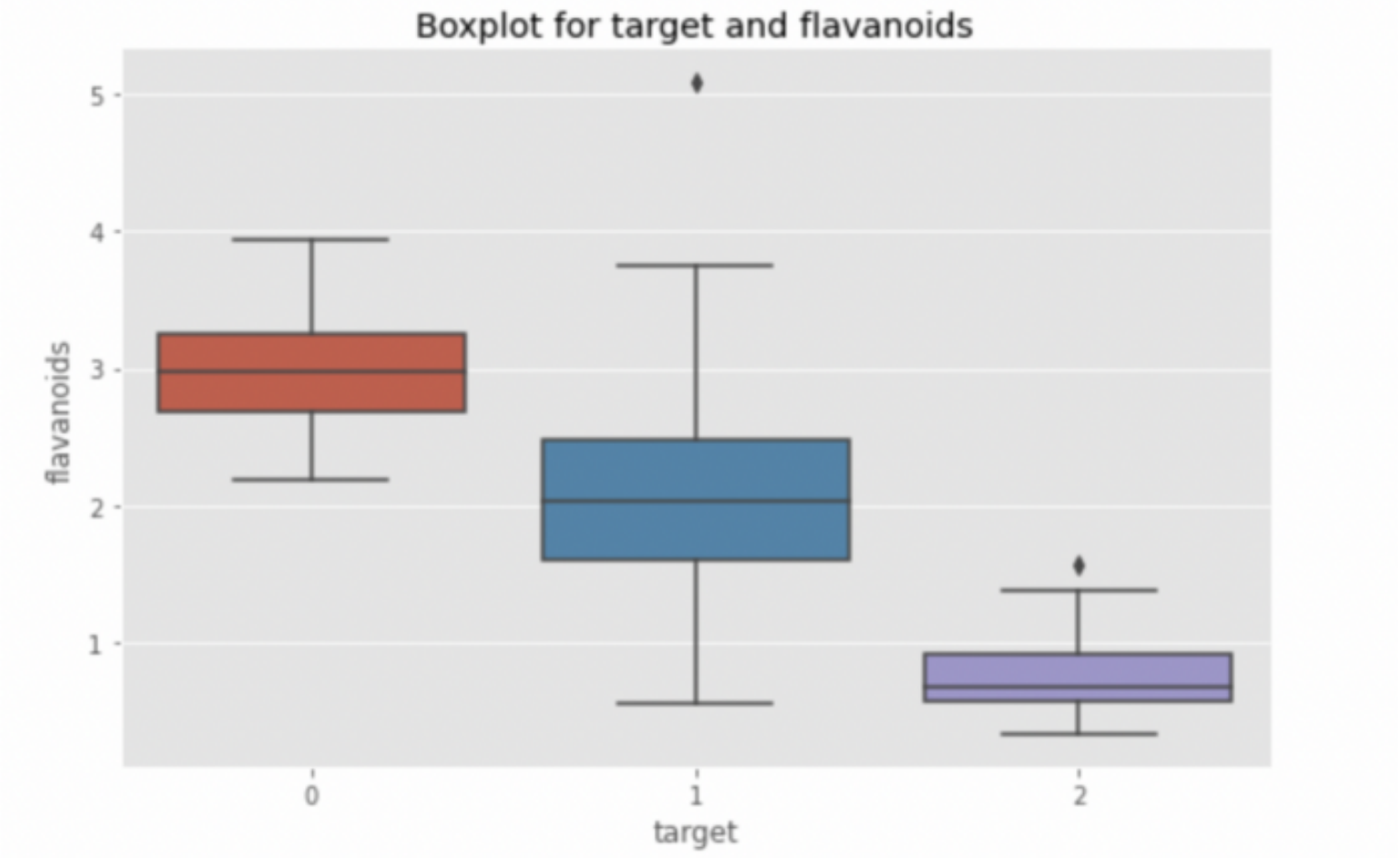
```
1  sns.catplot(x="target", y="proline", data=df, kind="box", aspect=1.5)
2  plt.title("Boxplot for target vs proline")
3  plt.show()
```

cateda.py hosted with ❤ by GitHub view raw



In fact, we see how the proline median of type 0 wine is bigger than that of the other two types. Could it be a differentiating factor? Too early to tell. There may be other variables to consider. Let's see *flavanoids* now

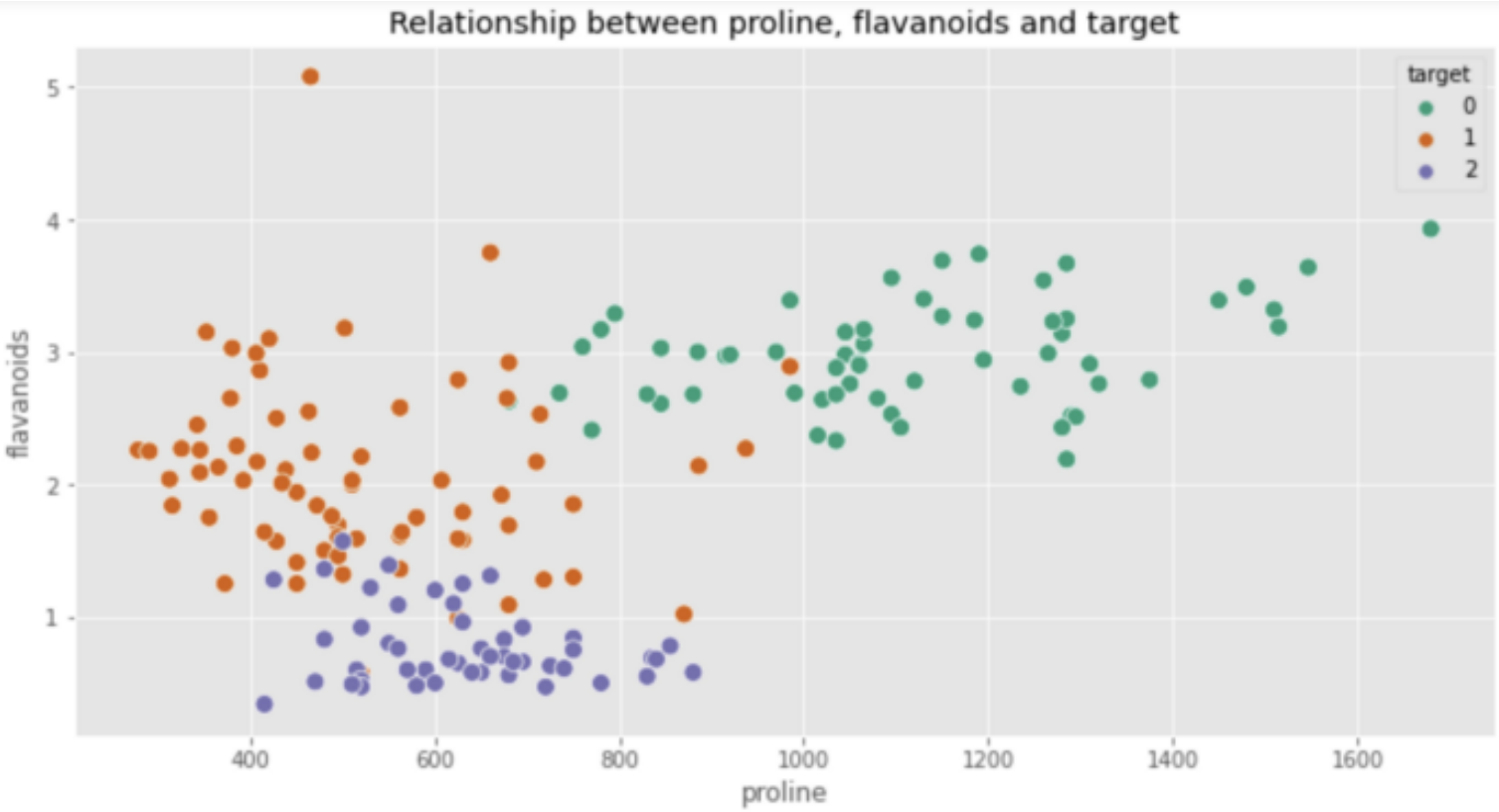
```
sns.catplot(x="target", y="flavanoids", data=df, kind="box", aspect=1.5)
plt.title("Boxplot for target and flavanoids")
plt.show()
```



Boxplot of target vs flavanoids. Image by author.

Here too, the type 0 wine seems to have higher values of flavanoids. Is it possible that type 0 wines have higher combined levels of proline and flavanoids? With Seaborn we can create a scatterplot and visualize which wine class a point belongs to. Just specify the *hue* parameter

```
sns.scatterplot(x="proline", y="flavanoids", hue="target", data=df, palette="Dark2", s=80)
plt.title("Relationship between proline, flavanoids and target")
plt.show()
```



Application of the hue parameter in the sns.scatterplot. Image by author.

Our intuition was on the right track! Type 0 wines show clear patterns of flavanoids and proline. In particular, the proline levels are much higher while the flavanoid level is stable around the value of 3.

Now let's see how Seaborn can again help us expand our exploration thanks to the *heatmap*. We are going to create a correlation matrix with Pandas and to isolate the most correlated variables

```
corrmat = df.corr()
hm = sns.heatmap(corrmat,
                 cbar=True,
                 annot=True,
                 square=True,
                 fmt='.2f',
                 annot_kws={'size': 10},
                 yticklabels=df.columns,
                 xticklabels=df.columns,
                 cmap="Spectral_r")
plt.show()
```

corr.py hosted with ❤ by GitHub view raw

Heatmap of the correlation matrix. Image by author.

The heat map is useful because it allows us to efficiently grasp which variables are strongly correlated with each other.

When the target variable decreases (which must be interpreted as a tendency to go to 0, therefore to the type of wine 0) the flavanoids, total phenols, proline and other proteins tend to increase. And viceversa.

We also see the relationships between the other variables, excluding the target. For example, there is a very strong correlation between alcohol and proline. High levels of alcohol correspond to high levels of proline.

We have collected a lot of data to support the hypothesis that class 0 wine has a particular chemical composition. It remains now is to isolate what are the conditions that differentiate type 1 from type 2. I will leave this exercise to the reader. At this point of the analysis we have several things we can do:

- create a report for the stakeholders
- do modeling
- continue with the exploration to further clarify business questions

The importance of asking the right questions

Regardless of the path we take after the EDA, **asking the right questions is what separates a good data analyst from a mediocre one**. We may be experts with the tools and tech, but these skills are relatively useless if we are unable to retrieve information from the data.

Asking the right questions allows the analyst to "be in sync" with the stakeholder, or to implement a predictive model that really works.

Again, I urge the interested reader to open up their favorite text editor and populate it with questions whenever doubts or specific thoughts arise. Be obsessive — if the answer is in the data, then it is up to us to find it and communicate it in the best possible way.

Conclusion

The process described so far is iterative in its nature. In fact, **the exploratory analysis goes on until we have answered all the business questions**. *It is impossible for me to show or demonstrate all the possible techniques of data exploration* —we don't have specific business requirements or valid real-world dataset. However, I can convey to the reader the importance of applying a template like the following to be efficient in the analysis.

If you want to support my content creation activity, feel free to follow my referral link below and join Medium's membership program. I will receive a portion of your investment and you'll be able to access Medium's plethora of articles on data science and more in a seamless way.

Join Medium with my referral link - Andrew D #datascience

What is your methodology for your exploratory analyses? What do you think of this one? Share your thoughts with a comment 👇

Thanks for your attention and see you soon! 👋