**A Guide to EDA in Python**

Important questions to ask about your data during exploratory data analysis

Creating machine learning models is cool. It’s tempting as a beginner (I know from experience) to jump straight to the cool part — after all, it’s the most important part too right?

What if you skipped straight to the climax of a movie that you’ve never seen before? Would you be confused? Would it even be enjoyable?

*Just as the first hour of character development is foundational in a movie, exploratory data analysis (EDA) is a crucial first step towards a good data science project.*

It’s time to grab your favorite blanket, snacks, and sweats and cozy up with your data. By the end of EDA, you should know your dataset as well as you know the characters of your favorite movies.

For this article, I made a list of the questions I try to answer when I conduct EDA at the beginning of a data science project. I’ve provided code snippets (all in Python, using the Pandas library) to give examples of some code you can run to answer these questions.



Action! (Photo by [Jakob Owens](https://unsplash.com/@jakobowens1?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral))

**0. Questions To Ask B*efore* You Download the Data**

I called this one step 0 because it happens before you import data into Python. It’s easy to forget, but if you can answer these questions it can save you a lot of time and frustration down the road.

* How was this data collected/where did it come from?
* Why am I interested in this data?
* What would be the target variable of interest? (if applicable)
* Is this data from a reputable source?
* Is there enough data here to make an ML model?
* Have other people conducted a similar analysis/modeling project on this dataset? Do I want to be able to learn from their conclusions or create a novel project?
* Is there a data dictionary for the dataset? Is it complete?
* Are there any additional challenges or problems that I anticipate if I use this data?

It’s helpful to use these questions like a filter when you have a choice on what dataset to use. It’s really tough to realize halfway through a project that you picked a bad dataset.

**1. Data Structure & Distributions**

For this section, I will just look at continuous variables. However, you would also want to repeat all of the questions that are applicable when you analyze your discrete variables.

Here are two lines of code I *always*run:

df.shape  
df.info()

And here is an example of the output from df.info() for reference:

<class ‘pandas.core.frame.DataFrame’>   
RangeIndex: 1302102 entries, 0 to 1302101   
Data columns (total 10 columns):   
unitid 1302102 non-null int64   
year 1302102 non-null int64   
gender 1302102 non-null object   
race 1302102 non-null object   
cohort 1302102 non-null object   
grad\_cohort 889380 non-null float64   
grad\_100 410069 non-null float64   
grad\_150 889380 non-null float64   
grad\_100\_rate 332061 non-null float64   
grad\_150\_rate 694869 non-null float64   
dtypes: float64(5), int64(2), object(3)   
memory usage: 99.3+ MB

**Questions to answer:**

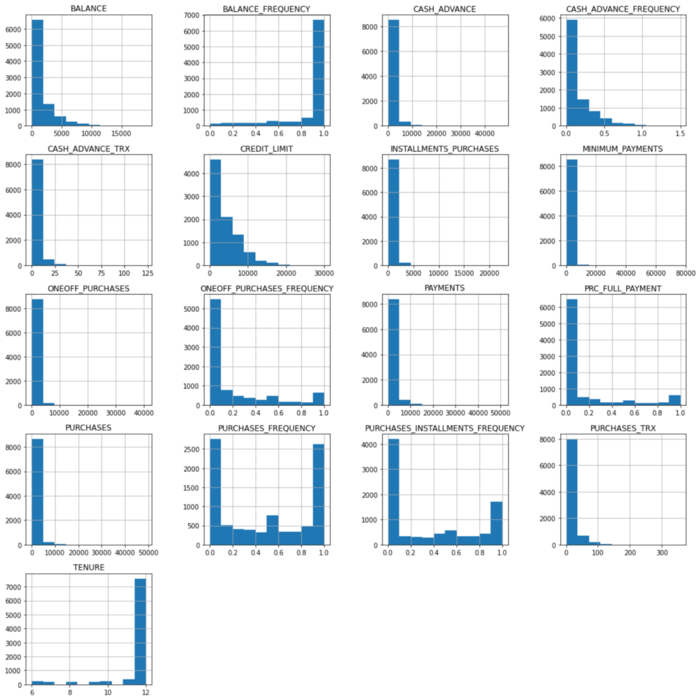
* How many features do you have?
* How many observations do you have?
* What is the data type of each feature?
* From what you know about the features of your dataset, do the data types make sense? Do you need to change any?

Example: Your data has a Customer ID number for every row, and each number is five digits long, stored as an integer. You will not ever be aggregating or analyzing the Customer ID like an integer, so you should change it to the “object” data type.

* Do you have null values? (to be fixed later)
* How much memory does this dataset use? Could this pose a problem for you later on?

And to look at variable distributions, these lines of code are **magic:**

df.hist(figsize=(15,15))   
#set a large figsize if you have > 9 variables  
plt.tight\_layout()  
plt.show()



Example output of code above

**Questions to answer:**

* What is the distribution of each variable?
* Do there appear to be outliers? (to be fixed later)
* Think about what the variables mean and what the histograms say about their values and their spread — are there any surprises?

And a final code snippet to look at summary statistics:

df.describe()

**Questions to answer:**

* Are the max/min values reasonable for the variables? Do you see any values that look like errors?
* What is the mean for each variable? What do the means tell you about your dataset as a whole?

**2. Null Values & Duplicates**

Let’s start with the easy one: duplicate values.

df.duplicated().sum()

After running this code, do you get a value greater than 0? If so, you can easily drop your duplicate rows with:

df.drop\_duplicates(inplace=True)

And now for null values. To print a list of every variable and the percentage of values in the dataset that are null (excluding 0), you can run:

null = df.isna().sum()/len(df)  
null[null > 0].sort\_values()

Here’s an example output:

aid\_value 0.000273   
aid\_percentile 0.000273   
pell\_value 0.000273   
pell\_percentile 0.000273   
ft\_pct 0.001093   
ft\_fac\_percentile 0.003347   
ft\_fac\_value 0.003347   
retain\_value 0.062022   
retain\_percentile 0.062022   
cohort\_size 0.069536   
grad\_100\_value 0.069536   
grad\_100\_percentile 0.069536   
grad\_150\_value 0.069536   
grad\_150\_percentile 0.069536   
grad\_150 0.121790   
grad\_cohort 0.121790   
grad\_150\_rate 0.279528

**Questions to answer:**

* Is the null value a result of the way data was recorded?

Example: Survey response data is recorded in columns as “yes”, “no,” and a null value for “prefer not to answer.” In this case, all nulls can be filled in with a single value like “no answer.”

* Can you drop the rows with null values without it significantly affecting your analysis?
* Looking at the distributions of the variables, can you justify filling in the missing values with the mean or median for that variable?

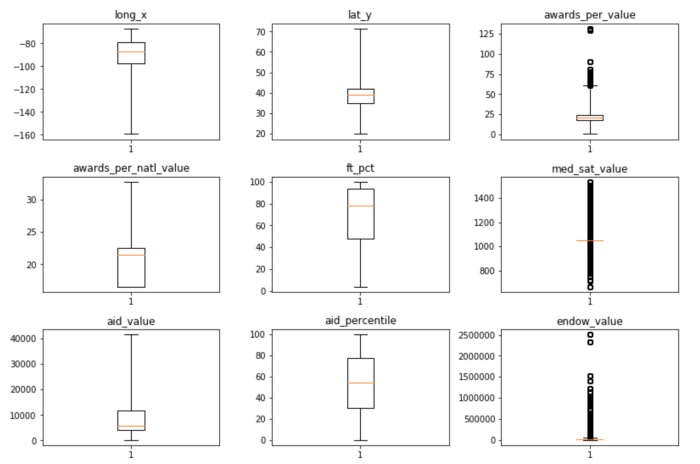
Be careful! You have to deal with missing values somehow, but sometimes it is better to drop rows rather than tinker with the original data because if you put bad data into a model you cannot get meaningful results.

* If your data is time-series data, can you fill the missing values with interpolation?
* Are there so many missing values for a variable that you should drop that variable from your dataset?

**3. Outliers**

I like making boxplots to visualize the outliers in a dataset. If your variables are roughly on the same scale, it is nice to use the Pandas df.boxplot() method. This creates boxplots of every continuous variable in your dataset on the same graph. If your variables have different scales, you can plot them with subplots in a loop:

continuous\_labels = list(continuous.columns)  
i = 1plt.figure(figsize=(15,30))for var in continuous\_labels: #plotting boxplot for each variable  
 plt.subplot(round(len(continuous\_labels),0)/3+3,4,i)  
 plt.boxplot(continuous[var],whis=5)  
 plt.title(var)  
 i+=1plt.tight\_layout()  
plt.show()



Partial picture of boxplots created with code above.

**Questions to ask:**

* Do you have outliers (represented as dark circles on the boxplots) in your variables?
* Why do you think you have outliers?
* Do the outliers represent real observations (i.e. not errors)?
* Should you exclude these observations? If not, should you winsorize the values?

This is a tricky question. I typically identify my outliers and then I leave them be until I have tried out some models. If I find the models have low accuracy, I will go back and re-evaluate whether I should winsorize the variable(s) with outliers (if I have no other options).

**4. Correlations/Relationships**

To create a matrix of correlations for continuous variables, all the code you need is:

df.corr()

If you want to include discrete variables, then you need to use one-hot-encoding (see next section) to transform these into numeric variables. Then they can also be included in the correlation matrix.

**Questions to ask:**

* Which variables are most correlated with your target variable? (If applicable)
* Is there multicollinearity? (Two features that have a correlation > 0.8) How will this affect your model?
* Do you have variables that represent the same information? Can one be dropped?

**5. Feature Engineering**

Let’s start with a commonly used feature engineering method: variable transformation.

**Variable Transformation**

The most common transformation is **one-hot-encoding** to transform categorical variables into numeric — binary, to be specific — variables. This is necessary because machine learning models cannot handle “object” data types. Pandas makes this easy to do:

new\_df = pd.get\_dummies(df,drop\_first=True)

Another common transformation (which is necessary for some models) is **standardizing** variables. Here is the code for that:

from sklearn.preprocessing import StandardScaler  
X\_std = StandardScaler().fit\_transform(X)

Finally, you may want to transform variables so that they follow a normal distribution, depending on the model you are using. For this, you can try np.log() , np.sqrt() , the box-cox transformation, and other functions to transform your data to better fit a normal distribution.

**Creating New Features**

This is the hardest section and requires the most critical thinking, in my opinion. There isn’t any code I can give that will apply to lots of projects — it really depends on the dataset.

Here are a couple of cases where you may want to try creating a new feature:

* You suspect that the relationship of an outcome and a feature depends on a second feature → Create an interaction variable
* You want to create linear relationships → Create quadratic or higher level functions
* You can think of variables/information that is missing from your dataset → Create this variable using a function of variables you do have

**The End**

Congratulations on making it to the end of EDA! Writing this article made me realize just how much you can learn about your data with some pretty simple lines of code. Hopefully, you have enough popcorn (and energy) left for the modeling process.