Exploratory Data Analysis: A Practical Guide and Template for Structured Data

Jiahao Weng Follow
Sep 24, 2019 · 8 min read ★

Very often, it is not the drawing that is most difficult, but rather what to draw on a blank white piece of paper that stumps people.

Similarly for data science, one may wonder how to get started after receiving a dataset. This is where Exploratory Data Analysis (EDA) comes to the rescue.

According to Wikipedia, EDA "is an approach to analyzing datasets to summarize their main characteristics, often with visual methods". In my own words, it is about knowing your data, gaining a certain amount of familiarity with the data, before one starts to extract insights from it.

Since EDA is such a crucial initial step for all data science projects, the lazy me decided to write a code template for performing EDA on structured datasets. The idea is to spend less time coding and focus more on the analysis of data itself. Scroll down to the bottom for the link to the code, but do read on to find out more about EDA and understand what the code does.

Where is EDA in the Data Science Process

Before we delve into EDA, it is important to first get a sense of where EDA fits in the whole data science process.

Source: Farcaster at English Wikipedia

Based on the process chart from Wikipedia above, after the data has been collected, it undergoes some processing before being cleaned and EDA is then performed. Notice that after EDA, we may go back to processing and cleaning of data, i.e., this can be an iterative process. Subsequently, we can then use the cleaned dataset and knowledge from EDA to perform modelling and reporting.

We can, therefore, understand the objectives of EDA as such:

To gain an understanding of data and find clues from the data,

1. *to formulate assumptions and hypothesis for our modelling; and*
2. *to check the quality of data for further processing and cleaning if necessary.*

To better illustrate the concept of EDA, we shall be using the Rossmann store sales "train.csv" data from Kaggle. Keeping in mind that the problem statement is to forecast store sales, our EDA objectives are, therefore:

1. To check for features that can help in forecasting sales; and
2. To check for anomalies or outliers that may impact our forecasting model.

General Outline of EDA

Our code template shall perform the following steps:

1. Preview data
2. Check total number of entries and column types
3. Check any null values
4. Check duplicate entries
5. Plot distribution of numeric data (univariate and pairwise joint distribution)
6. Plot count distribution of categorical data
7. Analyse time series of numeric data by daily, monthly and yearly frequencies

The necessary dependencies are as such:

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import missingno
%matplotlib inline
```

Preliminary Data Processing

To start off, we read in our dataset and generate a simple preview and statistics of our data.

```
df = pd.read_csv('train.csv')
df.info()
df.head()
```

The output shows that we have around 1 million entries with 9 columns. No null values but some of the columns' data type should be changed. As we shall see later, setting the data types correctly can aid us in our data science processes. In a nutshell, there are three common types of data type (categorical, numeric and datetime) and we have different EDA procedures for each of them.

In our preliminary processing, we changed the data types in the following way:

- Set the identifier *Store* as string.
- For columns that are categorical, i.e., columns that take on a limited, and usually fixed, number of possible values, we set their type as "category". E.g., gender, blood type and country are all categorical data.
- For columns that are numeric, we can either set their type as "int64" (integer) or "float64" (floating point number). E.g., sales, temperature and number of people are all numeric data.
- Set *Date* as "datetime64" data type.

```
# set identifier "Store" as string
df['Store'] = df['Store'].astype('str')

# set categorical data
df['DayOfWeek'] = df['DayOfWeek'].astype('category')
df['Open'] = df['Open'].astype('category')
df['Promo'] = df['Promo'].astype('category')
df['StateHoliday'] =
df['StateHoliday'].astype(str).str.strip().astype('category')
df['SchoolHoliday'] = df['SchoolHoliday'].astype('category')

# set datetime data
df['Date'] = pd.to_datetime(df['Date'])
```

Our EDA Begins

After setting the data types, we are ready to let the fun (a.k.a. EDA) begin.

To set up the party, we just have to copy, paste and run the code template that contains the various functions and subsequently, run the function *eda* that takes in a Pandas

dataframe as input.

```
eda(df)
```

And that's it! Simple :)

Let's now go through the salient parts of the output to understand how to make use of our EDA results.

Missing values and Duplicated Entries

In our case, there is no entry with missing values.

However, in the event of missing values, the code will generate a chart similar to the one below. Notice the white spaces under *CustomerID* and *Description*; those are the missing values. Thus, in one glance, we are able to know the extent of our missing values issue.

Missing entries chart

The portion of code relevant for checking missing values is as follows.

```
# generate preview of entries with null values
if len(df[df.isnull().any(axis=1)] != 0):
    print("\nPreview of data with null values:\nxxxxxxxxxxxxx")
    print(df[df.isnull().any(axis=1)].head(3))
```

```
    missingno.matrix(df)
    plt.show()
```

In our case, there are also no duplicated entries, which the code will point out directly by printing the output “No duplicated entries found”.

In the event of duplicated entries, the output will show the number of duplicated entries and a preview of these entries.

Duplicated entries sample output

Code for checking duplicated entries:

```
# generate count statistics of duplicate entries
if len(df[df.duplicated()]) > 0:
    print("No. of duplicated entries: ", len(df[df.duplicated()]))

print(df[df.duplicated(keep=False)].sort_values(by=list(df.columns)).
head())
else:
    print("No duplicated entries found")
```

Should there be any missing values or duplicated entries, you should decide on the cleaning steps required before proceeding further to the other parts of EDA.

For reference on how to treat missing values, you may wish to refer to this article by Jun Wu.

For duplicated entries, check that they are indeed duplicated and drop them by the following code.

```
df.drop_duplicates(inplace=True)
```

Categorical Data EDA

Our main EDA objective for categorical data is to know the unique values and their corresponding count.

Using the Rossmann store sales e.g., the column *Promo* indicates whether a store is running a promotion on that day. Running a count of *Promo* by its unique values shows that promotions are held quite often, taking up around 40% (388,080 / 1,017,209) of store days. This may indicate that *Promo* is an important feature in forecasting sales.

The function generating the EDA for categorical data is *categorical_eda*.

```
def categorical_eda(df):
    """Given dataframe, generate EDA of categorical data"""
    print("To check: Unique count of non-numeric data")
    print(df.select_dtypes(include=['category']).nunique())
    top5(df)
```

```
    # Plot count distribution of categorical data
    for col in df.select_dtypes(include='category').columns:
        fig = sns.catplot(x=col, kind="count", data=df)
        fig.set_xticklabels(rotation=90)
        plt.show()
```

Numeric Data EDA

For numeric data, our EDA approach is as follows:

1. Plot univariate distribution of each numeric data
2. If categorical data are available, plot univariate distribution by each categorical value
3. Plot pairwise joint distribution of numeric data

Prior to running the *eda* function, we created a new column *ave_sales* by dividing *Sales* on *Customers*, so that we can analyse the average sales per customer per store day.

The first output for our numeric data EDA shows some simple distribution statistics that include mean, standard deviation and quartiles.

Some of the points that we can derive from the output include:

- No negative values for all numeric data. If there is any negative value, it may mean that we have to further investigate since sales and number of customers are unlikely to be negative, and we may have to then clean the data.
- The max values for the numeric data are quite far off from the 75 percentile, indicating that we may have outliers.

The boxplots further confirm that there is/are outlier(s) with *Sales* more than 40,000 or *Customers* greater than 7,000. We may, therefore, conduct a check to see if this is an error or were there other special circumstances leading to the exceptional figures.

We have also plotted violin plots for each numeric data by category values to investigate for e.g., the effect of *Promo* on *Sales*.

From the violin plot above, we can infer that on days of promotion, sales typically increase as seen from the wider sections of the violin plot. The wide section of the plot near 0 *Sales* for *Promo*=0 is probably due to closed stores. Running the *eda* function

again later after removing entries with *Open*=0 shows that the wide section near 0 is no longer present, thereby confirming our hypothesis.

The last output for numeric data EDA is a pairwise joint distribution plot.

To generate further insights, we ran the function *numeric_eda* and added a parameter *hue='DayOfWeek'*. This allows us to color our pairwise plot by each day of the week.

```
numeric_eda(df, hue='DayOfWeek')
```

By incorporating the *DayOfWeek* into the plot, we noticed that Sunday's ave_sales (represent by pink color) is relatively more constant than the other days. This is, therefore, something that can be further investigated and considered as a predictor.

Time series EDA

Lastly, we also plotted time series charts to check for trends and seasonality.

For ease of analysis, our code automatically sums up the numeric data by daily, monthly and yearly frequency before plotting the charts. It achieves this through Pandas' resample method.

In our case, the yearly chart is not useful as the data only contains partial 2015 data, hence we have left it out from the screenshots.

From the daily plot, we can see that there are outliers towards the end of December 2013. This is also reflected in the monthly plot.

The monthly plot also shows that there is some seasonality, namely lower sales in February that is probably due to short month effect, as well as higher sales towards end of year. If we are predicting monthly sales, our forecasting would have to take this seasonality effect into consideration.

. . .

Through the examples above, we hope to have demonstrated how EDA is used to

formulate assumptions and hypothesis for our modelling, and to check the quality of data for further processing and cleaning.

In case you missed the code template link, here it is again.

Lastly, do note that depending on your data science problem statement, you may wish to perform additional EDA steps. Nonetheless, the code template should be able to cover the basic EDA and get you up to speed quickly.

Thanks for reading and I hope the code and article are useful. Please also feel free to comment with any questions or suggestions you may have.

References

- https://en.wikipedia.org/wiki/Exploratory_data_analysis#targetText=In%20statistics%2C%20exploratory%20data%20analysis,modeling%20or%20hypothesis%20testing%20task.
- https://towardsdatascience.com/exploratory-data-analysis-eda-techniques-for-kaggle-competition-beginners-be4237c3c3a9
- https://medium.com/@aiden.dataminer/the-data-science-method-dsm-exploratory-data-analysis-bc84d4d8d3f9

Data Science | Machine Learning | Python