<https://www.kite.com/blog/python/data-analysis-visualization-python/>

# Exploratory Data Analysis (EDA) and Data Visualization with Python

### Table of Contents

* [**Introduction**](https://www.kite.com/blog/python/data-analysis-visualization-python/#introduction)
* [**Defining Exploratory Data Analysis**](https://www.kite.com/blog/python/data-analysis-visualization-python/#defining_EDA)
* [**Overview**](https://www.kite.com/blog/python/data-analysis-visualization-python/#overview)
* [**A detailed explanation of EDA**](https://www.kite.com/blog/python/data-analysis-visualization-python/#detailed_explanation_of_EDA)
* [**Quick peek at functions**](https://www.kite.com/blog/python/data-analysis-visualization-python/#quick_peek_at_functions)
  + [**Types of variables and descriptive statistics**](https://www.kite.com/blog/python/data-analysis-visualization-python/#types-variables)
  + [**To summarize all the operations so far**](https://www.kite.com/blog/python/data-analysis-visualization-python/#summarize)
* [**Univariate and bivariate analysis**](https://www.kite.com/blog/python/data-analysis-visualization-python/#univariate_and_bivariate_analysis)
* [**Missing value analysis**](https://www.kite.com/blog/python/data-analysis-visualization-python/#missing_value_analysis)
  + [**To check if there are any null values in the dataset**](https://www.kite.com/blog/python/data-analysis-visualization-python/#check-null)
* [**Outlier detection analysis**](https://www.kite.com/blog/python/data-analysis-visualization-python/#outlier_detections_analysis)
* [**Percentile based outlier removal**](https://www.kite.com/blog/python/data-analysis-visualization-python/#percentile_based_outlier_removal)
* [**The correlation matrix**](https://www.kite.com/blog/python/data-analysis-visualization-python/#the_correlation_matrix)
* [**Conclusions**](https://www.kite.com/blog/python/data-analysis-visualization-python/#conclusions)

## Introduction

There is so much data in today’s world. Modern businesses and academics alike collect vast amounts of data on myriad processes and phenomena. While much of the world’s data is processed using Excel or (manually!), new data analysis and visualization programs allow for reaching even deeper understanding. The programming language Python, with its English commands and easy-to-follow syntax, offers an amazingly powerful (and free!) open-source alternative to traditional techniques and applications.

Data analytics allow businesses to understand their efficiency and performance, and ultimately helps the business make more informed decisions. For example, an e-commerce company might be interested in analyzing customer attributes in order to display targeted ads for improving sales. Data analysis can be applied to almost any aspect of a business if one understands the tools available to process information.

## How to analyze data using the Twitter API

If you’d like to see data analysis + data visualization in action, check out our intermediate-level tutorial on how to extract data using the Twitter API and map it out with Matplotlib and GeoPandas.

## Defining Exploratory Data Analysis

**Exploratory Data Analysis** – EDA – plays a critical role in understanding the what, why, and how of the problem statement. It’s first in the order of operations that a data analyst will perform when handed a new data source and problem statement.

Here’s a direct definition: exploratory data analysis is an approach to analyzing data sets by summarizing their main characteristics with visualizations. The EDA process is a crucial step prior to building a model in order to unravel various insights that later become important in developing a robust algorithmic model.

Let’s try to break down this definition and understand different operations where EDA comes into play:

* First and foremost, EDA provides a stage for breaking down problem statements into smaller experiments which can help understand the dataset
* EDA provides relevant insights which help analysts make key business decisions
* The EDA step provides a platform to run all thought experiments and ultimately guides us towards making a critical decision

## Overview

This post introduces key components of Exploratory Data Analysis along with a few examples to get you started on analyzing your own data. We’ll cover a few relevant theoretical explanations, as well as use sample code as an example so ultimately, you can apply these techniques to your own data set.

The main objective of the introductory article is to cover how to:

* Read and examine a dataset and classify variables by their type: quantitative vs. categorical
* Handle categorical variables with numerically coded values
* Perform univariate and bivariate analysis and derive meaningful insights about the dataset
* Identify and treat missing values and remove dataset outliers
* Build a correlation matrix to identify relevant variables

Above all, we’ll learn about the important API’s of the python packages that will help us perform various EDA techniques.

## A detailed explanation of an EDA on sales data

In this section, we’ll look into some code and learn to interpret key insights from the different operations that we perform.

Before we get started, let’s install and import all the relevant python packages which we would use for performing our analysis. Our requirements include the [**pandas**](https://www.kite.com/python/docs/pandas/), [**numpy**](https://www.kite.com/python/docs/numpy/), [**seaborn**](https://www.kite.com/python/docs/seaborn/), and [**matplotlib**](https://www.kite.com/python/docs/matplotlib/) python packages.

Python’s package management system called Pip makes things easier when it comes to tasks like installing dependencies, maintaining and shipping Python projects. Fire up your terminal and run the command below:

**import** python -m pip install --user numpy scipy matplotlib ipython pandas sympy nose statsmodels patsy seaborn

Note that you need to have Python and Pip already installed on your system for the above command to work, and the packages whose name look alien to you are the internal dependencies of the main packages that we intend to you, for now you can ignore those.

Having performed this step, we’re ready to install all our required Python dependencies. Next, we need to set up an environment where we can conduct our analysis – feel free to fire up your favorite text editing tool for Python and start with loading the following packages:

**import** pandas **as** pd

**import** numpy **as** np

**import** seaborn **as** sns

**import** matplotlib

**from** matplotlib **import** pyplot **as** plt

For reading data and performing EDA operations, we’ll primarily use the numpy and pandas Python packages, which offer simple API’s that allow us to plug our data sources and perform our desired operation. For the output, we’ll be using the Seaborn package which is a Python-based data visualization library built on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Data visualization is an important part of analysis since it allows even non-programmers to be able to decipher trends and patterns.

Let’s get started by reading the dataset we’ll be working with and deciphering its variables. For this blog post, we’ll be analyzing a [**Kaggle data set**](https://www.kaggle.com/flenderson/sales-analysis) on a company’s sales and inventory patterns. Kaggle is a great community of data scientists analyzing data together – it’s a great place to find data to practice the skills covered in this post.

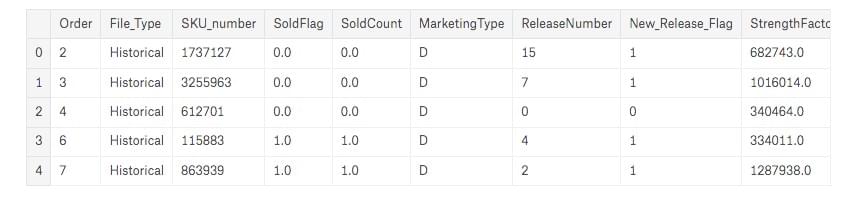
The dataset contains a detailed set of products in an inventory and the main problem statement here is to determine the products that should continue to sell, and which products to remove from the inventory. The file contains the observations of both historical sales and active inventory data. The end solution here is to create a model that will predict which products to keep and which to remove from the inventory – we’ll perform EDA on this data to understand the data better. You can follow along with a companion Kaggle notepad [**here**](https://www.kaggle.com/dvigneshwer/kernele7f4dbb964).

## Quick peek at functions: an example

Let’s analyze the dataset and take a closer look at its content. The aim here is to find details like the number of columns and other metadata which will help us to gauge size and other properties such as the range of values in the columns of the dataset.

sales\_data = pd.read\_csv('../input/SalesKaggle3.csv')

sales\_data.head()



The read\_csv function loads the entire data file to a Python environment as a Pandas dataframe and default delimiter is ‘,’ for a csv file.

The head() function returns the first 5 entries of the dataset and if you want to increase the number of rows displayed, you can specify the desired number in the head() function as an argument for ex: sales.data.head(10), similarly we can see the bottom rows of the Pandas dataframe with the command sales\_data.tail().

### Types of variables and descriptive statistics

Once we have loaded the dataset into the Python environment, our next step is understanding what these columns actually contain with respect to the range of values, learn which ones are categorical in nature etc.

To get a little more context about the data it’s necessary to understand what the columns mean with respect to the context of the business – this helps establish rules for the potential transformations that can be applied to the column values.

Here are the definitions for a few of the columns:

* **File\_Type:** The value “Active” means that the particular product needs investigation
* **SoldFlag:** The value 1 = sale, 0 = no sale in past six months
* **SKU\_number:** This is the unique identifier for each product.
* **Order:** Just a sequential counter. Can be ignored.
* **SoldFlag:** 1 = sold in past 6 mos. 0 = Not sold
* **MarketingType:** Two categories of how we market the product.
* **New\_Release\_Flag:** Any product that has had a future release (i.e., Release Number > 1)

sales\_data.describe()

The *describe* function returns a pandas series type that provides descriptive statistics which summarize the central tendency, dispersion, and shape of a dataset’s distribution, excluding NaN values. The three main numerical measures for the center of a distribution are the mode, mean(µ), and the median (M). The mode is the most frequently occurring value. The mean is the average value, while the median is the middle value.

sales\_data.describe(include='all')

When we call the describe function with include=’all’ argument it displays the descriptive statistics for all the columns, which includes the categorical columns as well.



Next, we address some of the fundamental questions:

The number of entries in the dataset:

print(sales\_data.shape)

We have 198917 rows and 14 columns.

Total number of products & unique values of the columns:

print(sales\_data.nunique())

nunique() would return the number of unique elements in each column

Count of the historical and active state, (we need only analyze the active state products):

print(sales\_data[sales\_data['File\_Type'] == 'Historical']['SKU\_number'].count())

print(sales\_data[sales\_data['File\_Type'] == 'Active']['SKU\_number'].count())

We use the count function to find the number of active and historical cases: we have 122921 active cases which needs to be analyzed. We then Split the dataset into two parts based on the flag type. To do this, we must pass the required condition in square brackets to the sales\_data object, which examines all the entries with the condition mentioned and creates a new object with only the required values.

sales\_data\_hist = sales\_data[sales\_data['File\_Type'] == 'Historical']

sales\_data\_act = sales\_data[sales\_data['File\_Type'] == 'Active']



### To summarize all the operations so far:

The dataset contains 198,917 rows and 14 columns with 12 numerical and 2 categorical columns. There are 122,921 actively sold products in the dataset, which is where we’ll focus our analysis.

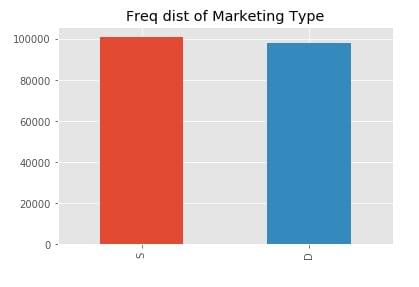
## Univariate and bivariate analysis

The data associated with each attribute includes a long list of values (both numeric and not), and having these values as a long series is not particularly useful yet – they don’t provide any standalone insight. In order to convert the raw data into information we can actually use, we need to summarize and then examine the variable’s **distribution**.

The univariate distribution plots are graphs where we plot the histograms along with the estimated probability density function over the data. It’s one of the simplest techniques where we consider a single variable and observe its spread and statical properties. The univariate analysis for numerical and categorical attributes are different.

For categorical columns we plot histograms, we use the value\_count() and plot.bar() functions to draw a bar plot, which is commonly used for representing categorical data using rectangular bars with value counts of the categorical values. In this case, we have two type of marketing types S and D. The bar plot shows comparisons among these discrete categories, with the x-axis showing the specific categories and the y-axis the measured value.

sales\_data['MarketingType'].value\_counts().plot.bar(title='Freq dist of Marketing Type')



Similarly, by changing the column name in the code above, we can analyze every categorical column.

Below is the code to plot the univariate distribution of the numerical columns which contains the histograms and the estimated PDF. We use displot of the seaborn library to plot this graph:

col\_names = ['StrengthFactor','PriceReg', 'ReleaseYear', 'ItemCount', 'LowUserPrice', 'LowNetPrice']

fig, ax = plt.subplots(len(col\_names), figsize=(16,12))

**for** i, col\_val **in** enumerate(col\_names):

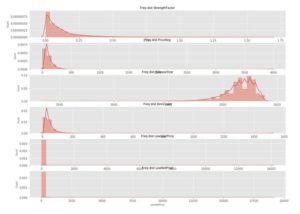
sns.distplot(sales\_data\_hist[col\_val], hist=True, ax=ax[i])

ax[i].set\_title('Freq dist '+col\_val, fontsize=10)

ax[i].set\_xlabel(col\_val, fontsize=8)

ax[i].set\_ylabel('Count', fontsize=8)

plt.show()



We can see that leaving the *ReleaseYear* column every other column is skewed to the left which indicates most of the values lie in the lower range values and vice versa in the case of a *ReleaseYear* attribute.

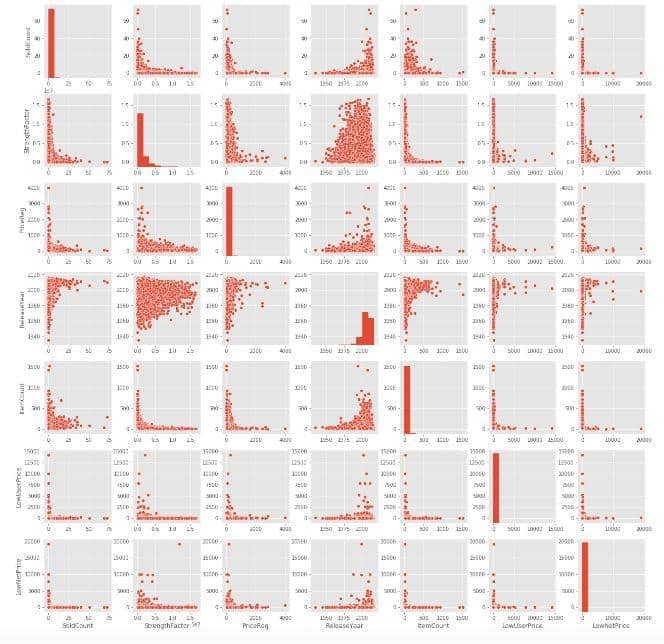
The bivariate distribution plots help us to study the relationship between two variables by analyzing the scatter plot, and we use the pairplot() function of the seaborn package to plot the bivariate distributions:

sales\_data\_hist = sales\_data\_hist.drop([

'Order', 'File\_Type','SKU\_number','SoldFlag','MarketingType','ReleaseNumber','New\_Release\_Flag'

], axis=1)

sns.pairplot(sales\_data\_hist)



We often look out for scatter plots that follow a clear linear pattern with an either increasing or decreasing slope so that we can draw conclusions, but don’t notice these patterns in this particular dataset. That said, there’s always room to derive other insights that might be useful by comparing the nature of the plots between the variables of interest.

## Missing value analysis

Missing values in the dataset refer to those fields which are empty or no values assigned to them, these usually occur due to data entry errors, faults that occur with data collection processes and often while joining multiple columns from different tables we find a condition which leads to missing values. There are numerous ways with which missing values are treated the easiest ones are to replace the missing value with the mean, median, mode or a constant value (we come to a value based on the domain knowledge) and another alternative is to remove the entry from the dataset itself.

In our dataset we don’t have missing values, thus we are not performing any operations on the dataset that said here are few sample code snippets that will help you perform missing value treatment in python.

### To check if there are any null values in the dataset

data\_frame.isnull().values.any()

If the above snippet returns true then there are null values in the dataset and false means there are none

data\_frame.isnull().sum()

The above snippet returns the total number of missing values across different columns

Now in order to replace the missing values, we use the fillna function of pandas to replace na values with the value of our interest and inplace=True command makes the permanently changes the value in that dataframe.

data\_frame['col\_name'].fillna(0, inplace=True)

## Outlier detection analysis

An outlier might indicate a **mistake** in the data (like a typo, or a measuring error, seasonal effects etc), in which case it should be corrected or removed from the data before calculating summary statistics or deriving insights from the data, failing to which will lead to incorrect analysis.

Below is the code to plot the box plot of all the column names mentioned in the list col\_names. The box plot allows us to visually analyze the outliers in the dataset.

The key terminology to note here are as follows:

* The range of the data provides us with a measure of spread and is equal to a value between the smallest data point (min) and the largest one (Max)
* The interquartile range (IQR), which is the range covered by the middle 50% of the data.
* IQR = Q3 – Q1, the difference between the third and first quartiles. The first quartile (Q1) is the value such that one quarter (25%) of the data points fall below it, or the median of the bottom half of the data. The third quartile is the value such that three quarters (75%) of the data points fall below it, or the median of the top half of the data.
* The IQR can be used to detect outliers using the 1.5(IQR) criteria. Outliers are observations that fall below Q1 – 1.5(IQR) or above Q3 + 1.5(IQR).

col\_names = ['StrengthFactor','PriceReg', 'ReleaseYear', 'ItemCount', 'LowUserPrice', 'LowNetPrice']

fig, ax = plt.subplots(len(col\_names), figsize=(8,40))

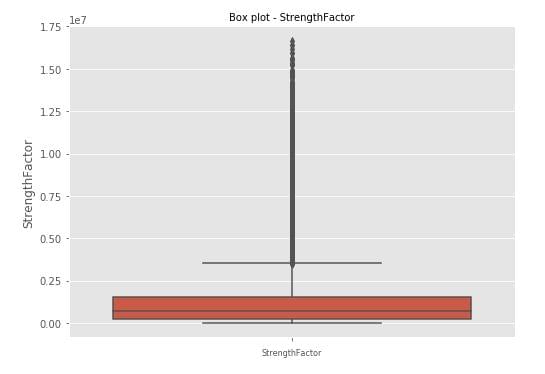
**for** i, col\_val **in** enumerate(col\_names):

sns.boxplot(y=sales\_data\_hist[col\_val], ax=ax[i])

ax[i].set\_title('Box plot - {}'.format(col\_val), fontsize=10)

ax[i].set\_xlabel(col\_val, fontsize=8)

plt.show()



Based on the above definition of how we identify outliers the black dots are outliers in the strength factor attribute and the red colored box is the IQR range.

## Percentile based outlier removal

The next step that comes to our mind is the ways by which we can remove these outliers. One of the most popularly used technique is the Percentile based outlier removal, where we filter out outliers based on fixed percentile values. The other techniques in this category include removal based on z-score, constant values etc

**def** **percentile\_based\_outlier**(data, threshold=95):

diff = (100 - threshold) / 2

minval, maxval = np.percentile(data, [diff, 100 - diff])

**return** (data < minval) | (data > maxval)

col\_names = ['StrengthFactor','PriceReg', 'ReleaseYear', 'ItemCount', 'LowUserPrice', 'LowNetPrice']

fig, ax = plt.subplots(len(col\_names), figsize=(8,40))

**for** i, col\_val **in** enumerate(col\_names):

x = sales\_data\_hist[col\_val][:1000]

sns.distplot(x, ax=ax[i], rug=True, hist=False)

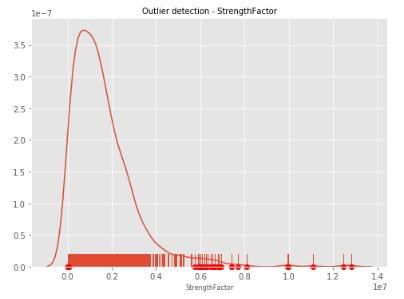
outliers = x[percentile\_based\_outlier(x)]

ax[i].plot(outliers, np.zeros\_like(outliers), 'ro', clip\_on=False)

ax[i].set\_title('Outlier detection - {}'.format(col\_val), fontsize=10)

ax[i].set\_xlabel(col\_val, fontsize=8)

plt.show()



The values marked with a dot below in the x-axis of the graph are the ones that are removed from the column based on the set threshold percentile (95 in our case), and is also the default value when it comes to percentile-based outlier removal.

## The correlation matrix

A correlation matrix is a table showing the value of the correlation coefficient (Correlation coefficients are used in statistics to measure how strong a relationship is between two variables. ) between sets of variables. Each attribute of the dataset is compared with the other attributes to find out the correlation coefficient. This analysis allows you to see which pairs have the highest correlation, the pairs which are highly correlated represent the same variance of the dataset thus we can further analyze them to understand which attribute among the pairs are most significant for building the model.

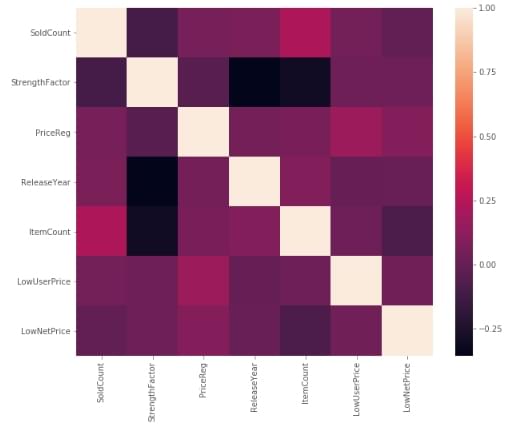
f, ax = plt.subplots(figsize=(10, 8))

corr = sales\_data\_hist.corr()

sns.heatmap(corr,

xticklabels=corr.columns.values,

yticklabels=corr.columns.values)



Above you can see the correlation network of all the variables selected, correlation value lies between -1 to +1. Highly correlated variables will have correlation value close to +1 and less correlated variables will have correlation value close to -1.

In this dataset, we don’t see any attributes to be correlated and the diagonal elements of the matrix value are always 1 as we are finding the correlation between the same columns thus the inference here is that all the numerical attributes are important and needs to be considered for building the model.

## Conclusions

Ultimately, there’s no limit to the number of experiments one can perform in the EDA process – it completely depends on what you’re analyzing, as well as the knowledge of packages such as Pandas and matplotlib our job becomes easier.

The code from our example is also available [**here**](https://www.kaggle.com/dvigneshwer/kernele7f4dbb964). The code is pretty straightforward, and you can clone the kernel and apply it to a dataset of your choice. If you’re interested in expanding your EDA toolkit even further, you may want to look into more advanced techniques such as advance missing value treatments that use regression-based techniques, or even consider exploring multivariate factor and cluster analysis.

These techniques are usually used when there are many attributes to analyze, and many of them represent the same information, often containing hundreds of variables – depending on the domain. Usually for model building, we consider 30-40 odd variables, in which case performing more advanced techniques is necessary to come up with factor variables that better represent the variance in the dataset.

Once you practice the example in this post, go forth and analyze your own data! Pretty much any process that generates data would benefit from the analysis techniques we used here, so there are a lot of opportunities to put your new skills to work. Share your progress in the comments below, I’d love to help if needed and hear about your experiences!

*Vigneshwer is a data scientist at Epsilon, where he crunches real-time data and builds state-of-the-art AI algorithms for complex business problems. He believes that technology needs to have a human-centric design to cater solutions to a diverse audience. He’s an official Mozilla TechSpeaker and is also the author of Rust Cookbook.*

*This post is a part of Kite’s new series on Python. You can check out the code from this and other posts on our*[***GitHub repository***](https://github.com/kiteco/kite-python-blog-post-code)*.*