

2020

Lab 5: Data Exploratory Analysis



Dr. Mohammed Al-Sarem
Taibah University, Information System
Department
2/22/2020

Lab 5: Data Exploratory Analysis

Lab Objectives:

Data mining tasks aim at extracting hidden information from the data. According to CRISP-DM, this process goes through well-defined steps. Data preparation is the most time-consuming and labor-intensive task. Before moving ahead, data scientist might explore the data to get a general overview. In this lab you will first get familiar with basic data visualization techniques. After completion of this module, you will be able to explore data graphically in Python using:

- histogram
- boxplot
- bar chart
- scatter plot

Methodology

In this lab we will use the Iris Flower Species Dataset. This dataset involves the prediction of iris flower species. Your task till now is to download the dataset and save it into your current working directory with the filename **iris.csv** (details how to download the dataset and where you find it is discussed in lab 4).

In class task:

At the end of this lab, the student will be able to:

- Load dataset to Python Jupyter.
- Write a complete Python code that explores data visually.

home task:

Starting from this lab you will be able to do your course project. To do that, first go to <https://archive.ics.uci.edu/ml/index.php> and explore the available data sets. Choose one dataset and download it. This lab will give you an overview on how to do Data Exploratory Analysis (DEA). Later, in the remain labs, more techniques and machine learning will be discussed.

References:

- https://www.shanelynn.ie/python-pandas-read_csv-load-data-from-csv-files/
- Open access data sets: <https://archive.ics.uci.edu/ml/index.php>
- NumPy library: <https://numpy.org/>
- Pandas library: <https://pandas.pydata.org/>
- Matplotlib library: <https://matplotlib.org/>
- Seaborn library: <https://seaborn.pydata.org/>

Lab 5: Data Exploratory Analysis

This tutorial is divided into the following parts:

- Exploring briefly what is Exploratory Data Analysis (EDA), how to perform EDA, and the required python libraries for EDA
- Working on the dataset: dealing with missing values, dropping irrelevant features, renaming data frames columns, etc.

1. Exploratory Data Analysis (EDA)

1.1. What is Exploratory Data Analysis?

Exploratory Data Analysis or (EDA) is understanding the data sets by summarizing their main characteristics often plotting them visually. This step is very important especially when we arrive at modeling the data in order to apply Machine learning. Plotting in EDA consists of Histograms, Box plot, Scatter plot and many more. It often takes much time to explore the data. Through the process of EDA, we can ask to define the problem statement or definition on our data set which is very important.

1.2. How to perform Exploratory Data Analysis?

This is one such question that everyone is keen on knowing the answer. Well, the answer is it depends on the data set that you are working. There is no one method or common methods in order to perform EDA, whereas in this tutorial you can understand some common methods and plots that would be used in the EDA process.

1.3. Importing the required libraries for EDA

From the previous lab, you should be familiar with *Numpy* library and you learnt how to apply it to find some important statistical values such mean, mode, standard division etc. In this lab, we will explore more python libraries that make the data scientist life very easier. Below are the libraries that are used in order to perform EDA:

- Numpy (for dealing with mathematical operations on arrays)
- Pandas (for data manipulation and analysis)
- Matplotlib (for data visualization)
- Seaborn (also for data visualization)

Lab 5: Data Exploratory Analysis

2. Working on data set

Let's first import the required libraries to *jupyter* notebook. In this lab, you have import the following libraries. Three new libraries: pandas, matplotlib and seaborn are introduced. To explore what they provide for data scientist, visit the URL at the first page). The code below presents how to import these libraries:

Import the required libraries

```
In [*]: # Importing required libraries.
import pandas as pd
import numpy as np
import seaborn as sns #visualisation
import matplotlib.pyplot as plt #visualisation
%matplotlib inline
sns.set(color_codes=True)
```

Since the *jupyter* notebook is a browser-based interactive data analysis tool that can combine narrative, code, graphics, HTML elements, and much more into a single executable document. Plotting interactively within an IPython notebook can be done with the `%matplotlib` command, and works in a similar way to the IPython shell. In the IPython notebook, you also have the option of embedding graphics directly in the notebook, with two possible options:

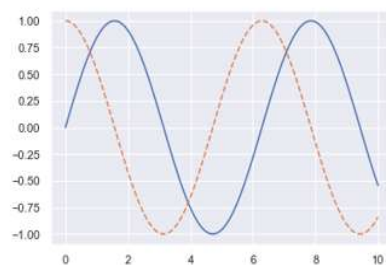
- `%matplotlib notebook` will lead to *interactive* plots embedded within the notebook
- `%matplotlib inline` will lead to *static* images of your plot embedded in the notebook

In this lab, we will generally opt for `%matplotlib inline`. Explore the code below:

```
In [2]: x = np.linspace(0, 10, 100)

fig = plt.figure()
plt.plot(x, np.sin(x), '-')
plt.plot(x, np.cos(x), '--')

Out[2]: [matplotlib.lines.Line2D at 0x2733baa8688>]
```



2.1 Loading the data into the data frame.

In the previous lab, you explored how to upload the iris.csv dataset using numpy. It is time now to explore a new way to upload the dataset into your jupyter notebook. In this time, instead of using numpy method `genfromtxt()`, we use pandas method `read_csv()`.

Lab 5: Data Exploratory Analysis

The code below reads the dataset from a directory defined by the user and displays the first five rows of data.

```
from sklearn import datasets
iris= datasets.load_iris()
df= pd.read_csv(iris.filename)
df.head(5)
```

Write down what is displayed on your browser!

A table of the first five rows in iris dataset

The `read_csv()` method is used to read the file from file *.csv. To explore the full functionality of `read_csv()` method, use `help()` method. The `read_csv()` has several parameters that you should learn how to use them properly. Among of them are:

- **filepath_or_buffer**
- **delimiter**
- **header**
- **names**

Exercise 1.1: Demonstrate how to use `read_csv()` method with the parameters listed above! Use `iris.csv` data set and display the outputs of each parameter separately!

filepath_or_buffer is used to specify the url for the dataset

delimiter is for the alias for the separation

header is for determining if the header is skipped or not and the row number to start from

names is for naming the columns of the dataset

Try to set the `header` parameter = `None`! Can you explain what was happen? the column names are now numbered from zero

Lab 5: Data Exploratory Analysis

Good! Get the name of iris dataset attributes. To do that, call `iris.feature_names`. It is expected to get the following: `['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']`. Now, set the value of `names` parameter to equal `['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']`. Write the code below:

```
df = pd.read_csv iris.filename, names=['sepal length (cm)', 'sepal width (cm)',  
    'petal length (cm)', 'petal width (cm)']
```

How many attributes found in *iris* data set??? 5 which of them is the target class? the fifth one. (hint: use `info()` method to explore number of columns, count of records, and dataset type!)

Exercise 1.2: Your observation, Is there any missing values?

No

2.2 Operations

The answer for exercise 1.2 can be also found by invoking the `shape` method. Below the code snippet which shows the shape of data present in data.

```
print(iris.shape) (150, 5) (call the right name in case you  
change the data frame name)
```

To know how many data points for a class, use the target class name that you got in the previous task as follows:

```
print(iris["target_calss"].value_counts())
```

Write your observation below. Is the data balanced? _____

150. Yes it is balanced

Lab 5: Data Exploratory Analysis

2.3 High-Level Statistics

Pandas *describe()* is used to view some basic statistical details like percentile, mean, std, etc. of a data frame or a series of numeric values.

```
In [70]: print(df_with_columns["Variety"].value_counts())
```

2	50
1	50
0	50

Name: Variety, dtype: int64

```
In [71]: print(df_with_columns.describe())
```

	sepal length (cm)	sepal width (cm)	petal length (cm) \
count	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000
std	0.828066	0.435866	1.765298
min	4.300000	2.000000	1.000000
25%	5.100000	2.000000	1.600000
50%	5.800000	3.000000	4.350000
75%	6.400000	3.300000	5.100000
max	7.900000	4.400000	6.900000

	petal width (cm)	Variety
count	150.000000	150.000000
mean	1.199333	1.000000
std	0.762238	0.819232
min	0.100000	0.000000
25%	0.300000	0.000000
50%	1.300000	1.000000
75%	1.800000	2.000000
max	2.500000	2.000000

3. Data Visualization

We now have a basic idea about the data. We need to extend that with some visualizations. We are going to look at two types of plots:

- Univariate plots to better understand each attribute.
- Bi-Variate plots to better understand the relationships between attributes.

3.1 Univariate Plots

We start with some univariate plots, that is, plots of each individual variable.

3.1.1 Distribution Plots

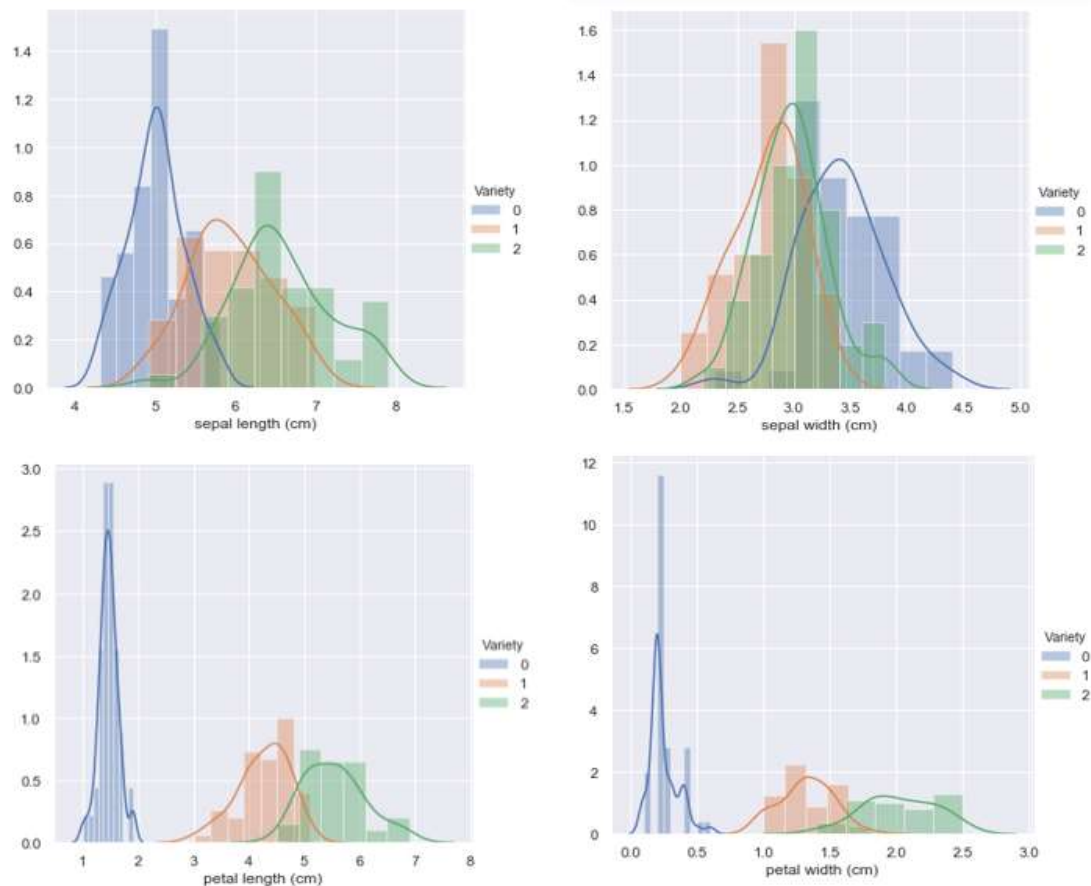
Distribution plots are used to visually assess how the data points are distributed with respect to its frequency.

- Usually the data points are grouped into bins and the height of the bars representing each group increases with increase in the number of data points lie within that group. (histogram).
- Probability Density Function (PDF) is the probability that the variable takes a value x. (smoothed version of the histogram)
- Kernel Density Estimate (KDE) is the way to estimate the PDF. The area under the KDE curve is 1.

The height of the bar denotes the percentage of data points under the corresponding group.

Lab 5: Data Exploratory Analysis

```
In [75]: for o, feature in enumerate(list(df_with_columns.columns)[:1]):  
         fg = sns.FacetGrid(df_with_columns, hue='Variety', height=5)  
         fg.map(sns.distplot, feature).add_legend()  
         plt.show()
```



3.1.2 Box Plots

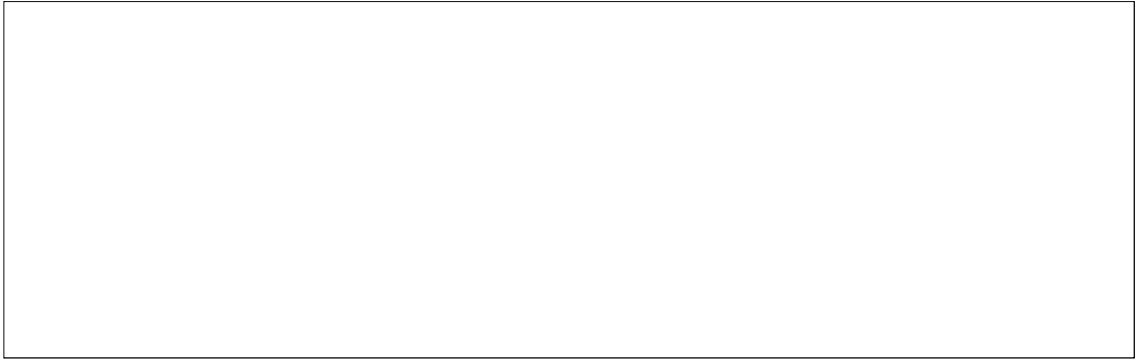
You can also analysis data using plot like Boxplot Contour and more, Seaborn library has wide variety of data plotting module. A boxplot is a graph that gives you a good indication of how the values in the data are spread out. Box plot takes a less space and visually represents the five number summary of the data points in a box. The outliers are displayed as points outside the box.

- $Q1 - 1.5 * IQR$
- $Q1$ (25th percentile)
- $Q2$ (50th percentile or median)
- $Q3$ (75th percentile)
- $Q3 + 1.5 * IQR$
- Inter Quartile Range = $Q3 - Q1$

```
sns.boxplot(x='Variety', y='petal length (cm)', data=df_with_columns)  
plt.show()
```


Lab 5: Data Exploratory Analysis

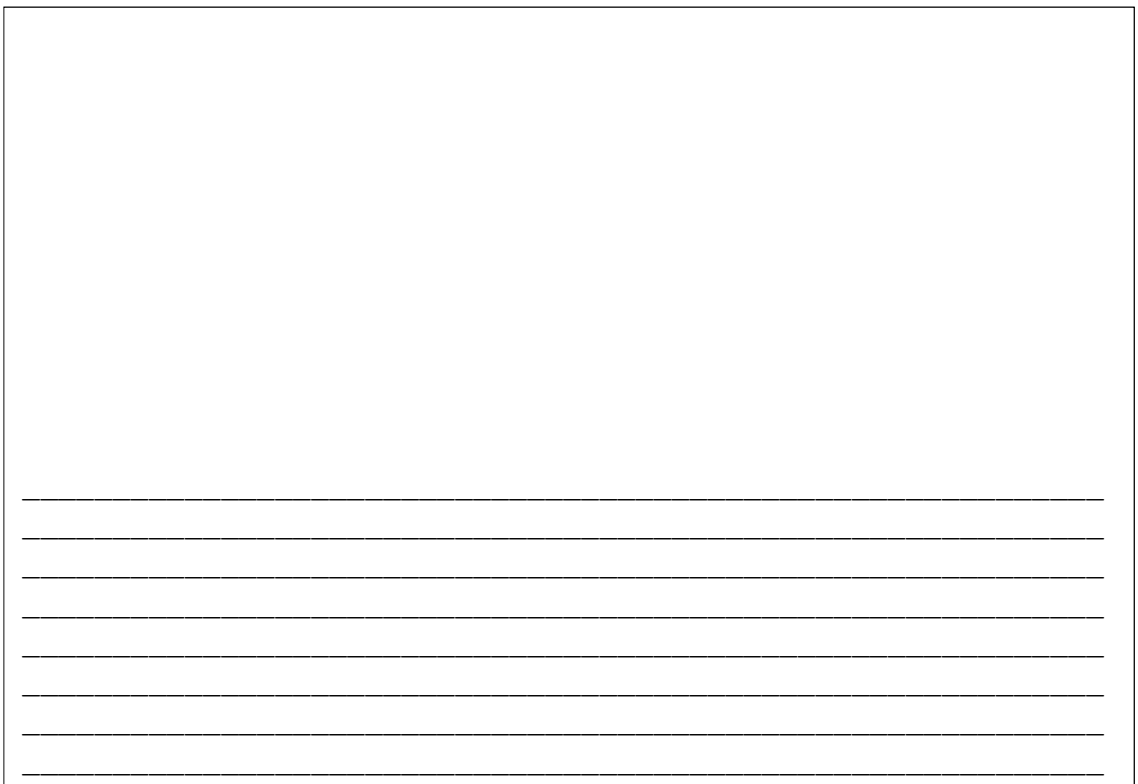
Set the figure below:



Write your observation below.

sns boxplot is useful when we want to compare data between groups

Exercise 2.1: Plot the graph for the other columns and explain your findings below:



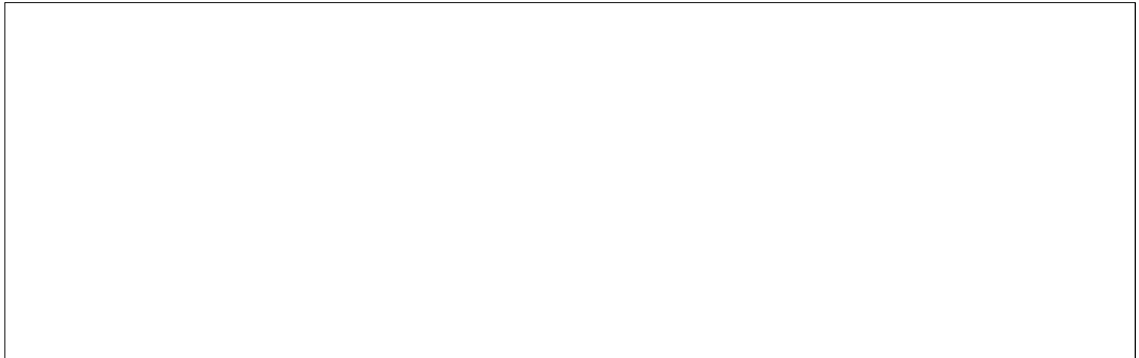
Lab 5: Data Exploratory Analysis

3.1.3 Violin Plots

Violin plot is the combination of a box plot and probability density function. It is same as Box whiskers plot, only difference is instead of box, histogram will represent **spread of data**. A violin plot is created using the *violinplot()* method, as follows:

```
sns.boxplot(x='Variety',y='petal length (cm)', data=df_with_columns)
plt.show()
```

Exercise 2.2: Set the figure and discuss your findings below!

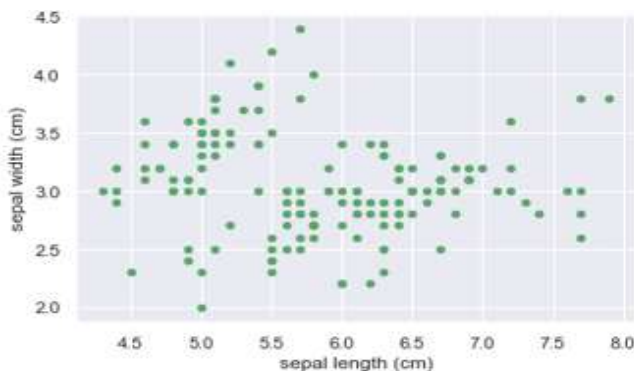


3.2 Bi-Variate analysis

3.2.1 Scatter Plots

A Scatter (XY) Plot has points that show the relationship between two sets of data. **Scatter plots** show how much one variable is affected by another.

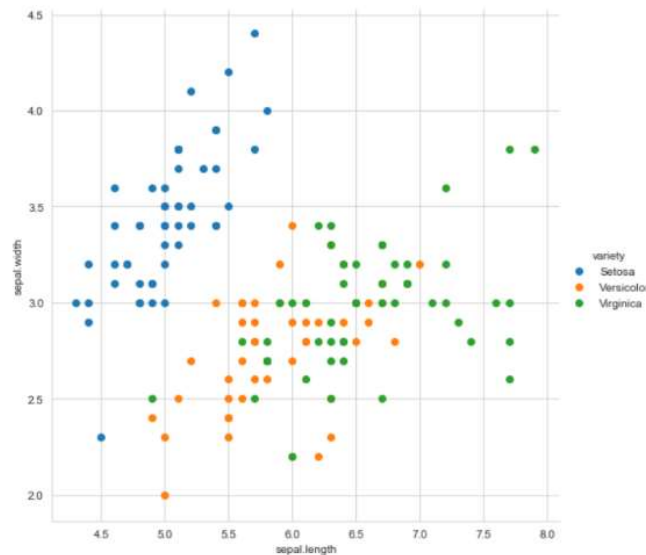
```
df_with_columns.plot(kind='scatter',x='sepal length (cm)',y='sepal width (cm)', c= 'g')
plt.show()
```



A scatter plot is a two-dimensional data visualization that uses dots to represent the values obtained for two different variables — one plotted along the x-axis and the other plotted along the y-axis.

Lab 5: Data Exploratory Analysis

```
sns.set_style("whitegrid");
sns.FacetGrid(df_with_columns, hue="Variety", height=10) \
    .map(plt.scatter, "sepal length (cm)", "sepal width (cm)") \
    .add_legend();
plt.show();
```



From above plot, we can see that Setosa is very well separated than that of Versicolor and Virginica. By using sepal.length and sepal.width we can distinguish Setosa flowers from others. Separating versicolor and virginica is very much harder as they have considerable overlap.

3.2.2 Pair Plots

A *pairplot* plot a pairwise relationships in a dataset. The *pairplot* function creates a grid of Axes such that each variable in data will be shared in the y-axis across a single row and in the x-axis across a single column.

```
sns.set_style("whitegrid");
sns.pairplot(df_with_columns, vars = iris.feature_names, hue = 'Variety', diag_kind = 'kde',
             plot_kws = {'alpha': 0.6, 's': 80, 'edgecolor': 'k'}, size = 2)
plt.show()
```

The plot between petal.length and petal.width is comparatively better. While Setosa can be easily identified Versicolor and Virginica have some overlap. We can use “if-else” condition to build a simple model to classify the flower types.

Lab 5: Data Exploratory Analysis

