Data Visualization Guide for Multi-dimensional Data

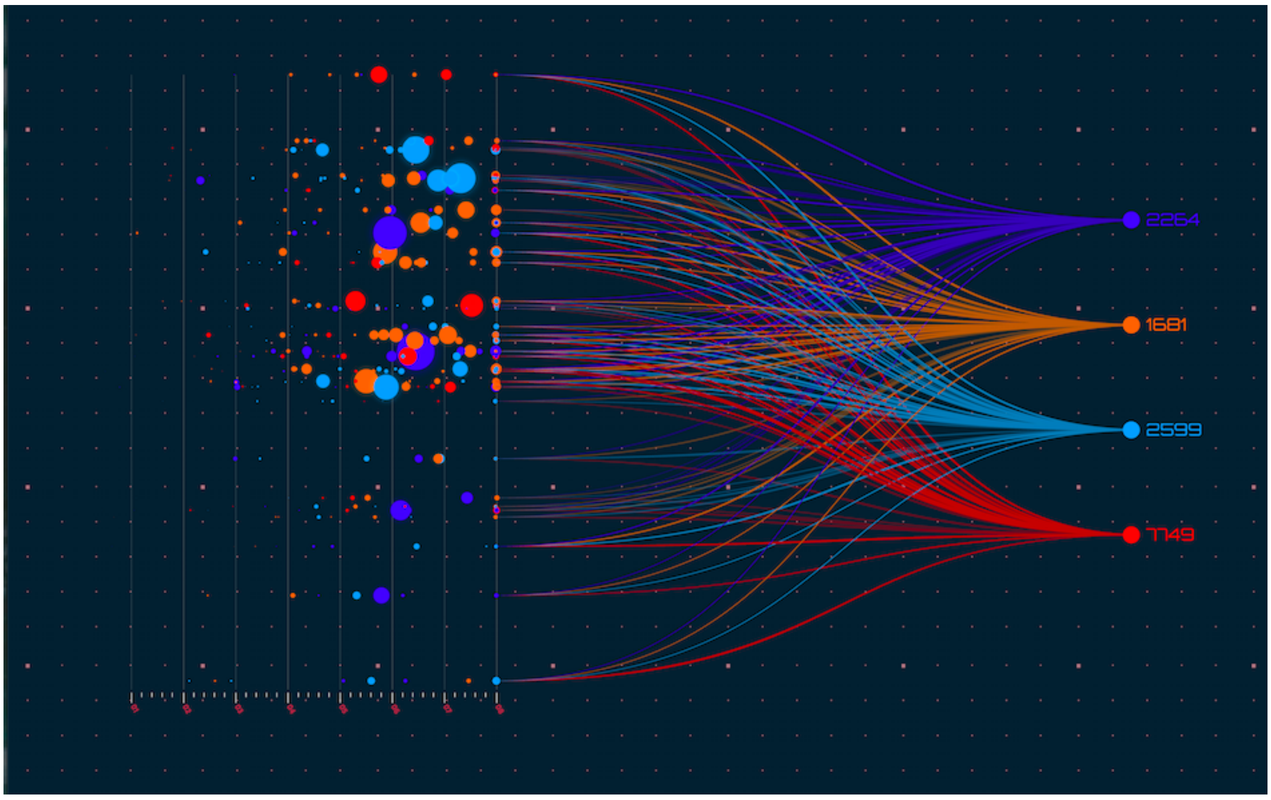
BEGINNER DATA VISUALIZATION GRAPH THEORY GRAPHS & NETWORKS R

This article was published as a part of the Data Science Blogathon.

Introduction

One can view numbers and play with numbers if they are statisticians to get some valuable insights from the raw data, but in the real world do we know whether our stakeholders are that much into statistics? Majorly answers will be straight **"no"**, to give your series of knowledge and insights a **storytelling** nature, we need to provide it in the form of **Data Visualization** hence converting your numeric data into insightful **graphs/plots/charts**.

In this article, our focus will be primarily on **multi-dimensional** data. I have dedicated my previous article solely to **one-dimensional** data; that's why focusing on N-D data is more important in enterprise case studies.



Source: Tree House Technology Group

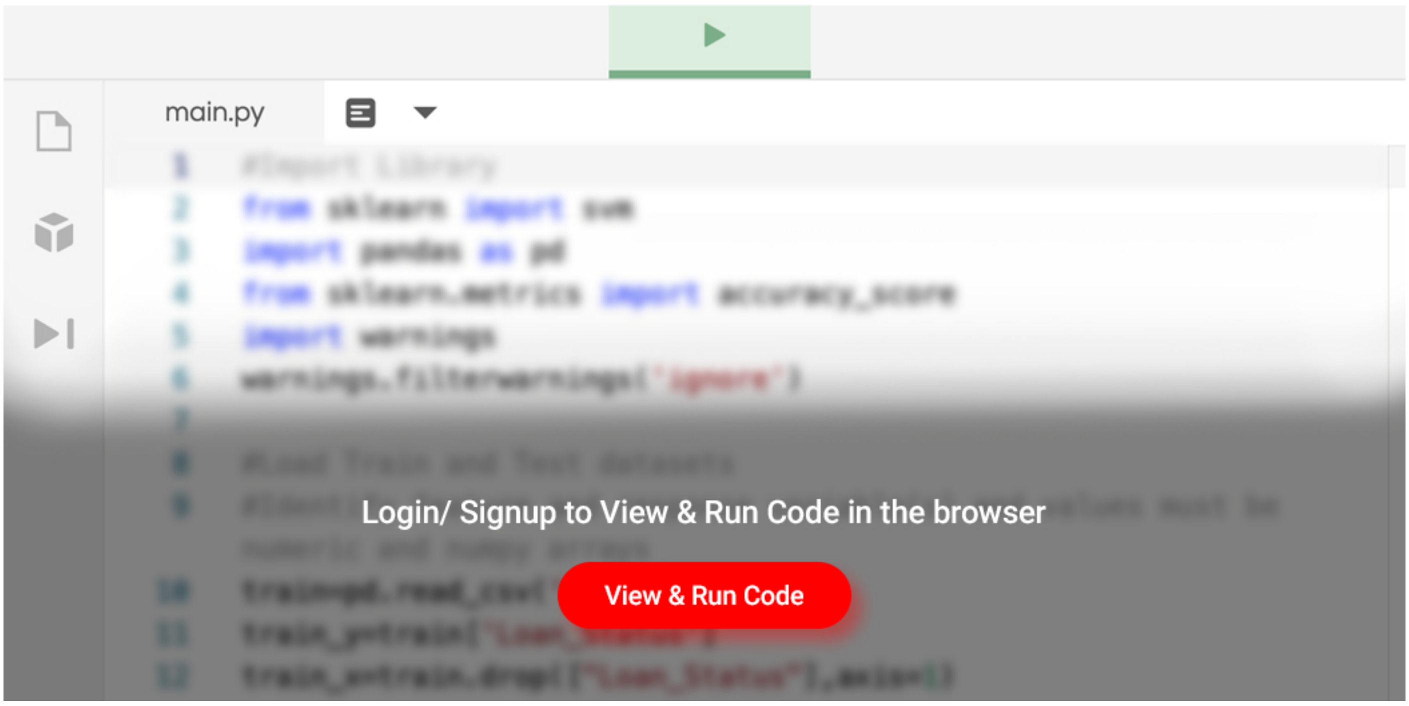
Importing all the Necessary Packages

Starting with importing the relevant libraries like **NumPy** (handling mathematical calculations), **pandas** (DataFrame manipulations), **matplotlib** (The OG visualization library closest to python interpreter and **"C"** development), and last but not least- **seaborn** (built on top of matplotlib it give way more options and better look and feels comparative).

To explore higher dimensional data and the relationships between data attributes, we'll load the file **Diabetes.csv**. It's from Kaggle.

Inference: We will be using two datasets for this article, one will be the **diabetes patients dataset,** and another one is the height and weight of the persons. In the above output, we can see a glimpse of the first dataset using the **head**() method.

Python Code:



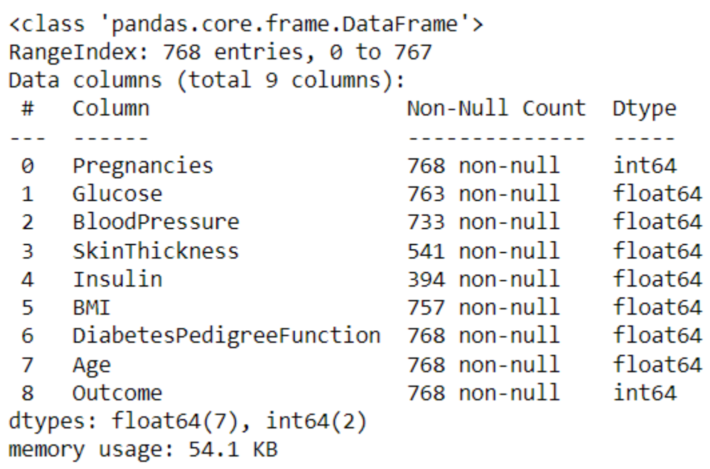
```
#Import Library
from sklearn import svm
import pandas as pd
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')

#Load Train and Test datasets
#Identi[illegible] values must be
numeric and numpy arrays
train=pd.read_csv([illegible]
train_y=train['Loan_Status']
train_x=train.drop(["Loan_Status"],axis=1)
```

Inference: Removing the **missing or junk** values from the dataset is always the priority step. Here we are first replacing the **0** value with **NaN** as we had 0 as the bad data in our **features** column.

```
df.info()
```

Output:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   763 non-null    float64
 2   BloodPressure             733 non-null    float64
 3   SkinThickness             541 non-null    float64
 4   Insulin                   394 non-null    float64
 5   BMI                       757 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    float64
 8   Outcome                   768 non-null    int64  
dtypes: float64(7), int64(2)
memory usage: 54.1 KB
```

Inference: Now we can see that in some of the columns, there are a few Null values like **Skin thickness, Insulin, BMI**, etc.

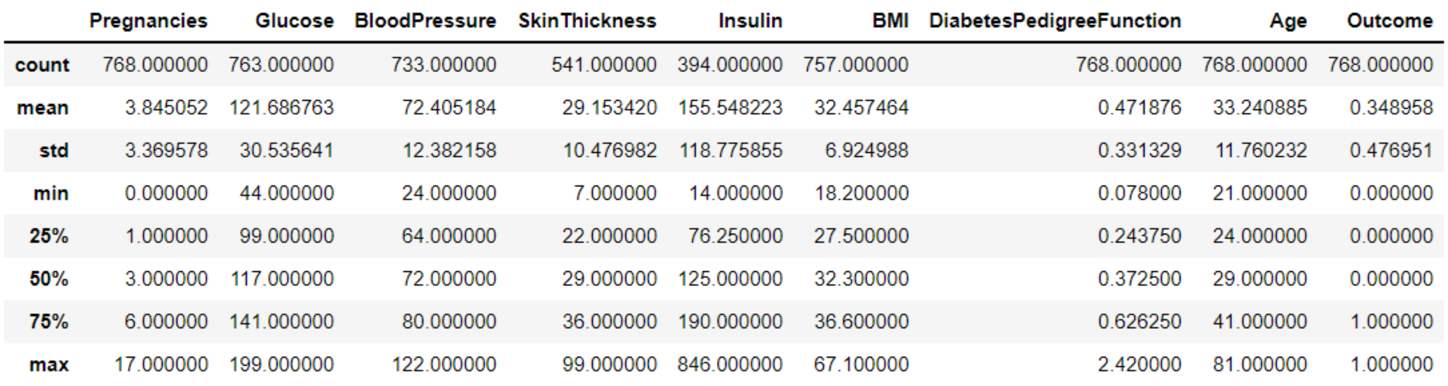
Viewing the Data

So, surprisingly no one it's useful to view the data. Straight up by using head, we can see that this dataset is utilizing **0** to represent no value – unless some poor unfortunate soul has a skin thickness of 0.

If we want to do more than expect the data, we can use the **described** function we talked about in the previous section.

```
df.describe()
```

Output:



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	763.000000	733.000000	541.000000	394.000000	757.000000	768.000000	768.000000	768.000000
mean	3.845052	121.686763	72.405184	29.153420	155.548223	32.457464	0.471876	33.240885	0.348958
std	3.369578	30.535641	12.382158	10.476982	118.775855	6.924988	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	64.000000	22.000000	76.250000	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	29.000000	125.000000	32.300000	0.372500	29.000000	0.000000
75%	6.000000	141.000000	80.000000	36.000000	190.000000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Scatter Matrix

Scatter Matrix is one of the best plots to determine the relationship (**linear** mostly) between the **multiple variables** of all the datasets; when you will see the linear graph between two or more variables that indicates the **high correlation** between those features, it can be either **positive** or **negative** correlation.

```
pd.plotting.scatter_matrix(df, figsize=(10, 10));
```

Output:

Inference: From the above plot, we can say that this plot alone is quite **descriptive** as it is showing us the linear relationship between all the variables in the dataset. For example, we can see **that skin Thickness** and **BMI** is sharing linear tendencies.

Note: Due to the big names of columns, we are facing a bit issue while reading the same though that can be improved (out of the scope of the article).

```
df2 = df.dropna() colors = df2["Outcome"].map(lambda x: "#44d9ff" if x else "#f95b4a")
pd.plotting.scatter_matrix(df2, figsize=(10,10), color=colors);
```

Output:

Inference: The scatter plot gives us both the histograms for the distributions **along the diagonal** and also a lot of 2D scatter plots **off-diagonal**. Not that this is a symmetric matrix, so I just look at the diagonal and below it normally. We can see that some variables have a lot of scattering, and some are correlated (ie, there is a direction in their scatter). This leads us to another type of plot i.e., **correlation plot**.

Correlation Plots

Before going into a deep discussion with the correlation plot, we first need to understand the correlation and for that reason, we are using the **pandas' corr()** method that will return the **Pearson's correlation coefficient** between two data inputs. In a nutshell, these plots easily quantify which variables or attributes are correlated with each other.

```
df.corr()
```

Output:

```
sb.set(rc={'figure.figsize':(11,6)}) sb.heatmap(df.corr());
```

Output:

Inference: In a seaborn or matplotlib supported **correlation plot,** we can compare the higher and lower correlation between the variables using its color palette and scale. In the above graph, **the lighter the color, the more the correlation and vice versa**. There are some drawbacks in this plot which we will get rid of in the very next graph.

```
sb.heatmap(df.corr(), annot=True, cmap="viridis", fmt="0.2f");
```

Output:

Inference: Now one can see this is a **symmetric matrix** too. But it immediately allows us to point out the most **correlated** and **anti-correlated attributes**. Some might just be common sense – Pregnancies v Age for example – but some might give us a real insight into the data.

Here, we have also used some parameters like **annot= True** so that we can see correlated values and some formatting as well.

2D Histograms

2D Histograms are mainly used for **image processing,** showing the **intensities of pixels** at a certain position of the image. Similarly, we can also use it for other problem statements, where we need to analyze two or more variables as **two-dimensional** or **three-dimensional** histograms, which provide multi-dimensional Data.

For the rest of this section, we're going to use a different dataset with more data.

Note: 2-D histograms are very useful when you have a **lot** of data. See here for the API.

```
df2 = pd.read_csv("height_weight.csv") df2.info() df2.describe()
```

Output:

```
plt.hist2d(df2["height"], df2["weight"], bins=20, cmap="magma") plt.xlabel("Height") plt.ylabel("Weight");
```

Output:

Inference: We have also worked with one-dimensional Histograms for multi-dimensional Data, but that is for **univariate analysis** now if we want to get the data distribution of more than one feature then we have to shift our focus to **2-D Histograms**. In the above 2-D graph height and weight is plotted against each other, keeping the C-MAP as **magma**.

Contour plots

Bit hard to get information from the 2D histogram, isn't it? Too much noise in the image. What if we try and contour diagram? We'll have to bin the data ourselves.

Every alternative comes into the picture when the original one has some drawbacks, Similarly, in the case with **2-D histograms** it becomes a bit hard to get the information from it as there is soo much noise in the graph. Hence now, we will go with **a contour plot**

Here is the resource that can help you deep dive, into this plot. The contour API is here.

```
hist, x_edge, y_edge = np.histogram2d(df2["height"], df2["weight"], bins=20) x_center = 0.5 * (x_edge[1:] +
x_edge[:-1]) y_center = 0.5 * (y_edge[1:] + y_edge[:-1])
```

```
plt.contour(x_center, y_center, hist, levels=4) plt.xlabel("Height") plt.ylabel("Weight");
```

Output:

Inference: Now we can see that this contour plot which is way better than a complex and noisy 2-D Histogram as it shows the clear distribution between **height and weight** simultaneously. There is still room for improvement. If we will use the **KDE plot from seaborn,** then the same contours will be smoothened and more clearly informative.

Conclusion

From the very beginning of the article, we are primarily focussing on data visualization for the multi-dimensional data, and in this journey, we got through all the important graphs/plots that could derive business-related insights from the numeric data from multiple features all at once. In the last section, we will cover all these graphs in a nutshell.

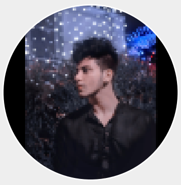
1. Firstly we got introduced to **a scatter matrix** that shows us the relationship of every variable with the other one. Then using seaborn **heat map** is used to get a better approach to **multivariable analysis**.
2. Then came the **2-D histograms,** where we can go with binary variable analysis, i.e., 2 variables can be simultaneously seen, and we can get insights from them.
3. At last, we got to know about **the Contour plot,** which helped us to get a better version of 2-D histograms as it **removes the noise** from the image and has a more clear interpretation.

Here's the repo link to this article. I hope you liked my article on **the Data visualization guide for multi-dimensional data.** If you have any opinions or questions, then comment below.

Connect with me on LinkedIn for further discussion.

The media shown in this article is not owned by Analytics Vidhya and is used at the Author's discretion.

Article Url - https://www.analyticsvidhya.com/blog/2022/09/data-visualization-guide-for-multi-dimensional-data/



Aman Preet Gulati