- **NumPy:** For handling mathematical operations without any hassle.
- **Matplotlib:** Library will help implement the **CDF plot** (we will discuss more on this).
- **Seaborn:** A Data visualization library built on top of matplotlib; we will use this one more widely in this article.
- **Pandas:** DataFrame manipulation library, we will use it for creating **DataFrame**.

```
d1 = np.loadtxt("example_1.txt")
d2 = np.loadtxt("example_2.txt")
print(d1.shape, d2.shape)
```

**Output:**

```
(500,) (500,)
```

**Inference:** Reading two text-based datasets from **the load txt** function of **NumPy** and looking at the output, one can point out that there are **500 data points** in each dataset.

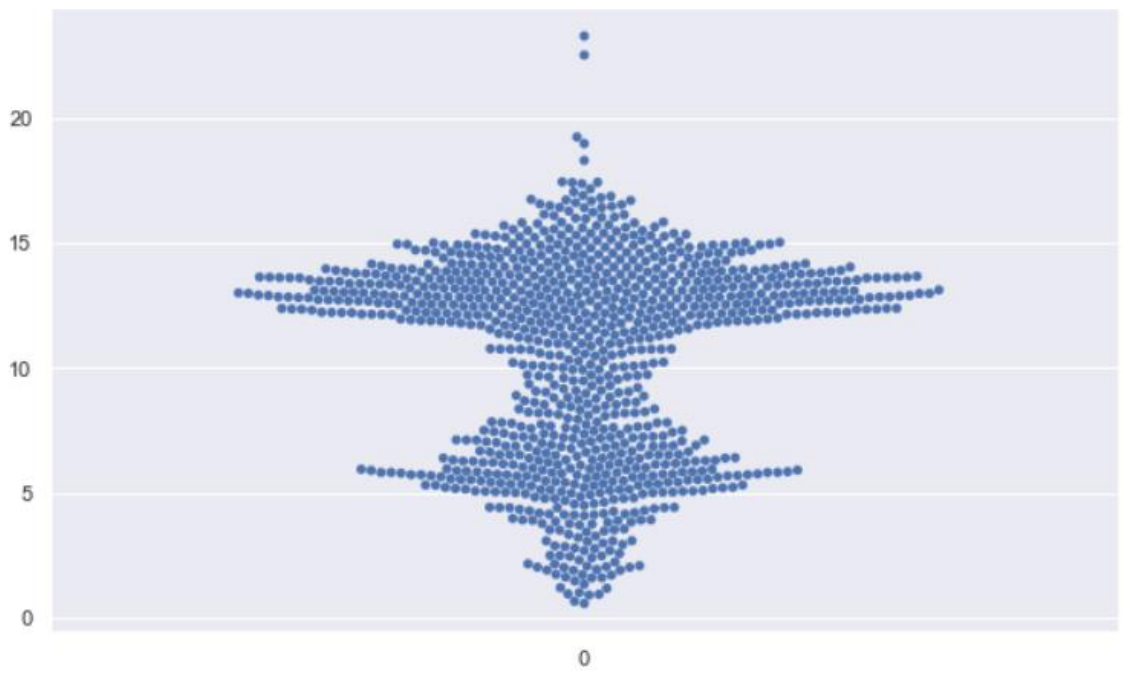
## Bee Swarm Plots in Data Visualization

Starting with the **Bee Swarm Plots**, The best thing about this plot is that along with showing the **distribution of the dataset,** we can also **see each data point** by managing the **size** of those data points; make sure you provide a suitable size; otherwise, for bigger data point distribution will tend to get out from canvas while for the relatively **smaller size** of data

**Inference:** Swarm plots are like the scatter plots (which show the complete data representation) but with **categorical data** in the axis, which differentiate them from the general **scatter plots**. Hence, for that reason, we are first creating **DataFrame,** which has some values and its categorical type.

```
sb.swarmplot(data = dataset["value"], size=5);
```

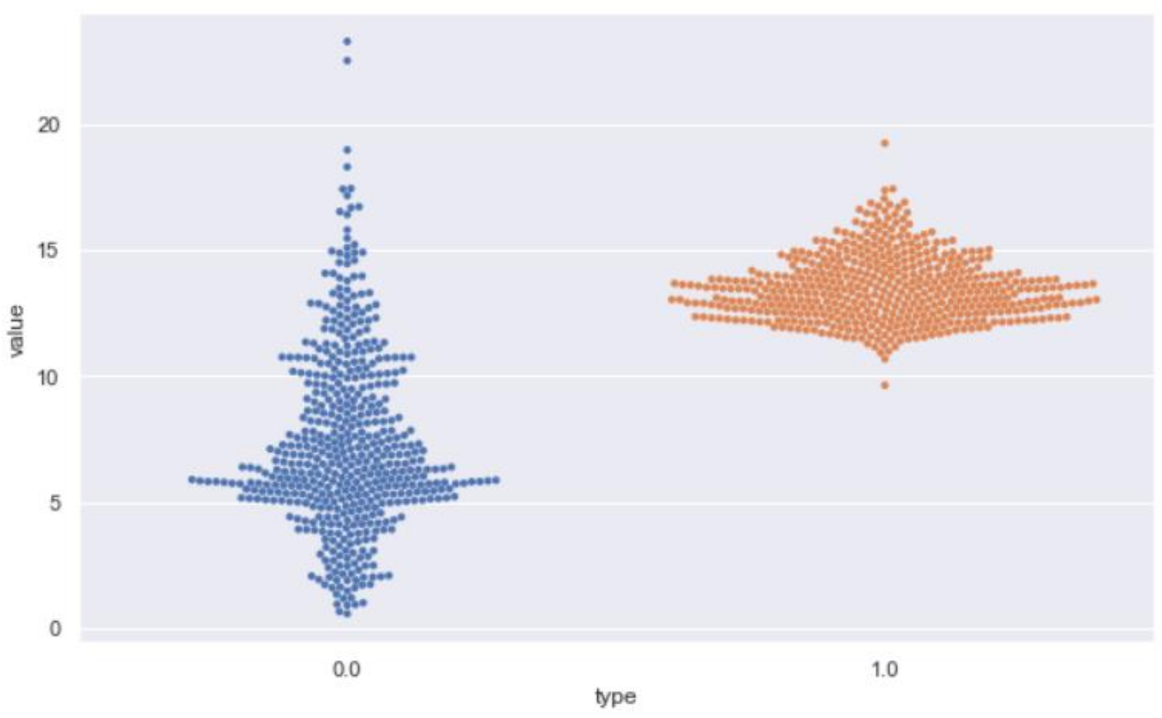
**Output:**



**Inference:** To explain the swarm plot and keep the simplicity in this example, we are plotting only **one categorical type**. In the above plot, we can see where the data points are more (denser) and where the data points diverge **away from the center** (both left and right).

```
sb.swarmplot(x="type", y="value", data=dataset, size=4);
```

**Output:**



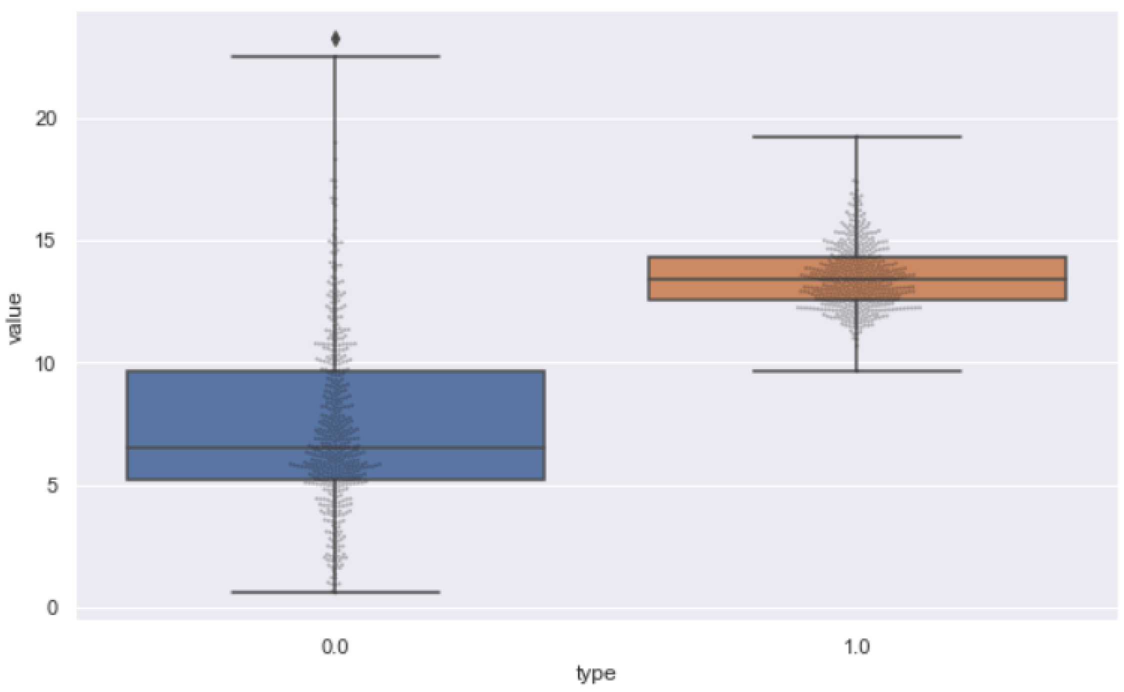
**Inference:** In the previous graph, we plotted only one category, this time, we are planning two, which are visible on the X axis labeled as **0.0 and 1.0,** and the concept is the same, i.e., the more the density, the more diverge will be data points from the center to both **left and right**.

## Box Plots in Data Visualization

Now it’s time to move forward and discuss another type of data visualization graph, i.e., **Box plot,** which is also widely known as box and whiskers plots as both the end has **cat-like whiskers**, along with that it put forward a wide variety of inferences in the table that we will discuss now.

```
sb.boxplot(x="type", y="value", data=dataset, whis=3.0);
sb.swarmplot(x="type", y="value", data=dataset, size=2, color="k", alpha=0.3);
```

**Output:**



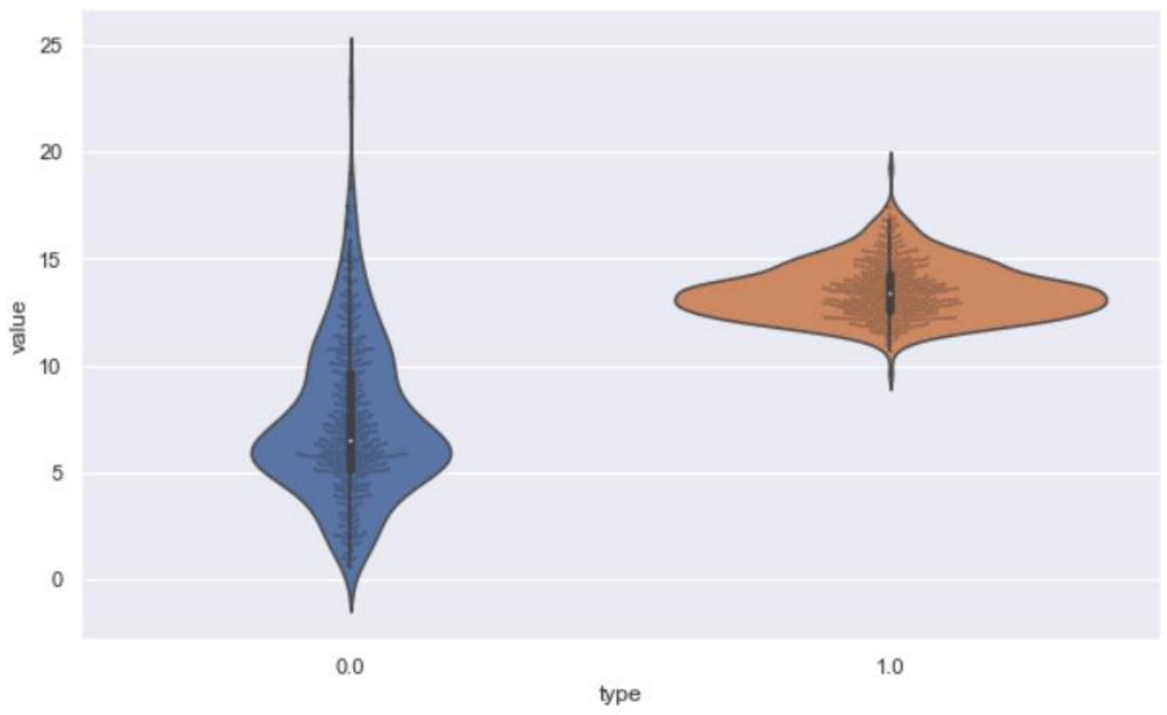
**Inference:** BoxPlot is yet another visualization plot best suited for **1-D** data, As it demonstrates a variety of statistical measures such as **outliers, median, 25th percentile, 50th percentile, and 75th percentile**. One can notice that along with the boxplot, we are also imputing the swarm plot on top of that, keeping the **alpha value less** so that we can see more of the boxplot and the swarm should not overshadow the same. This is a perfect combination where we can see the **accurate distribution** of the dataset along with its **density**.

## Violin Plots in Data Visualization

Okay, so a box plot doesn't display too much, does it? What if we want a hint more information? **The answer is** – We can use **the Violin plot** for the same; we summoned this plot because the boxplot could not show the complete distribution. All it was showing is a summary of statistics, but in the case of **the violin plot,** it is best in the case of **multimodal data** (when data has more than one peak).

```
sb.violinplot(x="type", y="value", data=dataset);
sb.swarmplot(x="type", y="value", data=dataset, size=2, color="k", alpha=0.3);
```

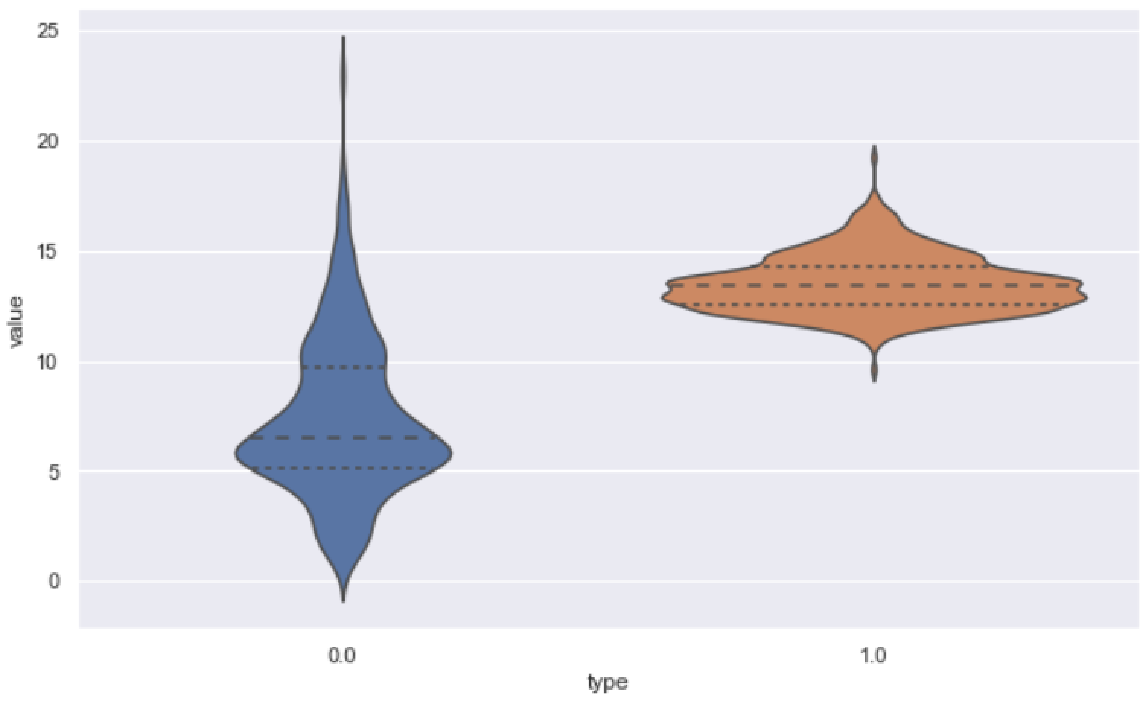
**Output:**



**Inference:** We have used a Swarm plot with a Violin plot where the swarm is doing its job, i.e., showing the **data representation** from each data on the other side violin plot shows us the **complete distribution** for both types we have in our self-created DataFrame.

```
sb.violinplot(x="type", y="value", data=dataset, inner="quartile", bw=0.2);
```

**Output:**



**Inference:** In **seaborn or matplotlib,** we always have the option to change the plot based on our needs by providing the required parameters. In violin plot also we have that flexibility, and for that, we have used two different parameters:

- **Inner**: This will stimulate how the data points will be represented in the violin plot (internally). We have multiple options like **box, quartile, point**, etc. A quartile is a value we chose, and the output is visible acc to us where the plot has internally changed the look and feel.
- **BW:** This one is to manage the **kernel bandwidth** and how **rigid and smooth** we want the corners to be in our violin plot.

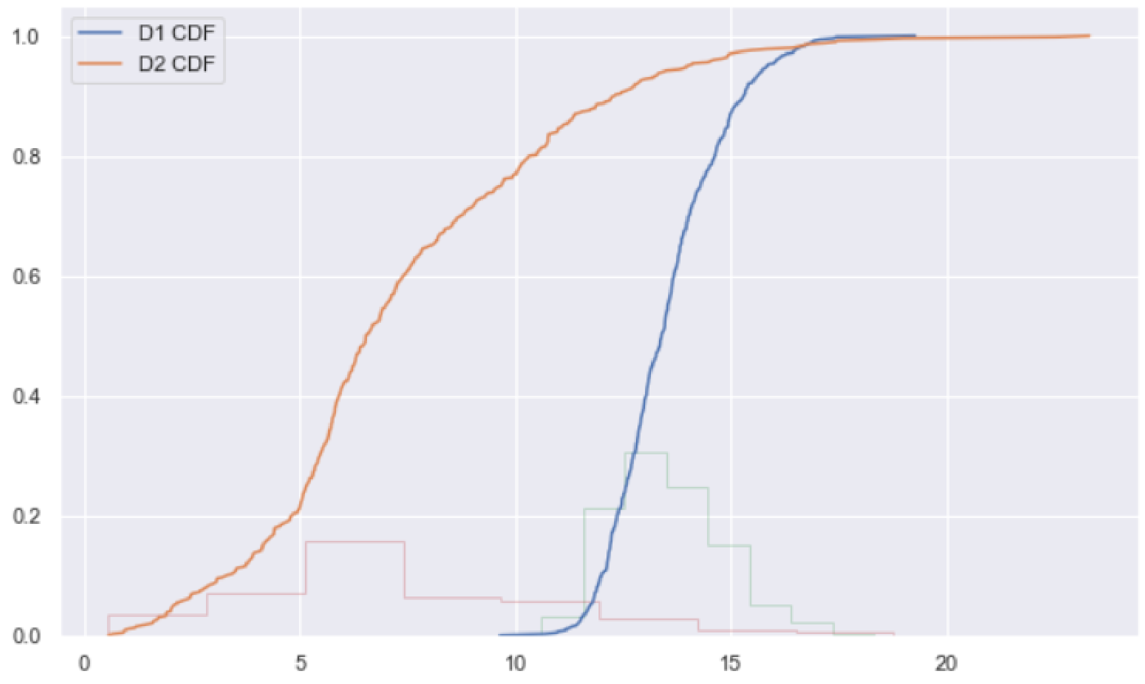
## Empirical Cumulative Distribution Functions

When you form a histogram, the fact that you have to bin data means that the looks can change significantly when you change in size. And each bin has statistical uncertainty. You can get past that using a CDF. It’s harder – visually – to see features in the PDF when looking at the CDF; however, it's generally more useful when you are trying to make quantitative comparisons between multiple distributions. We'll get on that later.

```
sd1 = np.sort(d1)
sd2 = np.sort(d2)
cdf = np.linspace(1/d1.size, 1, d1.size)

plt.plot(sd1, cdf, label="D1 CDF")
plt.plot(sd2, cdf, label="D2 CDF")
plt.hist(d1, histtype="step", density=True, alpha=0.3)
plt.hist(d2, histtype="step", density=True, alpha=0.3)
plt.legend();
```

**Output:**



**Inference:** CDF plots are not usually recommended for extracting insights for business needs. They are a bit complex to understand, but they can be a good resource for further analytics. There is an exciting relationship between **Histograms** and **CDF plots where** Histograms represent the **area of the bar** because of its nature of representation. At the same time, **CDF is cumulative** as it returns the **integral of PDF**.

## Conclusion

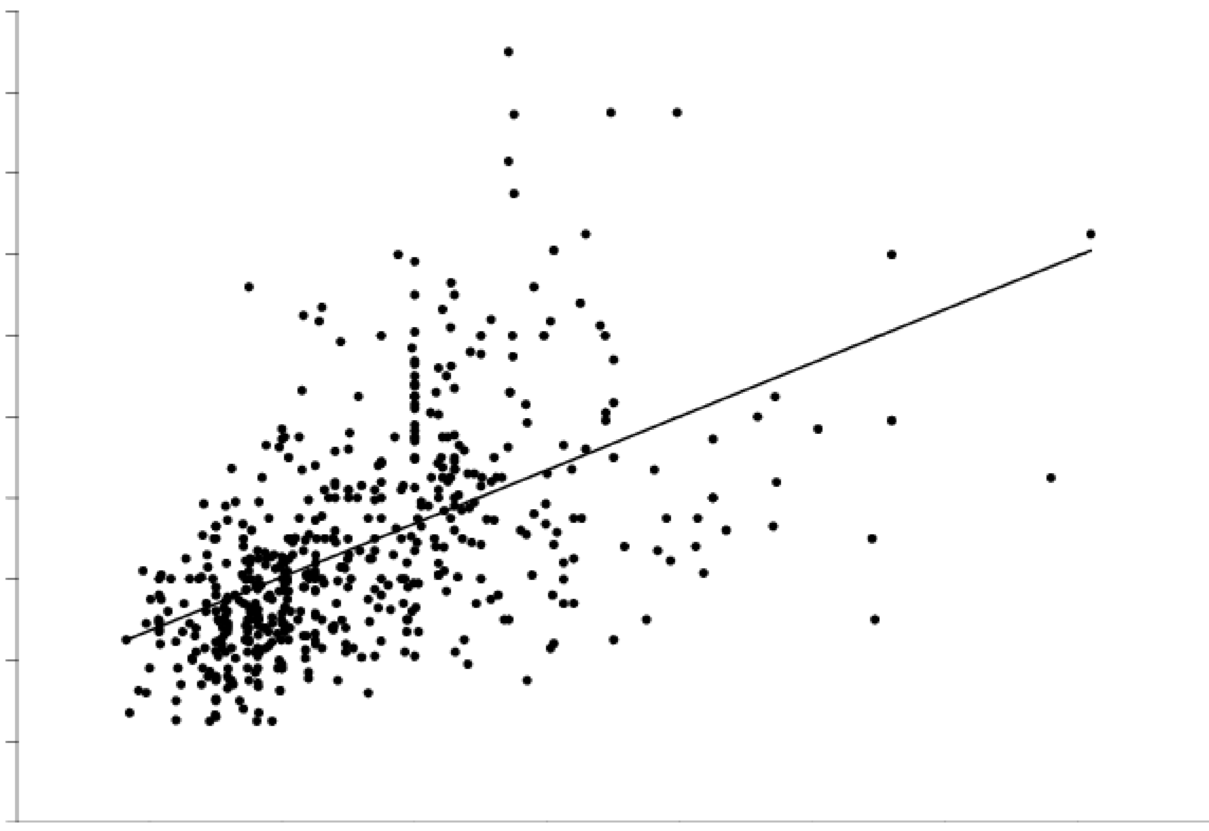
We are in the end game now. This is the last section of the article, where we will point out all the stuff that we have learned so far about **visualizing the data when it is in format of one-dimensional**. In a nutshell, we will give a brief explanation to get you to know the flow and key takeaways from the article.

1. Things started with **the Bee Swarm Plot,** where we learned about the perks and cons of this plot which helped us know **when to use** this plot to represent data distribution.
2. Then comes the **box and violin** plot, which echoed similar functionalities and output. Here we learned how the violin plot could be a better alternative when showing the **data distribution completely** and not only **summary statistics**.
3. The last plot was **Empirical Cumulative Distribution Functions (CDF),** where we saw the usage of this plot and also came to know the interesting relationship between **Histograms and CDF.**

## Data visualization on two dimensional data

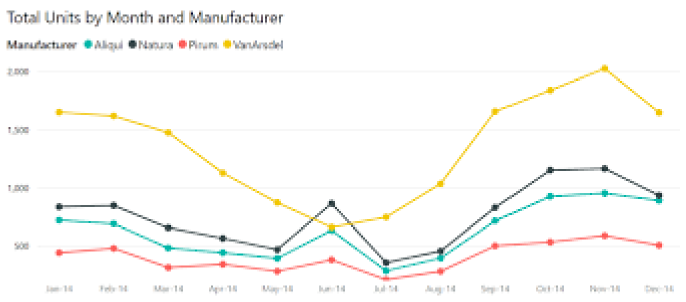
Data visualization is a powerful tool for exploring and communicating patterns, trends, and insights within two-dimensional data. Here are several common types of visualizations you can use for two-dimensional data:

1. **Scatter Plots:**
    - Scatter plots are effective for displaying the relationship between two continuous variables.
    - Each point on the plot represents an observation, with one variable on the x-axis and the other on the y-axis.

- 

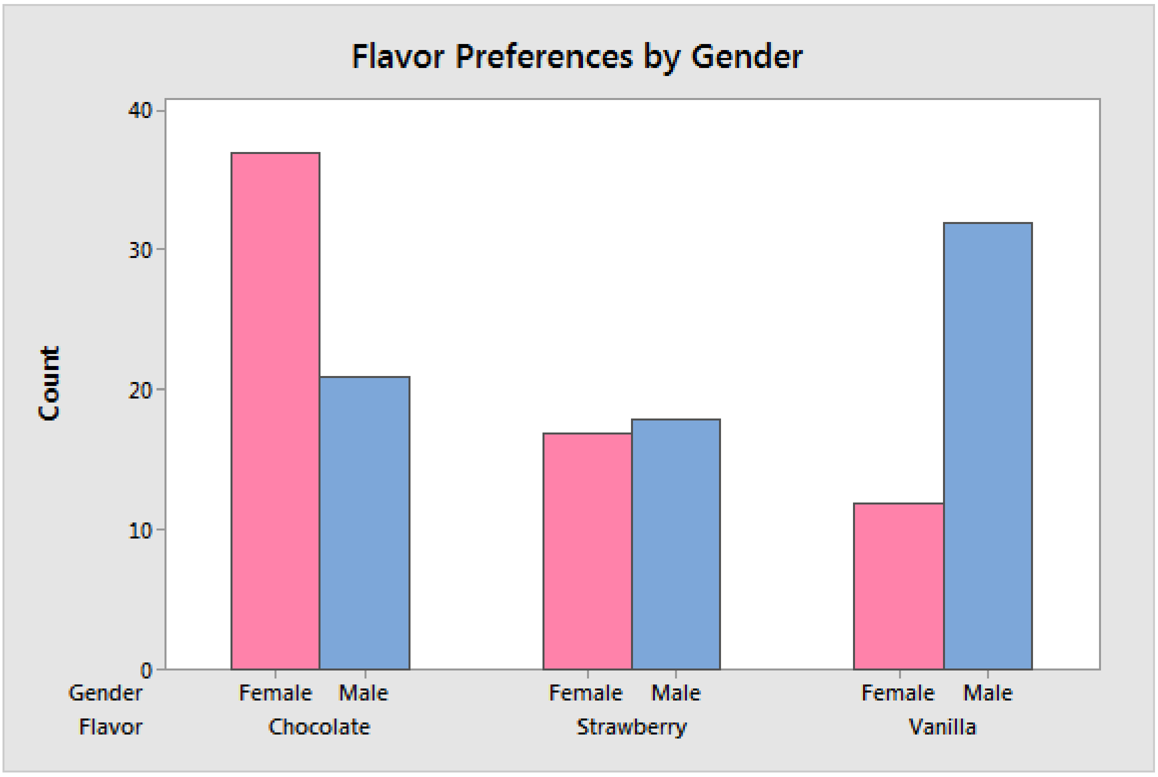
2. **Line Charts:**

   - Line charts are useful for showing trends or patterns over time.
   - They connect data points with lines, making it easy to observe changes in the two variables.



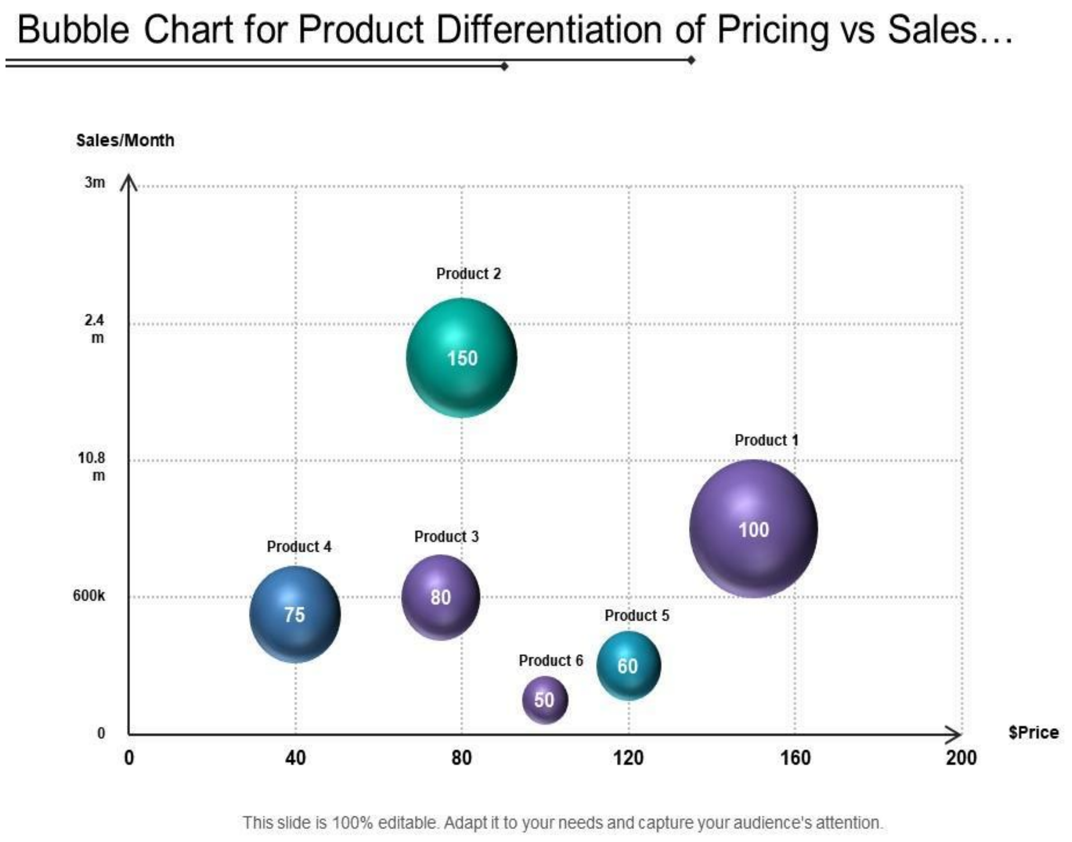
3. **Bar Charts:**

   - Bar charts are suitable for comparing the values of two variables across different categories.
   - The length of each bar represents the magnitude of the variable.



4. **Bubble Charts:**

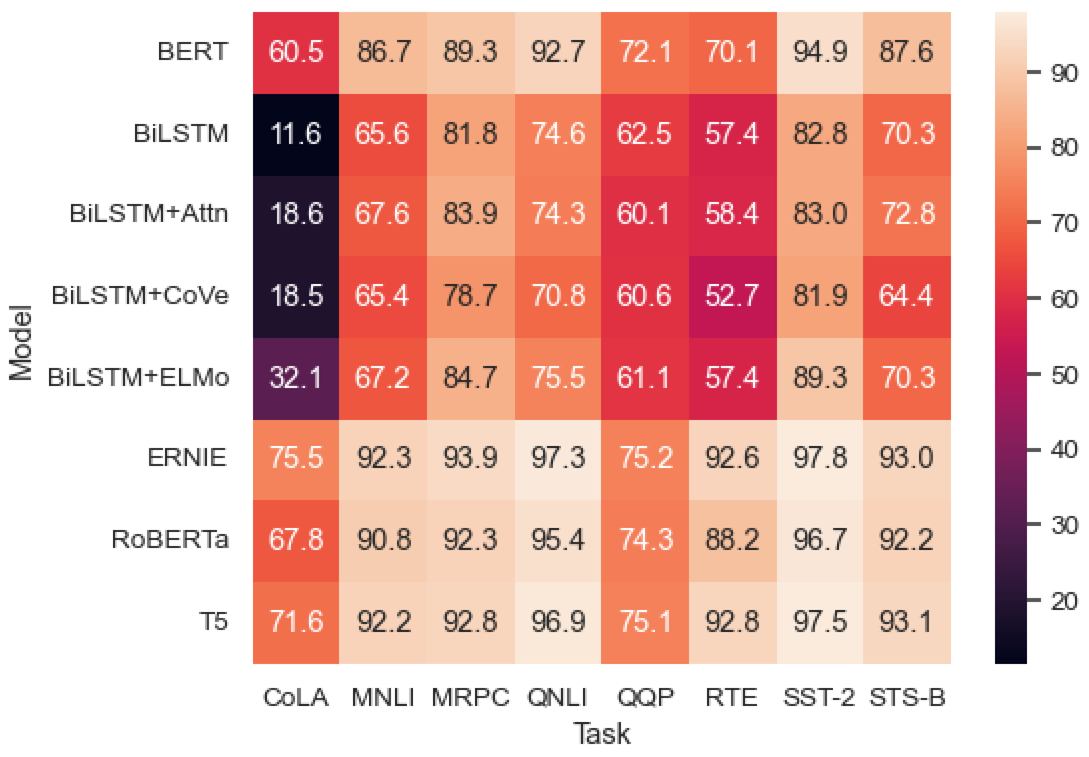
- Bubble charts extend the concept of scatter plots by adding a third dimension: the size of the bubble.
- The size of each bubble corresponds to a third variable, providing additional information.



-

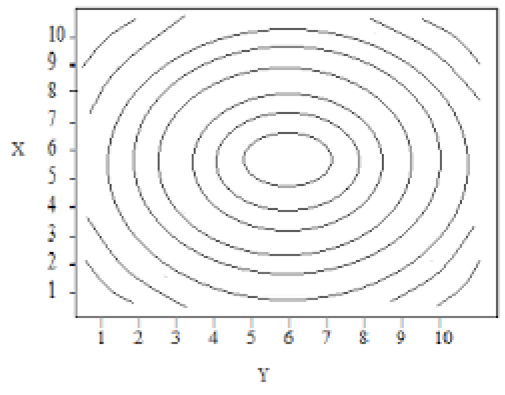
5. **Heatmaps:**

- Heatmaps are effective for visualizing the intensity of a relationship between two variables.
- Color gradients represent the magnitude of the relationship.



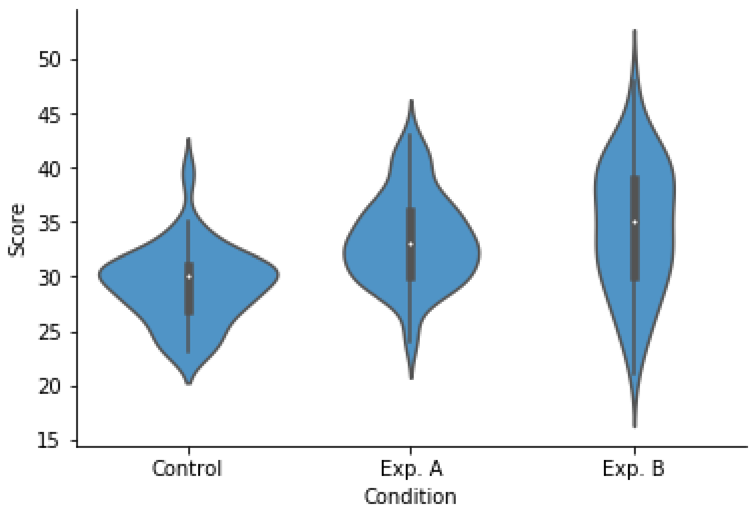
6. **Contour Plots:**

- Contour plots are helpful when you want to show the distribution of a third variable over a two-dimensional space.
- Contour lines connect points of equal value for the third variable.



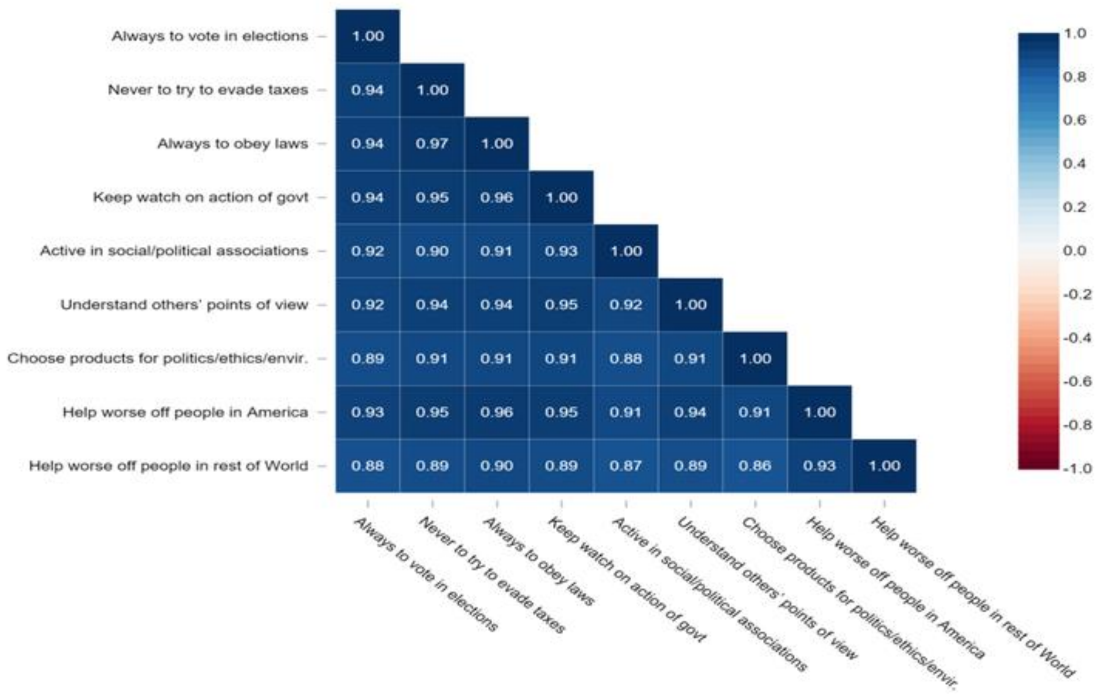
7. **Violin Plots:**

- Violin plots combine aspects of box plots and kernel density plots to show the distribution of a variable for different categories.



8. **Correlation Matrix:**

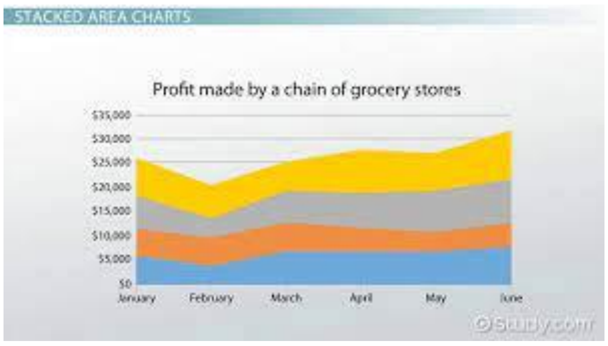
- A correlation matrix visually displays the correlation coefficients between pairs of variables, providing insights into relationships.



9. **Area Charts:**

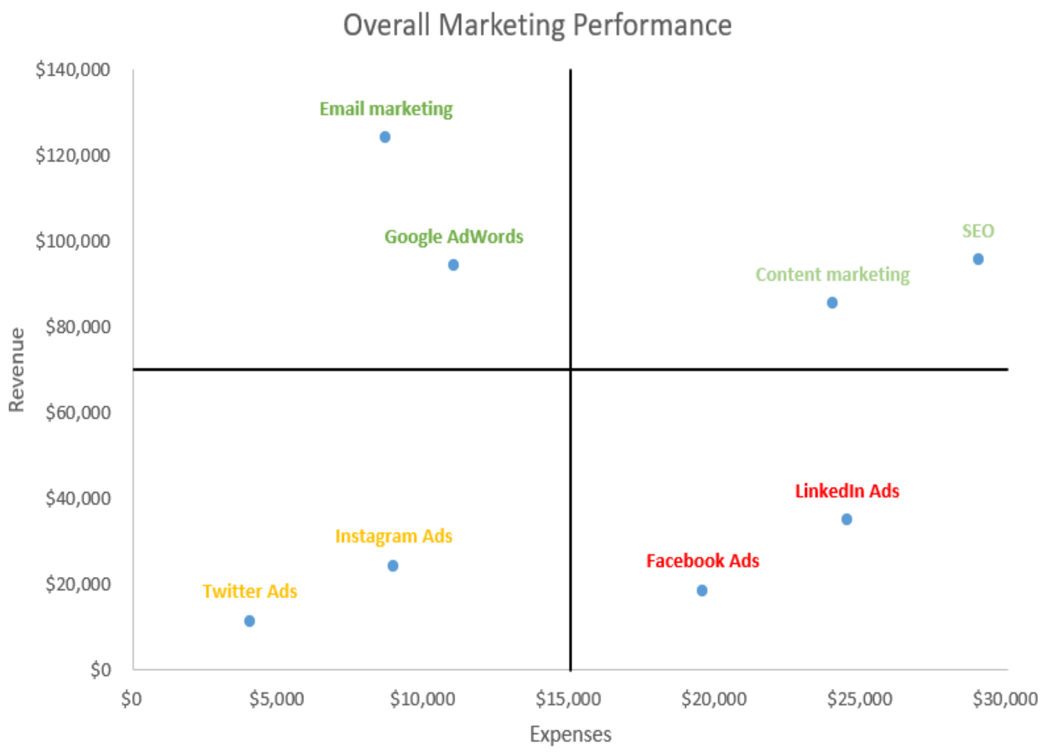
- Area charts can be used to represent the cumulative effect of two variables over time.

- The shaded area between the lines represents the combined impact.



10. **Quadrant Charts:**

- Quadrant charts divide a two-dimensional space into four quadrants, helping classify data points based on two variables.



## Data visualization on Multi dimensional data

## 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**.

## 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).

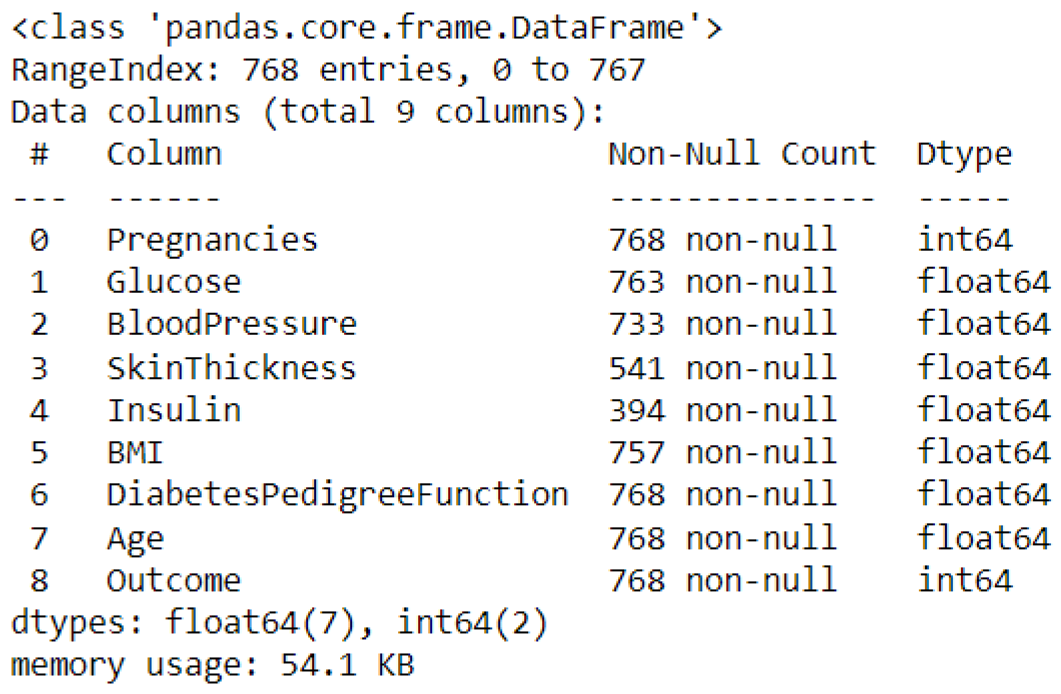
**Inference:** We will be using two datasets for this article, one will be the **diabetes patients dataset,** and another one is the height and weight of the persons. In the above output, we can see a glimpse of the first dataset using the **head**() method.

**Python Code:**

**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.

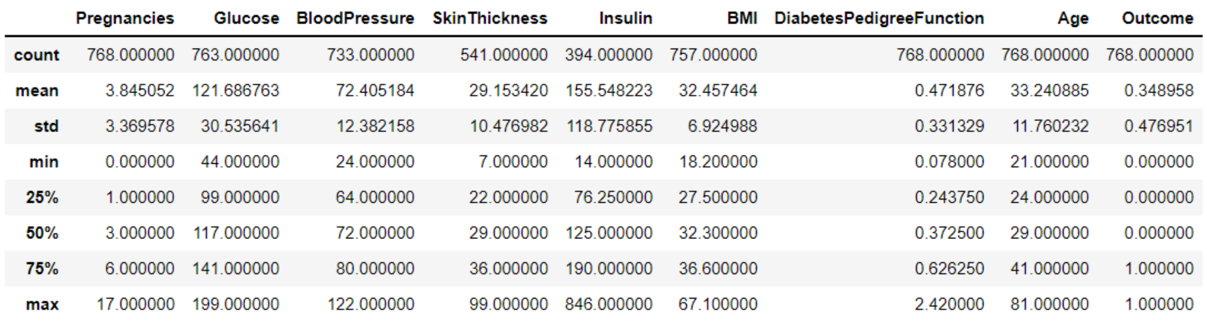
## Viewing the Data

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:**



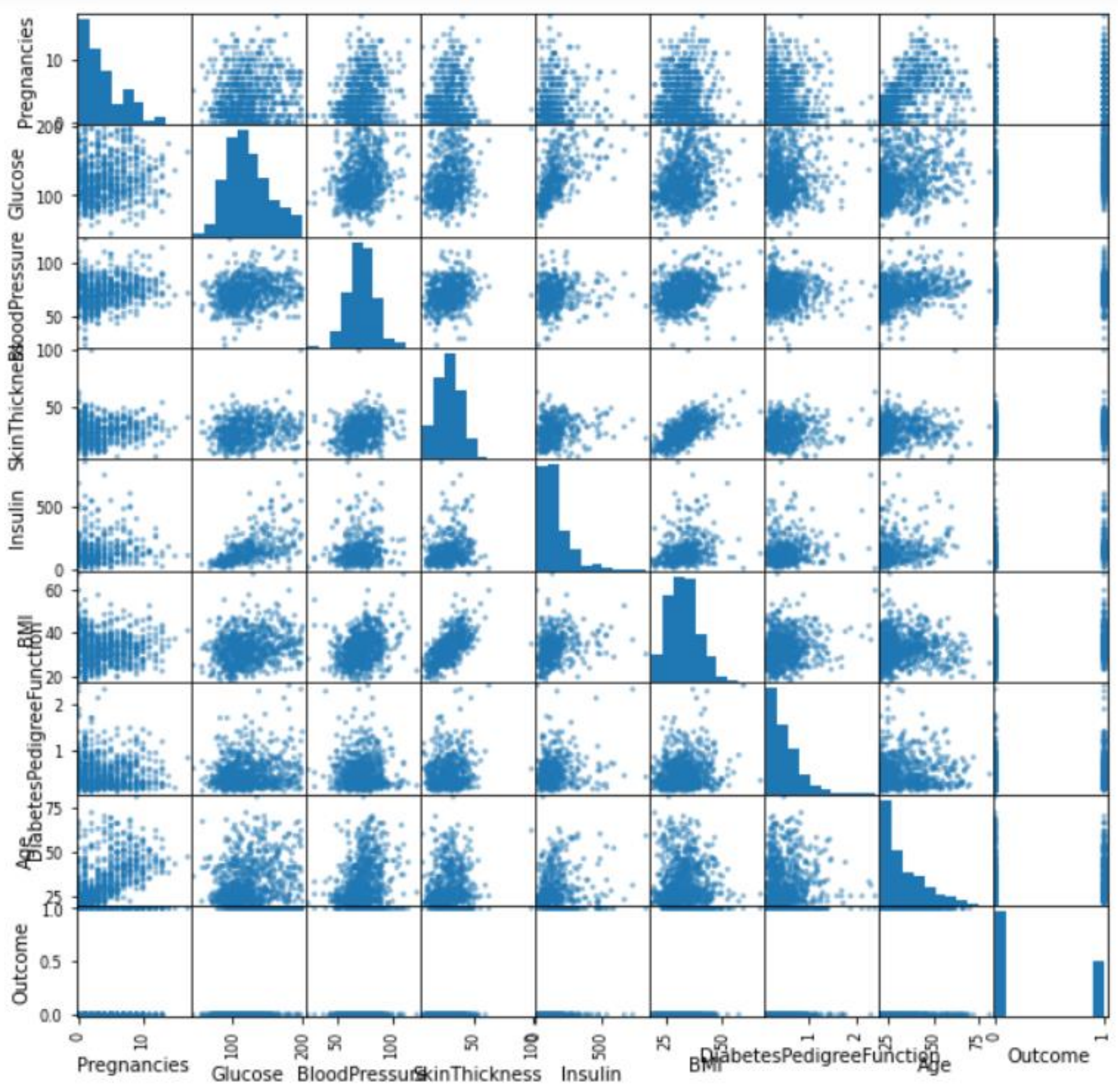
| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 768.000000 | 763.000000 | 733.000000 | 541.000000 | 394.000000 | 757.000000 | 768.000000 | 768.000000 | 768.000000 |
| **mean** | 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:**



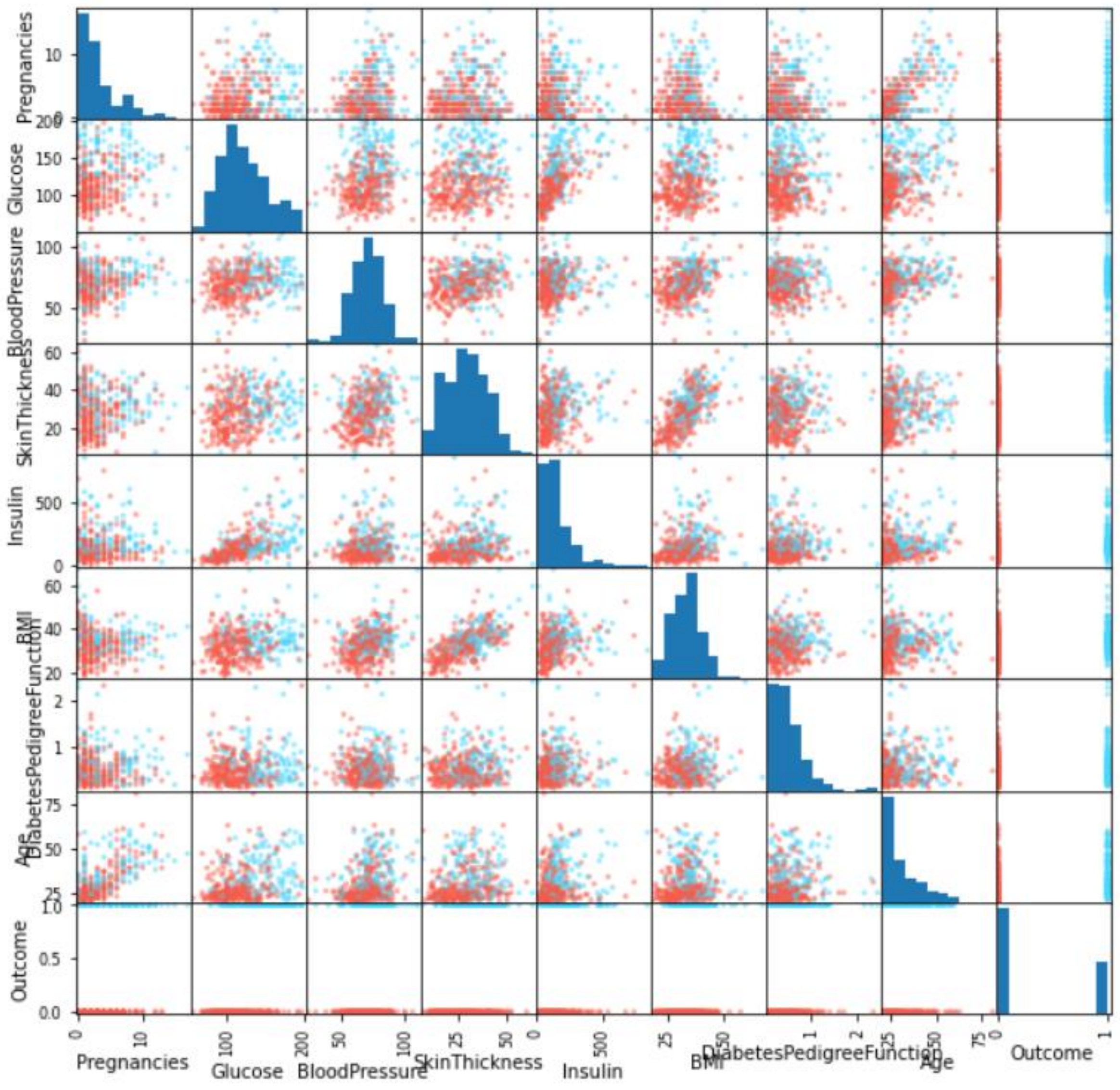
**Inference:** From the above plot, we can say that this plot alone is quite **descriptive** as it is showing us the linear relationship between all the variables in the dataset. For example, we can see **that skin Thickness** and **BMI** is sharing linear tendencies.

**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:**



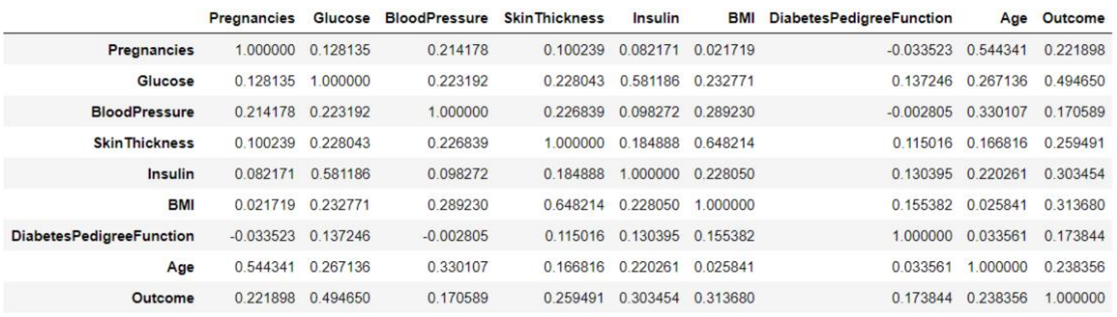
**Inference:** The scatter plot gives us both the histograms for the distributions **along the diagonal** and also a lot of 2D scatter plots **off-diagonal**. Not that this is a symmetric matrix, so I just look at the diagonal and below it normally. We can see that some variables have a lot of scattering, 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()
```

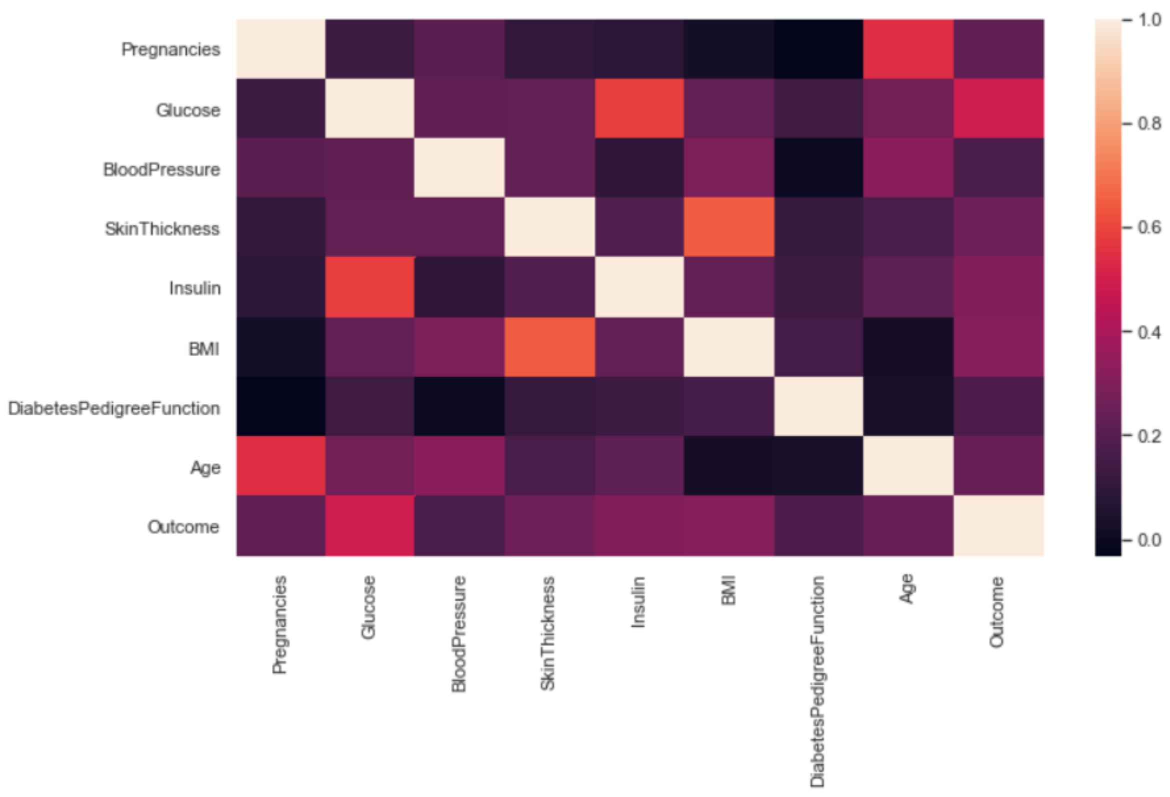
**Output:**



| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **Pregnancies** | 1.000000 | 0.128135 | 0.214178 | 0.100239 | 0.082171 | 0.021719 | -0.033523 | 0.544341 | 0.221898 |
| **Glucose** | 0.128135 | 1.000000 | 0.223192 | 0.228043 | 0.581186 | 0.232771 | 0.137246 | 0.267136 | 0.494650 |
| **BloodPressure** | 0.214178 | 0.223192 | 1.000000 | 0.226839 | 0.098272 | 0.289230 | -0.002805 | 0.330107 | 0.170589 |
| **SkinThickness** | 0.100239 | 0.228043 | 0.226839 | 1.000000 | 0.184888 | 0.648214 | 0.115016 | 0.166816 | 0.259491 |
| **Insulin** | 0.082171 | 0.581186 | 0.098272 | 0.184888 | 1.000000 | 0.228050 | 0.130395 | 0.220261 | 0.303454 |
| **BMI** | 0.021719 | 0.232771 | 0.289230 | 0.648214 | 0.228050 | 1.000000 | 0.155382 | 0.025841 | 0.313680 |
| **DiabetesPedigreeFunction** | -0.033523 | 0.137246 | -0.002805 | 0.115016 | 0.130395 | 0.155382 | 1.000000 | 0.033561 | 0.173844 |
| **Age** | 0.544341 | 0.267136 | 0.330107 | 0.166816 | 0.220261 | 0.025841 | 0.033561 | 1.000000 | 0.238356 |
| **Outcome** | 0.221898 | 0.494650 | 0.170589 | 0.259491 | 0.303454 | 0.313680 | 0.173844 | 0.238356 | 1.000000 |

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

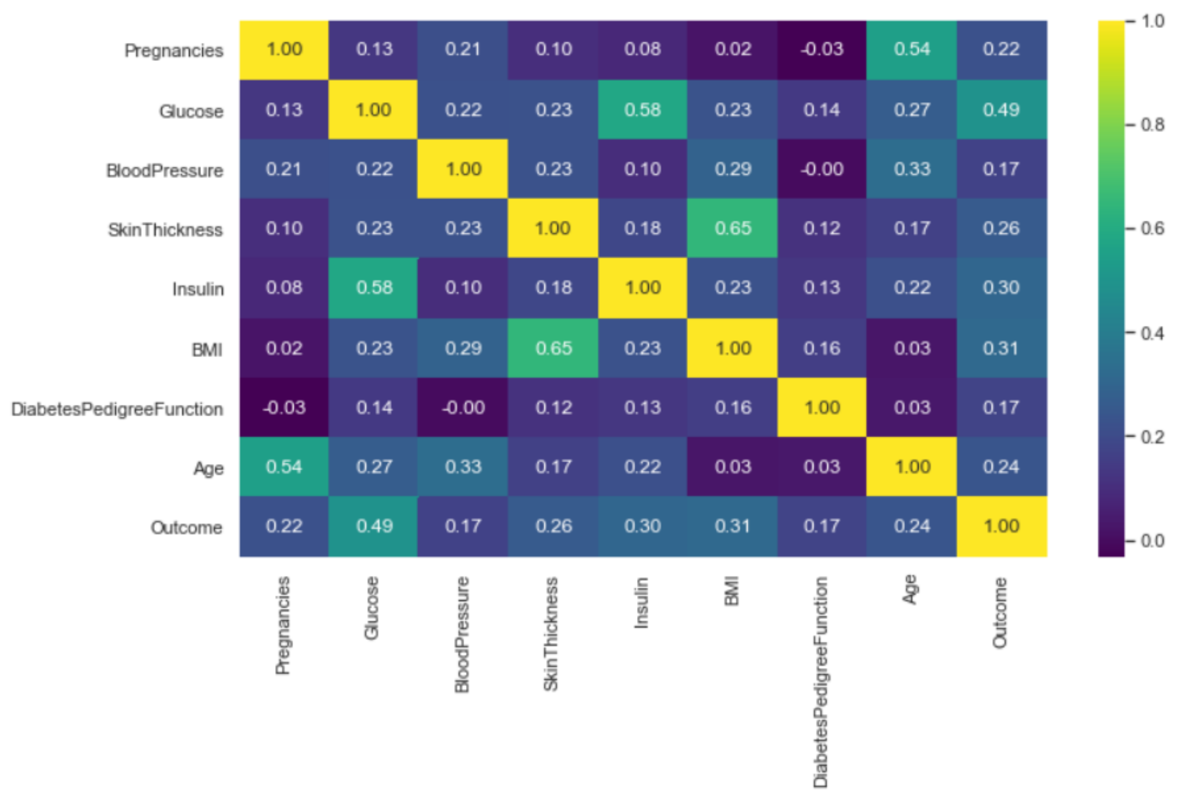
**Output:**



**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

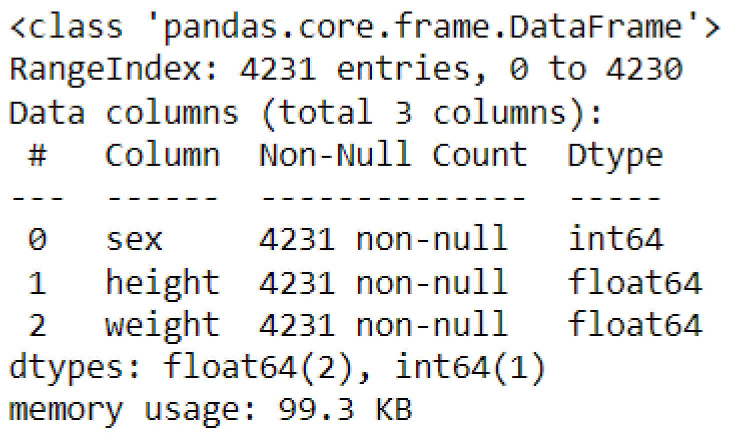
**2D Histograms** are mainly used for **image processing,** showing the **intensities of pixels** at a certain position of the image. Similarly, we can also use it for other problem statements, where we need to analyze two or more variables as **two-dimensional** or **three-dimensional** histograms,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

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

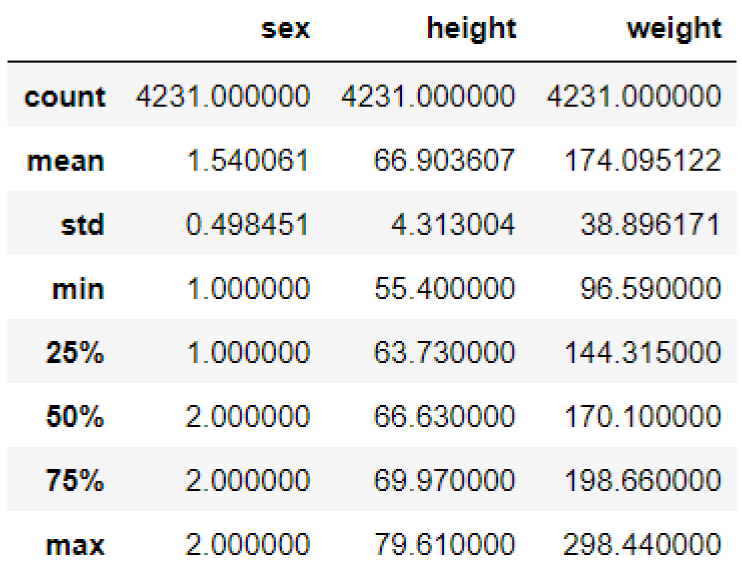
**Note:** 2-D histograms are very useful when you have a **lot** of data. See here for the API.

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

**Output:**



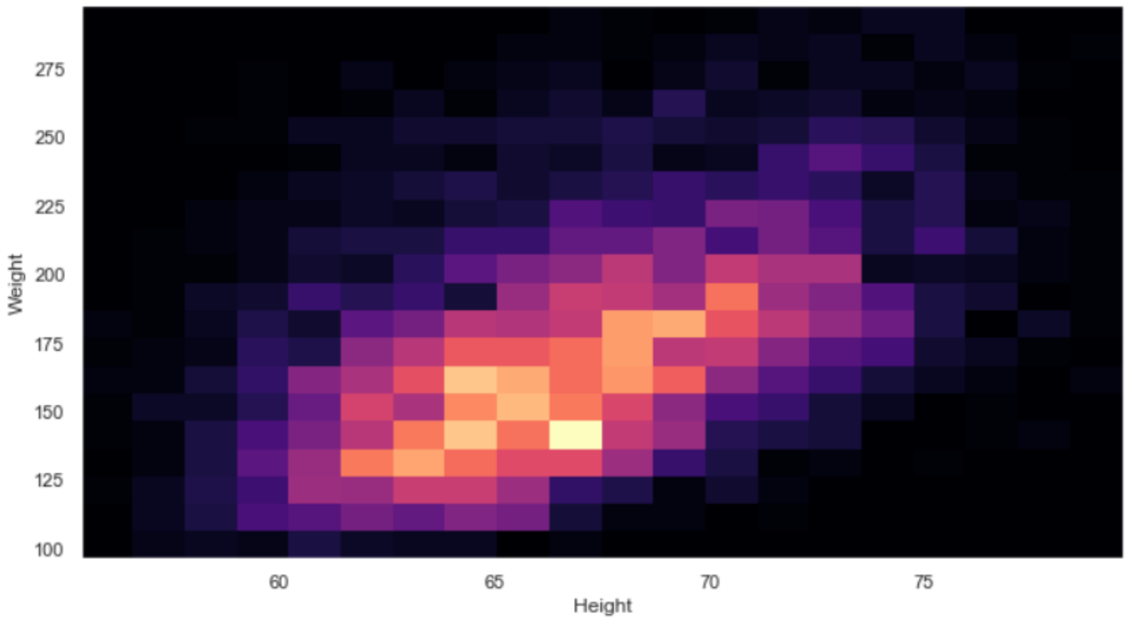
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4231 entries, 0 to 4230
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   sex     4231 non-null   int64  
 1   height  4231 non-null   float64
 2   weight  4231 non-null   float64
dtypes: float64(2), int64(1)
memory usage: 99.3 KB
```



| | sex | height | weight |
|---|---|---|---|
| **count** | 4231.000000 | 4231.000000 | 4231.000000 |
| **mean** | 1.540061 | 66.903607 | 174.095122 |
| **std** | 0.498451 | 4.313004 | 38.896171 |
| **min** | 1.000000 | 55.400000 | 96.590000 |
| **25%** | 1.000000 | 63.730000 | 144.315000 |
| **50%** | 2.000000 | 66.630000 | 170.100000 |
| **75%** | 2.000000 | 69.970000 | 198.660000 |
| **max** | 2.000000 | 79.610000 | 298.440000 |

```
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

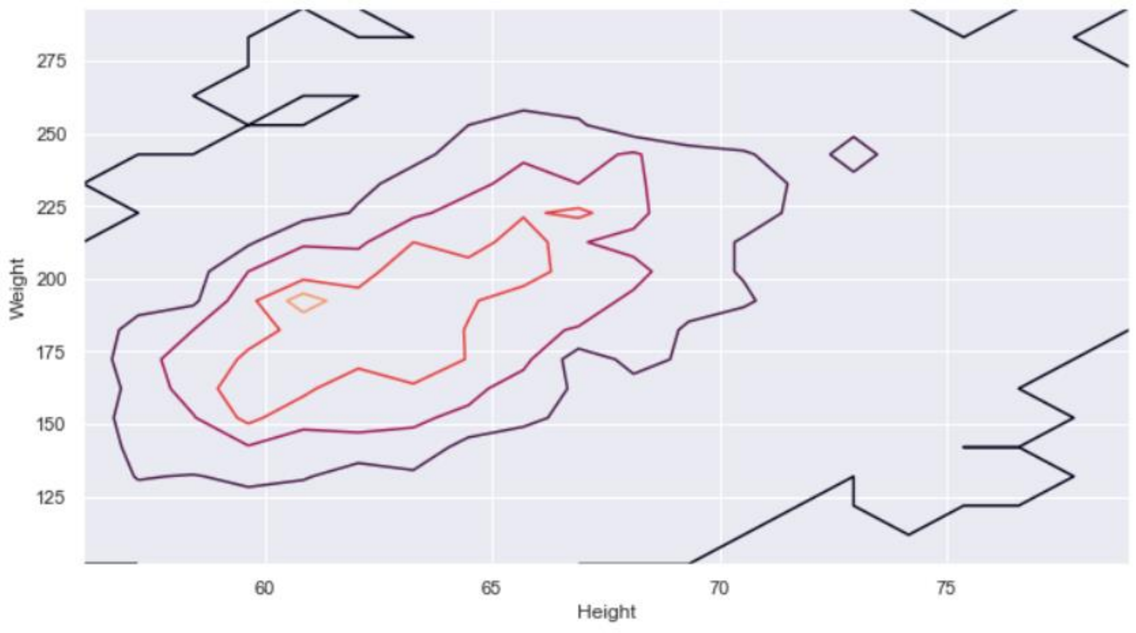
Bit hard to get information from the 2D histogram, isn't it? Too much noise in the image. What if we try and contour diagram? We'll have to bin the data ourselves.

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.