# EXPLORATORY DATA ANALYISIS

30 July 2023 01:57

**(Explore)**

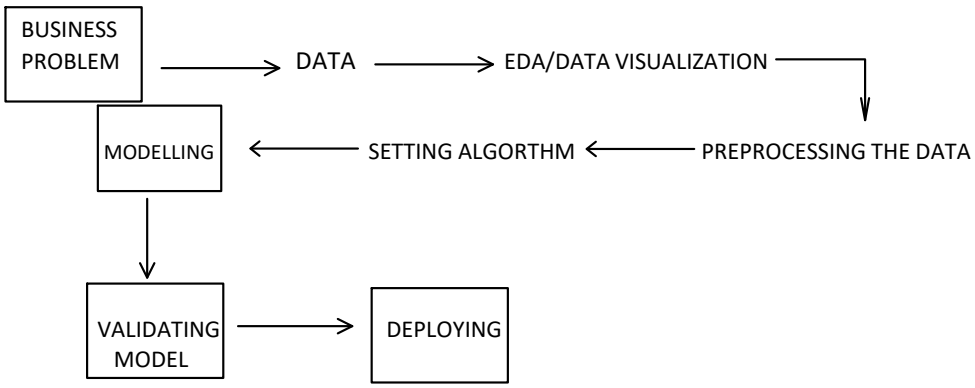
↓

Generate Insights & it also helps in Understanding the data how to preprocess it for modelling

Ex: Before Cooking the Rice(Data) we have to check and clean (look at what type of Rice and Amount of Water needed to cook). We have to remove Gravel(clean the data)

Generate Insights is looking at the Data Visualizing. Visualization Treatment and cleaning is **EDA**

**COMPLETE PROCESS**




**Deploying:** When model is Complete we have to Build In such a way That General Audience can use it or others terms in Production Phase.

*IT Starts With Visualization and end with Visualization.

## UNIVARIATE AND BIVARIATE DATA ANALYSIYS :

Data visualization is the process of representing data using visual elements like charts, graphs, etc. that helps in deriving meaningful insights from the data. It is aimed at revealing the information behind the data and further aids the viewer in seeing the structure in the data.

Univariate analysis is a statistical method used to analyse and summarize data involving a single variable. In other words, it focuses on examining the characteristics and properties of a single data set or variable without considering the relationships between that variable and others.

Common techniques and tools used in univariate analysis include:

1. Descriptive Statistics: This involves summarizing data using measures such as mean (average), median (middle value), mode (most common value), range (difference between the maximum and minimum values), variance, and standard deviation.
2. Frequency Distributions: A frequency distribution table or histogram can be used to display how often each value or range of values occurs in the dataset.
3. Measures of Central Tendency: These statistics, like the mean, median, and mode, help describe the center or typical value of the data.
4. Measures of Dispersion: These statistics, such as the range, variance, and standard deviation, describe how spread out the data is.
5. Measures of Shape: Skewness and kurtosis are used to describe the shape of the data distribution. Skewness measures the asymmetry of the distribution, while kurtosis measures its "tailed Ness."
6. Box Plots: A graphical representation that displays the median, quartiles, and potential outliers in a dataset.
7. Bar Charts and Histograms: These visualizations can provide a clear representation of the data distribution and frequency of values.

Univariate analysis is often the first step in data analysis and is useful for gaining insights into a single variable's characteristics. It can help identify outliers, assess data quality, and guide further statistical or machine learning analyses.

## Need for visualizing data :

- Understand the trends and patterns of data
- Analyze the frequency and other such characteristics of data
- Know the distribution of the variables in the data.
- Visualize the relationship that may exist between different variables

The number of variables of interest featured by the data classifies it as **univariate, bivariate, or multivariate**

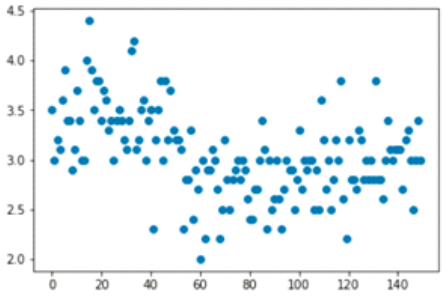
## Univariate enumerative Plots :

These plots enumerate/show every observation in data and provide information about the distribution of the observations on a single data variable. We now look at different enumerative plots.

## 1. UNIVARIATE SCATTER PLOT :

This plots different observations/values of the same variable corresponding to the index/observation number. Consider plotting of the variable ‘sepal Width(cm)’ :

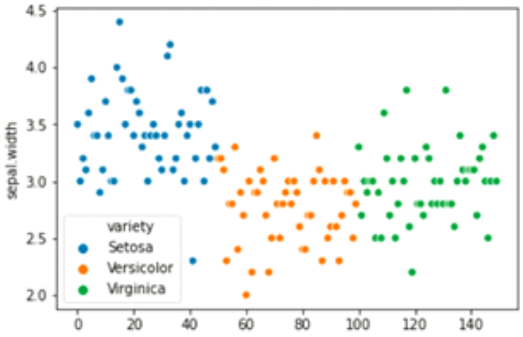
```
plt.scatter(df.index,df['sepal.width'])
plt.show()
```




Use the *plt.scatter()* function of matplotlib to plot a univariate scatter diagram. The scatter() function requires two parameters to plot. So, in this example, we plot the variable ‘sepal.width’ against the corresponding observation number that is stored as the index of the data frame (df.index).

Then visualize the same plot by considering its variety using the *sns.scatterplot()* function of the seaborn library.

```
sns.scatterplot(x=df.index,y=df['sepal.width'],hue=df['variety'])
```



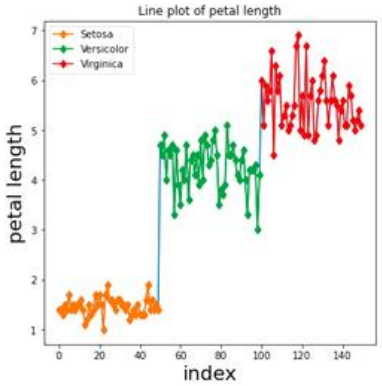
One of the interesting features in seaborn is the 'hue' parameter. In seaborn, **the hue parameter determines which column in the data frame should be used for color encoding**. This helps to differentiate between the data values according to the categories they belong to. **The hue parameter takes the grouping variable as it's input using which it will produce points with different colors**. The variable passed onto 'hue' can be either categorical or numeric, although color mapping will behave differently in the latter case.

## 2. LINE PLOT (with markers) :

A line plot visualizes data by connecting the data points via line segments. It is similar to a scatter plot except that the measurement points are ordered (typically by their x-axis value) and joined with straight line segments.

***Setting title, figure size, labels and font size in matplotlib***

```
plt.figure(figsize=(6,6))
plt.title('Line plot of petal length')
plt.xlabel('index',fontsize=20)
plt.ylabel('petal length',fontsize=20)
plt.plot(df.index,df['petal.length'],markevery=1,marker='d')
for name, group in df.groupby('variety'):
    plt.plot(group.index, group['petal.length'], label=name,markevery=1,marker='d')
plt.legend()
plt.show()
```




The matplotlib *plt.plot()* function by default plots the data using a line plot.

Previously, we discussed the hue parameter of seaborn. Though there is no such automated option available in matplotlib, one can use the *groupby()* function of pandas which helps in plotting such a graph.

Note:In the above illustration, the methods to set title, font size, etc in matplotlib are also implemented.

— Explanation of the functions used :

- *plt.figure(figsize=())* : To set the size of figure
- *plt.title()* : To set title
- *plt.xlabel()* / *plt.ylabel()* : To set labels on X-axis/Y-axis
- *df.groupby( )* : To group the rows of the data frame according to the parameter passed onto the function
- The groupby() function returns the data frames grouped by the criterion variable passed and the criterion variable.
- The *for loop* is used to plot each data point according to its variety.
- *plt.legend()*: Adds a legend to the graph (Legend describes the different elements seen in the graph).
- *plt.show()* : to show the plot.

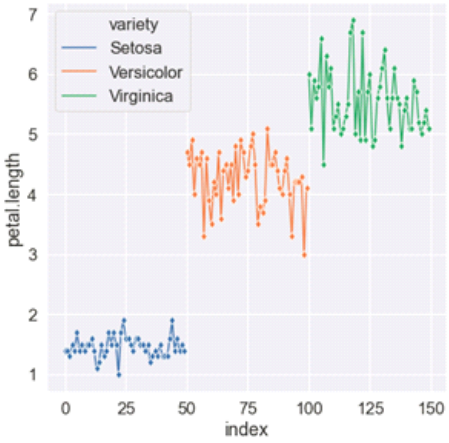
The 'markevery' parameter of the function *plt.plot()* is assigned to '1' which means it will plot every 1st marker starting from the first data point. There are various marker styles which we can pass as a parameter to the function.

The *sns.lineplot()* function can also visualize the line plot.

***Setting title, figure size,labels and font size in seaborn***

```
sns.set(rc={'figure.figsize':(7,7)})
sns.set(font_scale=1.5)
```

```
fig=sns.lineplot(x=df.index,y=df['petal.length'],markevery=1,marker='d',data=df,hue=df['variety'])
fig.set(xlabel='index')
```




In seaborn, the labels on axes are automatically set based on the columns that are passed for plotting. However if one desires to change it, it is possible too using the set() function.

### 3. STRIP PLOT :

The strip plot is similar to a scatter plot. It is often used along with other kinds of plots for better analysis. It is used to visualize the distribution of data points of the variable.

The *sns.striplot ( )* function is used to plot a strip-plot :

```
sns.stripplot(y=df['sepal.width'])
```

It also helps to plot the distribution of variables for each category as individual data points. By default, the function creates a vertical strip plot where the distributions of the continuous data points are plotted along the Y-axis and the categories are spaced out along the X-axis. In the above plot, categories are not considered. Considering the categories helps in better visualization as seen in the below plot.

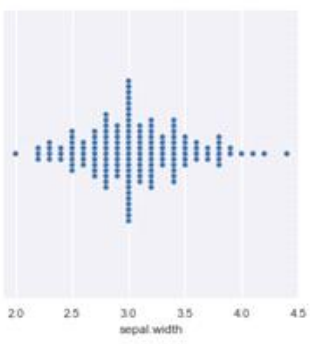
```
sns.stripplot(x=df['variety'],y=df['sepal.width'])
```

## 4. SWARM PLOT :

The swarm-plot, similar to a strip-plot, provides a visualization technique for univariate data to view the spread of values in a continuous variable. The only difference between the strip-plot and the swarm-plot is that the swarm-plot spreads out the data points of the variable automatically to avoid overlap and hence provides a better visual overview of the data.

The *sns.swarmplot( )* function is used to plot a swarm-plot :

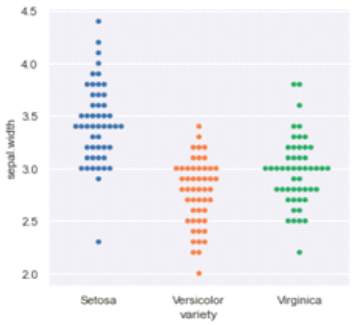
```
sns.set(rc={'figure.figsize':(5,5)})
sns.swarmplot(x=df['sepal.width'])
```




Distribution of the variable 'sepal.width' according to the categories :

Distribution of the variable 'sepal.width' according to the categories :

```
sns.swarmplot(x=df['variety'],y=df['sepal.width'])
```




## Uni-variate summary plots :

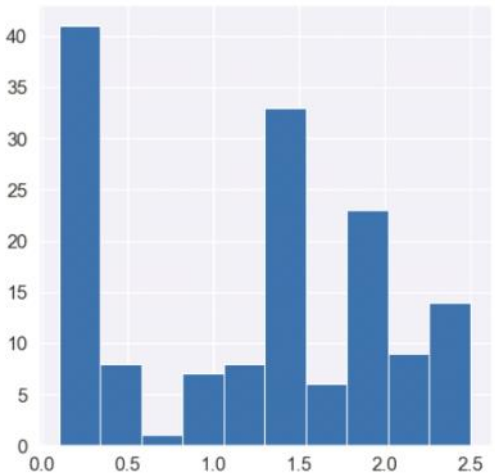
These plots give a more concise description of the location, dispersion, and distribution of a variable than an enumerative plot. It is not feasible to retrieve every individual data value in a summary plot, but it helps in efficiently representing the whole data from which better conclusions can be made on the entire data set.

# 5. HISTOGRAMS :

**Histograms are similar to bar charts which display the counts or relative frequencies of values falling in different class intervals or ranges. A histogram displays the shape and spread of continuous sample data. It also helps us understand the skewness and kurtosis of the distribution of the data.**

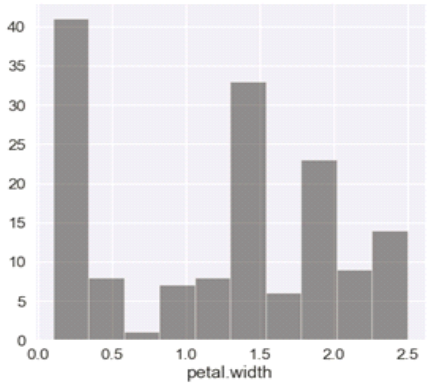
Plotting histogram using the matplotlib *plt.hist()* function :

```
In [12]: plt.hist(df['petal.width'])
Out[12]: (array([41.,  8.,  1.,  7.,  8., 33.,  6., 23.,  9., 14.]),
          array([0.1 , 0.34, 0.58, 0.82, 1.06, 1.3 , 1.54, 1.78, 2.02, 2.26, 2.5 ]),
          <a list of 10 Patch objects>)
```



The seaborn function *sns.distplot()* can also be used to plot a histogram.

```
sns.distplot(df['petal.width'],kde=False,color='black',bins=10)
```



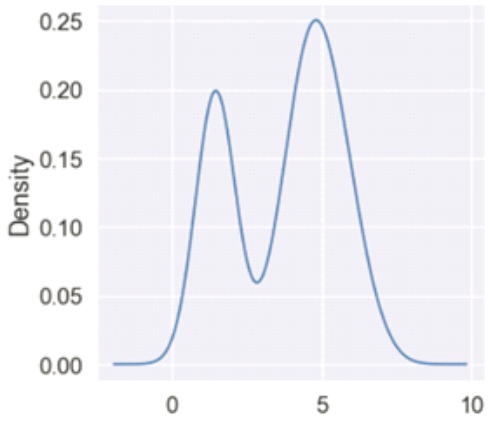

The kde (kernel density) parameter is set to False so that only the histogram is viewed. There are many parameters like bins (indicating the number of bins in histogram allowed in the plot), color, etc; which can be set to obtain the desired output.

## 6. DENSITY PLOTS :

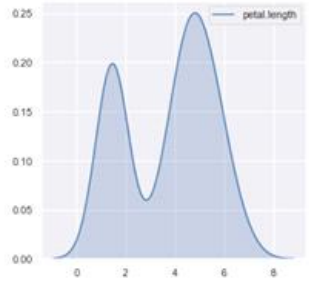
A density plot is like a **smoother version of a histogram**. Generally, **the kernel density estimate is used in density plots** to show the probability density function of the variable. A continuous curve, which is the kernel is drawn to generate a smooth density estimation for the whole data.

Plotting density plot of the variable ‘petal.length’ :

```
plt.figure(figsize=(5,5))
df['petal.length'].plot(kind='density')
```



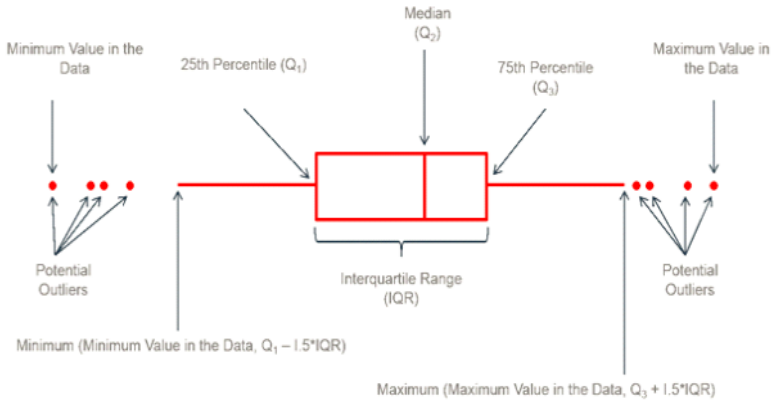
```
sns.set(rc={'figure.figsize':(5,5)})
sns.kdeplot(df['petal.length'],shade=True)
```




we use the pandas *df.plot()* function (built over matplotlib) or the seaborn library's *sns.kdeplot()* function to plot a density plot . Many features like shade, type of distribution, etc can be set using the parameters available in the functions. By default, the kernel used is Gaussian (this produces a Gaussian bell curve). Also, other graph smoothing techniques/filters are applicable.

# 8. BOX PLOTS :

A box-plot is a very useful and standardized way of displaying the distribution of data based on a five-number summary (minimum, first quartile, second quartile(median), third quartile, maximum). It helps in understanding these parameters of the distribution of data and is extremely helpful in detecting outliers.




**Plotting box plot of variable 'sepal.width'** : **plt.boxlplot(df.['Sepal  width'])**

Plotting box plots of all variables in one frame :

Since the box plot is for continuous variables, firstly create a data frame without the column 'variety'. Then drop the column from the DataFrame using the *drop( )* function and specify axis=1 to indicate it.

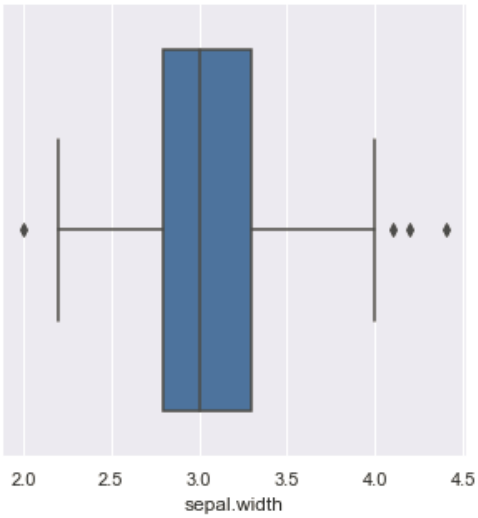
***Removing the column with categorical variables***

```
dfM=df.drop('variety',axis=1)
```

```
plt.figure(figsize=(9,9))
#Set Title
plt.title('Box plots of the 4 variables')
plt.boxplot(dfM.values,labels=['SepalLength','SepalWidth','PetalLength','PetalWidth'])
```

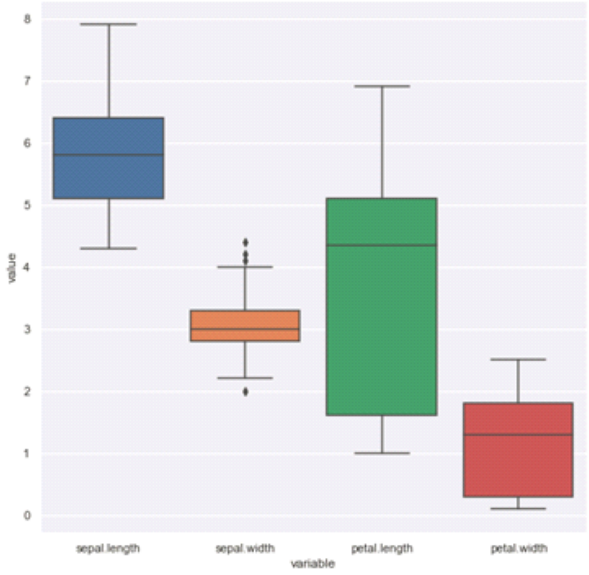
In matplotlib, mention the labels separately to display it in the output.

The plotting box plot in seaborn : **sns.boxplot(['Sepal  width'])**




Plotting the box plots of all variables in one frame :

```
sns.set(rc={'figure.figsize':(9,9)})
sns.boxplot(x="variable", y="value", data=pd.melt(dfM))
```




# VISUALIZING CATEGORICAL VARIABLES :

## 11. BAR CHART :

The bar plot is a univariate data visualization plot on a two-dimensional axis. One axis is the category axis indicating the category, while the second axis is the value axis that shows the numeric value of that category, indicated by the length of the bar.

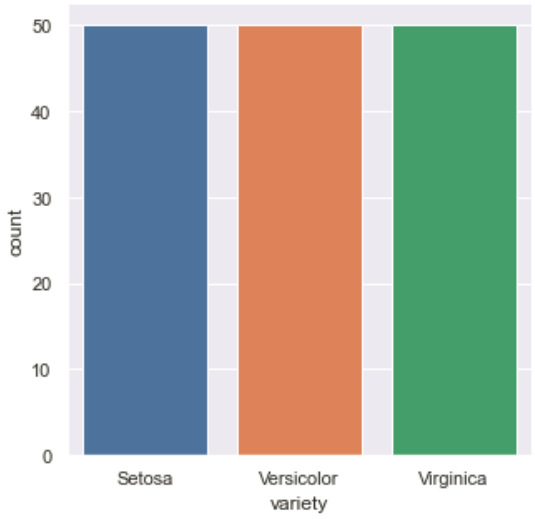
The *plot.bar()* function plots a bar plot of a categorical variable. The *value_counts()* returns a series containing the counts of unique values in the variable.

```
df['variety'].value_counts().plot.bar()
```

The *countplot()* function of the seaborn library obtains a similar bar plot. There is no need to separately calculate the count when using the *sns.countplot()* function.

Since the variety is equally distributed, we obtain bars with equal heights.

```
sns.countplot(df['variety'])
```

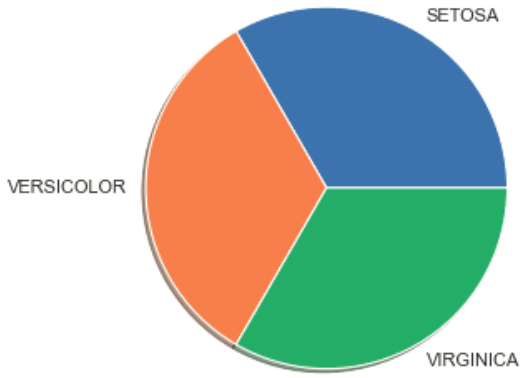



## 12. PIE CHART :

A pie chart is the most common way used to visualize the numerical proportion occupied by each of the categories.

Use the *plt.pie()* function to plot a pie chart. Since the categories are equally distributed, divide the sections in the pie chart is equally. Then add the labels by passing the array of values to the 'labels' parameter.

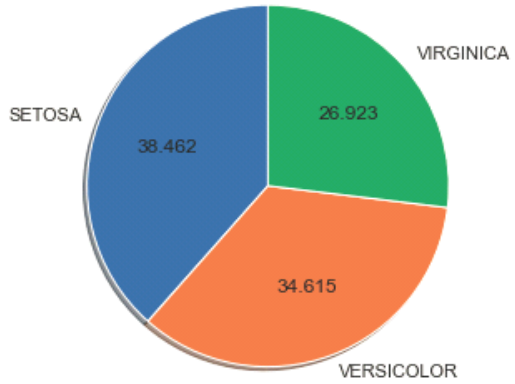
```
plt.pie(df['variety'].value_counts(),
        labels=['SETOSA','VERSICOLOR','VIRGINICA'],shadow=True)
```




The 'startangle' parameter of the *pie()* function rotates everything counter-clockwise at a specific angle. Further, the default value for startangle is 0. The 'autopct' parameter enables one to display the percentage value using Python string formatting.

```
df1=df.sample(frac=0.35)
```

```
plt.figure(figsize=(5,5))
plt.pie(df1['variety'].value_counts(),startangle=90,autopct='%.3f',
        labels=['SETOSA','VERSICOLOR','VIRGINICA'],shadow=True)
```




Most of the methods that help in the visualization of univariate data have been outlined in this article. As stated before the ability to see the structure and information carried by the data lies in its visual presentation.

## BIVARIATE ANALYSIS:

Bivariate analysis is a statistical analysis that involves the simultaneous analysis of two variables to determine the empirical relationship between them. In other words, it focuses on understanding the relationship between two different variables in a dataset. The primary goal of bivariate analysis is to examine whether there is a relationship, correlation, or association between the two variables.
Common techniques and tools used in bivariate analysis include:

1. Scatter plots: A graphical representation of data points in a Cartesian plane, where each point represents an observation for both variables. It helps visualize the relationship between the two variables.

2. Correlation analysis: This is used to quantify the strength and direction of the linear relationship between two variables. Common correlation measures include Pearson's correlation coefficient for linear relationships and Spearman's rank correlation coefficient for monotonic relationships.

3. Covariance: It measures the direction of the linear relationship between two variables. A positive covariance indicates that the variables move together, while a negative covariance indicates that they move inversely.

4. Contingency tables: These are used for categorical variables to display the frequency distribution of variables and to examine if there is any association between them.
Bivariate analysis is crucial for understanding how two variables are related and whether changes in one variable are associated with changes in the other. It forms the basis for more advanced analyses that involve multiple variables and helps in making informed decisions and predictions based on the relationships identified.

## Bivariate Analysis for Each Variable Type

There are essentially two types of variables in data – Categorical and continuous (numerical). So, in the case of bivariate analysis, there could be four combinations of analysis that could be done that is listed in the summary table below:

Summary Table:

| Type of Variables (Vs.) | Categorical (incl. discrete numerical) | Continuous |
|---|---|---|
| **Categorical (incl. discrete numerical)** | • Frequency of the two categories/ other continuous variables' range<br>  • Crosstab<br>  • Heatmaps<br>  • Stacked bar charts | • Range of continuous variable with respect to each category<br>  • Boxplots<br>  • Violin plots<br>  • Swam plots<br>  • Count plots<br>  • Bar plot |
| **Continuous** | • Range of continuous variable with respect to each category<br>  • Boxplots<br>  • Violin plots<br>  • Swam plots<br>  • Count plots<br>  • Bar plot | • How the increase or decrease in one variables changes with the other<br>  • Scatterplot<br>  • Line plots |

To develop a further hands-on understanding, the following is an example of bivariate analysis for each combination listed above in Python:

## Categorical vs Categorical Variables

This is used in case both the variables being analysed are categorical. In the case of classification, models say, for example, we are classifying a credit card fraud or not as Y variables and then checking if the customer is at his hometown or away or outside the country. Another example can be age vs gender and then counting the number of customers who fall in that category. It is important to note that the visualization/summary shows the count or some mathematical or logical aggregation of a 3rd variable/metric like revenue or cost and the like in all such analyses. It can be done using Crosstabs (heatmaps) or Pivots in Python. Let us look at examples below:

**Crosstabs:** It is used to count between categories, or get summaries between two categories. Pandas library has this functionality.

| | cust | categ_x | categ_y |
|---|---|---|---|
| **0** | 1 | a | a |
| **1** | 1 | a | b |
| **2** | 1 | b | a |
| **3** | 1 | b | b |
| **4** | 2 | a | a |
| **5** | 3 | a | a |
| **6** | 3 | a | b |
| **7** | 3 | b | a |
| **8** | 3 | b | b |
| **9** | 4 | b | b |
| **10** | 5 | b | b |

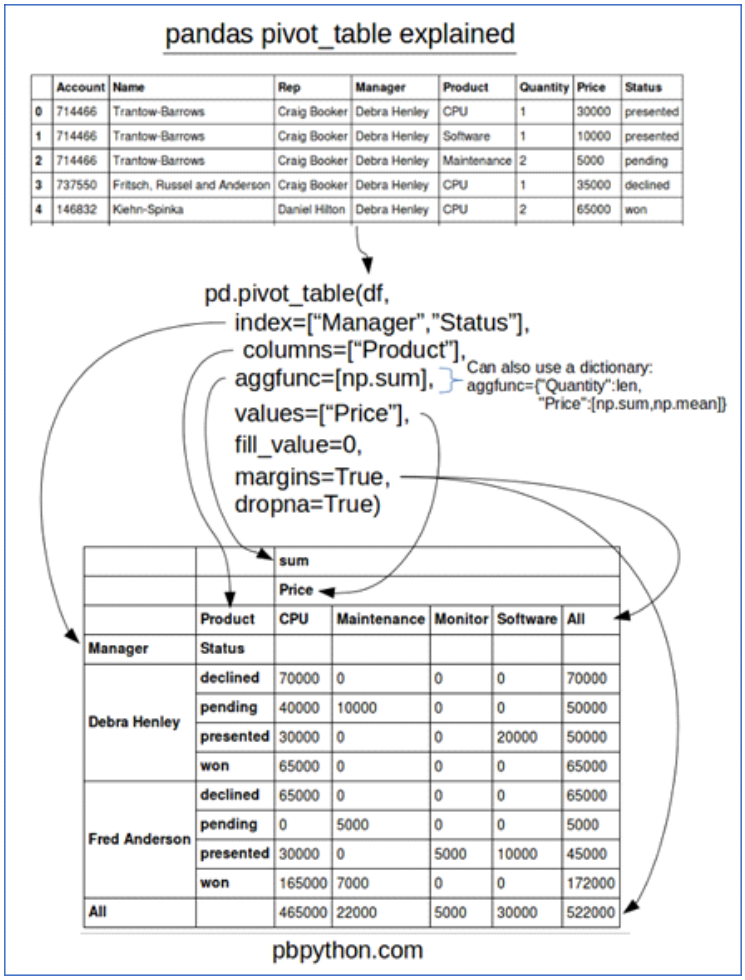
| categ_y | a | b | All |
|---|---|---|---|
| **categ_x** | | | |
| **a** | 3 | 2 | 5 |
| **b** | 2 | 4 | 6 |
| **All** | 5 | 6 | 11 |

```
Import pandas as pd
Import seaborn as sns
# need to summarize on x and y categorical variables
pd.crosstab(    dd.categ_x, dd.categ_y, margins=True,     values=dd.cust, aggfunc=pd.Series.count)
# 3 any other aggregation function can be used based on column type
```

**Margins=True means to display rows and columns total**

```
# to create a heatmap by enclosing the above in sns.heatmap
sns.heatmap(pd.crosstab(   dd.categ_x, dd.categ_y, margins=True,     values=dd.cust, aggfunc=pd.Series.count),
          cmap="YlGnBu", annot=True, cbar=False)
```

**Pivots:** Another useful functionality that can be applied to Pandas dataframes to get Excel like Pivot tables. This can work for 2+ categorical variables when placed in the proper hierarchy.




| | Account | Name | Rep | Manager | Product | Quantity | Price | Status |
|---|---|---|---|---|---|---|---|---|
| 0 | 714466 | Trantow-Barrows | Craig Booker | Debra Henley | CPU | 1 | 30000 | presented |
| 1 | 714466 | Trantow-Barrows | Craig Booker | Debra Henley | Software | 1 | 10000 | presented |
| 2 | 714466 | Trantow-Barrows | Craig Booker | Debra Henley | Maintenance | 2 | 5000 | pending |
| 3 | 737550 | Fritsch, Russel and Anderson | Craig Booker | Debra Henley | CPU | 1 | 35000 | declined |
| 4 | 146832 | Kiehn-Spinka | Daniel Hilton | Debra Henley | CPU | 2 | 65000 | won |

| | | sum | | | | |
|---|---|---|---|---|---|---|
| | | Price | | | | |
| | Product | CPU | Maintenance | Monitor | Software | All |
| Manager | Status | | | | | |
| Debra Henley | declined | 70000 | 0 | 0 | 0 | 70000 |
| | pending | 40000 | 10000 | 0 | 0 | 50000 |
| | presented | 30000 | 0 | 0 | 20000 | 50000 |
| | won | 65000 | 0 | 0 | 0 | 65000 |
| Fred Anderson | declined | 65000 | 0 | 0 | 0 | 65000 |
| | pending | 0 | 5000 | 0 | 0 | 5000 |
| | presented | 30000 | 0 | 5000 | 10000 | 45000 |
| | won | 165000 | 7000 | 0 | 0 | 172000 |
| All | | 465000 | 22000 | 5000 | 30000 | 522000 |

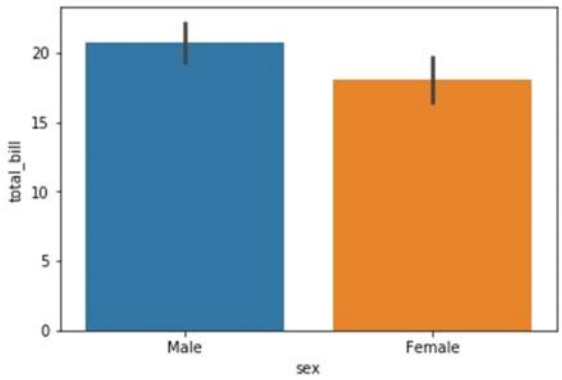
Python code: Assuming the above dataset, just this one line of code can produce the desired bivariate views.

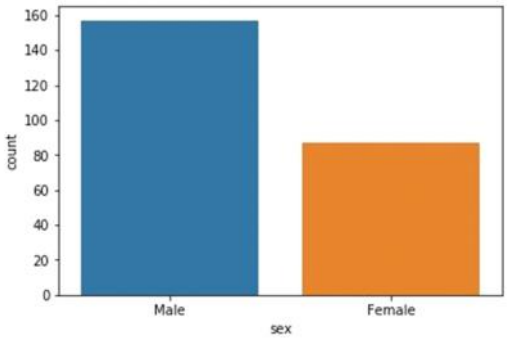
```
Pd.pivot_table(df,index =[“Manager”,”Status”],columns=[“Product”],aggfunc=[np.sum]
```

**Categorical vs continuous (numerical) variables:** It is an example of plotting the variance of a numerical variable in a class. Like how age varies in each segment or how do income and expenses of a household vary by loan re-payment status.

Categorical plot for aggregates of continuous variables: Used to get total or counts of a numerical variable. Ex: revenue for each month. PS: This can be used for counts of another categorical variable too instead of the numerical.

> Plots used are: bar plot and count plot
> sns.barplot(x='sex',y='total_bill',data=t)
> sns.countplot(x='sex',data=t)

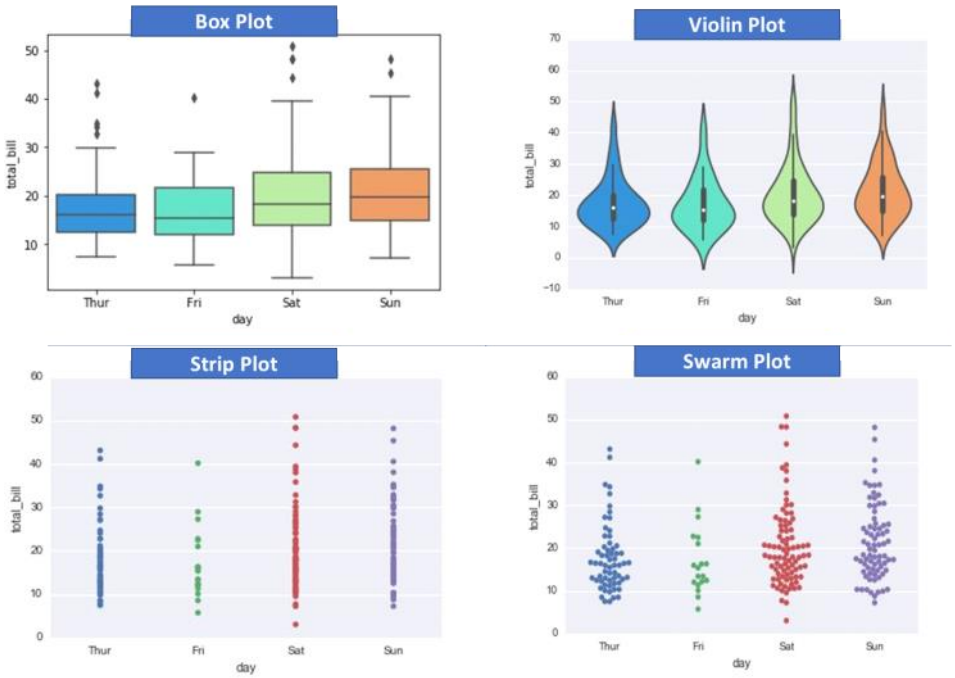



Plots for distribution of continuous (numerical) variables: Use to see the range and statistics of a numerical variable across categories

Plots used are – box plot, violin plot, swarm plot

```
sns.boxplot(x='day',y='total_bill',data=t,palette='rainbow')
sns.violinplot(x="day", y="total_bill", data=t,palette='rainbow')
sns.stripplot(x="day", y="total_bill", data=t)
sns.swarmplot(x="day", y="total_bill", data=t)
```




Source: Designed by the author for this writing

**Continuous vs continuous:** This is the most common use case of bivariate analysis and is used for showing the empirical relationship between two numerical (continuous) variables. This is usually more applicable in regression cases.

The following plots make sense in this case: scatterplot, regplot.

Code below:

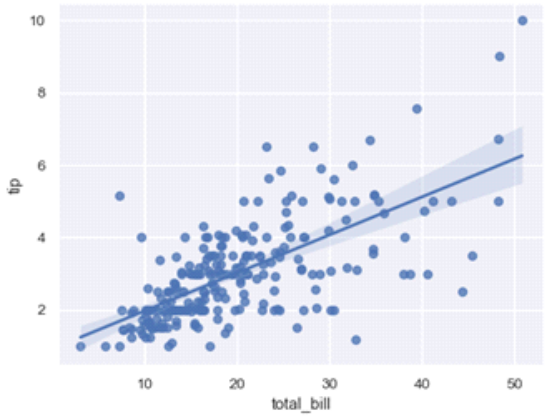
```
Import seaborn as sns
 Sns.regplot(x='a',y='b',data=df)
Plt.ylim(0,)
```

## Bivariate Analysis at Scale

In case we have large datasets with 30-70+ features (variables), there might not be sufficient time to run each pair of variables through bivariate analysis one by one. We could use the pairplot or pair grid from the seaborn library in such cases. It makes a grid where each cell is a bivariate graph, and Pairgrid also allows customizations. Code snippets and sample outputs below (assuming seaborn is imported and the iris dataset):

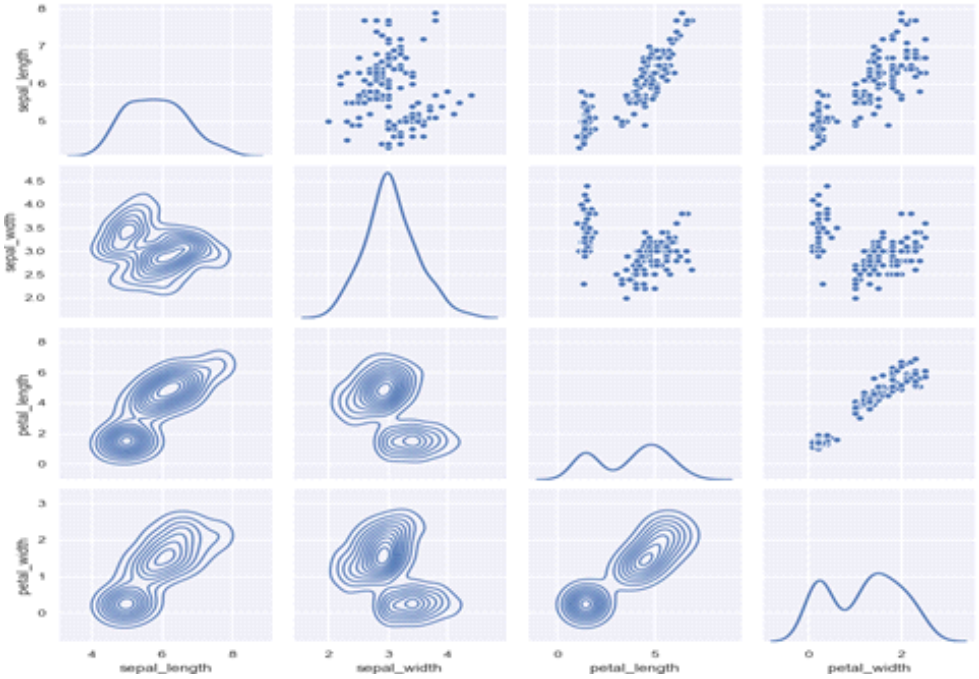
**Pairplot**

```
g = sns.pairplot(iris, diag_kind="kde", markers="+",
                 plot_kws=dict(s=50, edgecolor="b", linewidth=1),
                 diag_kws=dict(shade=True))
```




**Pairgrid**

```
g = sns.PairGrid(iris)
g = g.map_upper(sns.scatterplot)
g = g.map_lower(sns.kdeplot, colors="C0")
g = g.map_diag(sns.kdeplot, lw=2)
```




**Why not just run a correlation matrix here?**

The correlation matrix only provides a single numerical value without providing any information about the distribution which provides an in-depth picture of empirical relationships between variables in the bivariate analysis.

**Practical Tips:**

Please note this only works for numerical variables (to do it for categorical we need to first convert to numerical forms with techniques like one-hot encoding).

For very large datasets, group independent variables into groups of 10/15/20 and then run bivariate for each with respect to the dependent variable.

**EDA with one line of code!**

The pandas profiling library – a shorthand & quick way for EDA and bivariate analysis – more on this here. It does most of the univariate, bivariate and other EDA analyses.

## Multivariate analysis:

Multivariate analysis is a statistical technique used to analyze data that involves multiple variables simultaneously. Unlike univariate or bivariate analysis, which focus on one or two variables, multivariate analysis allows for the examination of more complex relationships between multiple variables. It is used to understand the relationships between several variables and to identify patterns and trends within complex datasets.

There are various types of multivariate analysis techniques, some of which include:

1. Principal Component Analysis (PCA): PCA is used to reduce the dimensionality of a dataset while retaining as much of the original variation as possible.
2. Factor Analysis: Factor analysis is used to identify underlying factors that explain the correlations among a set of observed variables.
3. Cluster Analysis: Cluster analysis is used to group similar observations or variables into clusters, based on defined criteria.
4. Multidimensional Scaling (MDS): MDS is used to analyze the structure of similarities or dissimilarities in data.
5. Multivariate Analysis of Variance (MANOVA): MANOVA is an extension of analysis of variance (ANOVA) that examines the differences between multiple dependent variables across different groups.
6. Canonical Correlation Analysis (CCA): CCA is used to examine the relationships between two sets of variables.

Multivariate analysis is essential in various fields, including social sciences, economics, marketing, and natural sciences, where complex relationships and interactions between multiple variables need to be studied and understood. By considering multiple variables simultaneously, researchers can gain deeper insights into the underlying patterns and structures within the data, leading to a more comprehensive understanding of the phenomena being studied.