

Data Visualisation with Python Programming

- Presentation By Uplatz
- Contact us: <https://training.uplatz.com>
- Email: info@uplatz.com
- Phone: +44 7836 212635

Data Visualisation using Seaborn

Learning outcomes:

- **Strip Plot**
- **Swarm Plot**
- **Plotting Bivariate Distribution:**
 - **Scatter plot, Hexbin plot, KDE, Regplot**
- **Visualizing Pairwise Relationship**
- **Box plot, Violin Plots, Point Plot**

Strip Plot

A strip plot is a graphical data analysis technique for summarizing a univariate data set.

A strip plot is a scatter plot where one of the variables is categorical. They can be combined with other plots to provide additional information.

Categorical data is represented in x-axis and values correspond to them represented through y-axis.

seaborn.stripplot(x,y,data) is used for creating strip plot. Here data set '**data**' is used if you want to use the values of '**x**' and '**y**' from data set '**data**'. You can also use **seaborn.stripplot(x,y)**.

Strip Plot

One problem with strip plots is how to display multiple points with the same value. Dataplot provides two options to address this.

With the **jitter** option, a small amount of random noise is added to the vertical coordinate. With the **stack** option, repeated values add a fixed increment to the vertical coordinate. So if there are 3 points with the same value, the y coordinates might be 1, 1.1, and 1.2. This gives the strip plot a histogram-like appearance.

Categorical variables represent types of data which may be divided into groups. Examples of categorical variables are **race, sex, age group, and educational level**.

Strip Plot

Example_1:

```
import matplotlib.pyplot as plt
import seaborn as sns
# x axis values
x=['sun', 'mon', 'tue', 'wed', 'thu','fri', 'sat']
y=[5, 6.7, 4, 6, 2, 4.9, 1.8] # y axis values
# plotting strip plot with seaborn
ax = sns.stripplot(x, y)
# giving labels to x-axis and y-axis
ax.set(xlabel='Days', ylabel='Amount_spend')
plt.title('My first graph');
plt.show()
```

Strip Plot

Example_2: Stripplot using inbuilt data-set given in seaborn

```
import matplotlib.pyplot as plt
import seaborn as sns
iris = sns.load_dataset('iris');
# use to set style of background of plot
sns.set(style = "whitegrid")
sns.stripplot(x = 'species', y = 'sepal_length', data =
iris);
plt.title('Graph');
plt.show()
```

Swarm Plot

Seaborn is an amazing visualization library for statistical graphics plotting in Python.

Seaborn swarm plot is probably similar to strip plot, only the points are adjusted so it won't get overlap to each other as it helps to represent the better representation of the distribution of values. A swarm plot can be drawn on its own, but it is also a good complement to a box plot, preferable because the associated names will be used to annotate the axes.

Swarm Plot

Example 1: Basic visualization of “fmri” dataset using swarmplot()

```
import matplotlib.pyplot as plt  
import seaborn as sns
```

```
sns.set(style = "whitegrid")  
fmri = sns.load_dataset("fmri")  
ax = sns.swarmplot(x='timepoint', y='signal',  
data=fmri)  
ax.set(xlabel = 'Time point', ylabel = 'Signal')  
plt.title('My first Swarm Plot');  
plt.show()
```

Swarm Plot

Example 2: Grouping data points on the basis of **category**, here as **region**

```
import matplotlib.pyplot as plt  
import seaborn as sns
```

```
sns.set(style = "darkgrid")  
fmri = sns.load_dataset("fmri")  
ax = sns.swarmplot(x='timepoint', y='signal', hue  
="region", data=fmri)  
ax.set(xlabel = 'Time point', ylabel = 'Signal')  
plt.title('My first Swarm Plot');  
plt.show()
```

Plotting Bivariate Distribution

Bivariate Distribution is used to determine the relation between two variables. This mainly deals with relationship between two variables and how one variable is behaving with respect to the other. The best way to analyze Bivariate Distribution in seaborn is by using the **jointplot()** function. Jointplot creates a multi-panel figure that projects the bivariate relationship between two variables and also the univariate distribution of each variable on separate axes.

Plotting Bivariate Distribution

Scatter Plot:

Scatter plot is the most convenient way to visualize the distribution where each observation is represented in two-dimensional plot via x & y axis.

Example:

```
import seaborn as sb
from matplotlib import pyplot as plt
df = sb.load_dataset('iris')
sb.jointplot(x = 'petal_length', y = 'petal_width', data
= df)
plt.show()
```

Plotting Bivariate Distribution

Hexbin Plot:

Hexagonal binning is used in bivariate data analysis when the data is sparse in density i.e., when the data is very scattered and difficult to analyze through scatterplots.

An additional parameter called '**kind**' and value '**hex**' plots the **hexbin** plot.

Plotting Bivariate Distribution

Hexbin Plot:

Example:

```
import seaborn as sb
from matplotlib import pyplot as plt
df = sb.load_dataset('iris')
sb.jointplot(x = 'petal_length', y = 'petal_width', data
= df, kind = 'hex')
plt.show()
```

Plotting Bivariate Distribution

Kernel Density Estimation:

Kernel density estimation is a non-parametric way to estimate the distribution of a variable. In seaborn, we can plot a kde using **jointplot()**. Pass value 'kde' to the parameter 'kind' to plot kernel plot.

Plotting Bivariate Distribution

Kernel Density Estimation:

Example:

```
import seaborn as sb
from matplotlib import pyplot as plt
df = sb.load_dataset('iris')
sb.jointplot(x = 'petal_length',y = 'petal_width',data
= df,kind = 'kde')
plt.show()
```


Plotting Bivariate Distribution

Regplots: Plot data & a linear regression model fit. Two main functions in Seaborn are used to visualize a linear relationship as determined through regression. These functions [regplot\(\)](#) and [lmpplot\(\)](#) are closely related, and share much of their core functionality. In terms of core functionality, **regplot()** is pretty similar to **lmpplot()** and solves similar purpose of visualizing a linear relationship as determined through the **Regression**. In the simplest invocation, both functions draw a scatterplot of two variables, x and y, and then fit the regression model $y \sim x$ and plot the resulting regression line and a 95% confidence interval for that regression:

Plotting Bivariate Distribution

Regplots:

Example_1: Creating regplot using `kind='reg'`

```
import seaborn as sb
```

```
from matplotlib import pyplot as plt
```

```
df = sb.load_dataset('iris')
```

```
sb.jointplot(x = 'petal_length', y = 'petal_width', data  
= df, kind = 'reg')
```

```
plt.show()
```

Plotting Bivariate Distribution

Regplots:

Example_2: Creating Regplot using `regplot()`

```
import seaborn as sb
```

```
from matplotlib import pyplot as plt
```

```
df = sb.load_dataset('iris')
```

```
sb.regplot(x="petal_length", y="petal_width",  
data=df)
```

```
plt.show()
```

Visualizing Pairwise Relationship

Datasets under real-time study contain many variables. In such cases, the relation between each and every variable should be analysed.

When you generalize joint plots to datasets of larger dimensions, you end up with **pair plots**. This is very useful for exploring correlations between multidimensional data, when you'd like to plot all pairs of values against each other.

To plot multiple pairwise bivariate distributions in a dataset, you can use the **pairplot()** function.

This shows the relationship for (n,2) combination of variable in a DataFrame as a matrix of plots and the diagonal plots are the univariate plots.

Visualizing Pairwise Relationship

Syntax for using pairplot():

seaborn.pairplot(data, hue, palette, kind, diag_kind)

data : Dataframe

hue: Variable in data to map plot aspects to different colors.

palette: Set of colors for mapping the hue variable

Kind: Kind of plot for the non-identity relationships.

{'scatter', 'reg'}

diag_kind: Kind of plot for the diagonal subplots.

{'hist', 'kde'}

Except **data**, all other parameters are **optional**. There are few other parameters which **pairplot** can accept.

Visualizing Pairwise Relationship

Example_1:

```
import seaborn as sb
```

```
import matplotlib.pyplot as plt
```

```
df = sb.load_dataset('tips')
```

```
print(df)
```

```
sb.pairplot(df, hue = 'sex') # pairplot with hue sex
```

```
plt.show() # to show the plot
```

Visualizing Pairwise Relationship

Example_2:

```
import seaborn as sb
import matplotlib.pyplot as plt
df = sb.load_dataset('iris')
sb.pairplot(df,hue = 'species',diag_kind =
"kde",kind = "scatter",palette = "husl")
plt.show()
```

We can observe the variations in each plot. The plots are in matrix format where the row name represents x axis and column name represents the y axis. The diagonal plots are kernel density plots where the other plots are scatter plots as mentioned.

Box Plot

Boxplot is a convenient way to visualize the distribution of data through their quartiles. Box plots usually have vertical lines extending from the boxes which are termed as whiskers. These whiskers indicate variability outside the upper and lower quartiles, hence Box Plots are also termed as **box-and-whisker** plot and **box-and-whisker** diagram. Any Outliers in the data are plotted as individual points.

We are going to use `seaborn.boxplot()` to create the box plot through seaborn library.

Box Plot

Example_1:

A box plot consist of 5 things.

Minimum, First Quartile or 25%, Median (Second Quartile) or 50%, Third Quartile or 75%, Maximum.

```
import seaborn as sb
from matplotlib import pyplot as plt
df = sb.load_dataset('tips')
print(df)
sb.boxplot(x = "day", y = "total_bill", data = df)
plt.show()
```

Box Plot

Example_1:

Let's take the first box plot i.e, *blue box plot* of the figure and understand these statistical things:

Bottom black horizontal line of blue box plot is **minimum** value. *First black* horizontal line of rectangle shape of blue box plot is **First quartile** or 25%

Second black horizontal line of rectangle shape of blue box plot is Second quartile or 50% or **median**.

Third black horizontal line of rectangle shape of blue box plot is **third quartile** or 75%

Top black horizontal line of rectangle shape of blue box plot is **maximum** value.

Small diamond shape of blue box plot is outlier data or **erroneous** data.

Violin Plots

Violin Plots are a combination of the box plot with the kernel density estimates. So, these plots are easier to analyze and understand the distribution of the data.

We are going to use `seaborn.violinplot()` to create the violin plot through seaborn library.

Violin Plots

Example_1:


Let us use tips dataset called to learn more into violin plots. This dataset contains the information related to the tips given by the customers in a restaurant.

```
import seaborn as sb
from matplotlib import pyplot as plt
df = sb.load_dataset('tips')
sb.violinplot(x = "day", y = "total_bill", data=df)
plt.show()
```

Violin Plots

Example_1:

The quartile and whisker values from the boxplot are shown inside the **violin**. As the **violin** plot uses KDE, the wider portion of violin indicates the higher density and narrow region represents relatively lower density. The Inter-Quartile range in boxplot and higher density portion in kde fall in the same region of each category of violin plot.

The above plot shows the distribution of total_bill on four days of the week. But, in addition to that, if we want to see how the distribution behaves with respect to sex, let's explore it in below example. 

Violin Plots

Example_2:

```
import seaborn as sb
from matplotlib import pyplot as plt
df = sb.load_dataset('tips')
sb.violinplot(x = "day", y = "total_bill", hue = 'sex',
data = df)
plt.show()
```

Point Plots

Point plots serve same as bar plots but in a different style. Rather than the full bar, the value of the estimate is represented by the point at a certain height on the other axis.

Remember, to create bar plot using seaborn library we need to use `seaborn.barplot()`

To create point plot using seaborn library we need to use `seaborn.pointplot()`

Point Plots

Example:

```
import seaborn as sb
from matplotlib import pyplot as plt
df = sb.load_dataset('titanic')
sb.pointplot(x = "sex", y = "survived", hue = "class",
data = df)
plt.show()
```




Thank you