Data Visualization

Nouman M Durrani

Seaborn

- “If Matplotlib “tries to make easy things easy and hard things possible”, seaborn tries to make a well-defined set of hard things easy too” – Michael Waskom (Creator of **Seaborn**).

- When working with Pandas, Matplotlib doesn’t serve well with DataFrames, while Seaborn functions actually work on DataFrames.

- Python Seaborn library is used to ease the challenging task of data visualization
  - It is based on Matplotlib.

Seaborn

- Seaborn allows the creation of statistical graphics through the following functionalities:
  - An API that is based on datasets allowing comparison between multiple variables

  - Supports multi-plot grids
  - Univariate and bivariate visualizations
  - Availability of different color palettes
  - Estimates and plots linear regression automatically

Seaborn to visualize a Pandas DataFrame

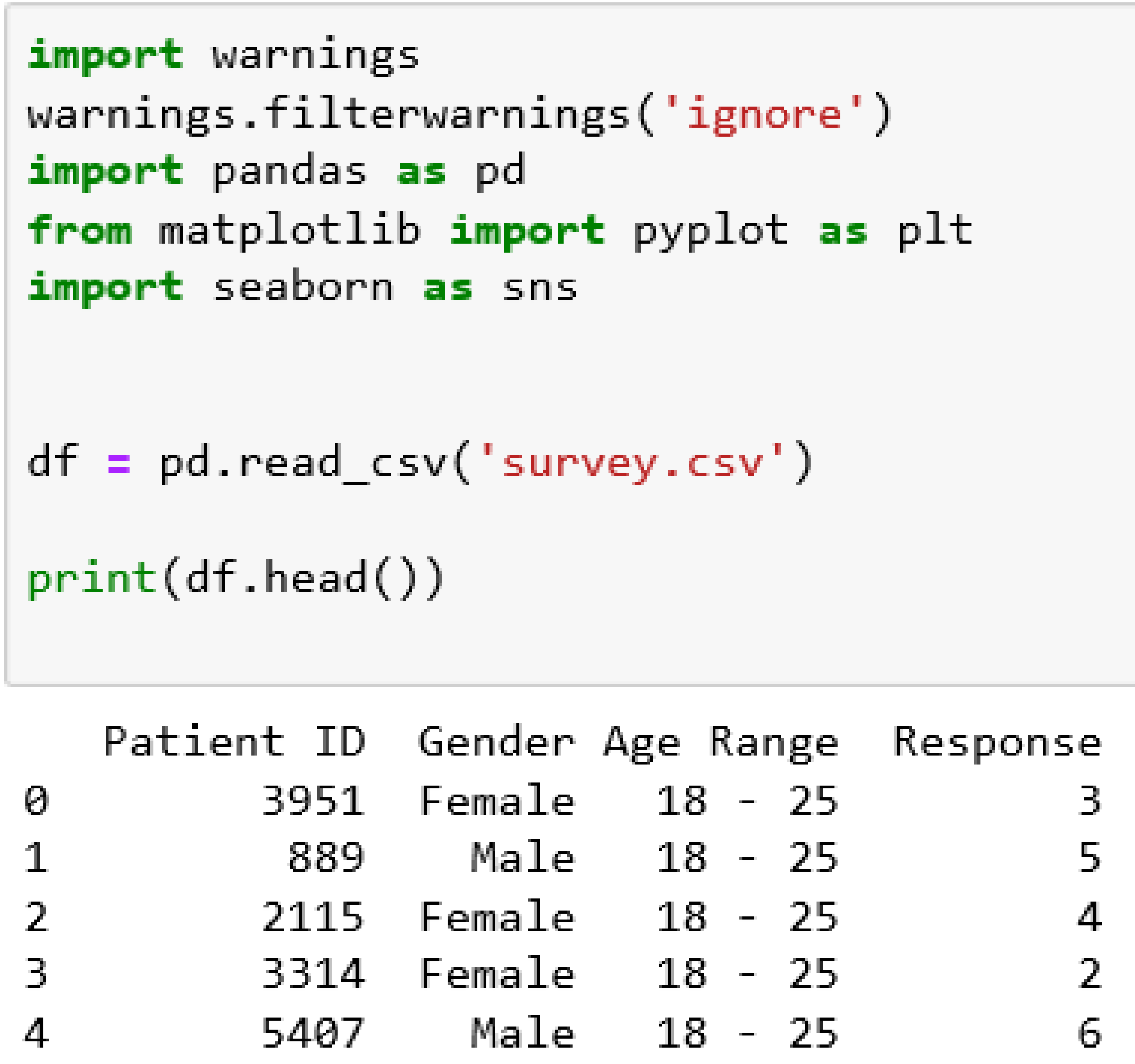
- DataFrames contain data structured into rows and columns.

- You can create a DataFrame from a local CSV file (CSV files store data in a tabular format).

```
df = pd.read_csv('file_name.csv')
```

Seaborn to visualize a Pandas DataFrame

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns


df = pd.read_csv('survey.csv')


print(df.head())
```



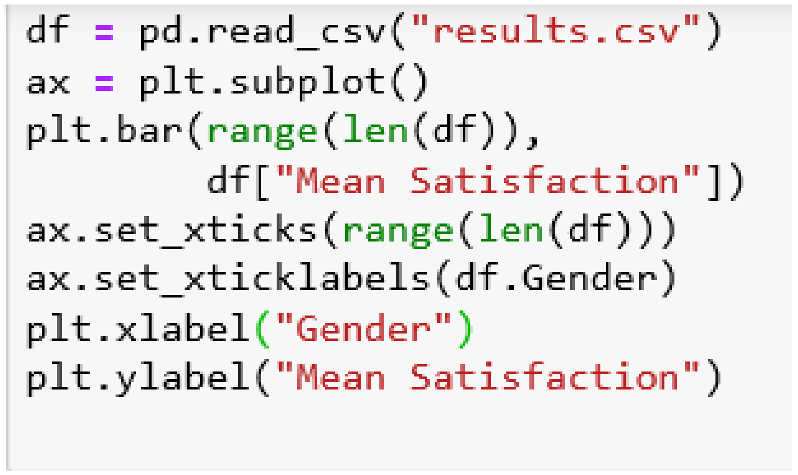
```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns


df = pd.read_csv('survey.csv')

print(df.head())
```

```
   Patient ID  Gender Age Range  Response
0        3951  Female   18 - 25         3
1         889    Male   18 - 25         5
2        2115  Female   18 - 25         4
3        3314  Female   18 - 25         2
4        5407    Male   18 - 25         6
```

Plotting Bars with Matplotlab vs Seaborn

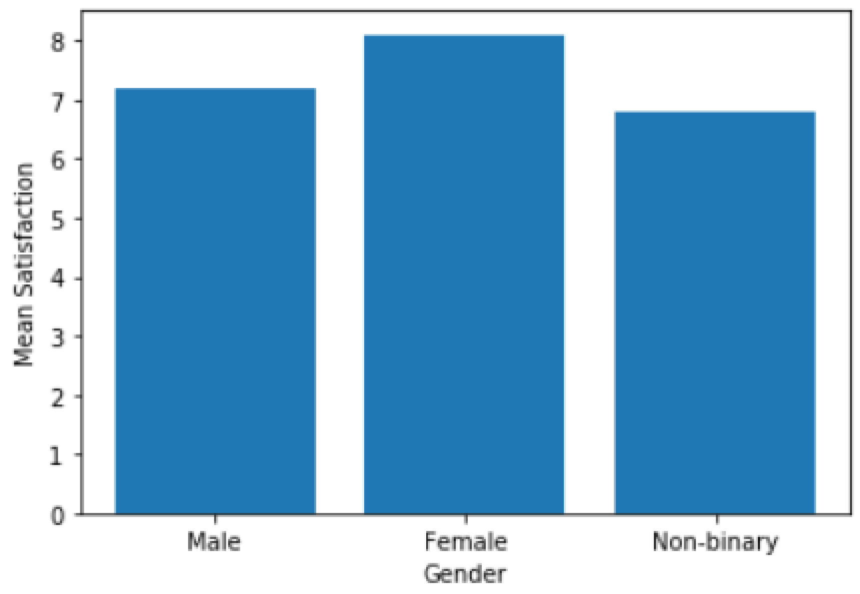
Problem: Suppose we are analyzing data from a survey: we asked 1,000 patients at a hospital how satisfied they were with their experience. Their response was measured on a scale of 1 - 10, with 1 being extremely unsatisfied, and 10 being extremely satisfied. We have summarized that data in a CSV file called results.csv.

```
df = pd.read_csv("results.csv")

ax = plt.subplot()

plt.bar(range(len(df)), df["Mean Satisfaction"])

ax.set_xticks(range(len(df)))

ax.set_xticklabels(df.Gender)

plt.xlabel("Gender")

plt.ylabel("Mean Satisfaction")
```



```
df = pd.read_csv("results.csv")
ax = plt.subplot()
plt.bar(range(len(df)),
        df["Mean Satisfaction"])
ax.set_xticks(range(len(df)))
ax.set_xticklabels(df.Gender)
plt.xlabel("Gender")
plt.ylabel("Mean Satisfaction")
```

```
Text(0, 0.5, 'Mean Satisfaction')
```



Plotting Bars with Matplotlab vs Seaborn

- The Seaborn function sns.barplot(), takes at least three keyword arguments:
  - data: a Pandas DataFrame that contains the data (in this example, data=df)
  - x: a string that tells Seaborn which column in the DataFrame contains other x-labels (in this case, x="Gender")
  - y: a string that tells Seaborn which column in the DataFrame contains the heights we want to plot for each bar (in this case y="Mean Satisfaction")
- By default, Seaborn will aggregate and plot the mean of each category.

Task 1:

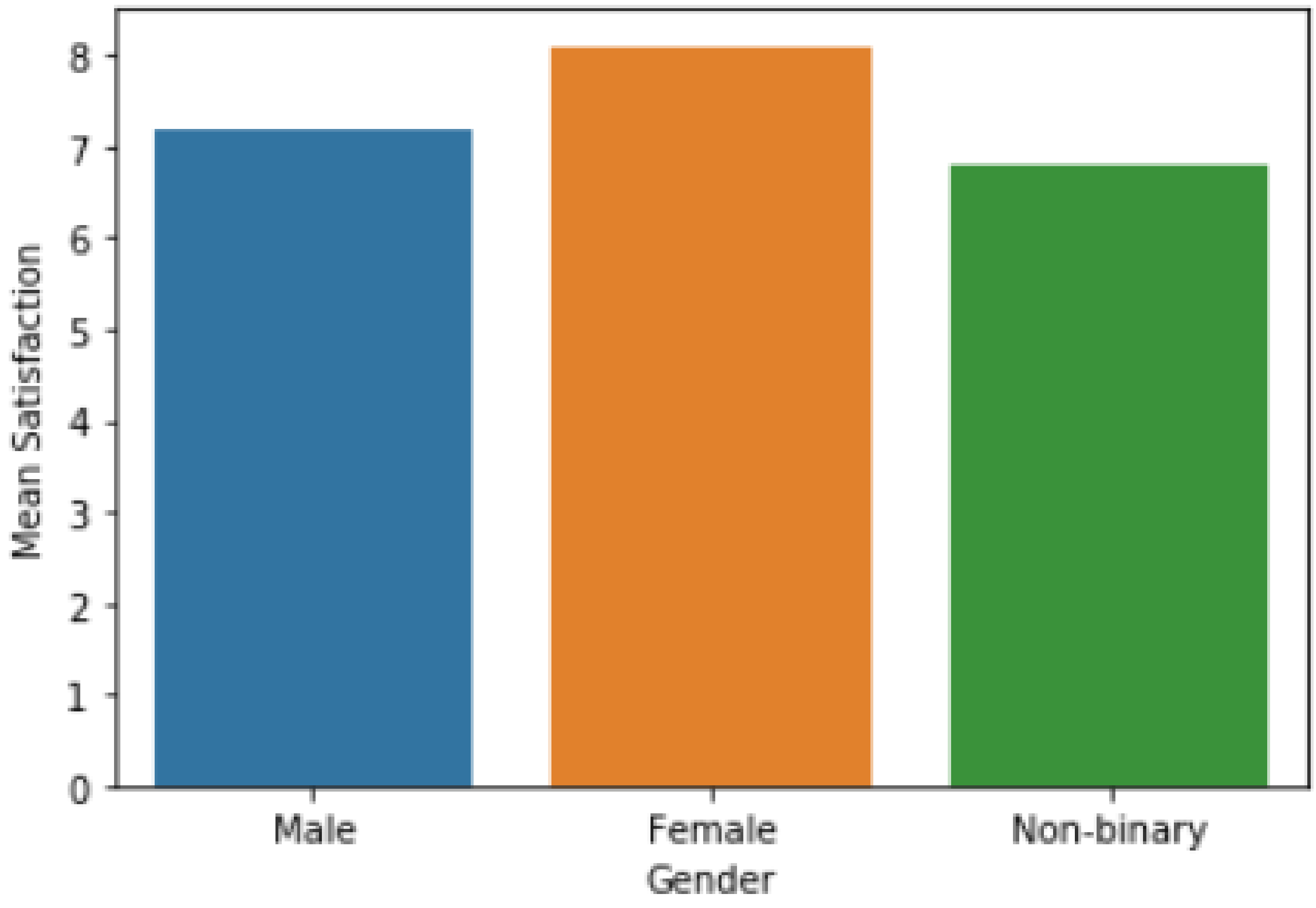
1. Use Pandas to load in the data from results.csv and save it to the variable df.
2. Display df using print
3. Remove all of the # characters from in front of the sns.barplot command and fill in the missing values.
4. Type plt.show() to display the completed bar plot.

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

# Load results.csv here:
# sns.barplot(
        # data= ,
        # x= ,
        # y=
# )
```

Task 1 Solution:

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

# Load results.csv here:
df = pd.read_csv('results.csv')
print(df)

sns.barplot(
            data=df,
            x='Gender',
            y='Mean Satisfaction'
)

plt.show()
```

```
      Gender  Mean Satisfaction
0       Male                7.2
1     Female                8.1
2 Non-binary                6.8
```



Understanding Aggregates

- Seaborn can also calculate aggregate statistics for large datasets.
- a grade book with columns student, assignment_name, and grade

student	assignment_name	grade
Amy	Assignment 1	75
Amy	Assignment 2	82
Bob	Assignment 1	99
Bob	Assignment 2	90
Chris	Assignment 1	72
Chris	Assignment 2	66
…	…	…

student	grade
Amy	78.5
Bob	94.5
Chris	69
…	…

assignment_name	grade
Assignment 1	82
Assignment 2	79.3
…	…

Task 2:

```
import codecademylib3_seaborn
import pandas as pd
from matplotlib import pyplot as plt

gradebook = pd.read_csv("gradebook.csv")
```

- In Python, you can compute aggregates fairly quickly and easily using Numpy.

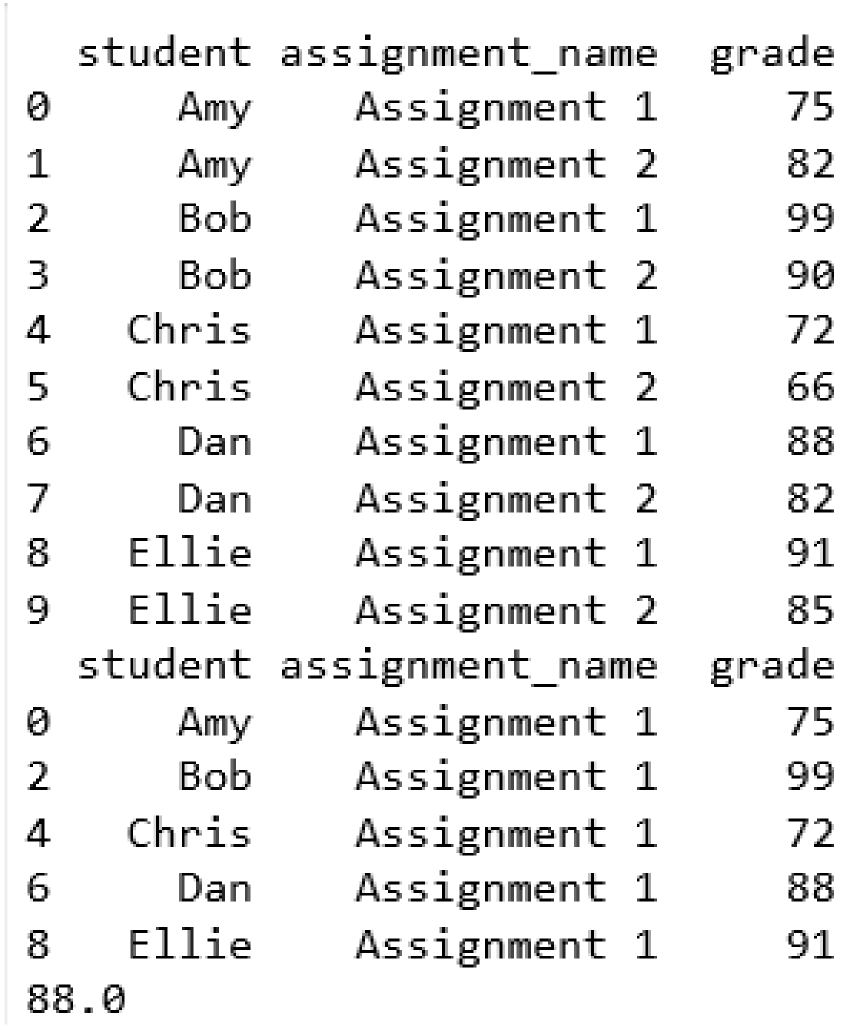
1. To calculate aggregates using Numpy, you'll first need to import the Numpy library

   ```
   import numpy as np
   ```

1. The DataFrame gradebook contains the complete gradebook for a hypothetical classroom. Use print to examine gradebook.
2. Select all rows from the gradebook DataFrame where assignment_name is equal to Assignment 1. Save the result to the variable assignment1. Print assignment1.
3. Now use Numpy to calculate the median grade in assignment1. Use np.median() to calculate the median of the column grade from assignment1 and save it to asn1_median.
4. Display asn1_median using print. What is the median grade on Assignment 1?

Task 2 Solution:

```
import codecademylib3_seaborn
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np

gradebook = pd.read_csv('gradebook.csv')
print(gradebook)

assignment1 = gradebook[gradebook.assignment_name ==
'Assignment 1']
print(assignment1)

asn1_median = np.median(assignment1.grade)
print(asn1_median)
```



```
  student assignment_name  grade
0     Amy    Assignment 1     75
1     Amy    Assignment 2     82
2     Bob    Assignment 1     99
3     Bob    Assignment 2     90
4   Chris    Assignment 1     72
5   Chris    Assignment 2     66
6     Dan    Assignment 1     88
7     Dan    Assignment 2     82
8   Ellie    Assignment 1     91
9   Ellie    Assignment 2     85
  student assignment_name  grade
0     Amy    Assignment 1     75
2     Bob    Assignment 1     99
4   Chris    Assignment 1     72
6     Dan    Assignment 1     88
8   Ellie    Assignment 1     91
88.0
```

Plotting Aggregates using Seaborn

student	assignment_name	grade
Amy	Assignment 1	75
Amy	Assignment 2	82
Bob	Assignment 1	99
Bob	Assignment 2	90
Chris	Assignment 1	72
Chris	Assignment 2	66
…	…	…

- Suppose this data is stored in a Pandas DataFrame called df.
- The Seaborn command (sns.barplot()) will plot this data in a bar plot and automatically aggregate the data:

```
sns.barplot(data=df, x="student", y="grade")
```

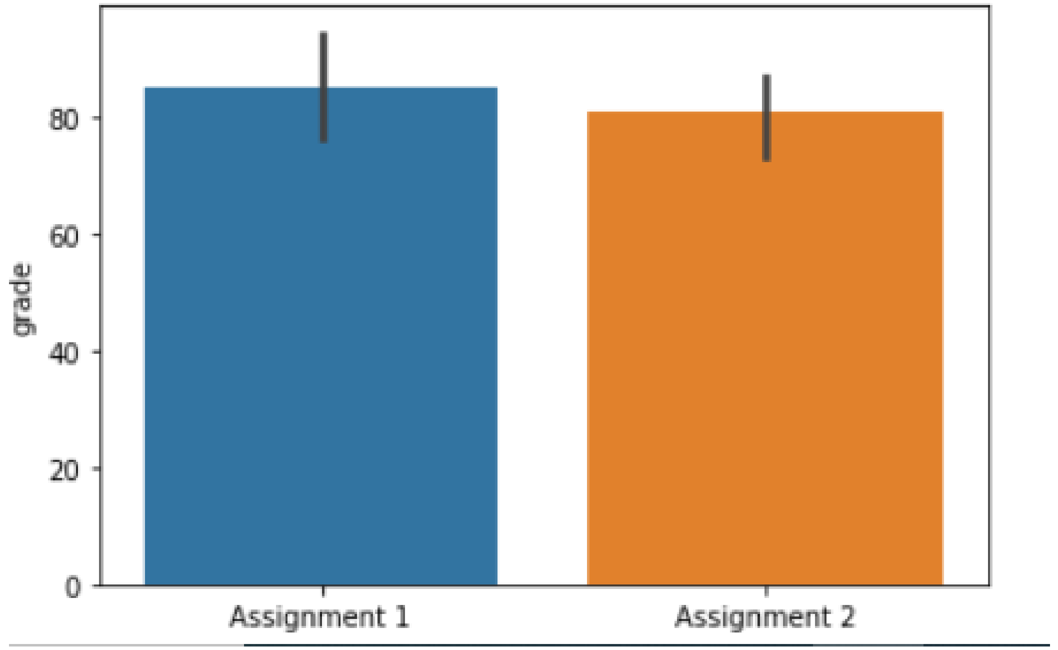
- In the example above, Seaborn will aggregate grades by student, and plot the average grade for each student.

Task 3:

1. Use Seaborn to plot the average grade for each assignment. Take a look at gradebook.csv for the column names.
2. Use plt.show() to display the graph.

Task 3 Solution:

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

gradebook = pd.read_csv("gradebook.csv")

sns.barplot(data=gradebook, x="assignment_name", y="grade")

plt.show()
```



Modifying Error Bars

- By default, Seaborn will place error bars on each bar when you use the barplot() function.

- For example, an error bar might indicate what grade we expect an average student to receive on this assignment.

- By default, Seaborn uses a **bootstrapped confidence interval**.

Modifying Error Bars

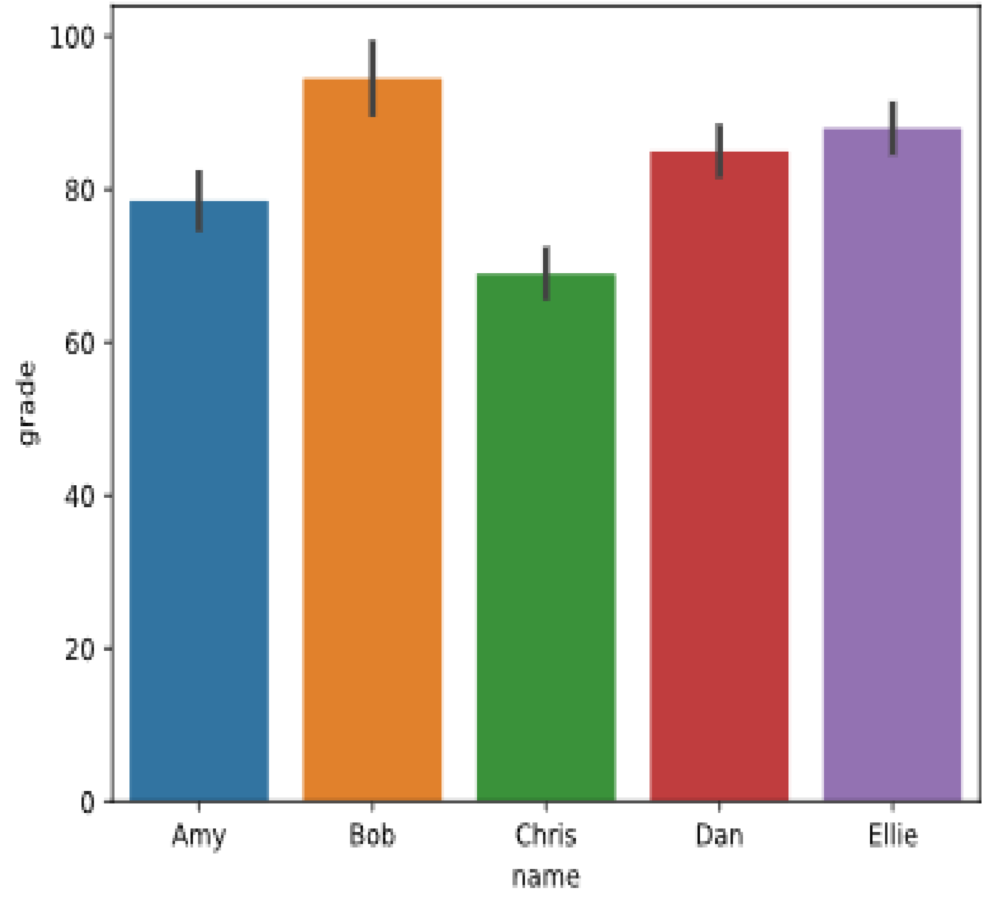
- ci="sd" to sns.barplot() represent one standard deviation.

```
sns.barplot(data=gradebook, x="name", y="grade", ci="sd")
```

Task 4:

Modify the bar plot so that the error bars represent one standard deviation, rather than 95% confidence intervals.

```
gradebook = pd.read_csv("gradebook.csv")

sns.barplot(data=gradebook, x="name", y="grade", ci="sd")
plt.show()
```



Calculating Different Aggregates

- If our data has many outliers, we may want to plot the median.
- If our data is categorical, we might want to count how many times each category appears

- Use the keyword argument estimator, which accepts any function that works on a list.

- To calculate the median, you can pass in np.median to the estimator keyword:

```
sns.barplot(data=df,
  x="x-values",
  y="y-values",
  estimator=np.median)
```

- To calculate the number of times a particular value appears in the Response column , we pass in len:

```
sns.barplot(data=df,
  x="Patient ID",
  y="Response",
  estimator=len)
```

Task 5:

1. Consider our hospital satisfaction survey data, which is loaded into the Pandas DataFrame df. Use print to examine the data.

2. We'd like to know how many men and women answered the survey. Use **sns.barplot()** with:

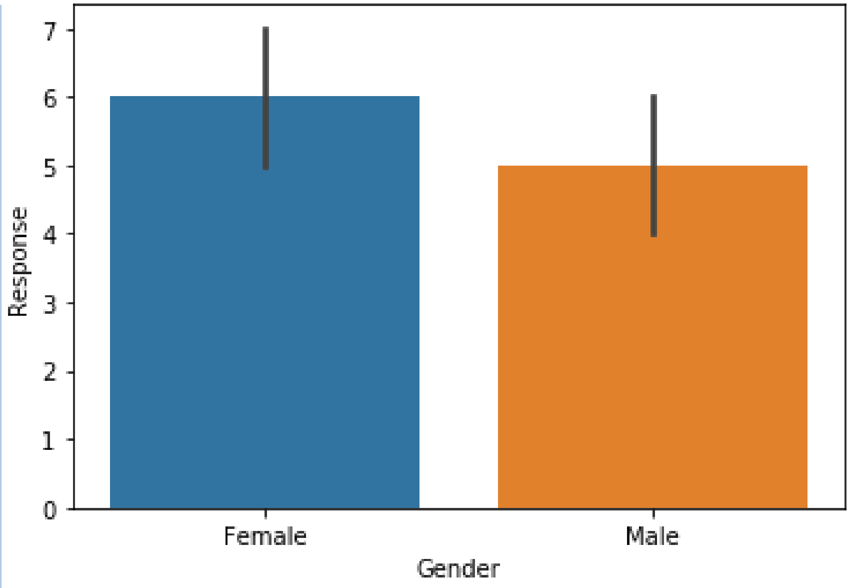
- data equal to df
- x equal to Gender
- y equal to Response
- estimator equal to len

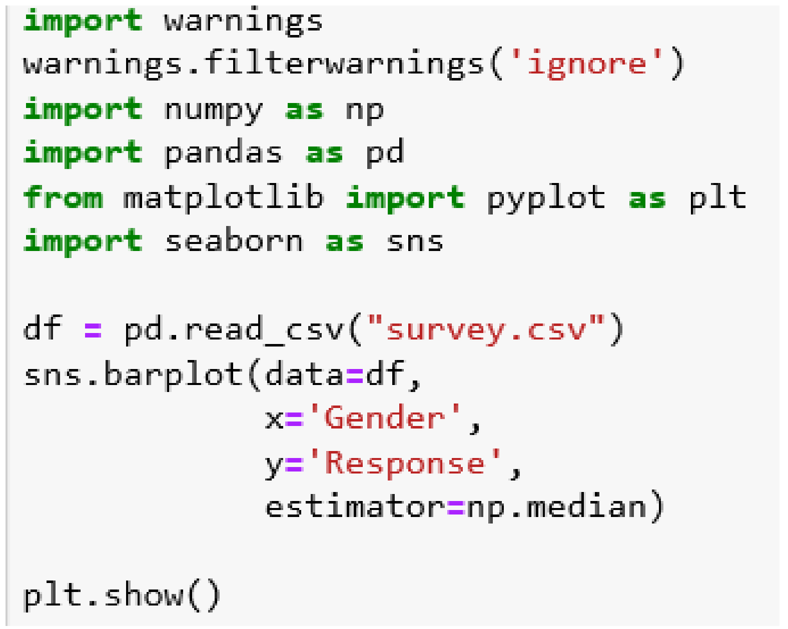
3. Change sns.barplot() to graph the median Response aggregated by Gender using estimator=np.median.

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

df = pd.read_csv("survey.csv")
```

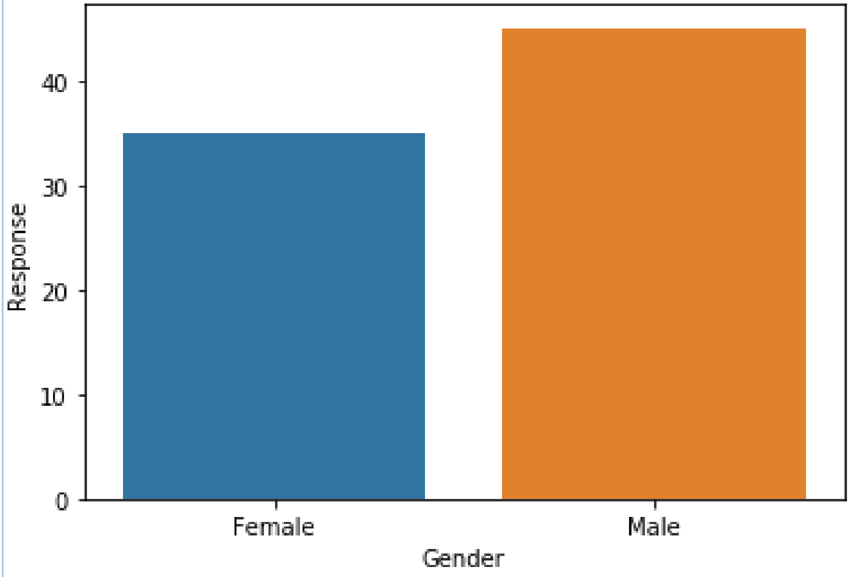
Task 4 Solution:

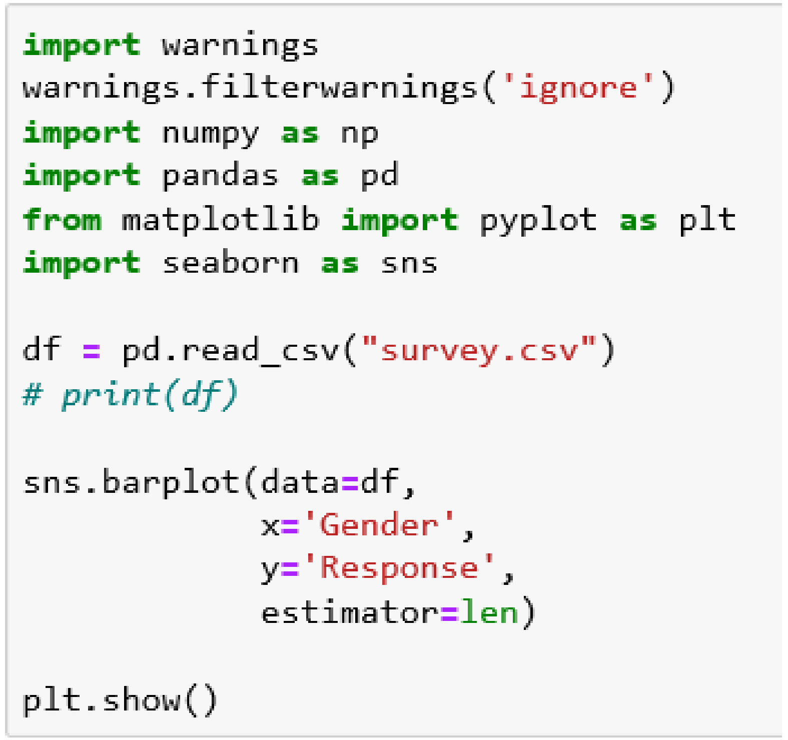
```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

df = pd.read_csv("survey.csv")
print(df)

#sns.barplot(data=df,
#           x='Gender',
#           y='Response',
#           estimator=len)
sns.barplot(data=df,
          x='Gender',
          y='Response',
          estimator=np.median)

plt.show()
```





```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

df = pd.read_csv("survey.csv")
sns.barplot(data=df,
            x='Gender',
            y='Response',
            estimator=np.median)

plt.show()
```



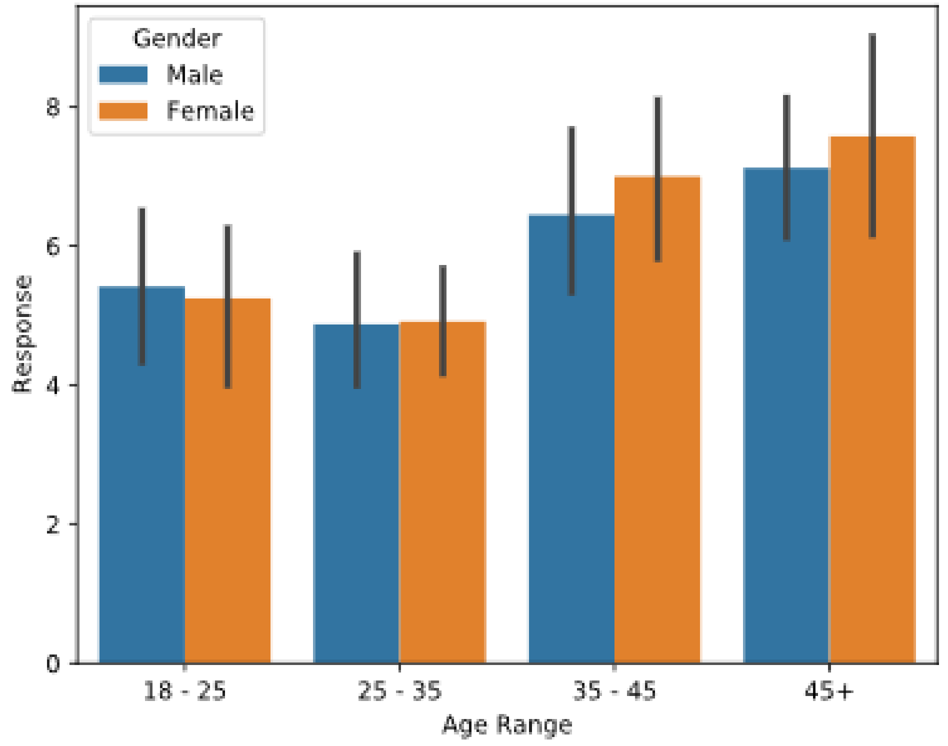


```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

df = pd.read_csv("survey.csv")
# print(df)

sns.barplot(data=df,
            x='Gender',
            y='Response',
            estimator=len)

plt.show()
```

Aggregating by Multiple Columns

- For example, consider our hospital survey data. The mean satisfaction seems to depend on Gender, but it might also depend on another column: Age Range.
- Compare both the Gender and Age Range factors at once by using the keyword hue.

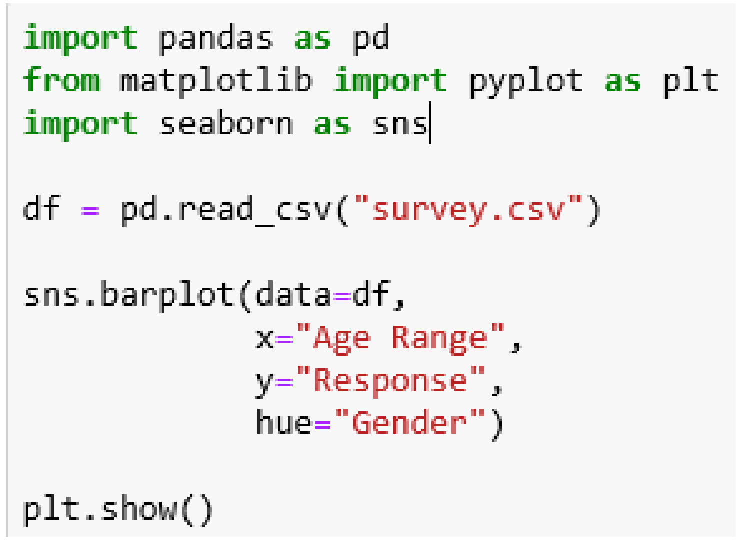
```
sns.barplot(data=df,
            x="Gender",
            y="Response",
            hue="Age Range")
```



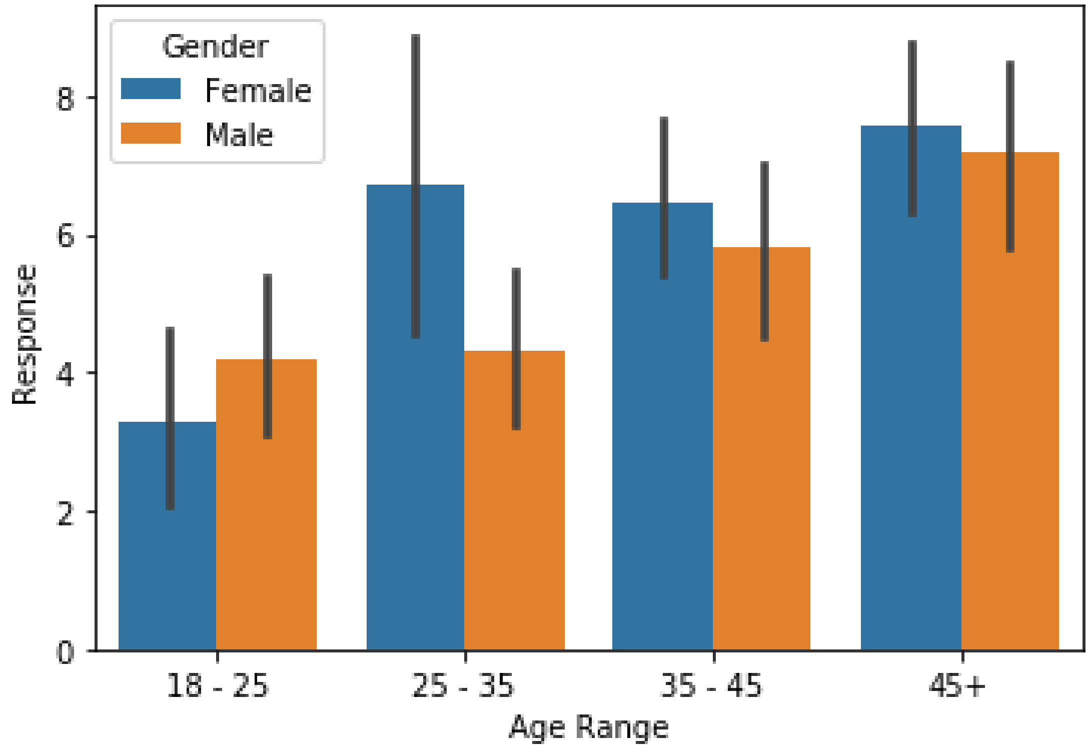
Visualizing survey results by gender with age range nested

Visualizing survey results by gender with age range nested

```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

df = pd.read_csv("survey.csv")

sns.barplot(data=df,
        x="Age Range",
        y="Response",
        hue="Gender")

plt.show()
```



```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

df = pd.read_csv("survey.csv")

sns.barplot(data=df,
            x="Age Range",
            y="Response",
            hue="Gender")

plt.show()
```



Load_dataset with Seaborn

- Import the required libraries as follows:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="darkgrid")
```

- Style attribute is also customizable

- Seaborn allows you to load any dataset from GIT using the load_dataset() function.

```
f = sns.load_dataset("flights")
```

- You can also view all the available datasets using get_dataset_names() function as follows:

```
sns.get_dataset_names()
```

Built-in Themes

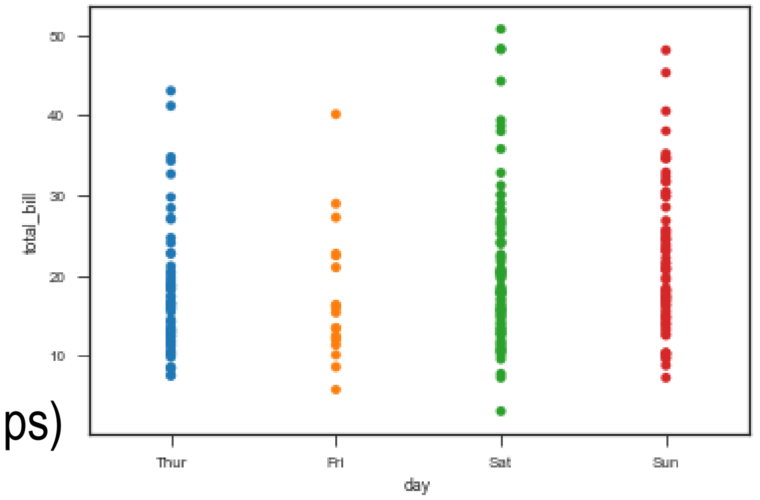
- Seaborn has five built-in themes to style its plots: darkgrid, whitegrid, dark, white, and ticks. Seaborn defaults to using the darkgrid theme for its plots, but you can change this styling
- To use any of the preset themes pass the name of it to sns.set_style().
  - Style attribute can take any value such as darkgrid, ticks, etc.

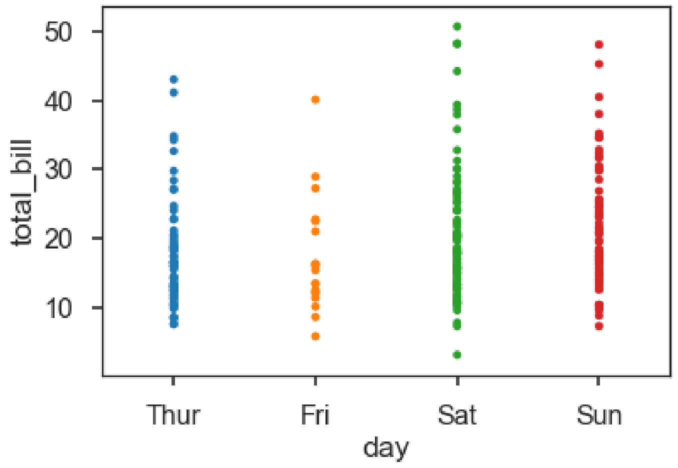
```
sns.set_style("darkgrid")
sns.stripplot(x="day", y="total_bill", data=a)
```

Scaling Figure Styles for Different Mediums

- Styling plots for different presentation purposes is difficult in Matplotlib.
- Seaborn makes it easy to produce the same plots in a variety of different visual formats so you can customize the presentation of your data for the appropriate context.
- Seaborn has four presets (paper, notebook, talk, and poster) which set the size of the plot and allow you to customize your figure depending on how it will be presented.
- The notebook style is the default.

```
sns.set_style("ticks")

# Smallest context:
sns.set_context("paper")
sns.stripplot(x="day", y="total_bill", data=tips)
```





relplot():

- This is a figure-level-function that makes use of two other axes functions

    - scatterplot()
    - lineplot()

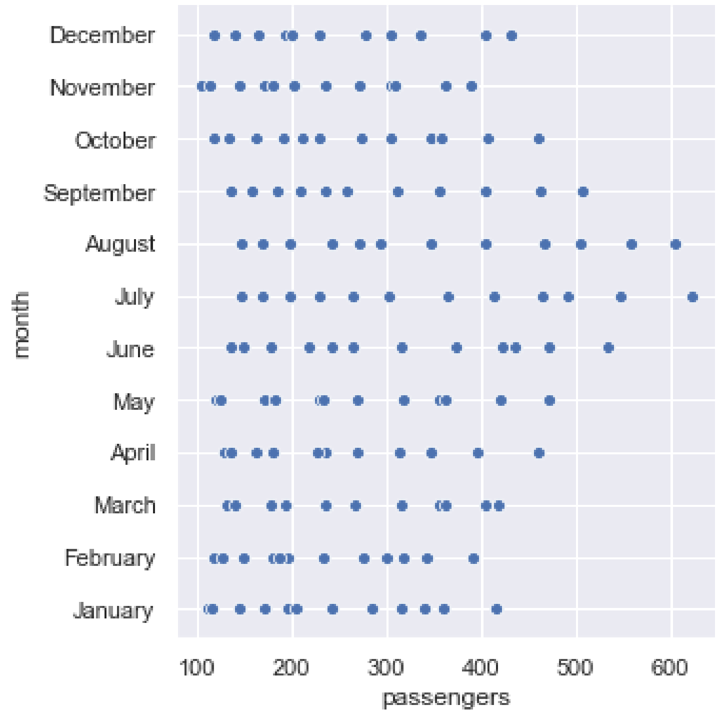
- These functions can be specified using the ‘kind’ parameter of relplot().
- It takes the default one which is scatterplot().

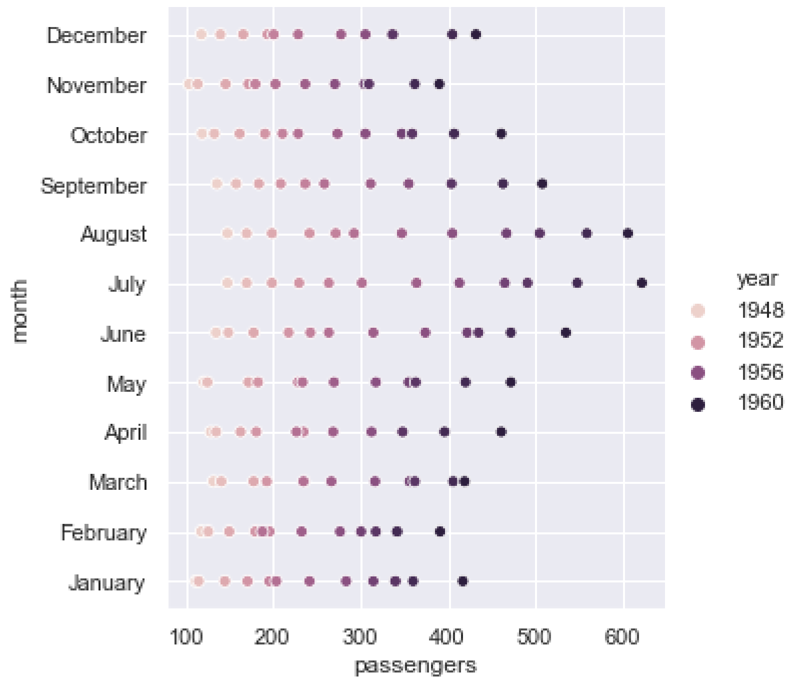
relplot():

```
f = sns.load_dataset("flights")
sns.relplot(x="passengers", y="month", data=f);
```

We can add another dimension using the ‘hue’.

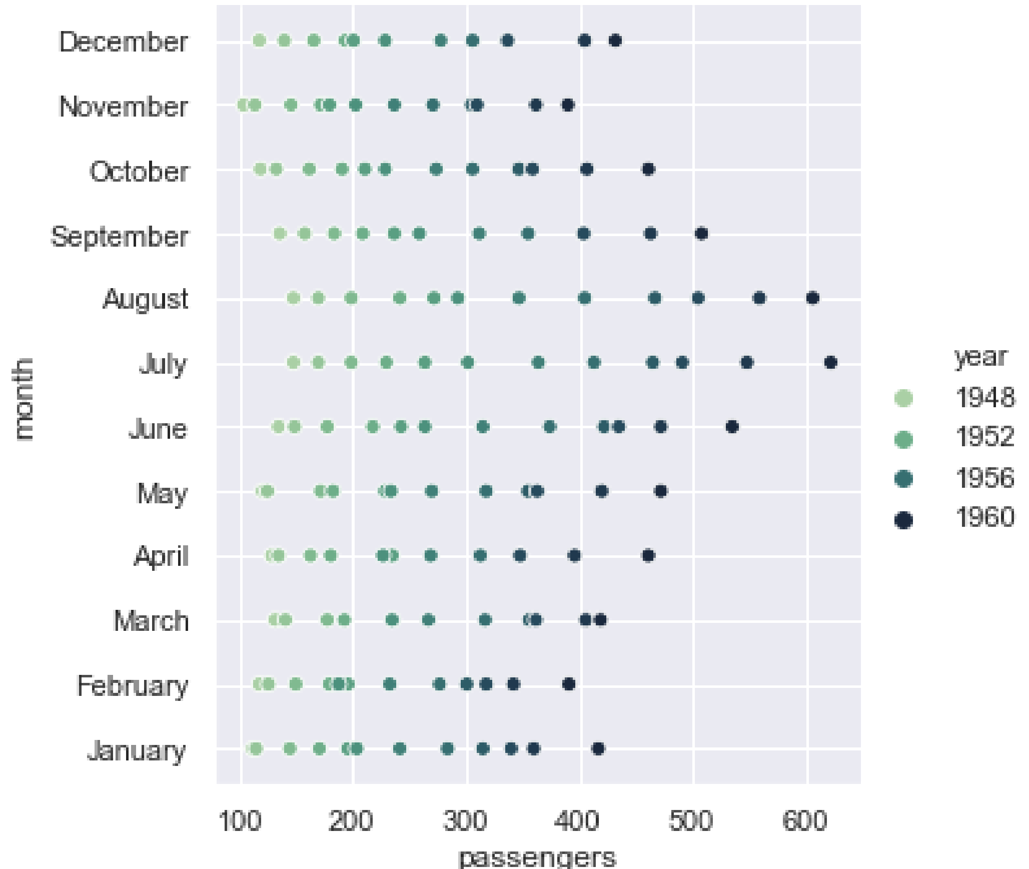
```
f = sns.load_dataset("flights")
sns.relplot(x="passengers", y="month", hue="year", data=f);
```





More customizations

```
sns.set(style="darkgrid")
f = sns.load_dataset("flights")
sns.relplot(x="passengers", y="month", hue="year", palette="ch:r=-.5,l=.75", data=f);
```

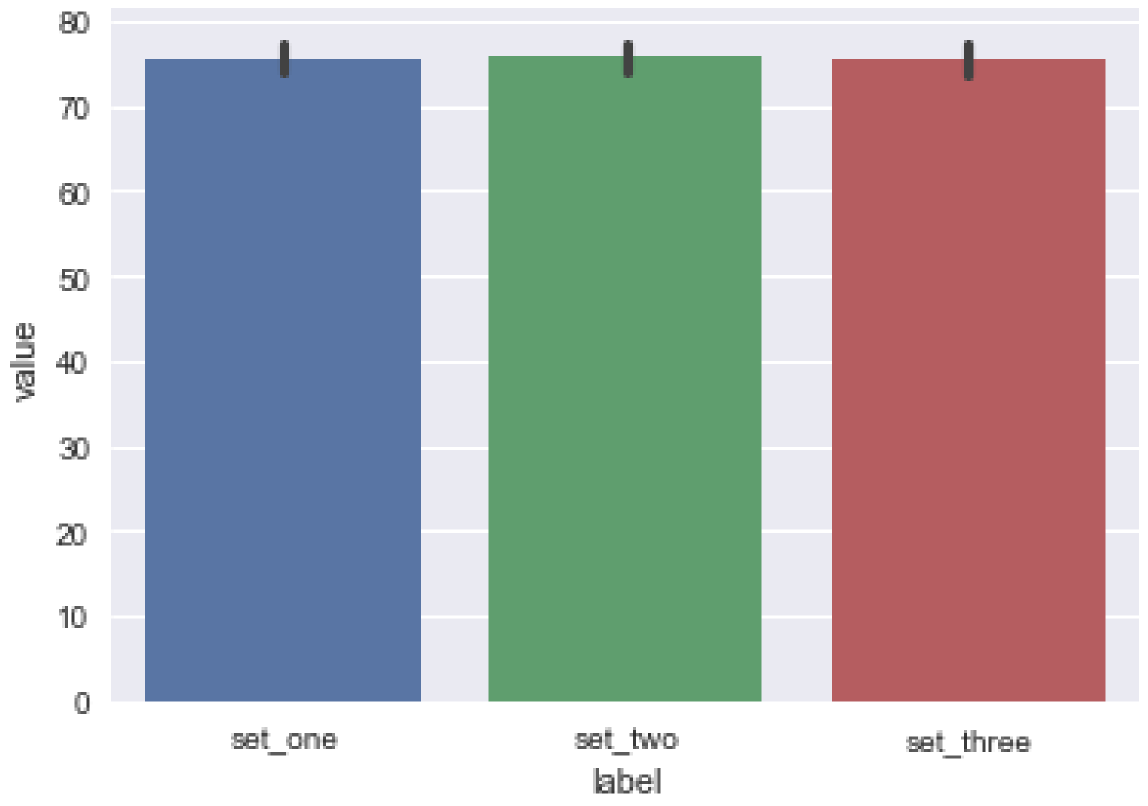


Plotting Distributions with Seaborn

- Seaborn is optimized to work with large datasets
- One of the most powerful aspects of Seaborn is its ability to visualize and compare distributions.
    - Distributions provide us with more information about our data — how spread out it is, its range, etc.
- Seaborn includes KDE plots, box plots, and violin plots.

Plotting Distributions with Seaborn

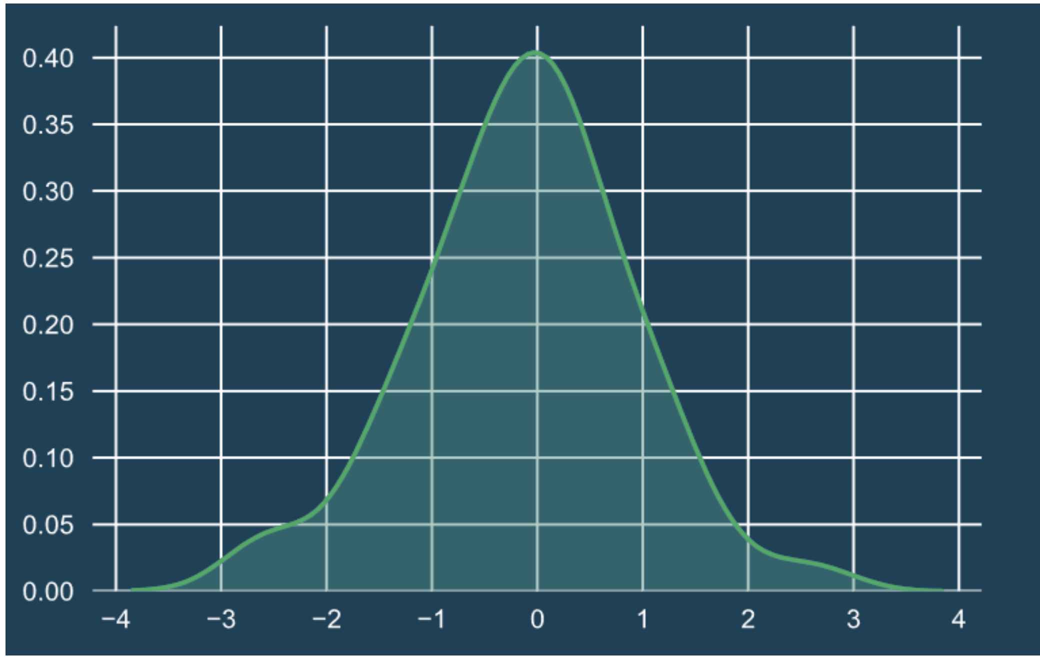
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
n = 500
dataset1 = np.genfromtxt("dataset1.csv", delimiter=",")
dataset2 = np.genfromtxt("dataset2.csv", delimiter=",")
dataset3 = np.genfromtxt("dataset3.csv", delimiter=",")
df = pd.DataFrame({
    "label": ["set_one"] * n + ["set_two"] * n + ["set_three"] * n,
    "value": np.concatenate([dataset1, dataset2, dataset3])
})

sns.set()
sns.barplot(data=df, x='label', y='value')
plt.show()
```



KDE Plots

- KDE stands for Kernel Density Estimator. A KDE plot gives us the sense of a univariate as a curve.

- A univariate dataset only has one variable

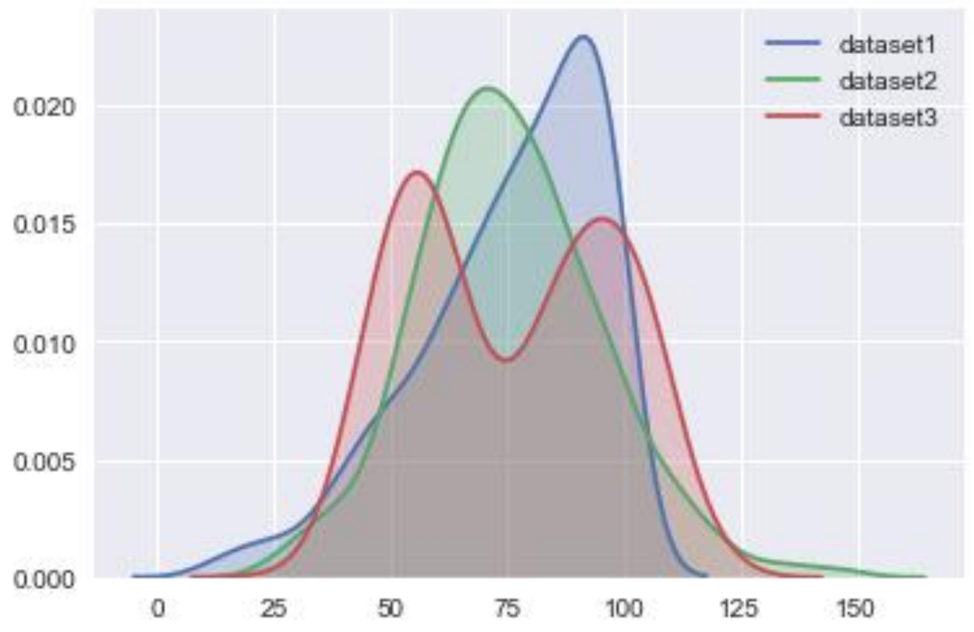


KDE Plots

- To plot a KDE in Seaborn, we use the method sns.kdeplot().
- A KDE plot takes the following arguments:
  - data - the univariate dataset being visualized, like a Pandas DataFrame, Python list, or NumPy array
  - shade - a boolean that determines whether or not the space underneath the curve is shaded
- Let's examine the KDE plots of our three datasets:

```
sns.kdeplot(dataset1, shade=True)
sns.kdeplot(dataset2, shade=True)
sns.kdeplot(dataset3, shade=True)
plt.legend()
plt.show()
```

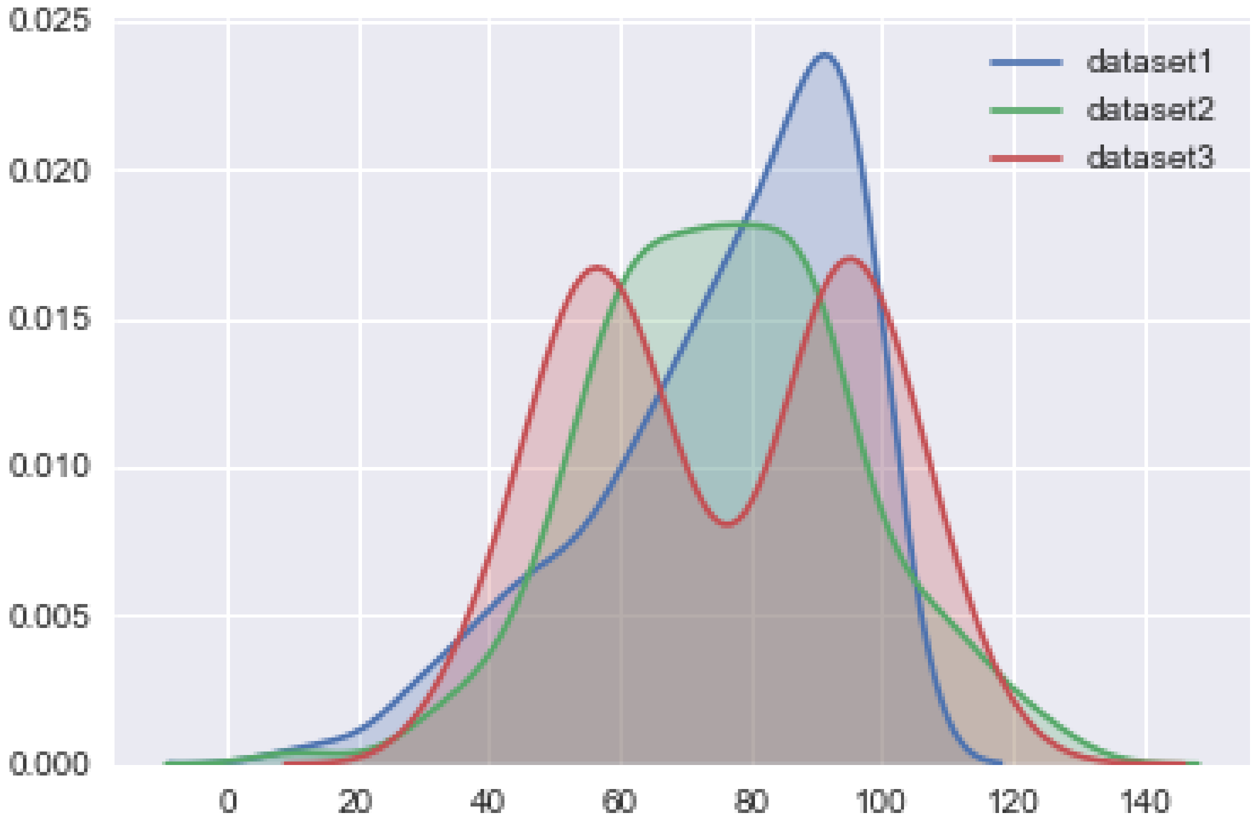
- Dataset 1 is skewed left
- Dataset 2 is normally distributed
- Dataset 3 is bimodal (it has two peaks)



KDE Plots

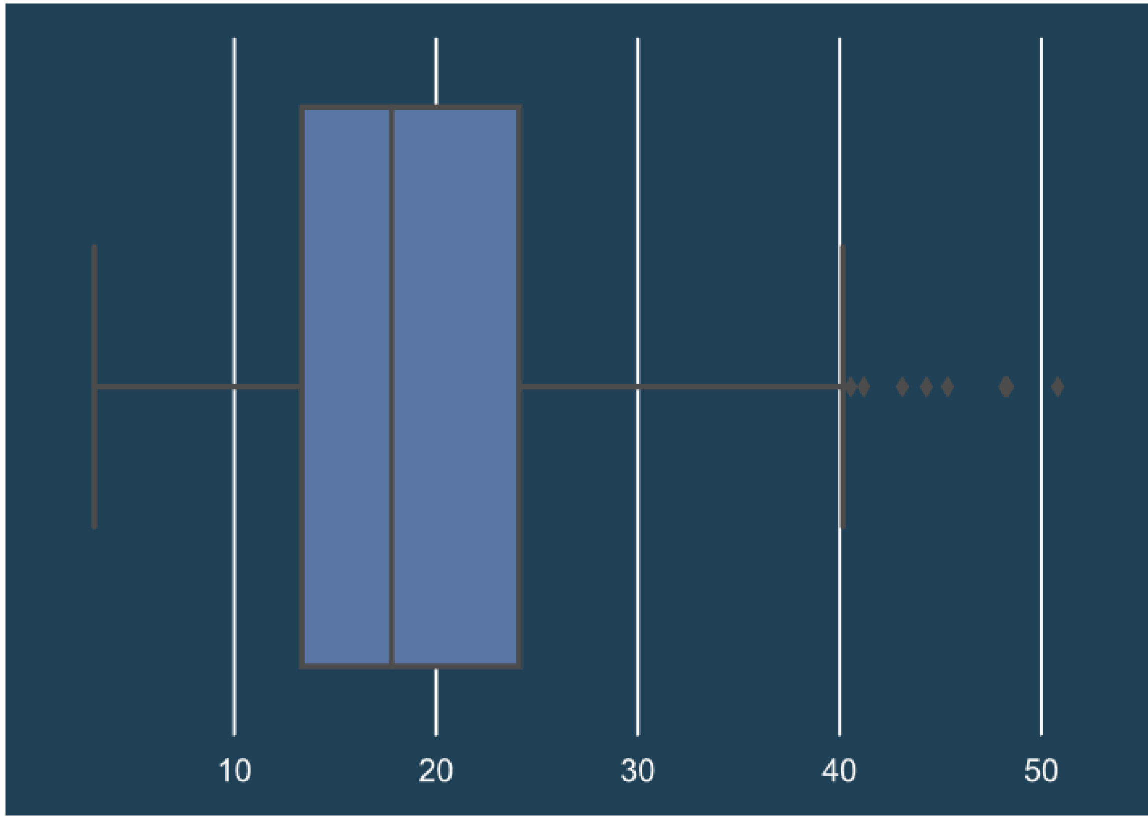
- We can use barplots to find out information about the mean - but it doesn't give us a sense of how spread out the data is in each set.
- To find out more about the distribution, we can use a KDE plot.

```
sns.kdeplot(dataset1, shade=True, label="dataset1")
sns.kdeplot(dataset2, shade=True, label="dataset2")
sns.kdeplot(dataset3, shade=True, label="dataset3")

plt.legend()
plt.show()
```



Box Plots

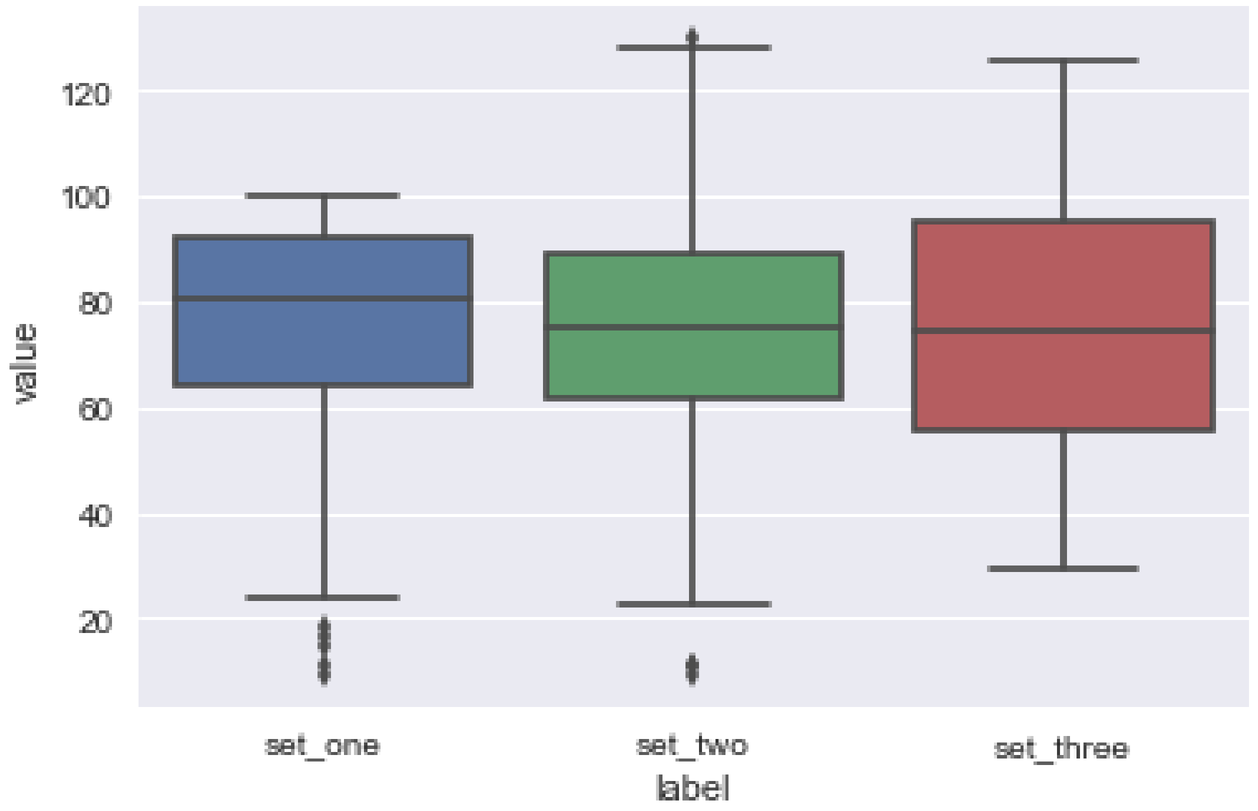
- The box plot (also known as a box-and-whisker plot) can tell us about how our dataset is distributed.
- It gives us an idea about where a significant portion of our data lies, and whether or not any outliers are present.
    - The **box** represents the interquartile range
    - The **line in the middle** of the box is the median
    - The **end lines** are the first and third quartiles
    - The **diamonds** show outliers
- However, we lose the ability to determine the shape of the data



Box Plots

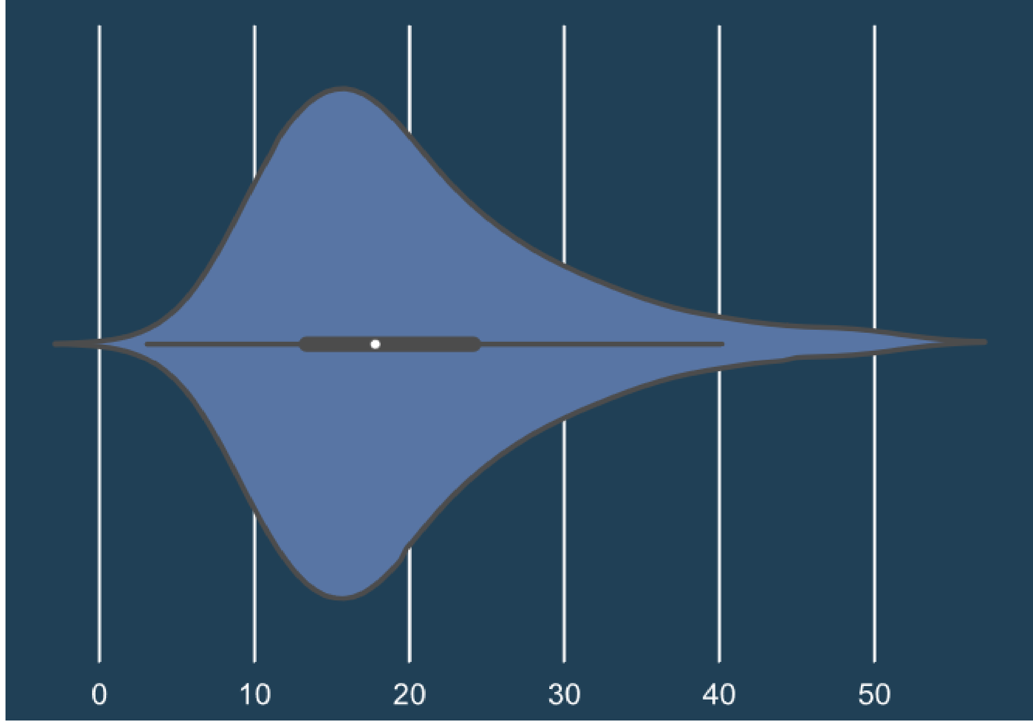
- A box plot takes the following arguments:
  - data - the dataset we're plotting, like a DataFrame, list, or an array
  - x - a one-dimensional set of values, like a Series, list, or array
  - y - a second set of one-dimensional data

```
sns.boxplot(data=df, x='label', y='value')
plt.show()
```



- if you use the value Series as your y value data, Seaborn will automatically apply that name as the y-axis label.

Violin plots

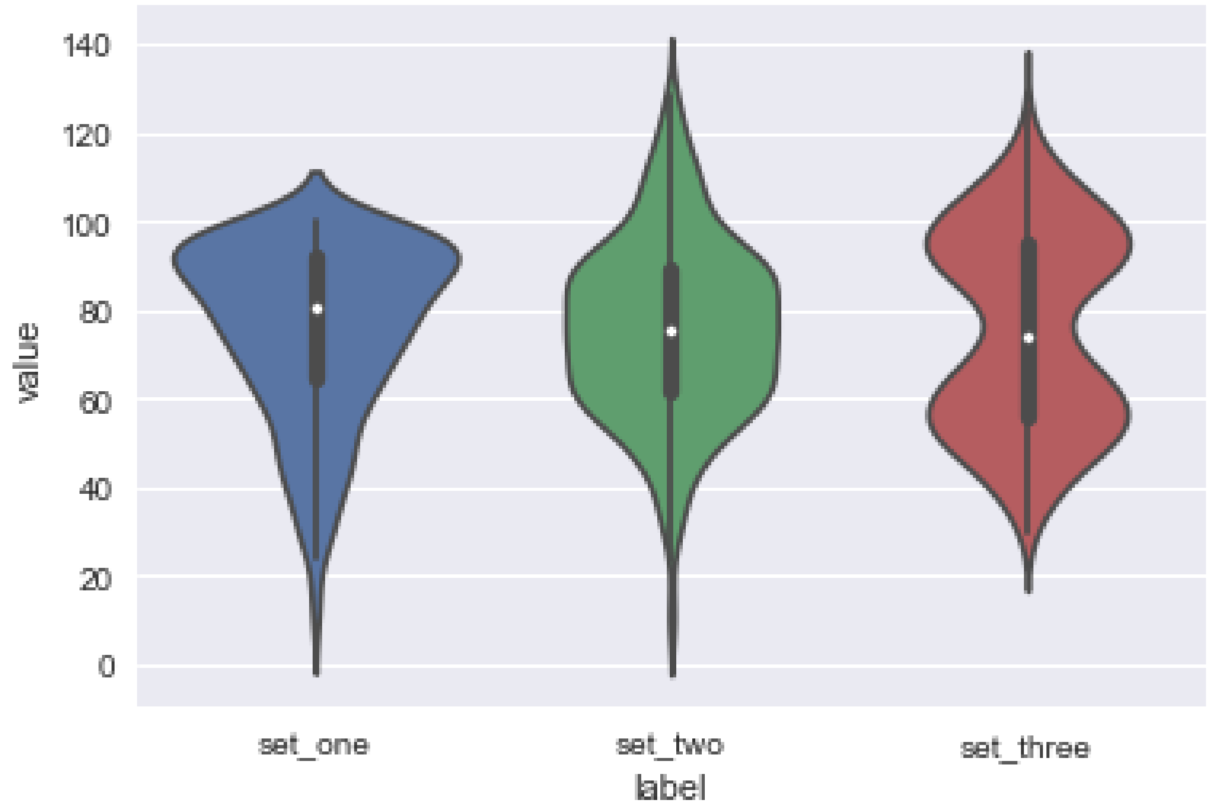


- Violin plots provide more information than box plots because instead of mapping each individual data point, we get an estimation of the dataset using the KDE.
  - There are two KDE plots that are symmetrical along the center line.
  - A white dot represents the median.
  - The thick black line in the center of each violin represents the interquartile range.
  - The lines that extend from the center are the confidence intervals - just as we saw on the bar plots, a violin plot also displays the 95% confidence interval.

Violin plots

- A violin plot brings together shape of the KDE plot with additional information that a box plot provides.

```
sns.violinplot(data=df, x="label", y="value")

plt.show()
```



Seaborn Styling, Part 1: Figure Style and Scale

- Styling is the process of customizing the overall look of your visualization, or figure.
- Making intentional decisions about the details of the visualization will increase their impact
- Seaborn enables you to change the presentation of your figures by changing the style of elements like the background color, grids, and spines.

1. Built-in Themes:

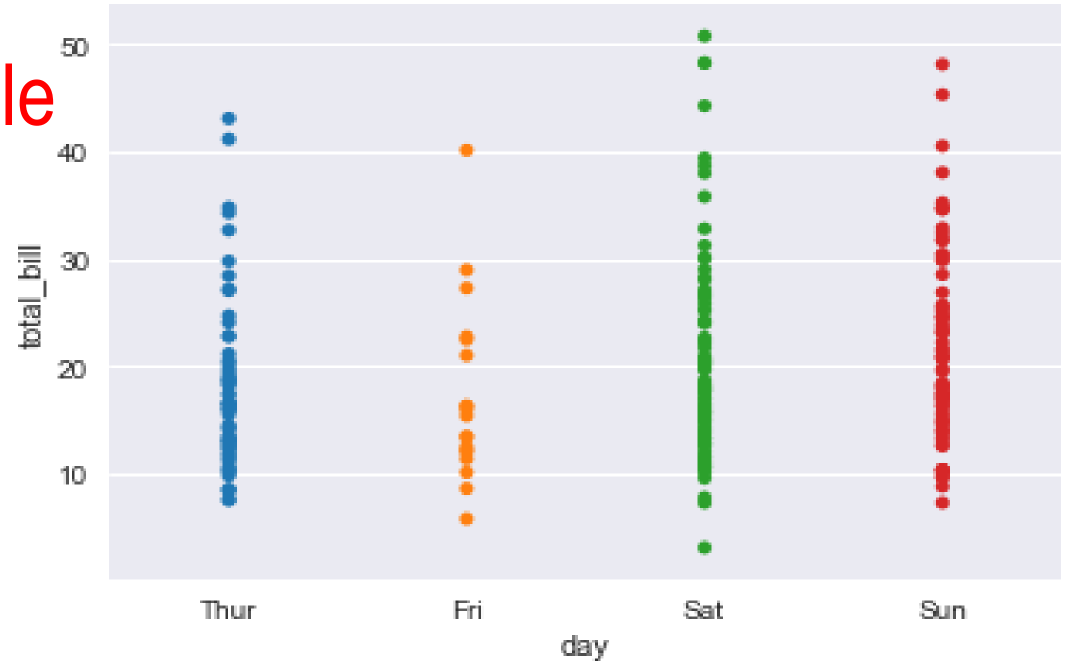
Seaborn has five built-in themes to style its plots: darkgrid, whitegrid, dark, white, and ticks.

Seaborn defaults to using the darkgrid theme for its plots

Seaborn Styling, Part 1: Figure Style and Scale

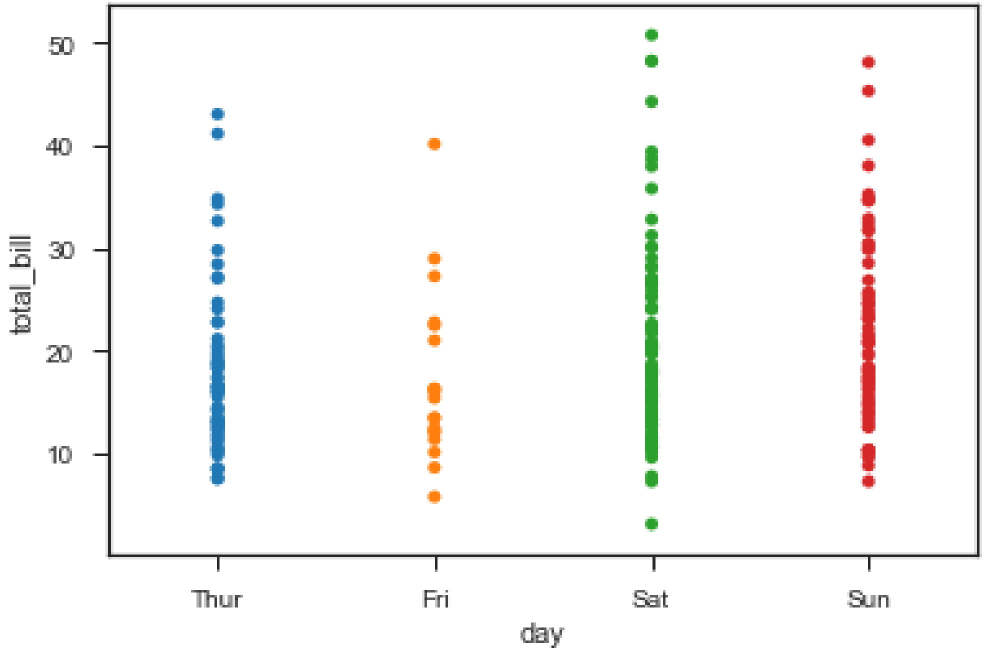
- To use any of the preset themes pass the name of it to sns.set_style().

```
sns.set_style("darkgrid")

sns.stripplot(x="day", y="total_bill", data=tips)
```



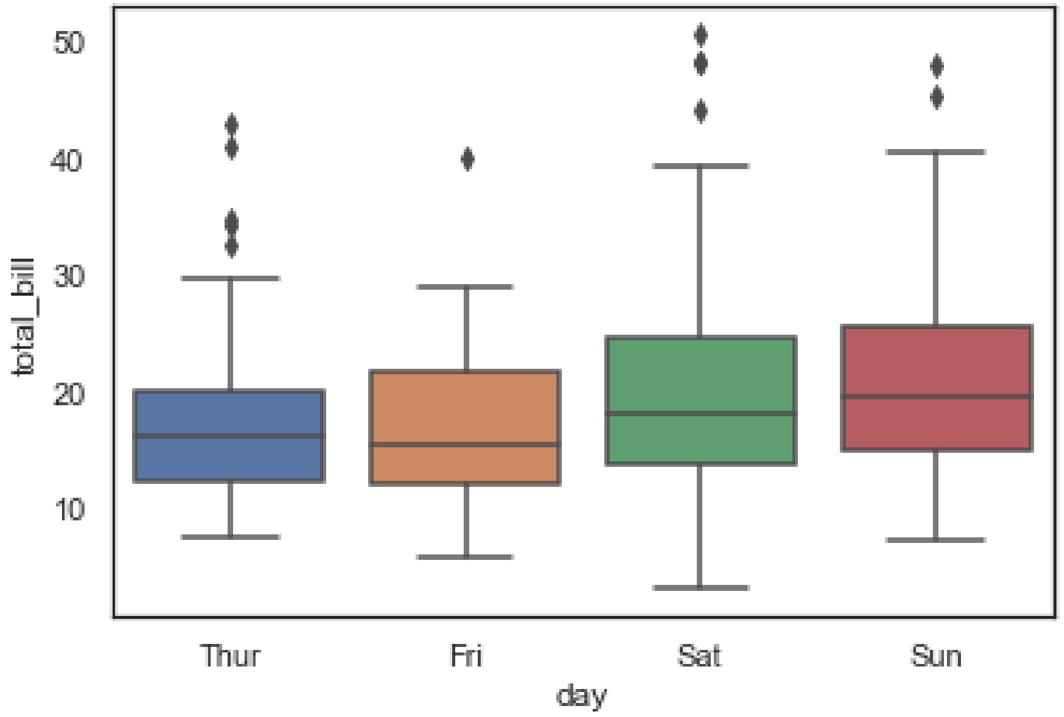
- The white and tick themes will allow the colors of your dataset to show more visibly and provides higher contrast so your plots are more legible:

```
sns.set_style("ticks")

sns.stripplot(x="day", y="total_bill", data=tips)
```



Styling:

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="white", color_codes=True)
a = sns.load_dataset("tips")
sns.boxplot(x="day", y="total_bill", data=a);
```



Theme is changed to white

Seaborn Styling, Part 1: Figure Style and Scale

2. Grids:

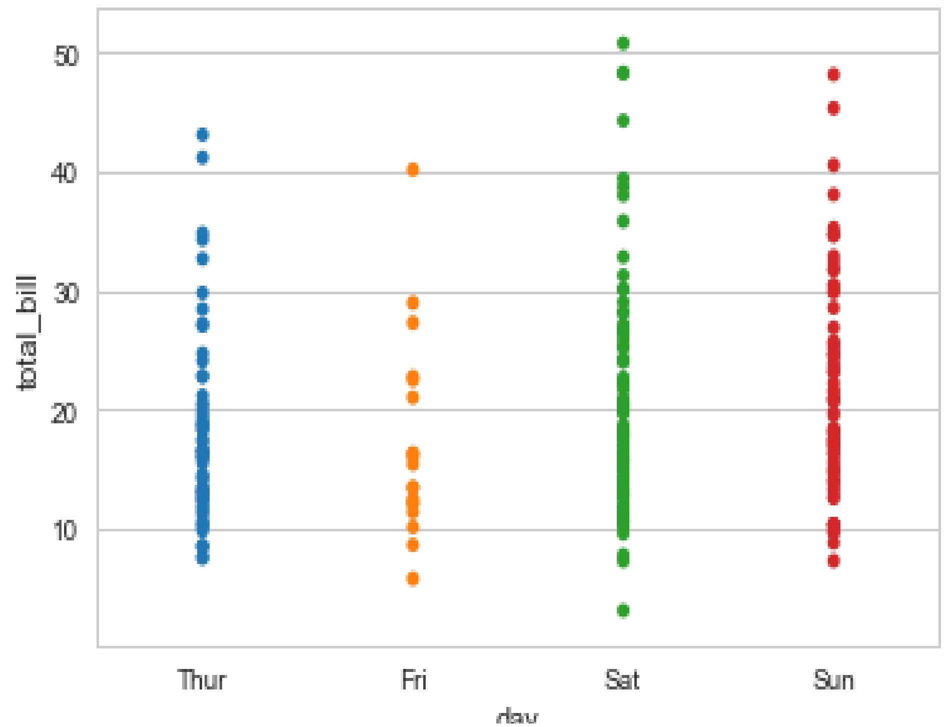
- To define the background color of your figure, you can also choose whether or not to include a grid.

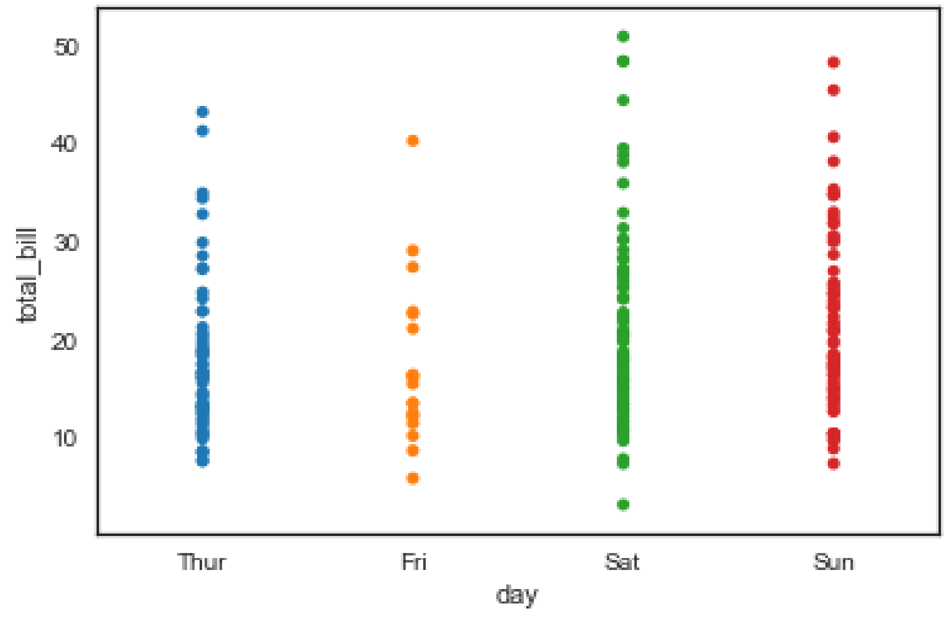
- A grid allows the audience to read your chart and get specific information about certain values. Research papers and reports are a good example of when you would want to include a grid.

```
sns.set_style("whitegrid")

sns.stripplot(x="day", y="total_bill", data=tips)
```

- There are also instances where it would make more sense to not use a grid.

```
sns.set_style("white")

sns.stripplot(x="day", y="total_bill", data=tips)
```





Seaborn Styling, Part 1: Figure Style and Scale

3. Despine

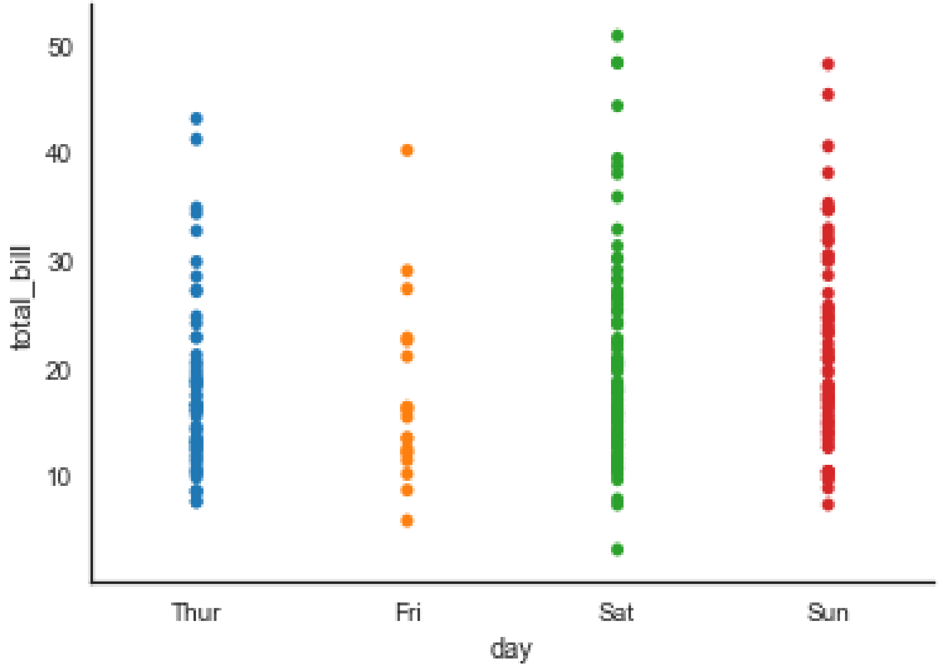
- Spines are the borders of the figure that contain the visualization.
  - By default, an image has four spines.

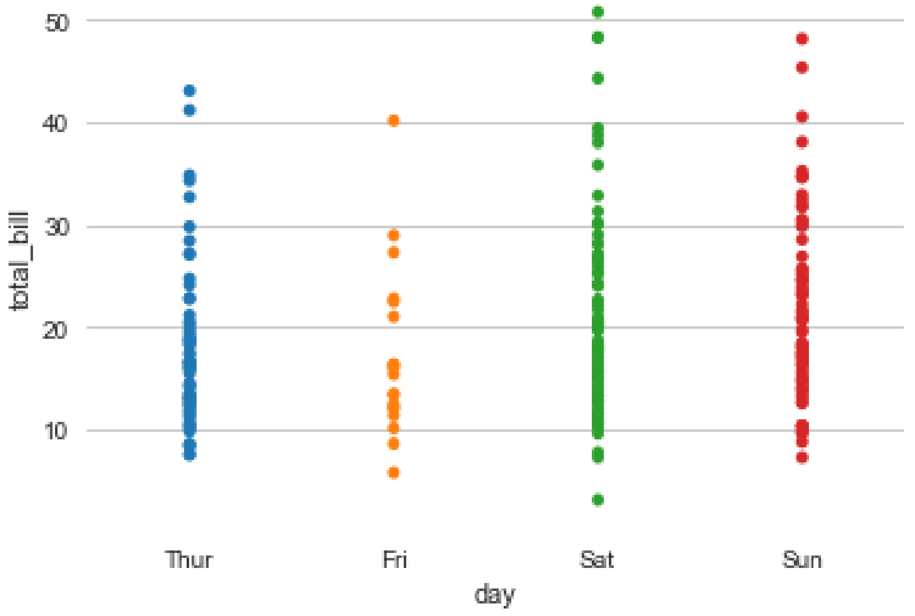
- You can automatically take away the top and right spines using the sns.despine()function.

```
sns.set_style("white")

sns.stripplot(x="day", y="total_bill", data=tips)

sns.despine()
```

- You can also specify how many spines you want to include by calling despine() and passing in the spines you want to get rid of, such as: left, bottom, top, right.

```
sns.set_style("whitegrid")

sns.stripplot(x="day", y="total_bill", data=tips)

sns.despine(left=True, bottom=True)
```



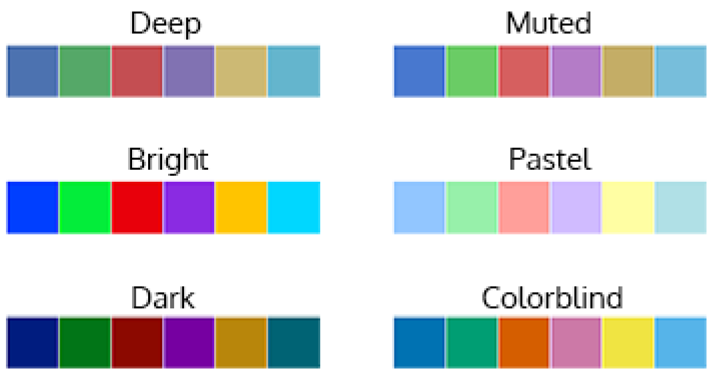


Seaborn Styling, Part 1: Figure Style and Scale

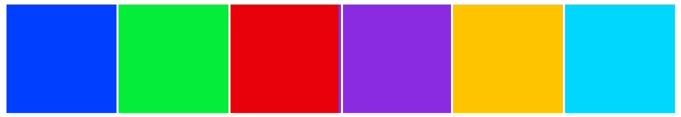
4. Working with Palettes

- You can build color palettes using the function sns.color_palette().
- This function can take any of the Seaborn built-in palettes (see below). You can also build your own palettes

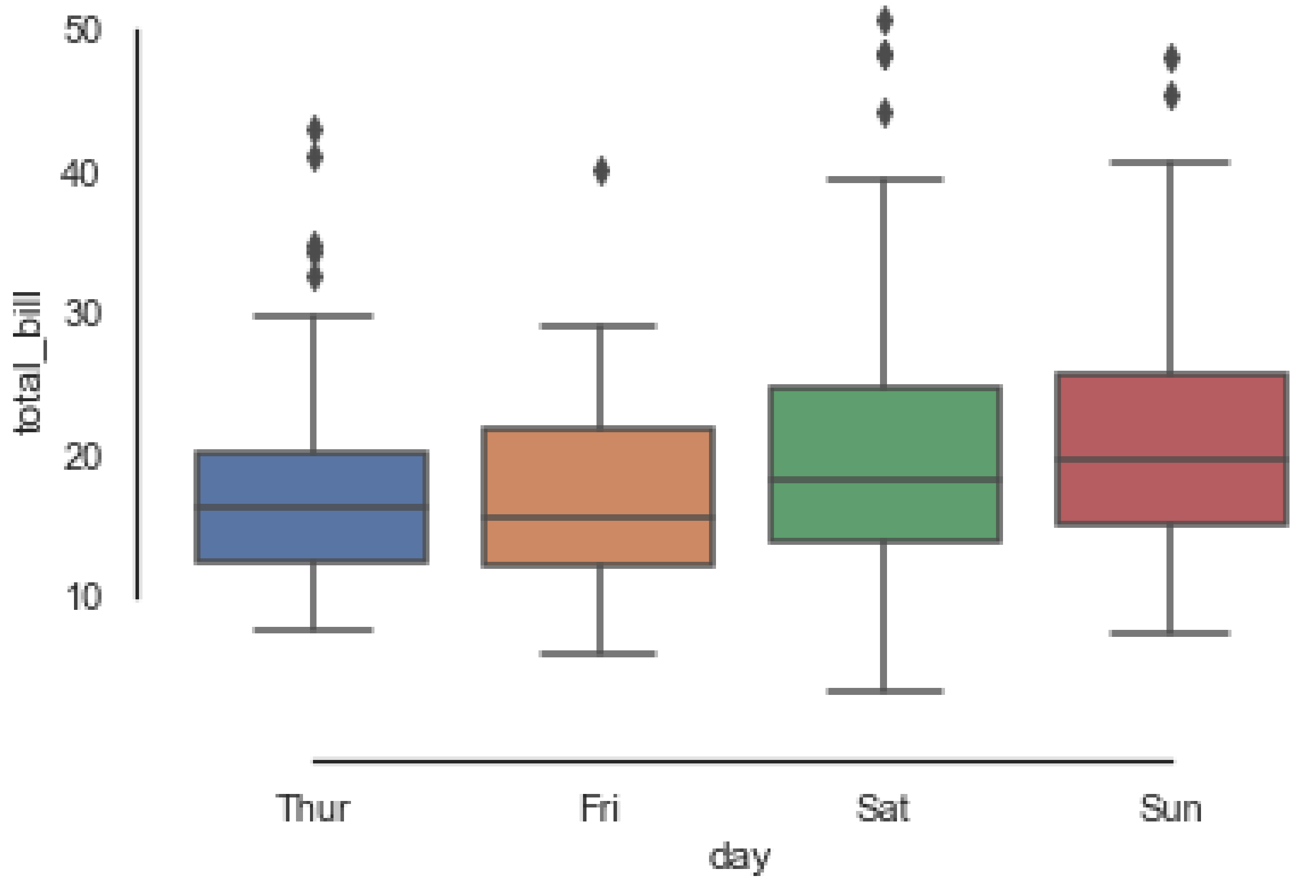
- use the function sns.palplot() to plot a palette as an array of colors



```
# Save a palette to a variable:
palette = sns.color_palette("bright")

# Use palplot and pass in the variable:
sns.palplot(palette)
```



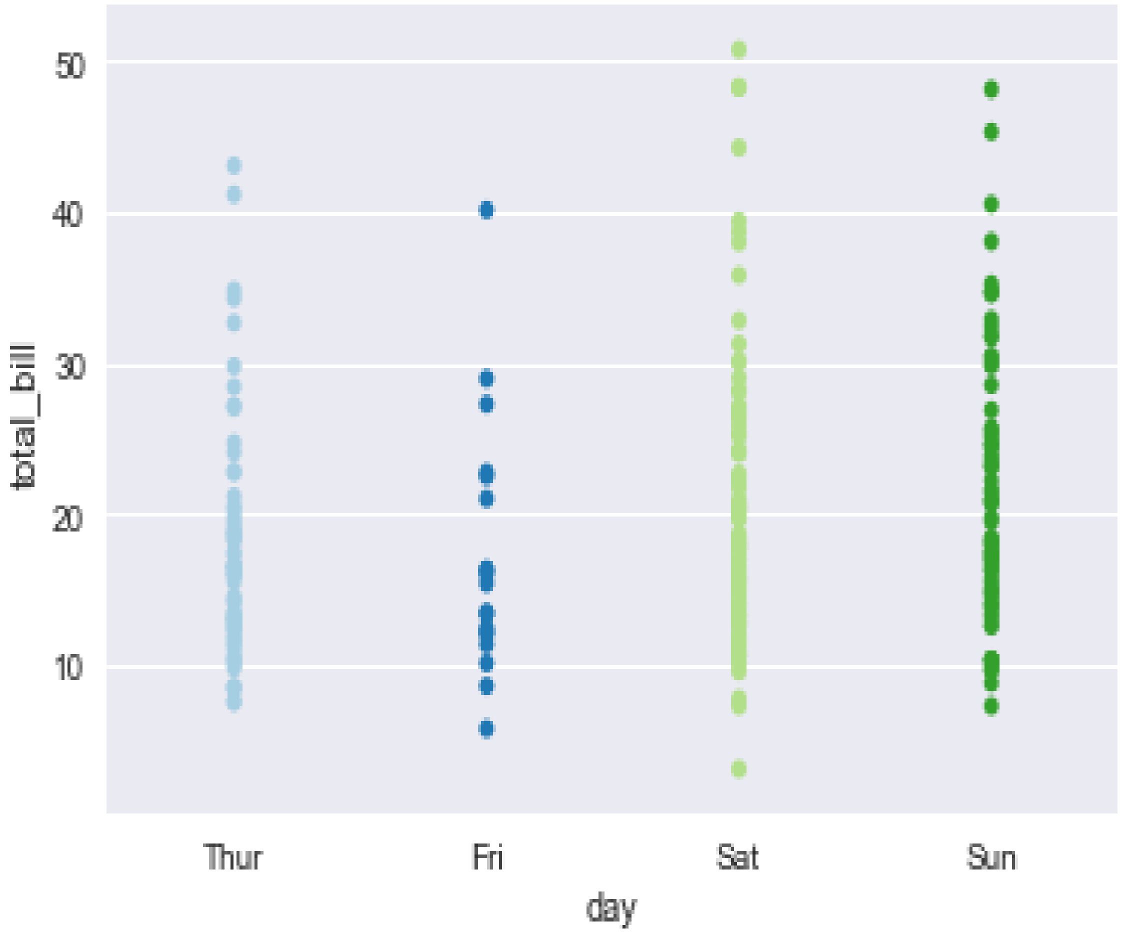
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="white", color_codes=True)
a = sns.load_dataset("tips")
sns.boxplot(x="day", y="total_bill", data=a);
sns.despine(offset=10, trim=True);
```



4. Working with Palettes

- To select and set a palette in Seaborn, use the command sns.set_palette() and pass in the name of the palette that you would like to use:

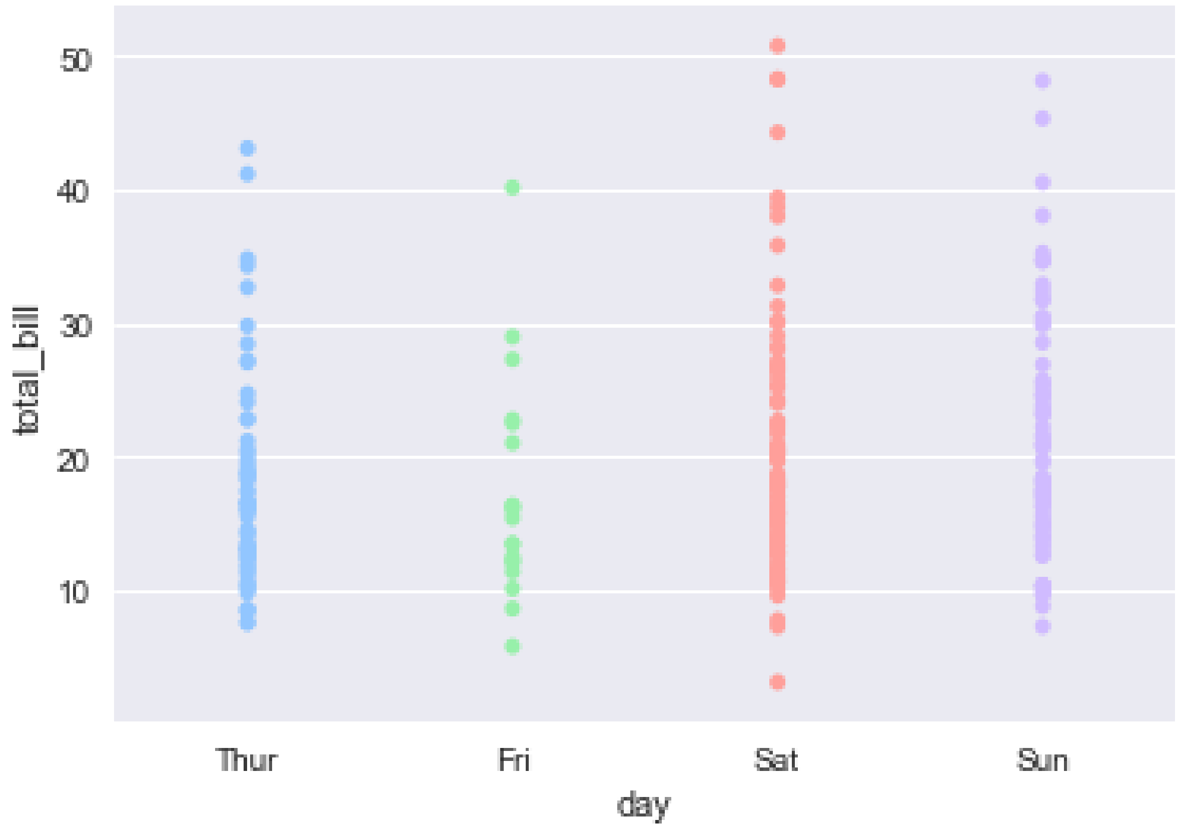
```
# Set the palette using the name of a palette:
sns.set_palette("Paired")

# Plot a chart:
sns.stripplot(x="day", y="total_bill", data=tips)
```



4. Working with Palettes

- To use one of these palettes, pass the name into sns.set_palette():

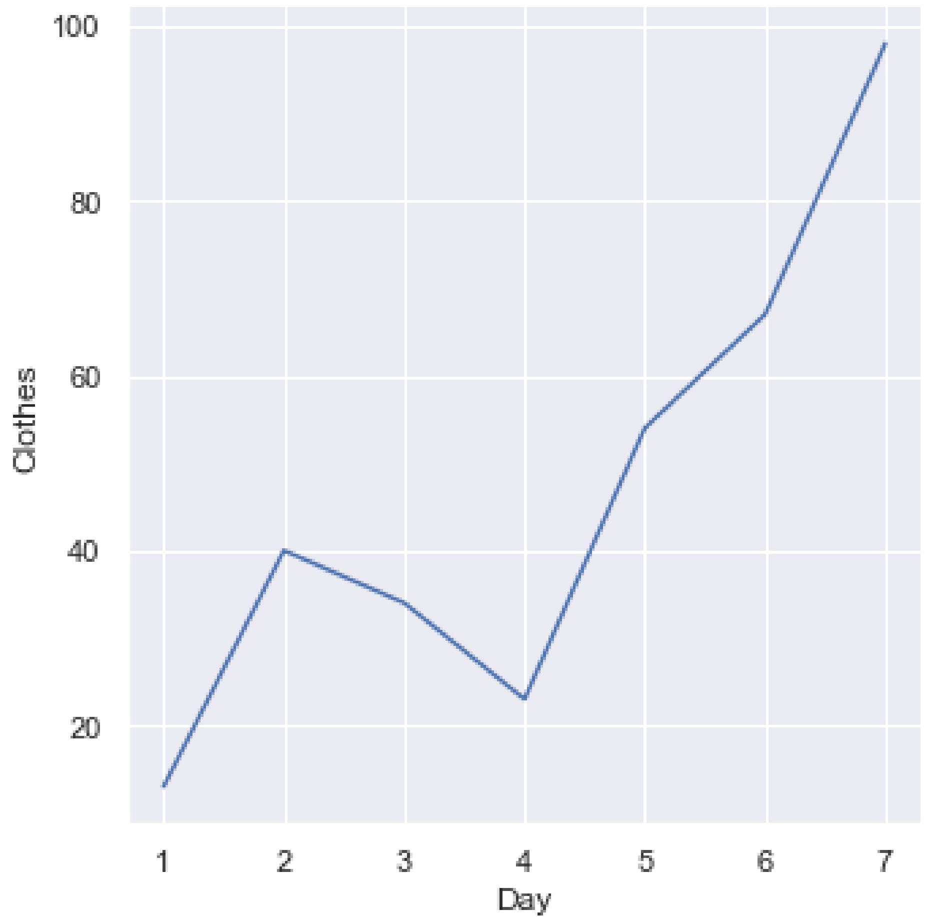
```
# Set the palette to the "pastel" default palette:
sns.set_palette("pastel")

# plot using the "pastel" palette
sns.stripplot(x="day", y="total_bill", data=tips)
```



lineplot():

This function will allow you to draw a continuous line for your data. You can use this function by changing the ‘kind’ parameter as follows:

```
a=pd.DataFrame({'Day':[1,2,3,4,5,6,7],'Grocery':[30,80,45,23,51,46,76],'Clothes':[13,40,34,23,54,67,98],'Utensils':[12,32,27,56,87,54,34]},index=[1,2,3,4,5,6,7])
sns.relplot(x="Day", y="Clothes", kind="line", data=a)
```



Plotting with Categorical Data:

- This approach is used when our main variable is further divided into discrete groups (categorical).
- This can be achieved using the catplot() function.

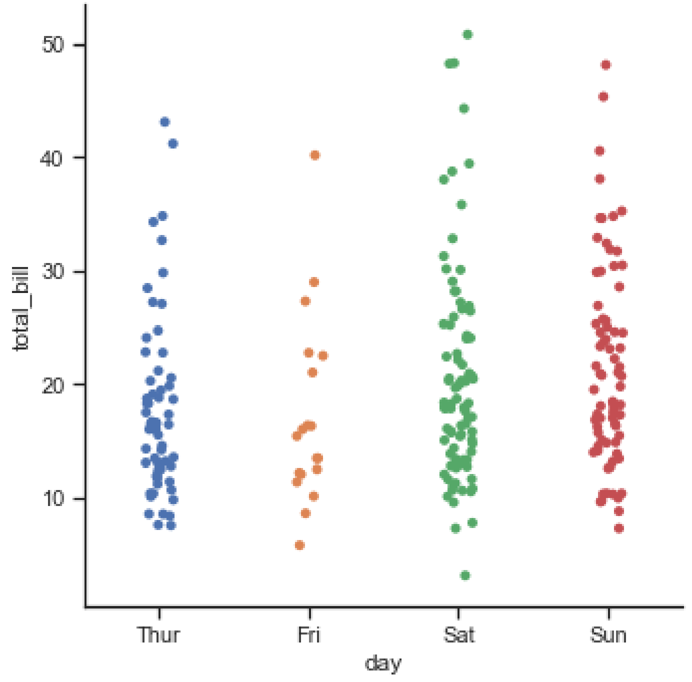
catplot():

It can be characterized by three families of axes level functions namely:

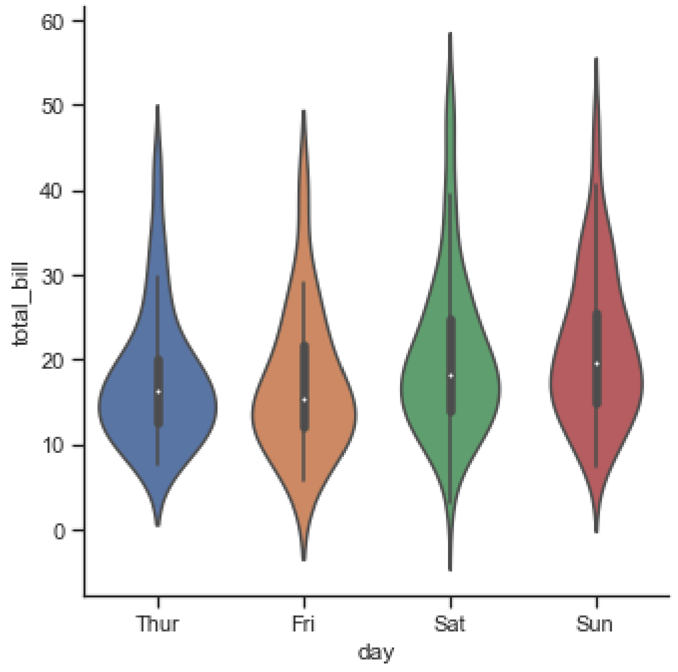
- Scatterplots – These include stripplot(), swarmplot()
- Distribution Plots – which are boxplot(), violinplot(), boxenplot()
- Estimateplots – namely pointplot(), barplot(), countplot()

Plotting with Categorical Data:

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="ticks", color_codes=True)
a = sns.load_dataset("tips")
sns.catplot(x="day", y="total_bill", data=a);
```



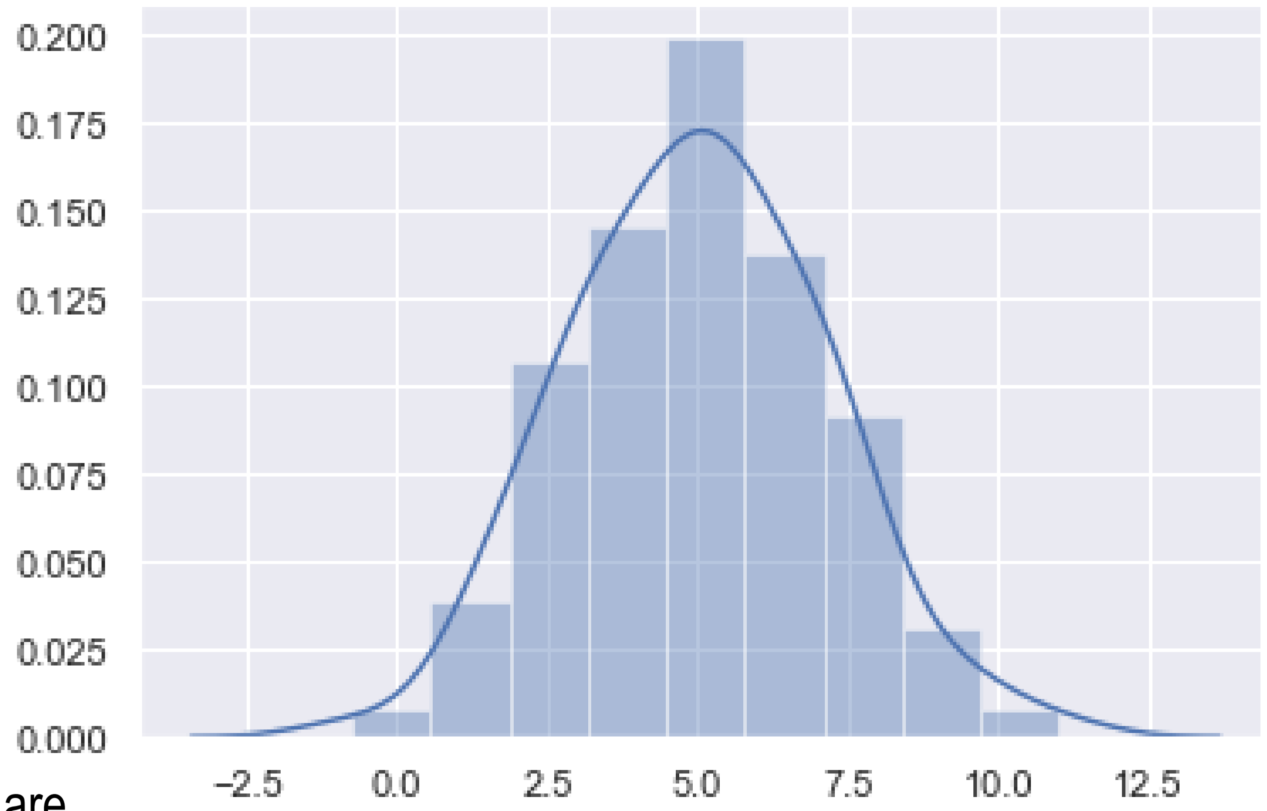
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="ticks", color_codes=True)
a = sns.load_dataset("tips")
sns.catplot(x="day", y="total_bill", kind="violin", data=a);
```



Visualizing the distribution of a dataset:

- Plotting Univariate distributions: To plot them, you can make use of distplot() function as follows:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
sns.set(color_codes=True)

a = np.random.normal(loc=5,size=100,scale=2)
sns.distplot(a);
```

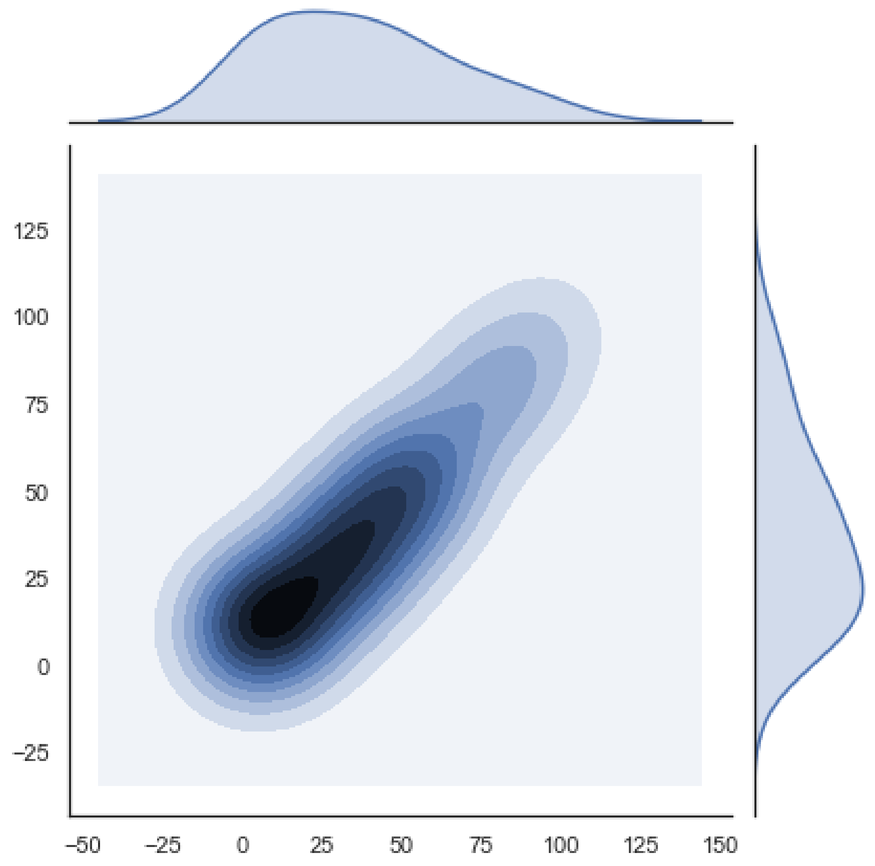


- We have plotted a graph for the variable a whose values are generated by the normal() function using distplot

Plotting bivariate distributions:

- When you have two random independent variables resulting in some probable event, then you can use a bivariate distribution
- The best function to plot these type of graphs is jointplot().

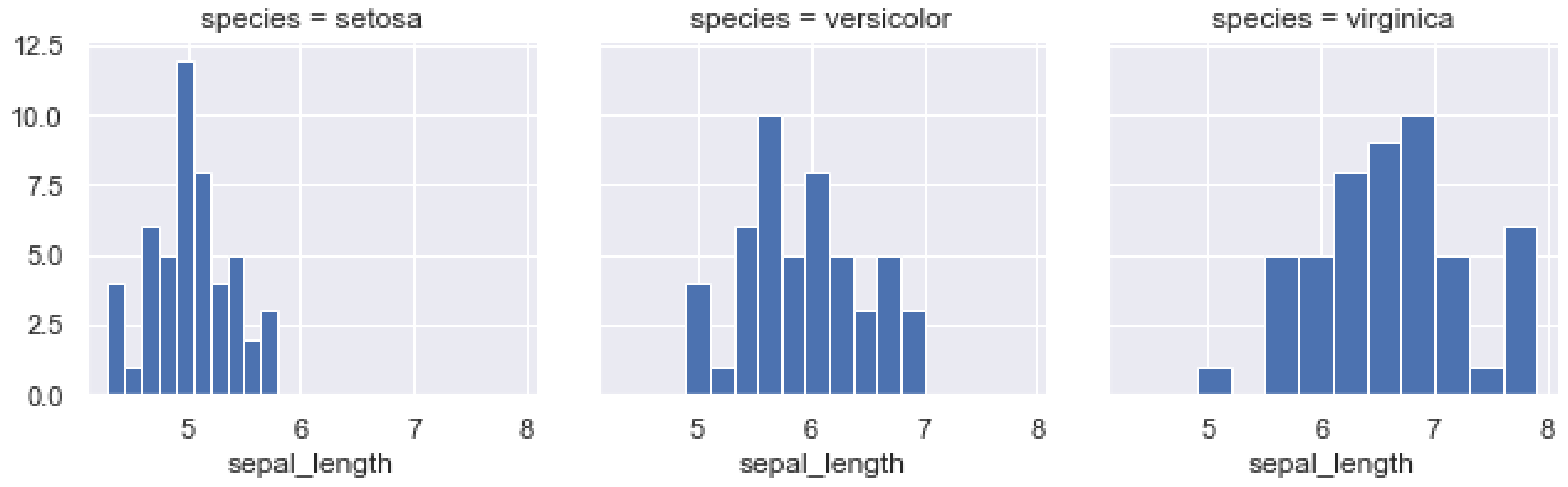
```
x=pd.DataFrame({'Day':[1,2,3,4,5,6,7],'Grocery':[30,80,45,23,51,46,76],'Clothes'
:[13,40,34,23,54,67,98],'Utensils':[12,32,27,56,87,54,34]},index=[1,2,3,4,5,6,7])
y=pd.DataFrame({'Day':[8,9,10,11,12,13,14],'Grocery':[30,80,45,23,51,46,76],'Cl
othes':[13,40,34,23,54,67,98],'Utensils':[12,32,27,56,87,54,34]},index=[8,9,10,1
1,12,13,14])
mean, cov = [0, 1], [(1, .5), (.5, 1)]
data = np.random.multivariate_normal(mean, cov, 200)
with sns.axes_style("white"):
   sns.jointplot(x=x, y=y, kind="kde", color="b");
```



Multi-Plot Grids:

- Python Seaborn allows you to plot multiple grids side-by-side.
- These are basically plots or graphs that are plotted using the same scale and axes to aid comparison between them.
- Use facetgrid() function to plot these graphs.

```
sns.set(style="darkgrid")

a = sns.load_dataset("iris")

b = sns.FacetGrid(a, col="species")

b.map(plt.hist, "sepal_length");
```

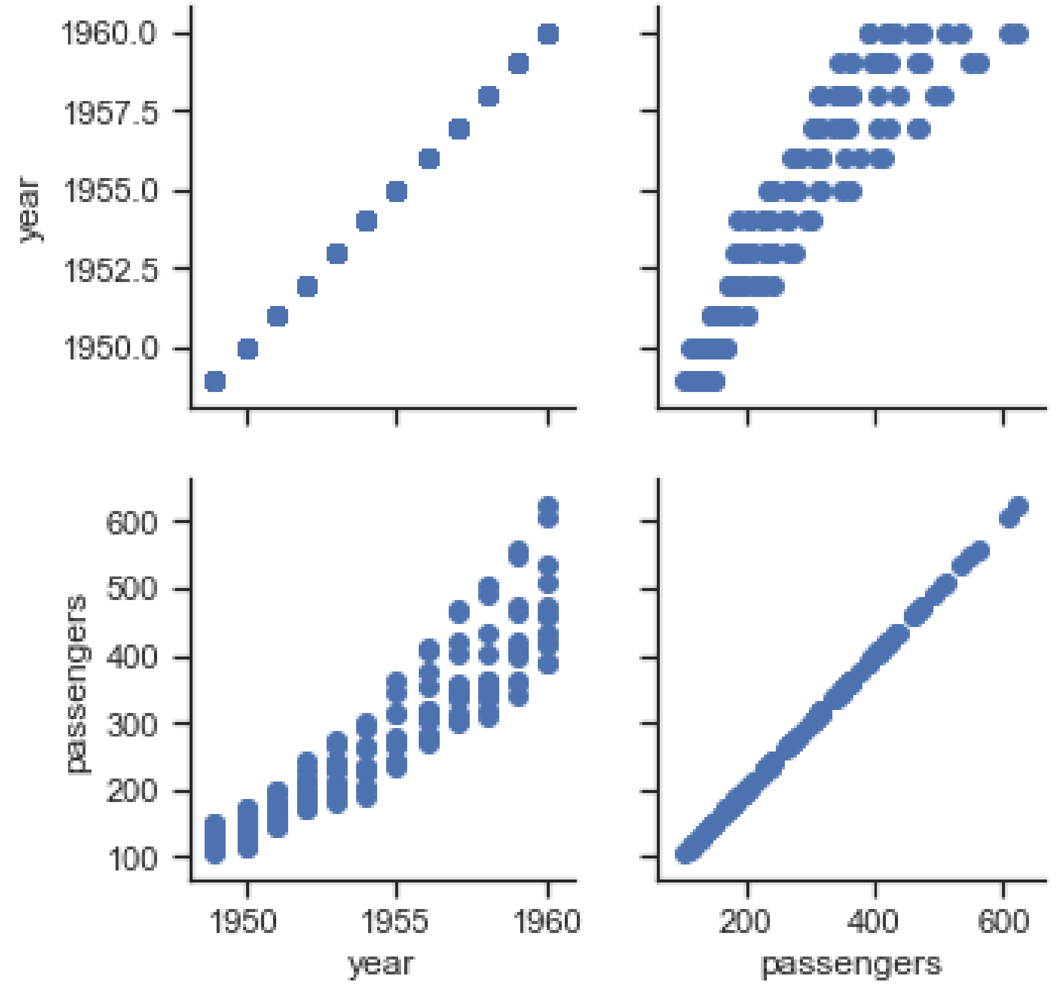


Multi-Plot Grids:

- You can also plot using PairGrid function when you have a pair of variables to compare.

```
sns.set(style="ticks")

a = sns.load_dataset("flights")

b = sns.PairGrid(a)

b.map(plt.scatter);
```

- The output clearly compares between the year and the number of passengers in different ways.


year
passengers
year
passengers