#20

Using Seaborn for Visualization

MIS561 Data Visualization
Original Author: Lusi Yang

Introduction to Seaborn

Unlike **Matplotlib**, **Seaborn** is not a standalone Python library. It is built on top of Matplotlib and provides a higher-level abstraction to make visually appealing statistical visualizations.

A neat feature of Seaborn is the ability to integrate with DataFrames from the pandas library.

The most prominent features of Seaborn:

- Beautiful out of the box plots with different themes
- Built-in color palettes that can be used to reveal patterns in the dataset
- A high-level abstraction that still allows for complex visualizations

Advantages of Seaborn

- Seaborn is built to operate on DataFrames and full dataset arrays, which makes this process of exploring dataset simpler.

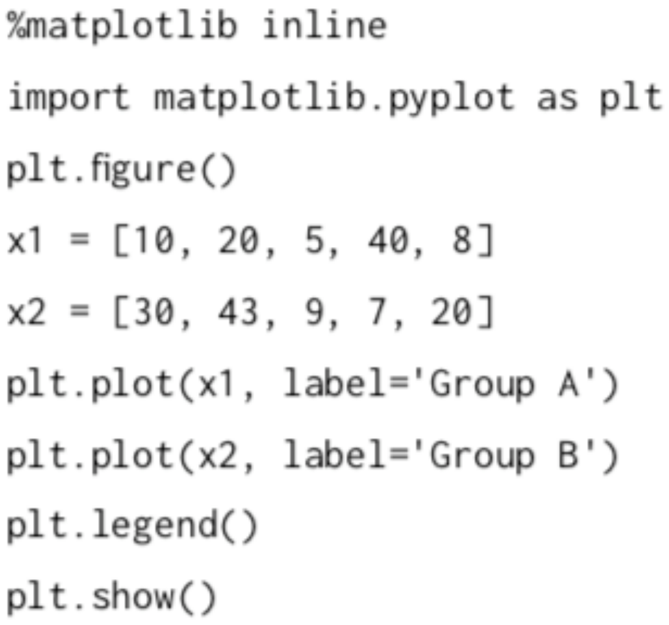
- It internally performs the necessary semantic mappings and statistical aggregation to produce informative plots.

- The default parameters in Seaborn provide better visualizations without additional customization.

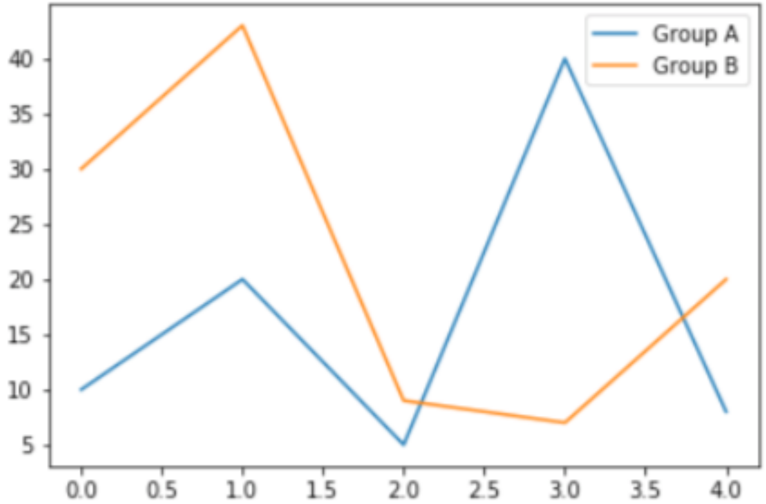
Controlling Figure Aesthetics

- contrast between matplotlib and seaborn

Matplotlib is highly customizable. But it is difficult to know what settings to tweak to achieve a visually appealing plot.



```
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure()
x1 = [10, 20, 5, 40, 8]
x2 = [30, 43, 9, 7, 20]
plt.plot(x1, label='Group A')
plt.plot(x2, label='Group B')
plt.legend()
plt.show()
```



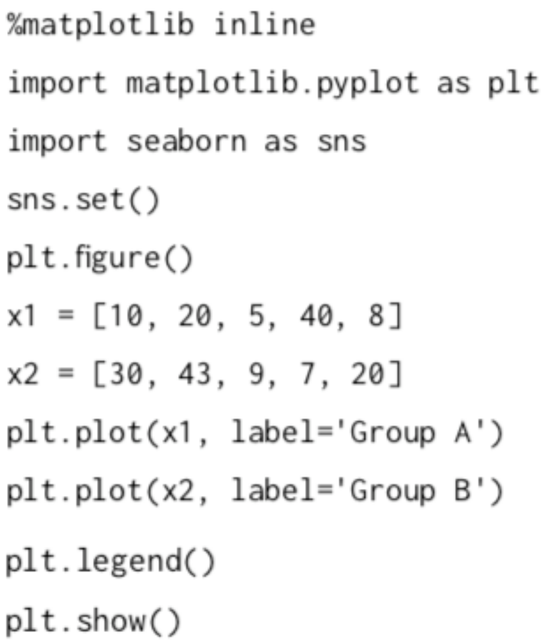
Matplotlib line plot

Controlling Figure Aesthetics

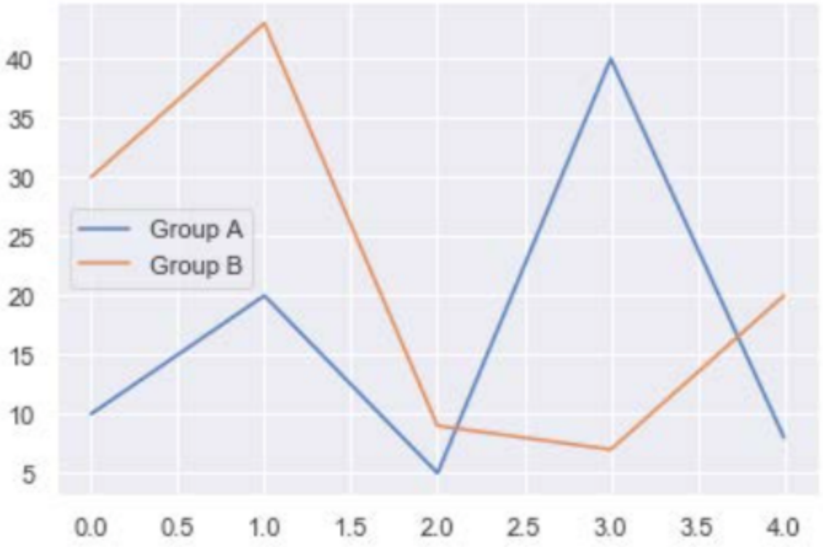
- contrast between matplotlib and seaborn

Seaborn provides several customized themes and a high-level interface for controlling the appearance of Matplotlib figures.

To switch to the Seaborn defaults, simply call the **set()** function.



```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
plt.figure()
x1 = [10, 20, 5, 40, 8]
x2 = [30, 43, 9, 7, 20]
plt.plot(x1, label='Group A')
plt.plot(x2, label='Group B')
plt.legend()
plt.show()
```



Seaborn line plot

Seaborn Figure Styles

To control the style, Seaborn provides two methods: **set_style(style, [rc])** and **axes_ style(style, [rc])**.

seaborn.set_style(style, [rc]) sets the aesthetic style of the plots.

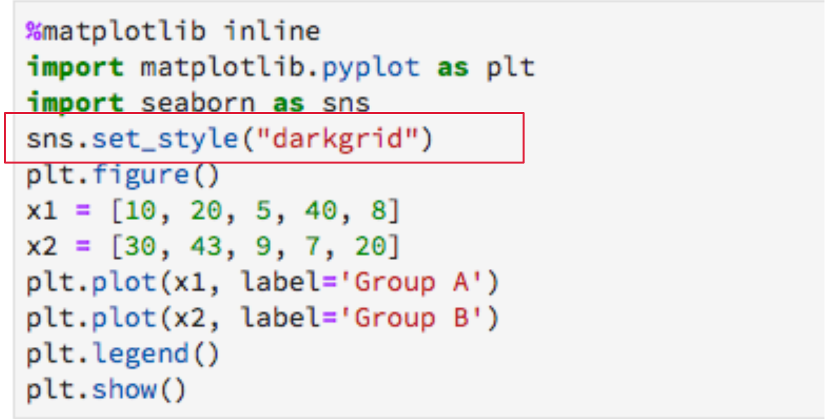
seaborn.axes_style(style, [rc]) returns a parameter dictionary for the aesthetic style of the plots.

Parameters:

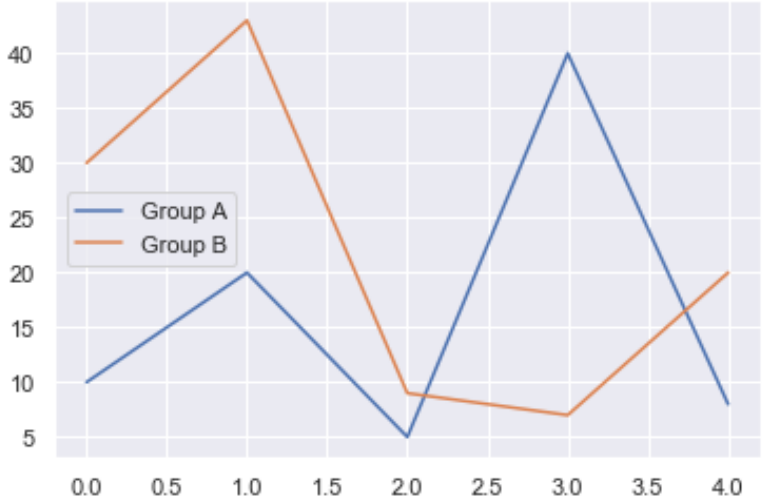
- **style** – A dictionary of parameters or the name of one of the following preconfigured sets: **darkgrid**, **whitegrid**, **dark**, **white**, or **ticks**
- **rc** (optional) – Parameter mappings to override the values in the preset Seaborn style dictionaries

Examples of Changing Figure Style

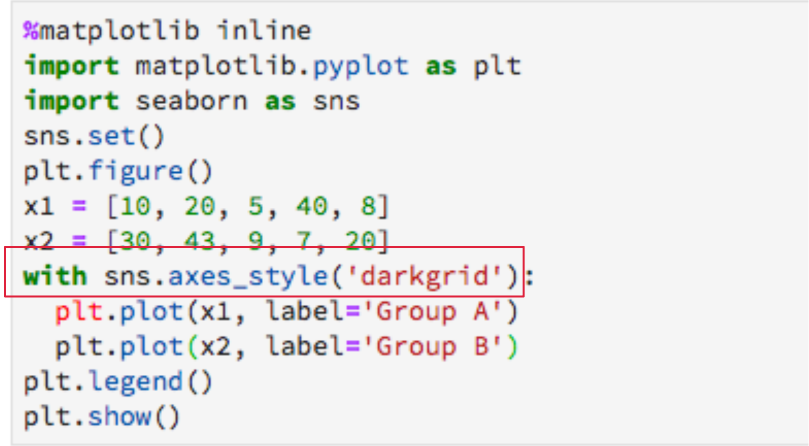
Approach 1



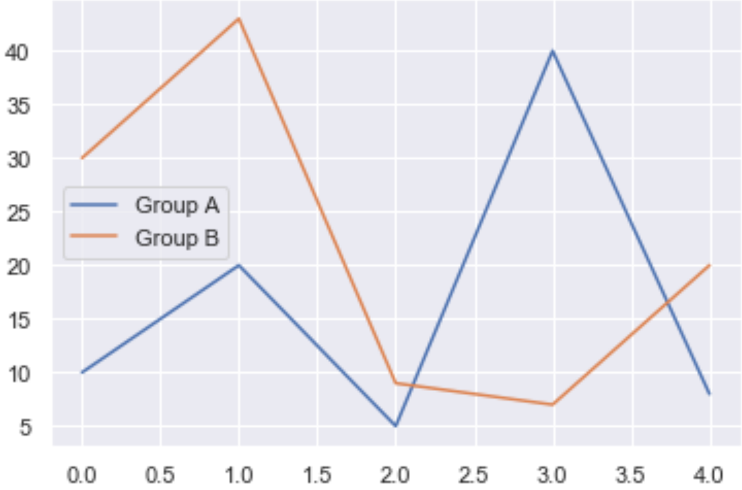
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")
plt.figure()
x1 = [10, 20, 5, 40, 8]
x2 = [30, 43, 9, 7, 20]
plt.plot(x1, label='Group A')
plt.plot(x2, label='Group B')
plt.legend()
plt.show()
```



Approach 2



```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
plt.figure()
x1 = [10, 20, 5, 40, 8]
x2 = [30, 43, 9, 7, 20]
with sns.axes_style('darkgrid'):
  plt.plot(x1, label='Group A')
  plt.plot(x2, label='Group B')
plt.legend()
plt.show()
```

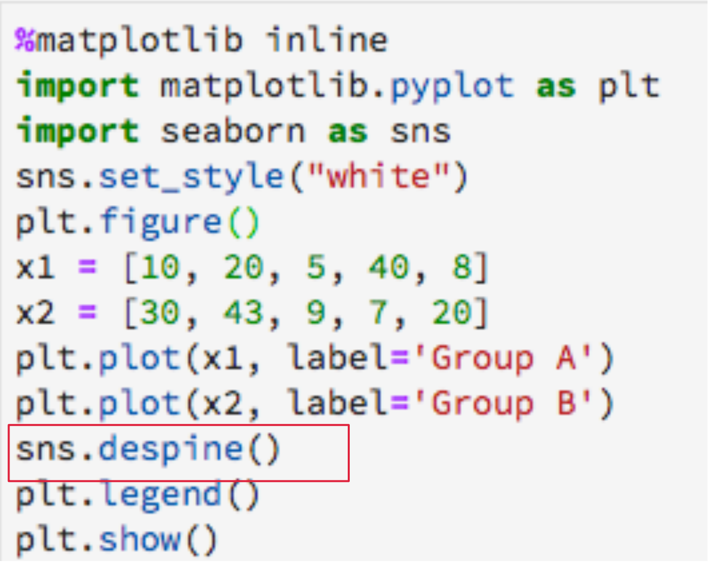


Plot with “darkgrid” style

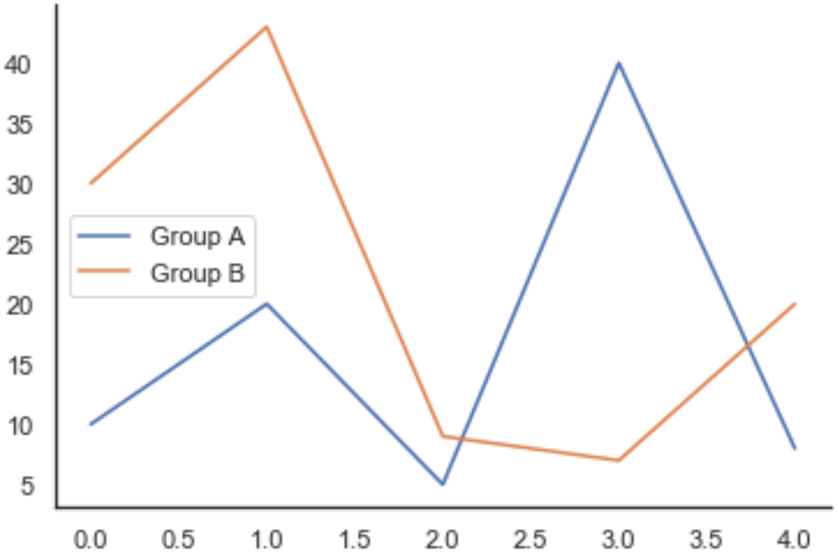
Removing Axes Spines

Sometimes, it might be desirable to remove the top and right axes spines.

seaborn.despine(fig=None, ax=None, top=True, right=True, left=False, bottom=False, offset=None, trim=False) removes the top and right spines from the plot.



```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("white")
plt.figure()
x1 = [10, 20, 5, 40, 8]
x2 = [30, 43, 9, 7, 20]
plt.plot(x1, label='Group A')
plt.plot(x2, label='Group B')
sns.despine()
plt.legend()
plt.show()
```



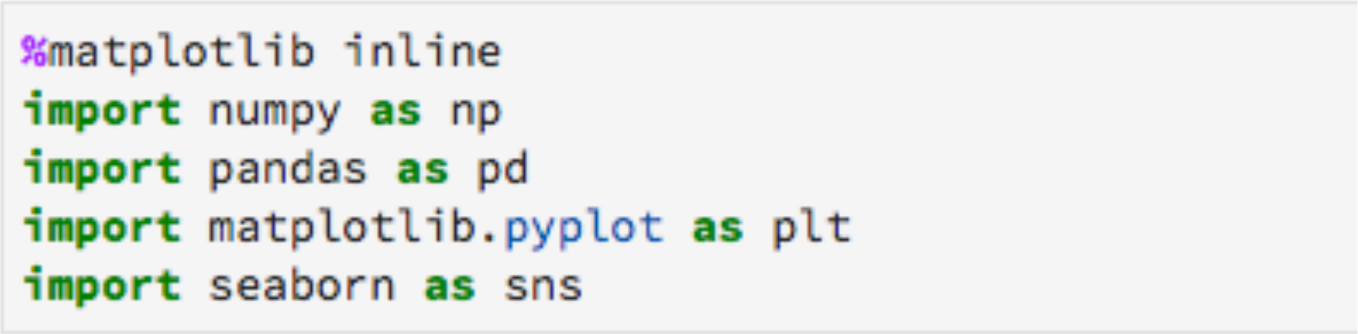
Despined Seaborn line plot

Activity 1

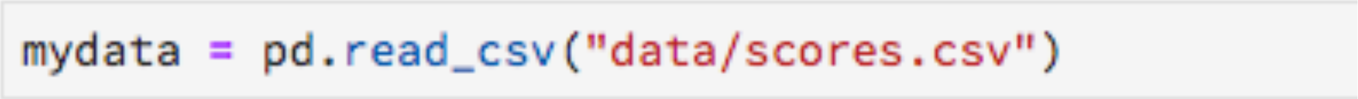
Revisit – Comparing IQ Scores for Different Test Groups by Using a Box Plot

Objective

In this activity, we will compare IQ scores among different test groups using a box plot of the Seaborn library

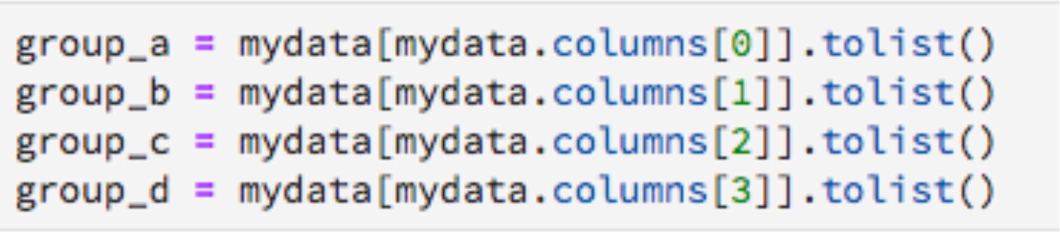


```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```



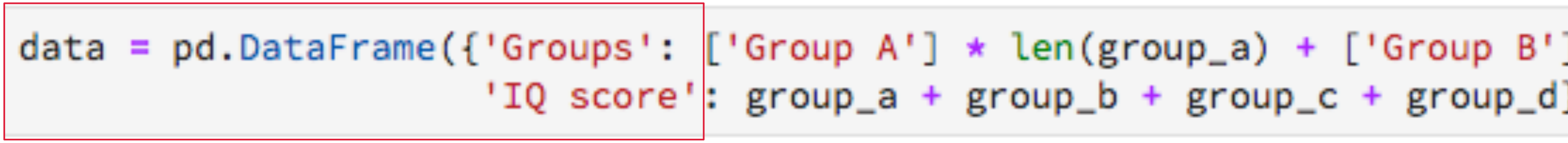
```
mydata = pd.read_csv("data/scores.csv")
```

Read data located in the data folder



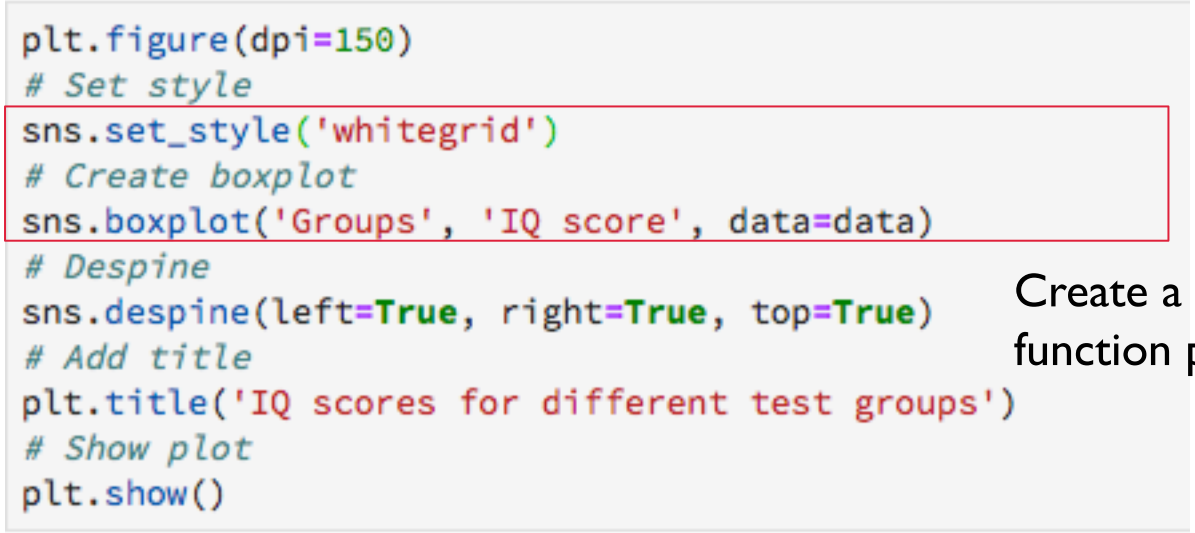
```
group_a = mydata[mydata.columns[0]].tolist()
group_b = mydata[mydata.columns[1]].tolist()
group_c = mydata[mydata.columns[2]].tolist()
group_d = mydata[mydata.columns[3]].tolist()
```

Access the data of each test group in the column. Convert them into a list using the **tolist()** method.



```
data = pd.DataFrame({'Groups': ['Group A'] * len(group_a) + ['Group B']
                     'IQ score': group_a + group_b + group_c + group_d]
```

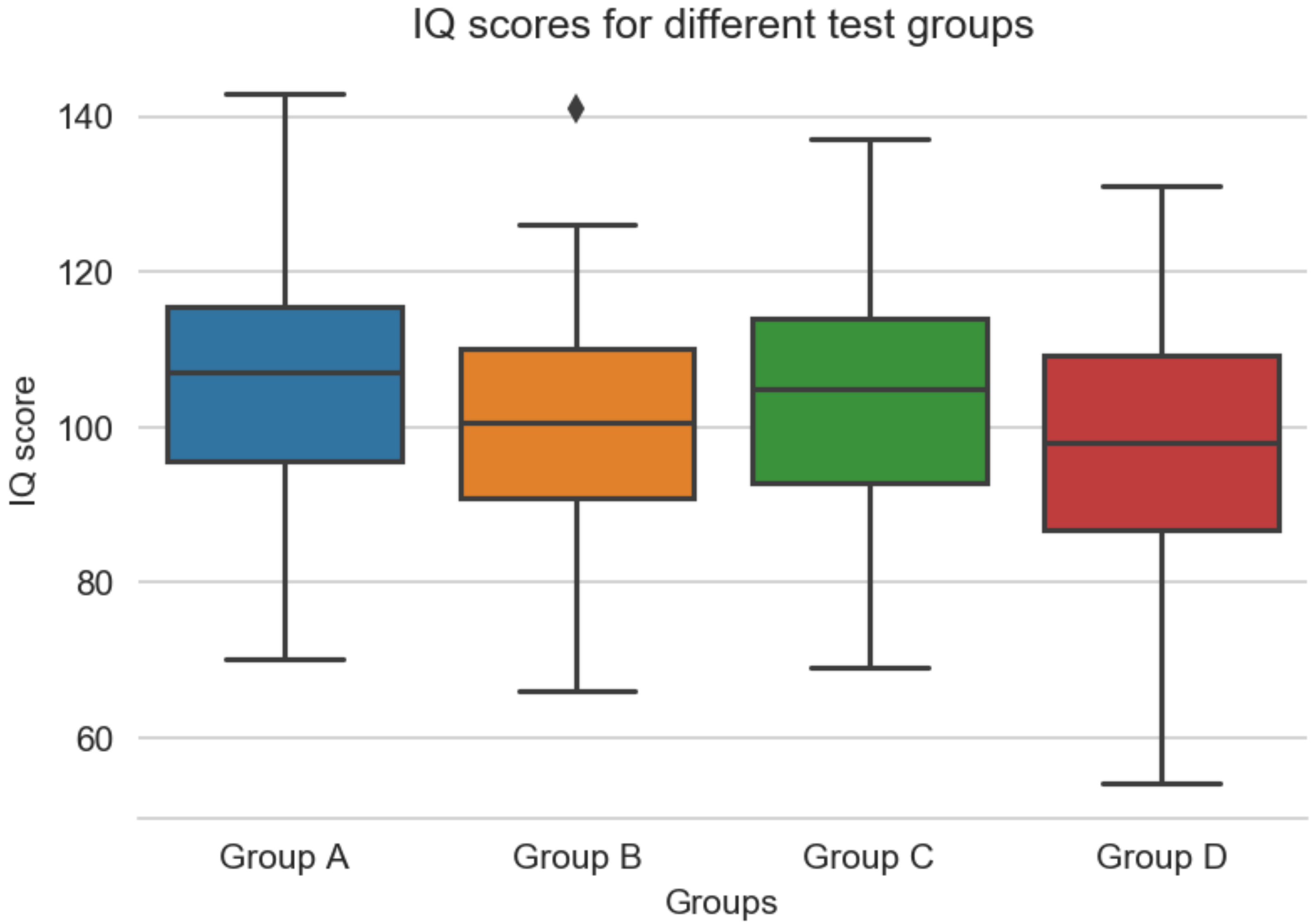
Construct a dataframe using **pd.DataFrame()** function



```
plt.figure(dpi=150)
# Set style
sns.set_style('whitegrid')
# Create boxplot
sns.boxplot('Groups', 'IQ score', data=data)
# Despine
sns.despine(left=True, right=True, top=True)
# Add title
plt.title('IQ scores for different test groups')
# Show plot
plt.show()
```

Create a boxplot using the **boxplot()** function provided by Seaborn.

Expected view



Color Palettes

Seaborn makes it easy to select and use **color palettes** that are suited to your task.

The **color_palette()** function provides an interface for many of the possible ways to generate colors.

seaborn.color_palette([palette], [n_colors], [desat]) returns a list of colors, thus defining a color palette.

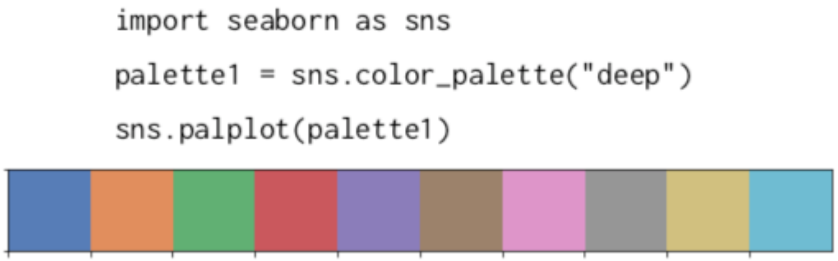
Parameters:

- **palette** (optional) – name of palette or None to return the current palette.
- **n_colors** (optional) – Number of colors in the palette. If the specified number of colors is larger than the number of colors in the palette, the colors will be cycled.
- **desat** (optional) – Proportion to desaturate each color by.

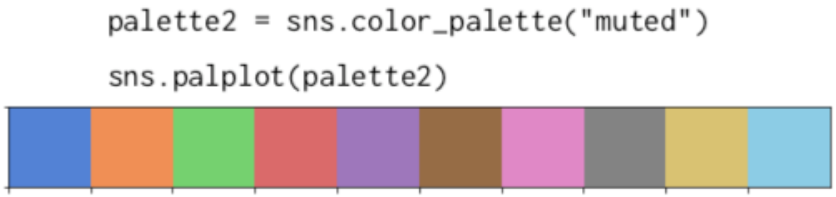
Categorical Color Palettes

Categorical palettes are best for distinguishing discrete data that does not have an inherent ordering.

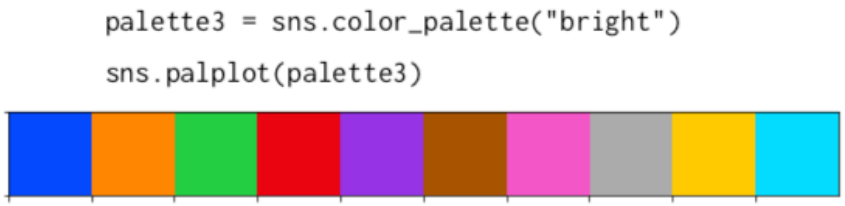
There are six default themes in Seaborn: **deep**, **muted**, **bright**, **pastel**, **dark**, and **colorblind**.



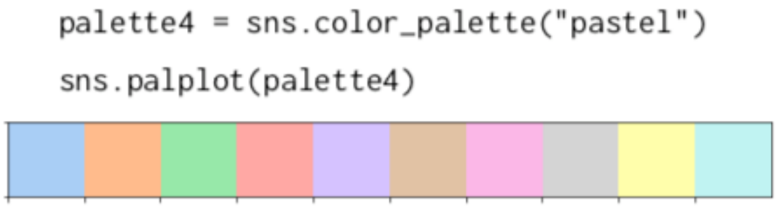
```
import seaborn as sns
palette1 = sns.color_palette("deep")
sns.palplot(palette1)
```



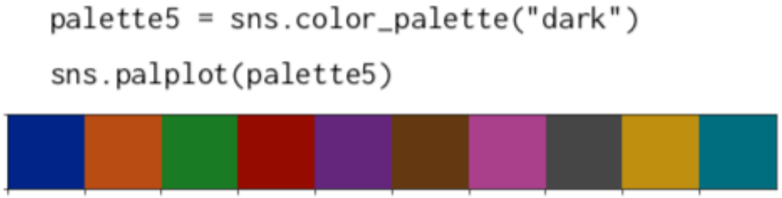
```
palette2 = sns.color_palette("muted")
sns.palplot(palette2)
```



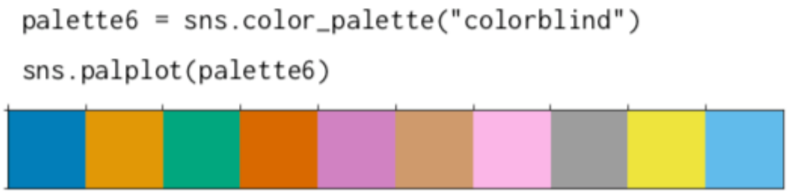
```
palette3 = sns.color_palette("bright")
sns.palplot(palette3)
```



```
palette4 = sns.color_palette("pastel")
sns.palplot(palette4)
```



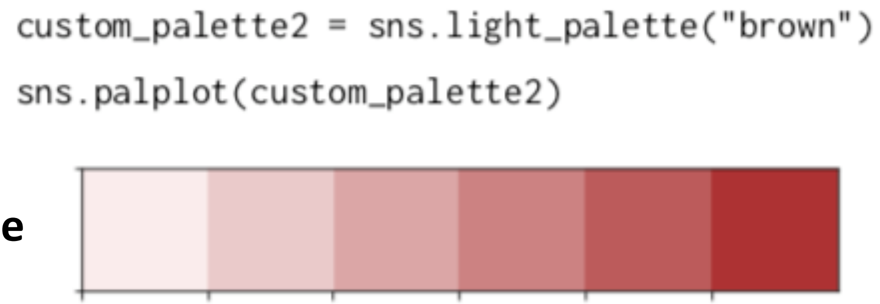
```
palette5 = sns.color_palette("dark")
sns.palplot(palette5)
```



```
palette6 = sns.color_palette("colorblind")
sns.palplot(palette6)
```

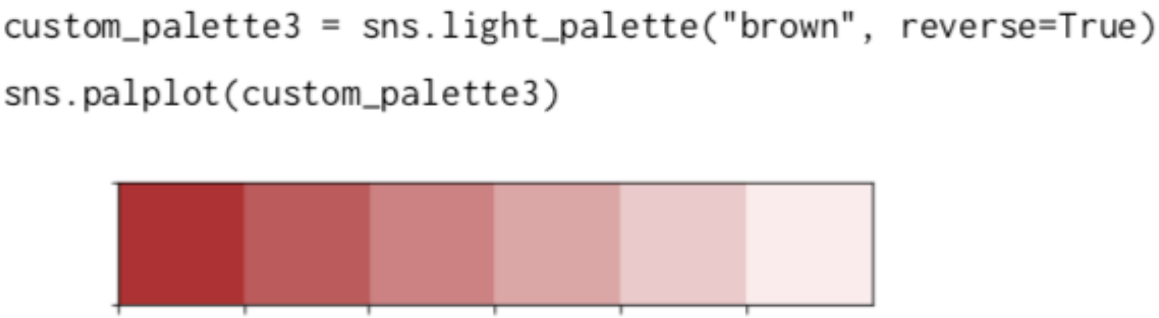
Sequential Color Palettes

Sequential color palettes are appropriate when the data ranges from relatively low or uninteresting values to relatively high or interesting values.



```
custom_palette2 = sns.light_palette("brown")
sns.palplot(custom_palette2)
```

Custom brown color palette

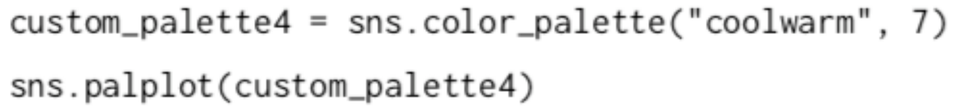


```
custom_palette3 = sns.light_palette("brown", reverse=True)
sns.palplot(custom_palette3)
```

Custom reversed brown color palette

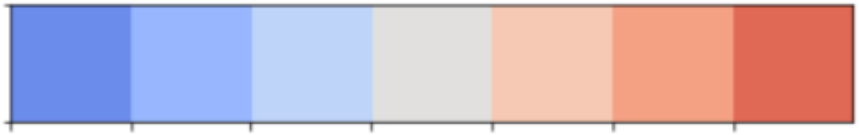
Diverging Color Palettes

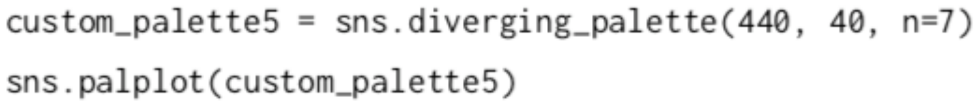
Diverging color palettes are used for data that consists of a well-defined midpoint. Emphasis is being laid on both high and low values.



```
custom_palette4 = sns.color_palette("coolwarm", 7)
sns.palplot(custom_palette4)
```

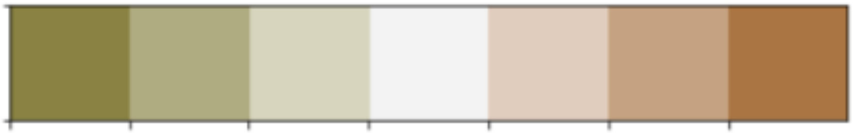
Coolwarm color palette





```
custom_palette5 = sns.diverging_palette(440, 40, n=7)
sns.palplot(custom_palette5)
```

Custom diverging color palette



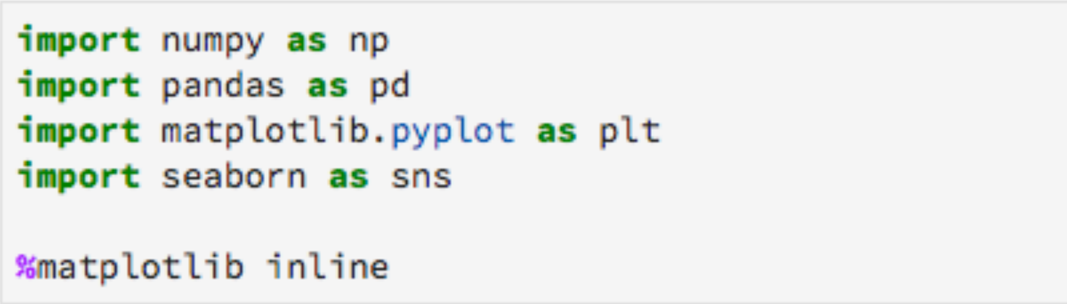
We can use the **diverging_palette()** function to create custom diverging palettes. We can pass two **hues** in degrees as parameters, along with the total number of palettes.

Activity 2

Using Heatmaps to Find Patterns in Flight Passengers' Data

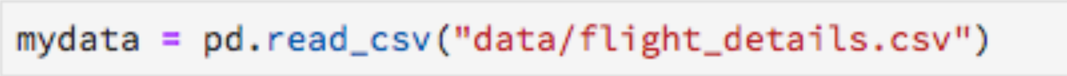
Objective

In this activity, we will use a heatmap to find the patterns in the flight passengers' data.

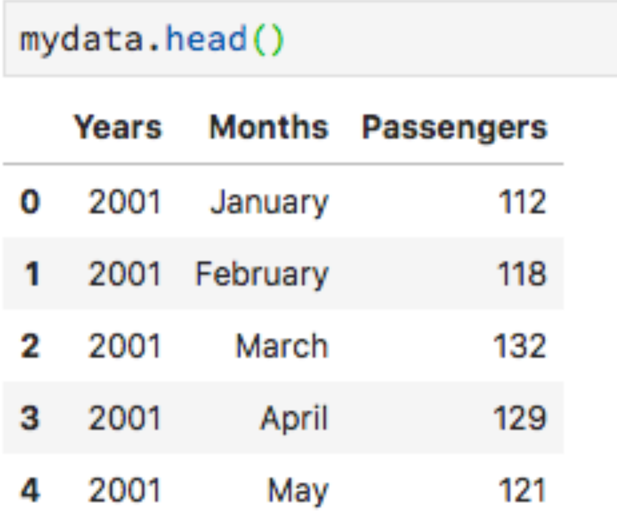


```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```



```
mydata = pd.read_csv("data/flight_details.csv")
```

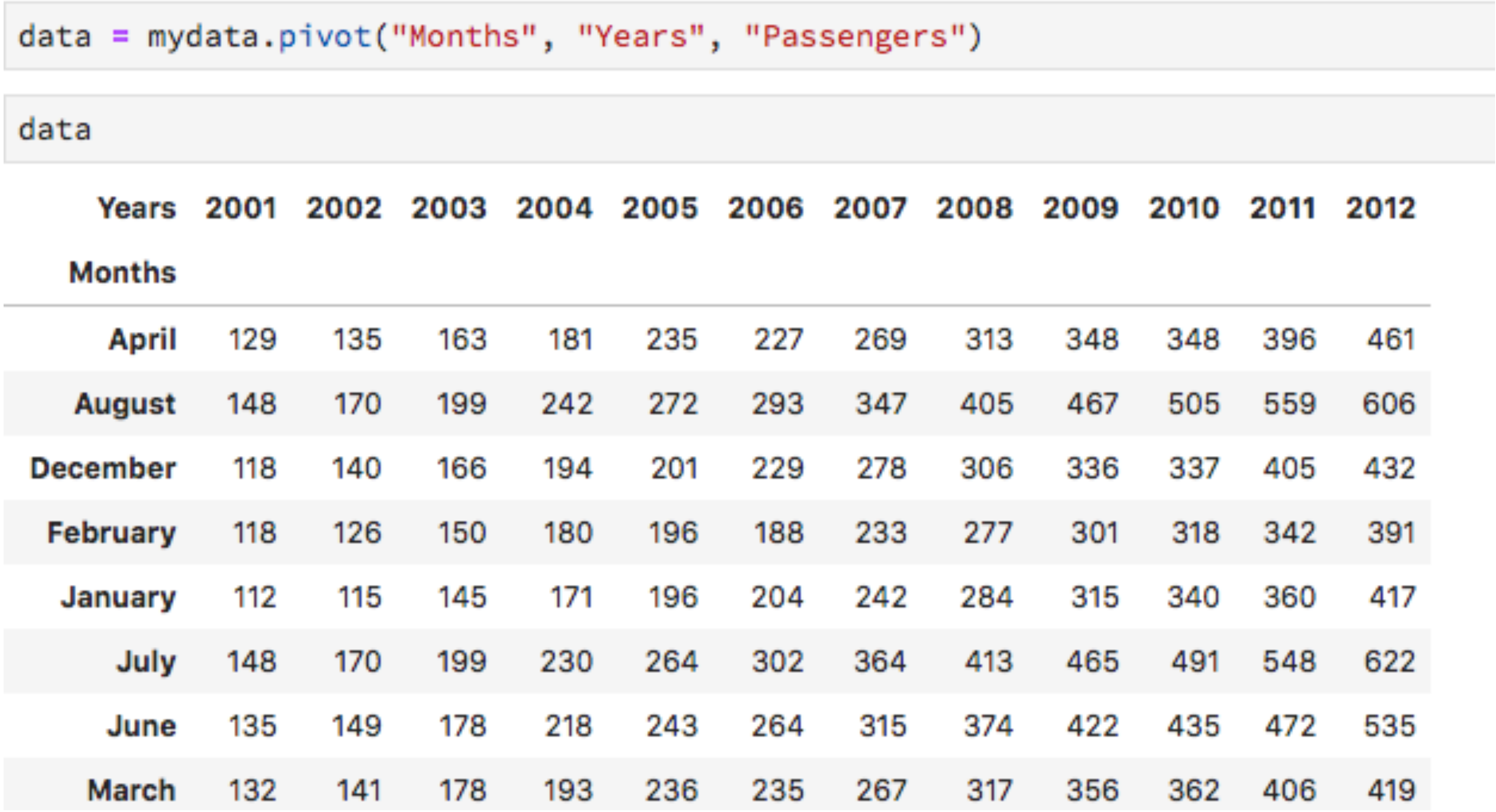


```
mydata.head()
```

	Years	Months	Passengers
0	2001	January	112
1	2001	February	118
2	2001	March	132
3	2001	April	129
4	2001	May	121

Read data located in the data folder

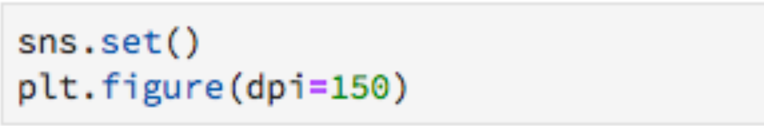
Provide meaningful data labels to our DataFrame



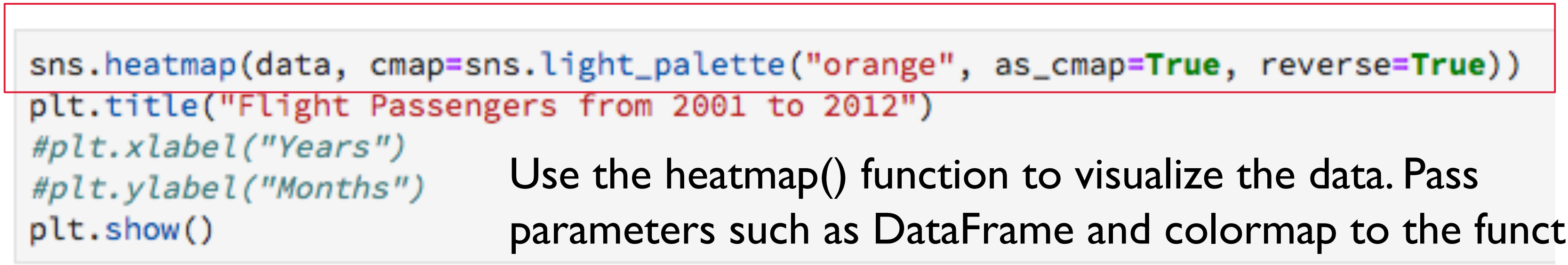
```python
data = mydata.pivot("Months", "Years", "Passengers")
```

```python
data
```

Years	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012
Months												
April	129	135	163	181	235	227	269	313	348	348	396	461
August	148	170	199	242	272	293	347	405	467	505	559	606
December	118	140	166	194	201	229	278	306	336	337	405	432
February	118	126	150	180	196	188	233	277	301	318	342	391
January	112	115	145	171	196	204	242	284	315	340	360	417
July	148	170	199	230	264	302	364	413	465	491	548	622
June	135	149	178	218	243	264	315	374	422	435	472	535
March	132	141	178	193	236	235	267	317	356	362	406	419

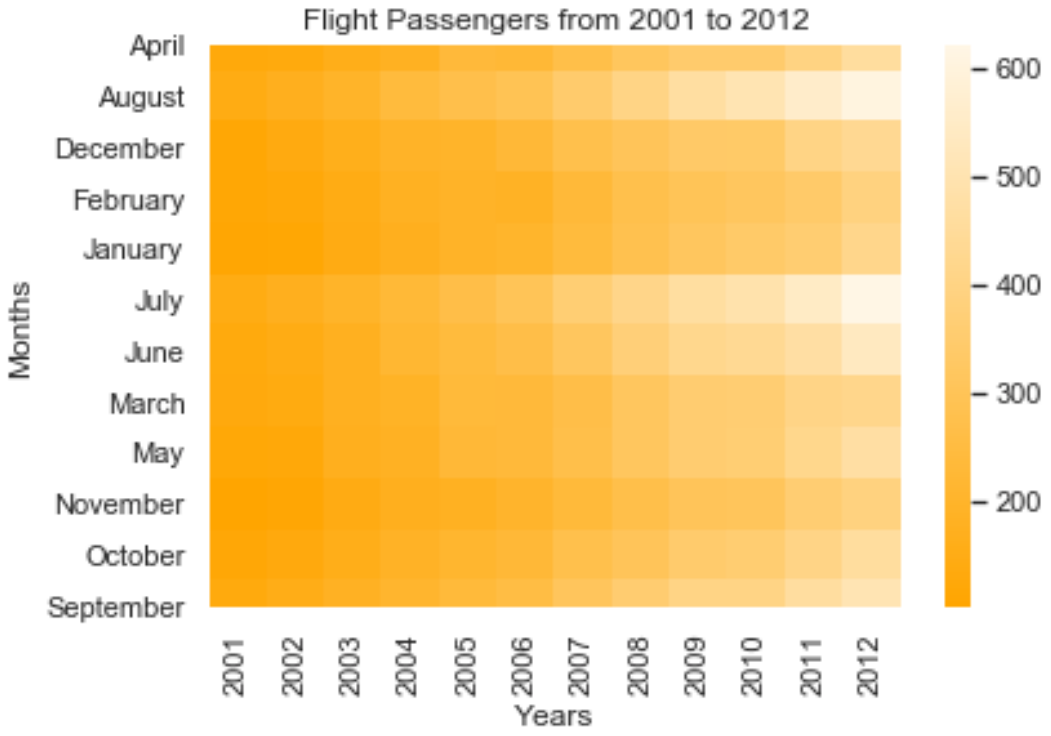


```
sns.set()
plt.figure(dpi=150)
```



```
sns.heatmap(data, cmap=sns.light_palette("orange", as_cmap=True, reverse=True))
plt.title("Flight Passengers from 2001 to 2012")
#plt.xlabel("Years")
#plt.ylabel("Months")
plt.show()
```

Use the heatmap() function to visualize the data. Pass parameters such as DataFrame and colormap to the function.



Violin Plots

Violin plots can be used to visualizing statistical measures.

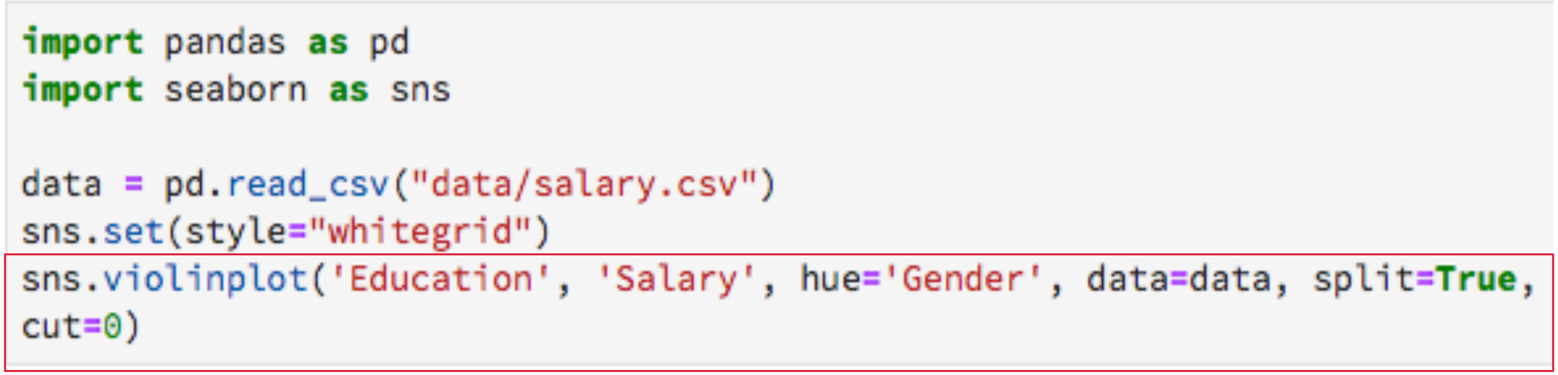
Violin plots combine box plots with the kernel density estimation procedure. It provides a richer description of the variable's distribution.

The quartile and whisker values from the box plot are also shown inside the violin.

An interpretation of violin plot can be found here:
https://mode.com/blog/violin-plot-examples

More details of kernel density estimation can be found here:
https://en.wikipedia.org/wiki/Kernel_density_estimation

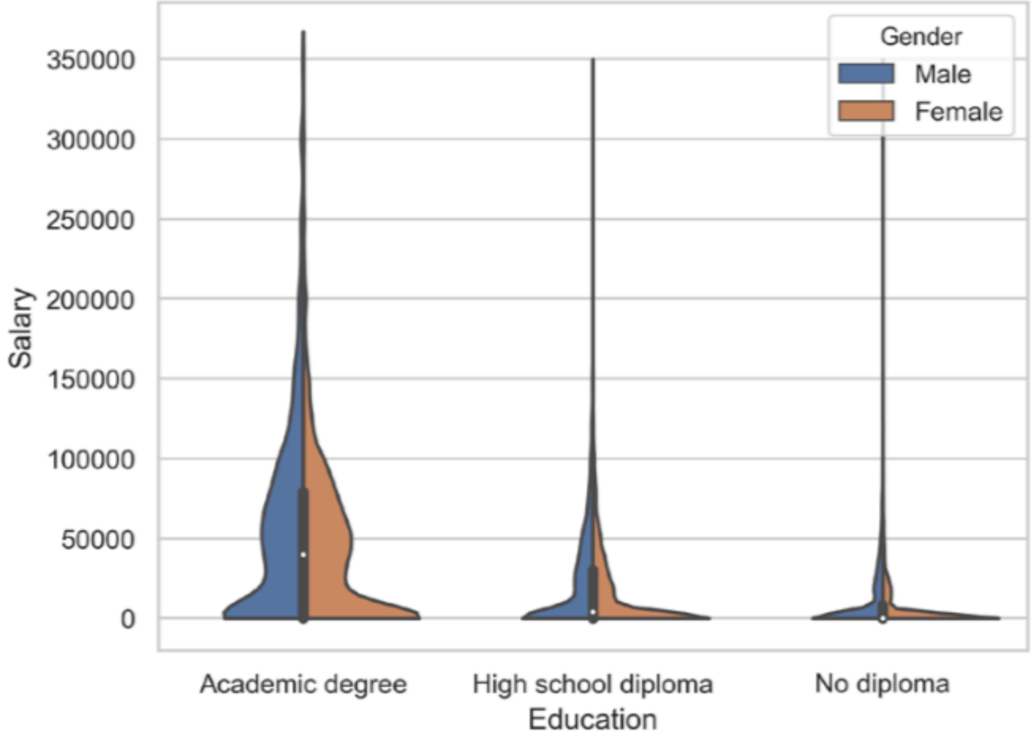
Violin Plots – example



```python
import pandas as pd
import seaborn as sns

data = pd.read_csv("data/salary.csv")
sns.set(style="whitegrid")
sns.violinplot('Education', 'Salary', hue='Gender', data=data, split=True,
cut=0)
```

We use **violinplot()** function provided by seaborn to create a violin plot.

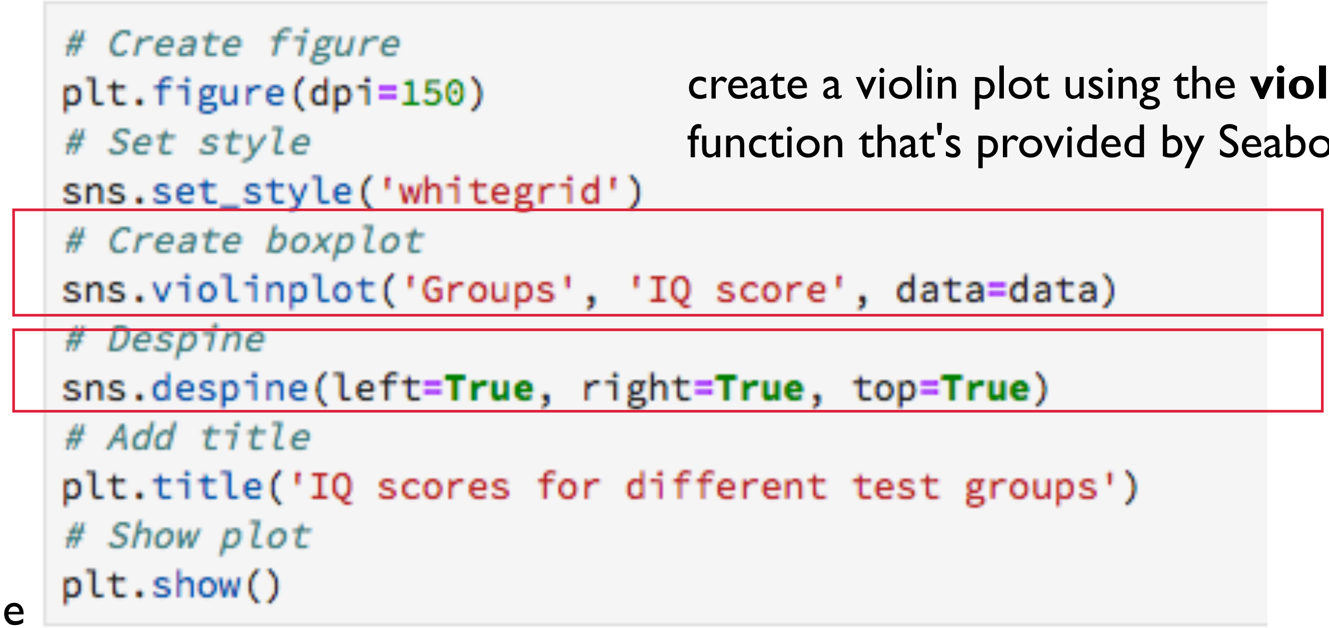


Activity 3

Revisit – Comparing IQ Scores for Different Test Groups by Using a Violin Plot

Objective

In this activity, we will compare IQ scores among different test groups using violin plots.

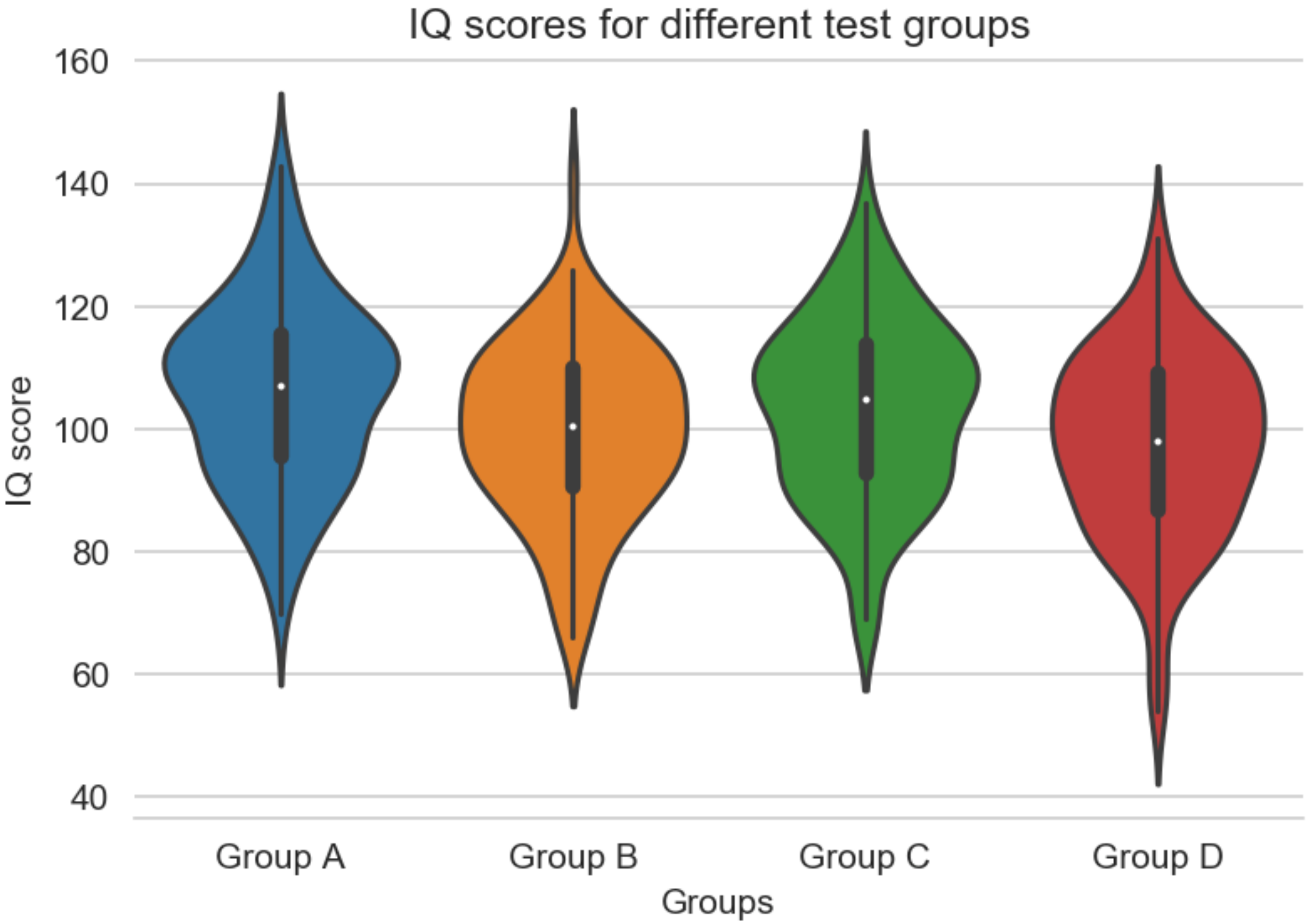


```
# Create figure
plt.figure(dpi=150)
# Set style
sns.set_style('whitegrid')
# Create boxplot
sns.violinplot('Groups', 'IQ score', data=data)
# Despine
sns.despine(left=True, right=True, top=True)
# Add title
plt.title('IQ scores for different test groups')
# Show plot
plt.show()
```

create a violin plot using the **violinplot()** function that's provided by Seaborn

Remove the top and right spines from the plot.

Expected view



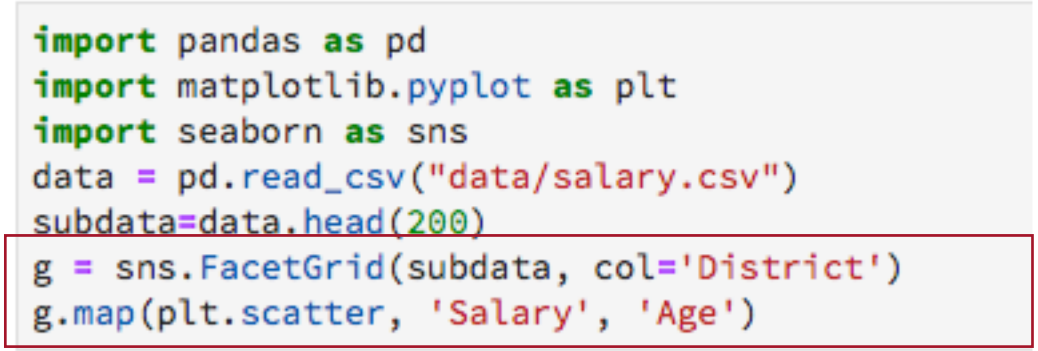
Multi-Plots in Seaborn

seaborn.FacetGrid(data, row, col, hue, ...) initializes a multi-plot grid for plotting conditional relationships.

Here are some interesting parameters:

- **data** – A tidy ("long-form") DataFrame where each column corresponds to a variable and each row corresponds to an observation
- **row, col, hue** – Variables that define subsets of the given data, which will be drawn on separate facets in the grid
- **sharex, sharey** (optional) – Share **x/y** axes across rows/columns
- **height** (optional) – Height (in inches) of each facet

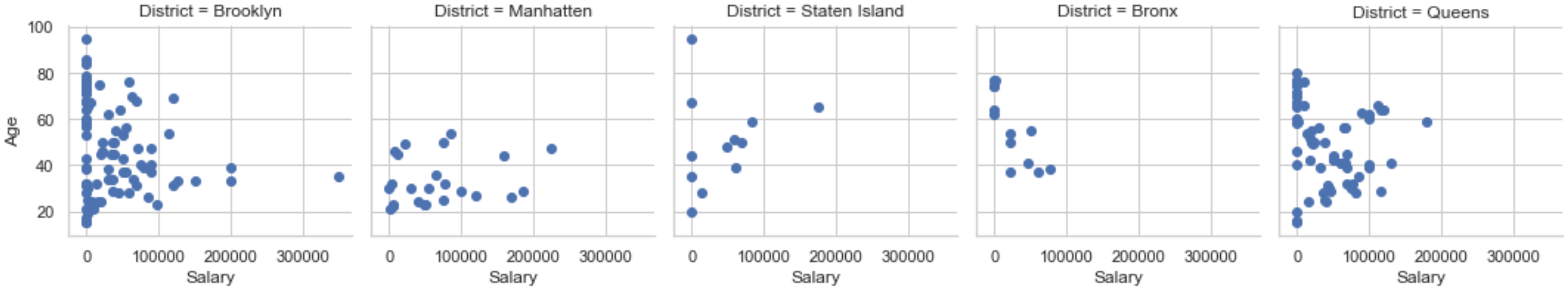
Multi-Plots in Seaborn – example



```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv("data/salary.csv")
subdata=data.head(200)
g = sns.FacetGrid(subdata, col='District')
g.map(plt.scatter, 'Salary', 'Age')
```

Create a FacetGrid with multiple plots. Each column represents a district.

Then, plot a scatterplot for each district.



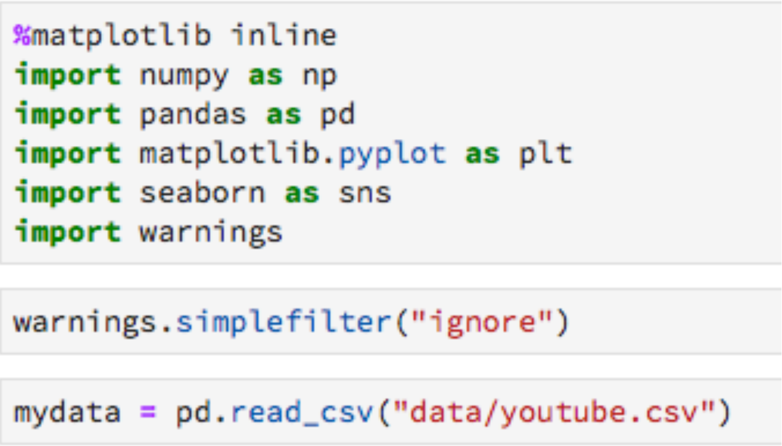
Activity 4

Top 30 YouTube Channels

Objective

In this activity, we will visualize the total number of subscribers and the total number of views for the top 30 YouTube channels by using the **FacetGrid()** function that's provided by the Seaborn library.

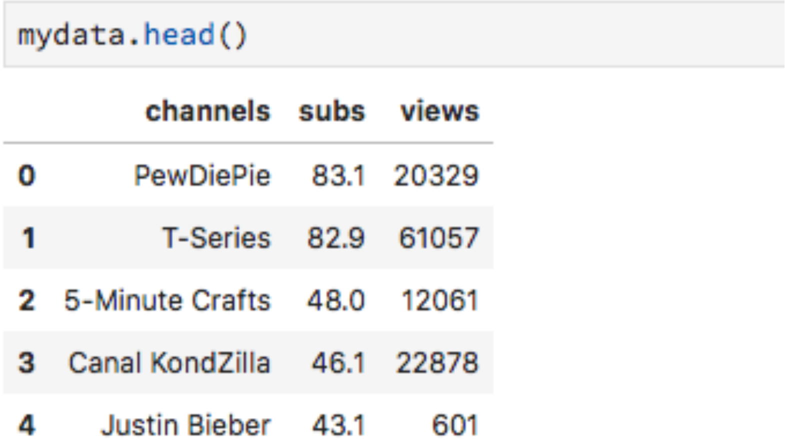
Visualize the given data using a FacetGrid with two columns. The first column should show the number of subscribers for each YouTube channel, whereas the second column should show the number of views.



```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
```

```
warnings.simplefilter("ignore")
```

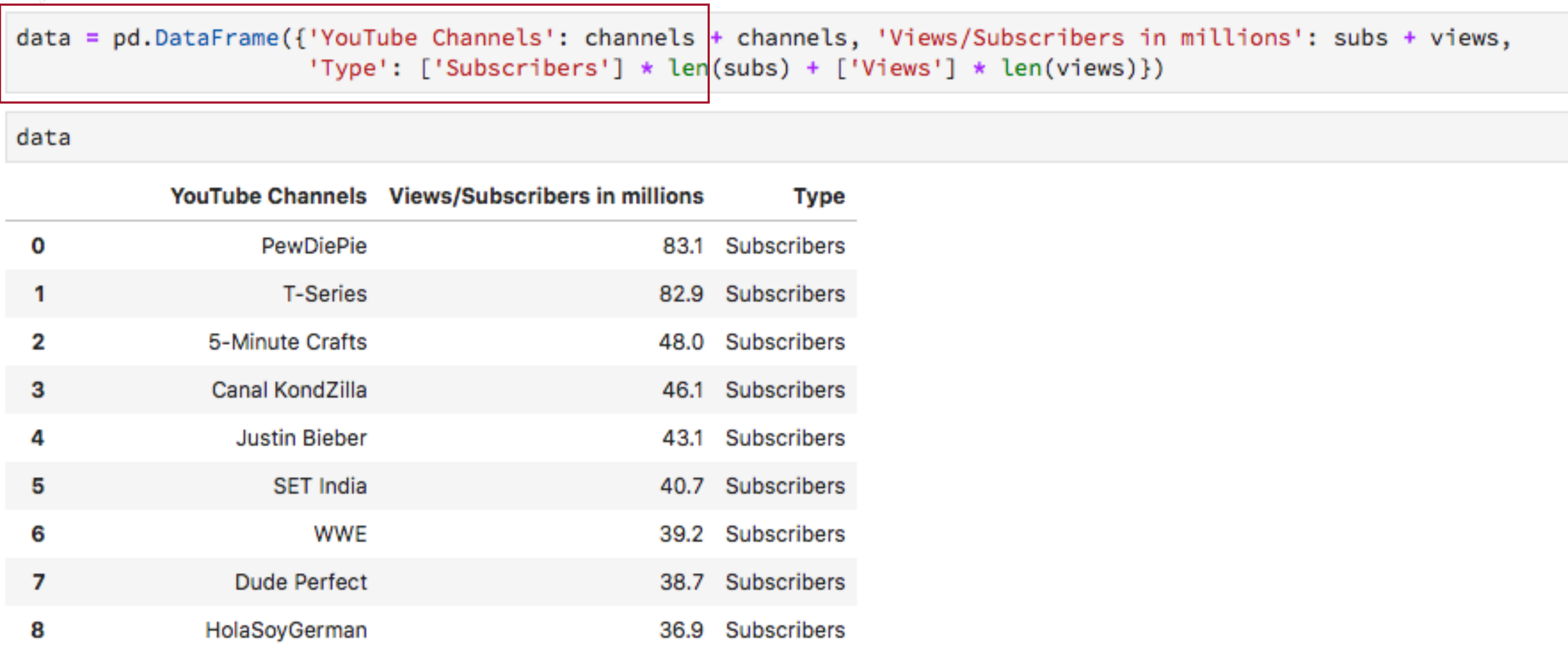
```
mydata = pd.read_csv("data/youtube.csv")
```



```
mydata.head()
```

	channels	subs	views
0	PewDiePie	83.1	20329
1	T-Series	82.9	61057
2	5-Minute Crafts	48.0	12061
3	Canal KondZilla	46.1	22878
4	Justin Bieber	43.1	601

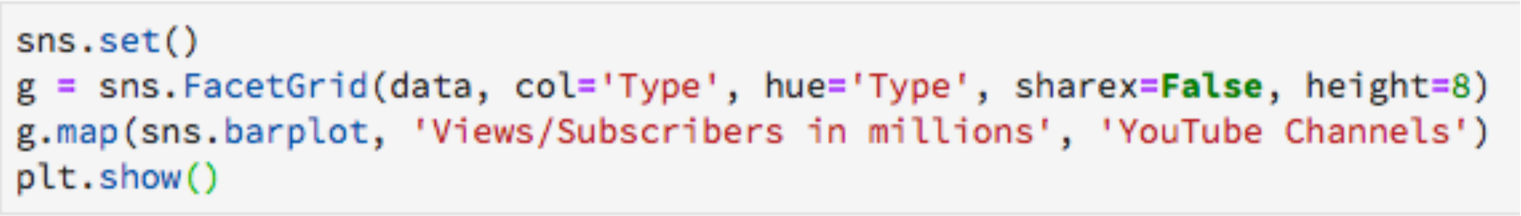
Construct a dataframe from the given data



```
data = pd.DataFrame({'YouTube Channels': channels + channels, 'Views/Subscribers in millions': subs + views,
                     'Type': ['Subscribers'] * len(subs) + ['Views'] * len(views)})
```

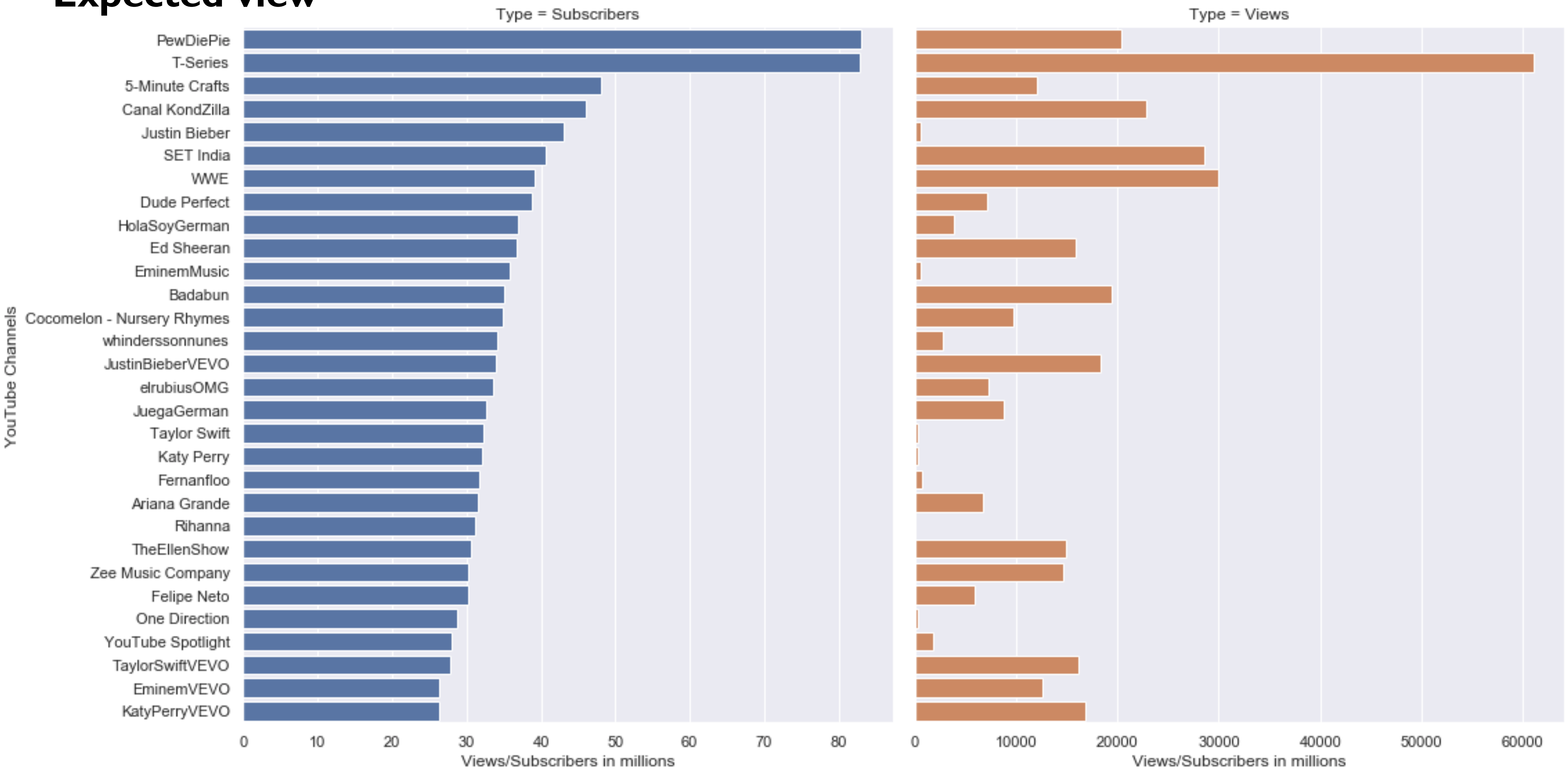
```
data
```

	YouTube Channels	Views/Subscribers in millions	Type
0	PewDiePie	83.1	Subscribers
1	T-Series	82.9	Subscribers
2	5-Minute Crafts	48.0	Subscribers
3	Canal KondZilla	46.1	Subscribers
4	Justin Bieber	43.1	Subscribers
5	SET India	40.7	Subscribers
6	WWE	39.2	Subscribers
7	Dude Perfect	38.7	Subscribers
8	HolaSoyGerman	36.9	Subscribers



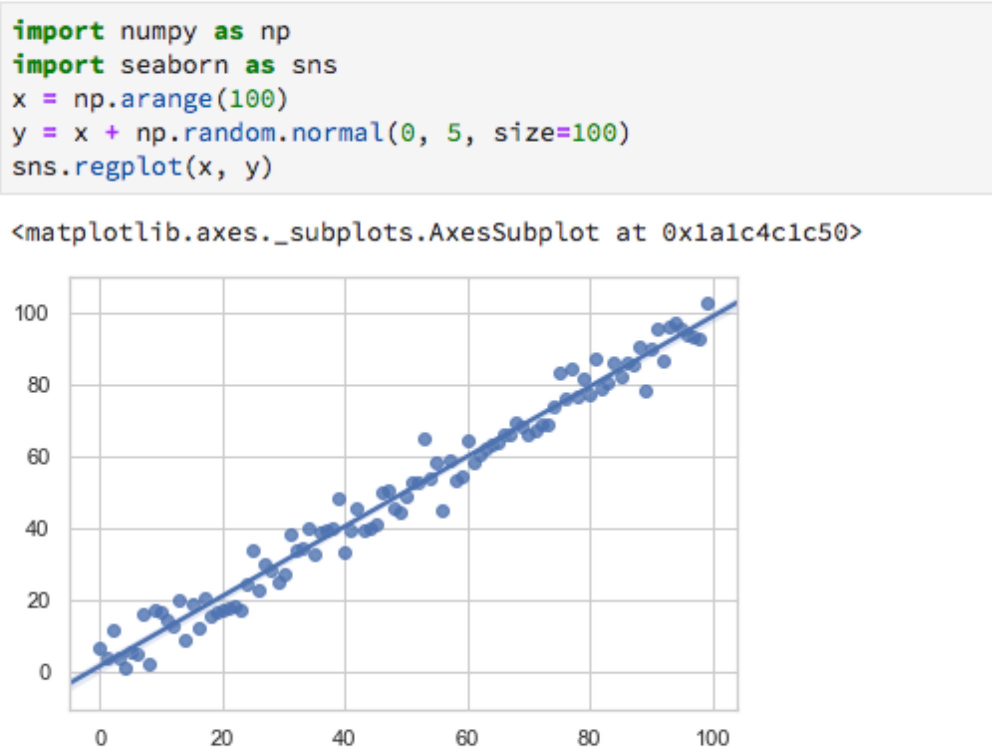
```
sns.set()
g = sns.FacetGrid(data, col='Type', hue='Type', sharex=False, height=8)
g.map(sns.barplot, 'Views/Subscribers in millions', 'YouTube Channels')
plt.show()
```

Expected view



Regression Plots

To visualize linear relationships, determined through linear regression, the **regplot()** function is offered by Seaborn.



```
import numpy as np
import seaborn as sns
x = np.arange(100)
y = x + np.random.normal(0, 5, size=100)
sns.regplot(x, y)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1c4c1c50>
```

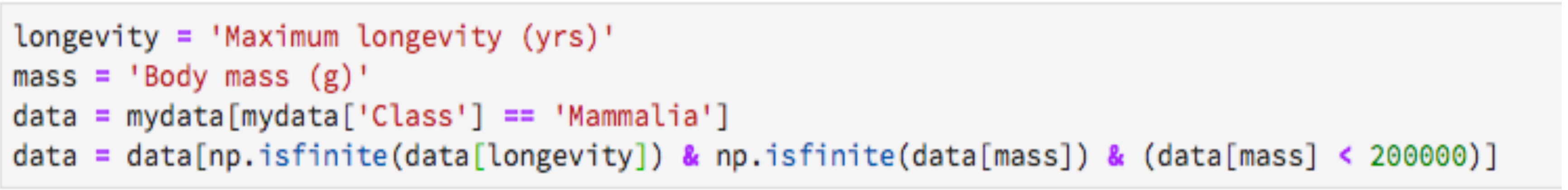
The regplot() function draws a scatter plot, a regression line, and a 95% confidence interval for that regression

Activity 5

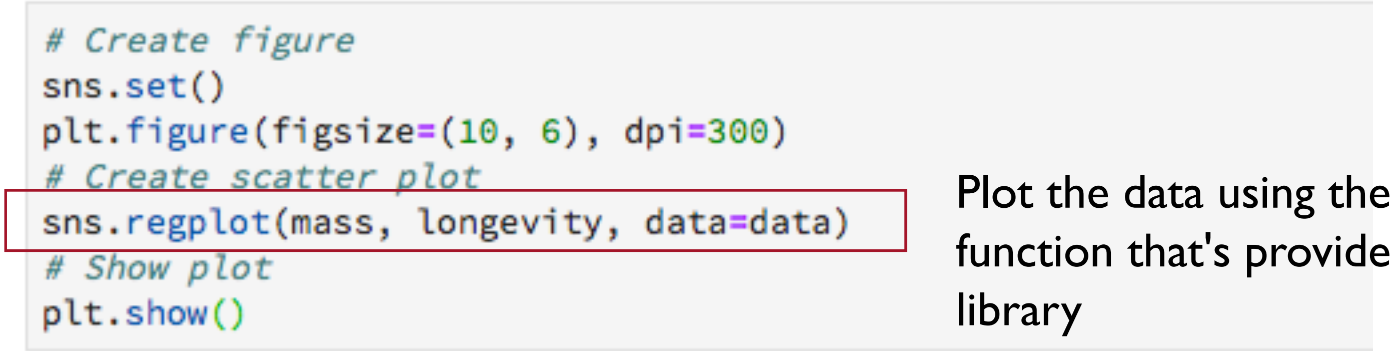
Linear Regression

Objective

In this activity, we will use a regression plot to visualize the linear relationship between animal body mass and maximum longevity.



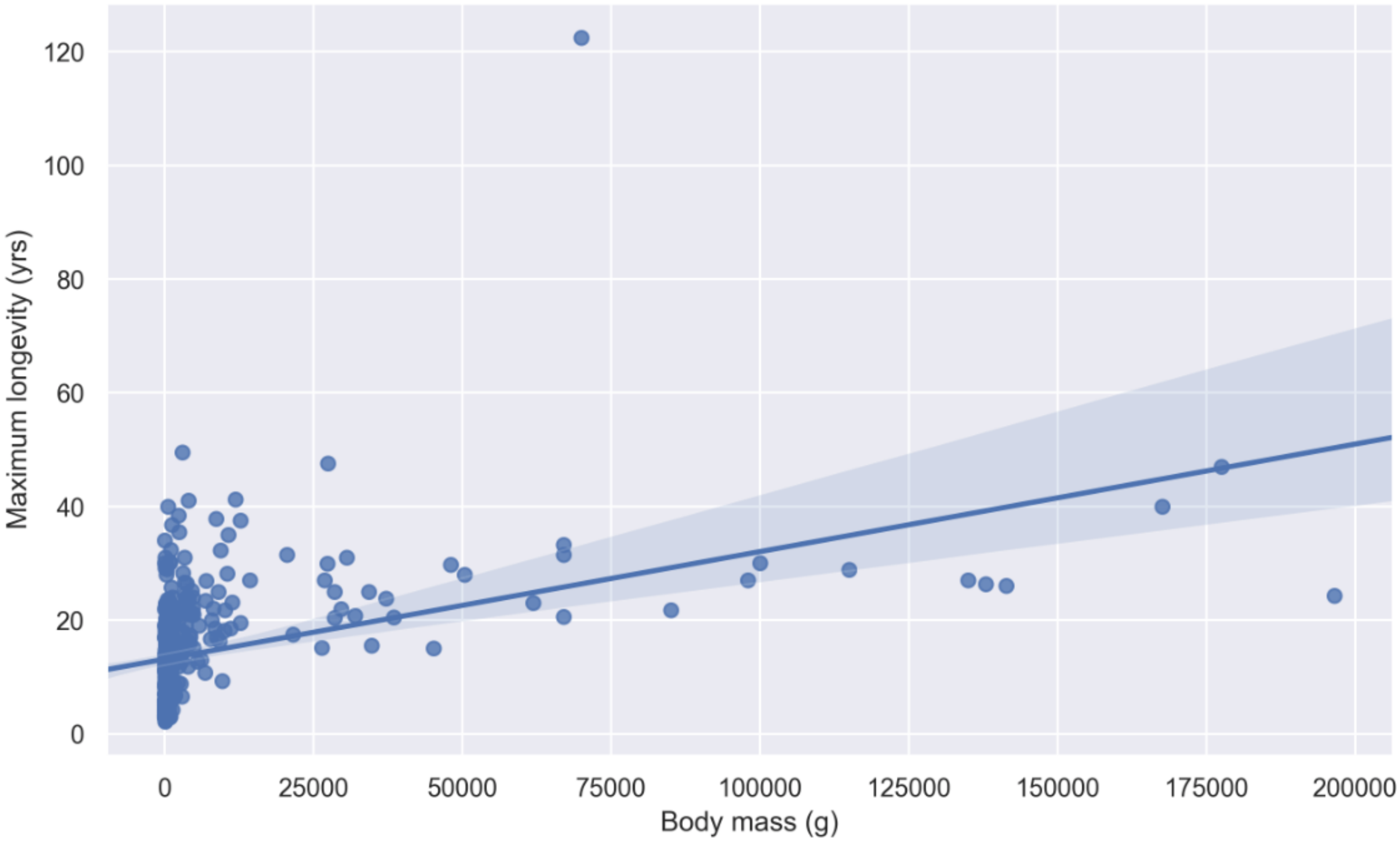
```
longevity = 'Maximum longevity (yrs)'
mass = 'Body mass (g)'
data = mydata[mydata['Class'] == 'Mammalia']
data = data[np.isfinite(data[longevity]) & np.isfinite(data[mass]) & (data[mass] < 200000)]
```



```
# Create figure
sns.set()
plt.figure(figsize=(10, 6), dpi=300)
# Create scatter plot
sns.regplot(mass, longevity, data=data)
# Show plot
plt.show()
```

Plot the data using the **regplot()** function that's provided by the Seaborn library

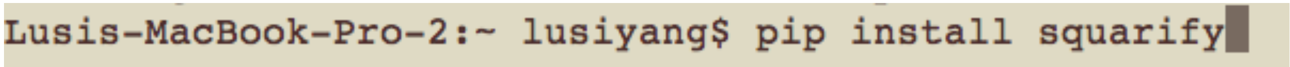
Expected view



Squarify

Tree maps display hierarchical data as a set of nested rectangles. Each group is represented by a rectangle, of which its area is proportional to its value. Using color schemes, it is possible to represent hierarchies: groups, subgroups, and so on.

Compared to pie charts, tree maps efficiently use space. Matplotlib and Seaborn do not offer tree maps, and so the **Squarify** library that is built on top of Matplotlib is used.

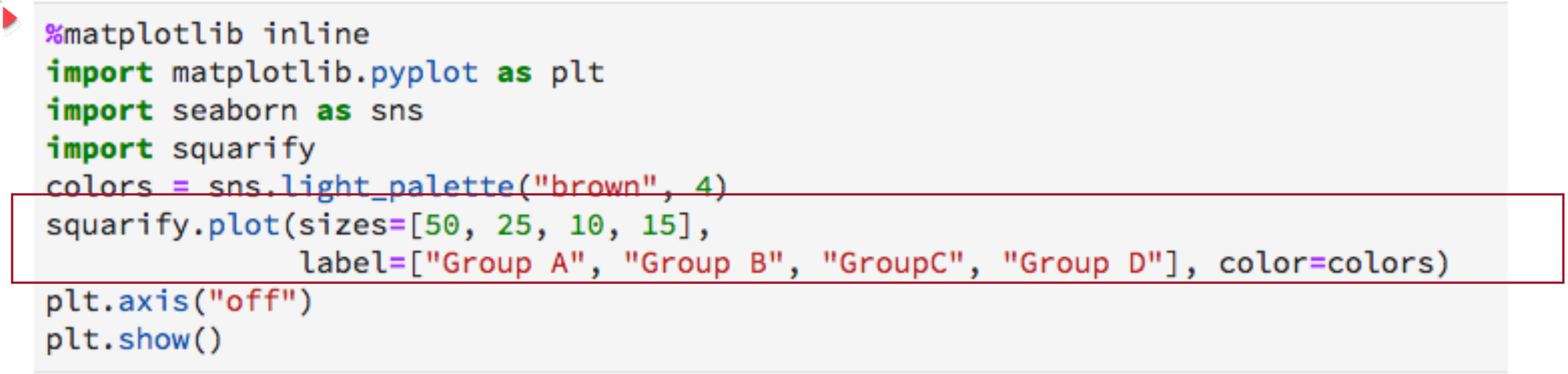


```
Lusis-MacBook-Pro-2:~ lusiyang$ pip install squarify
```

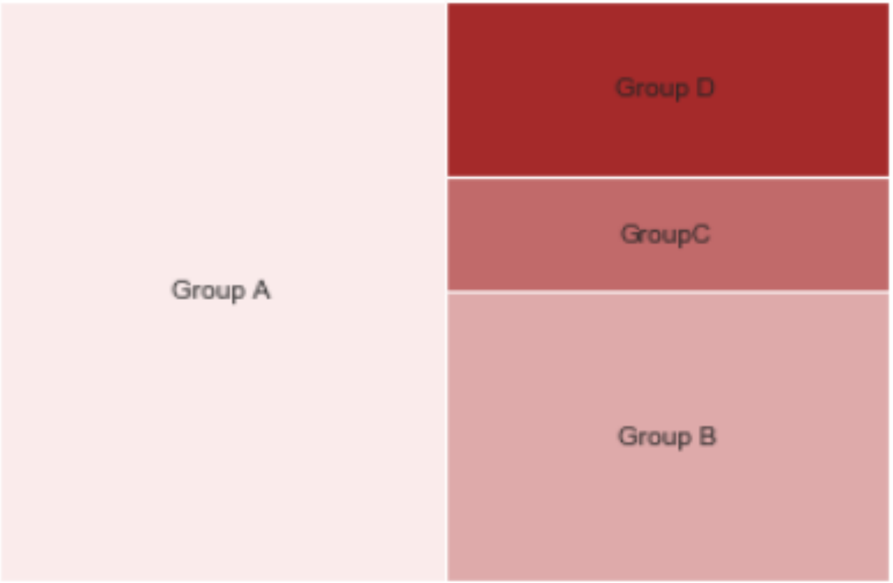
Install Squarify library:

Type “pip install squarify” in command line

Squarify – an example of treemap



```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import squarify
colors = sns.light_palette("brown", 4)
squarify.plot(sizes=[50, 25, 10, 15],
              label=["Group A", "Group B", "GroupC", "Group D"], color=colors)
plt.axis("off")
plt.show()
```



We use the **plot()** function provided by Squarify to create a treemap. Pass the numbers, group labels, and color palette to the function.

On your own: create a correlogram

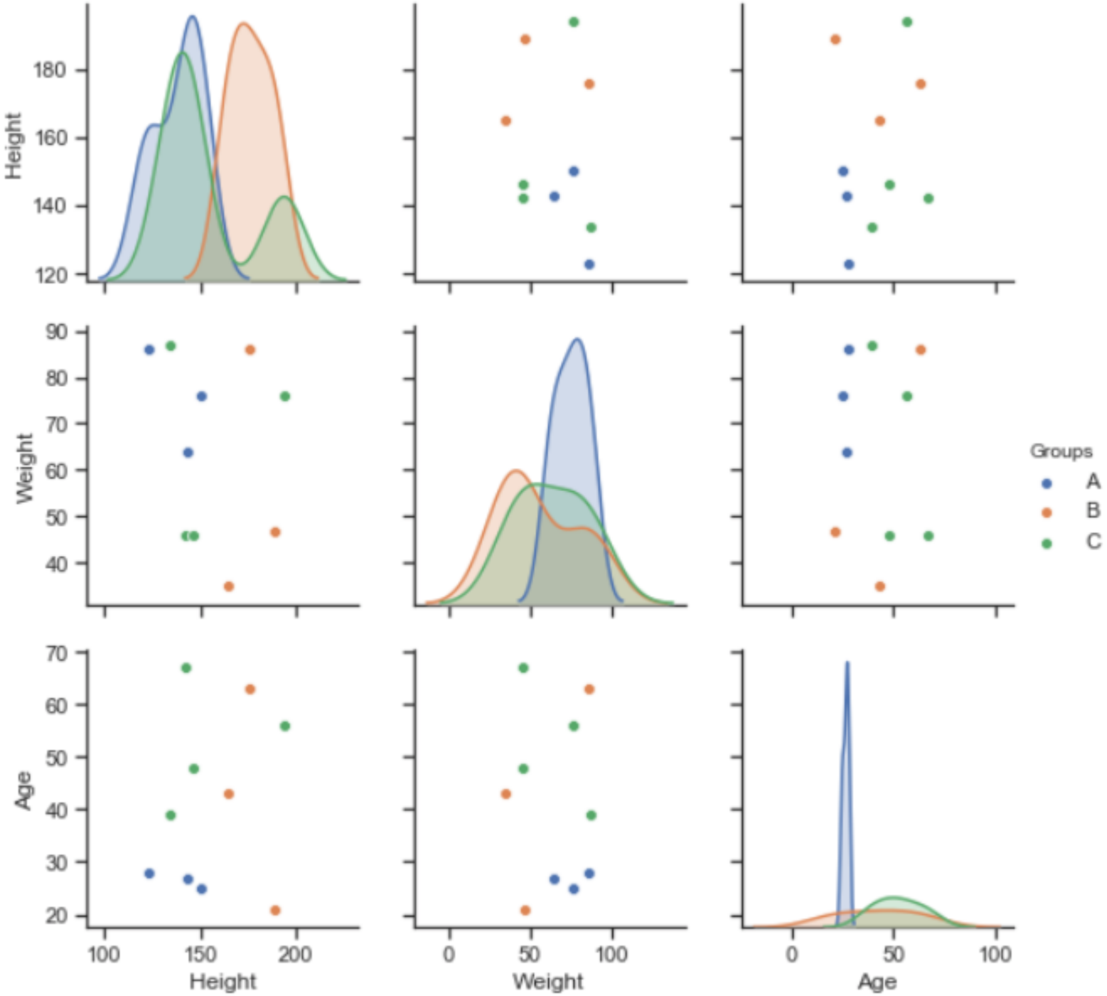
For visualizing multiple pairwise bivariate distributions in a dataset, Seaborn offers the **pairplot()** function.

This function creates a matrix where off-diagonal elements visualize the relationship between each pair of variables and the diagonal elements show the marginal distributions.

Your task:

Use the dataset "basic_details.csv" to create a correlogram, as shown on the next slide.

Expected view



The solution of this exercise will be shared on this Friday.

Summary

What we have learnt today?

- How Seaborn helps create visually appealing figures.
- Various options for controlling figure aesthetics, such as figure style, and controlling spines.
- Color palettes in Seaborn.
- FacetGrids – creating multi-plots, and regression plots as a way to analyze the relationships between two variables.
- Squarify library – create tree maps