Visualization in Python

Data Visualization with Matplotlib

- Data visualizations let you derive insights from data and let you communicate about the data with others!

Customization

- Markers, line style, and color:

```
fig, ax = plt.subplots()
ax.plot(weather['month'], weather['Tavg'], marker='o', linestyle='--', color
='r')
```

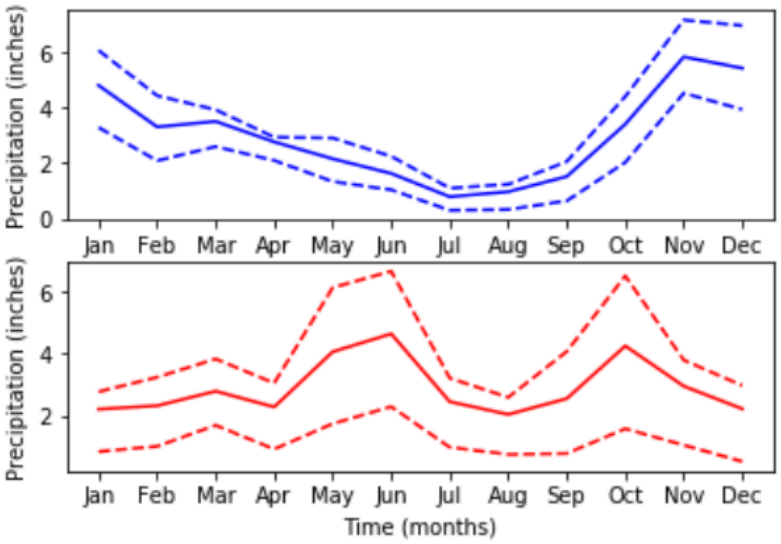
- Labels: (axis has many methods that start with set)

```
ax.set_xlabel("Time")
ax.set_ylabel("Temperature")
```

- Title:

```
ax.set_title("Weather in Egypt")
```

Subplots




```
fig, ax = plt.subplots(2, 1)
ax[0].plot(A_weather['month'], weather['MLY'], color = 'b')
ax[0].plot(A_weather['month'], weather['MLY_25_quantile'], color = 'b', line
style='--')
ax[0].plot(A_weather['month'], weather['MLY_75_quantile'], color = 'b', line
style='--')
ax[1].plot(B_weather['month'], weather['MLY'], color = 'r')
ax[1].plot(B_weather['month'], weather['MLY_25_quantile'], color = 'r', lines
tyle='--')
ax[1].plot(B_weather['month'], weather['MLY_75_quantile'], color = 'r', lines
tyle='--')
```

- If two figures share the same y-axis, we use the argument **sharey**:

```
fig, ax = plt.subplots(2, 1, sharey = True)
```

Visualizing Time Series Data:

- Visualizing time series is a great method to detect patterns in data.
- To make Python recognize data series, the date column should be a date type, and the dates have to be the index.

```
fig, ax = plt.subplots()
ax.plot(climate_change.index, climate_change['CO2']
```

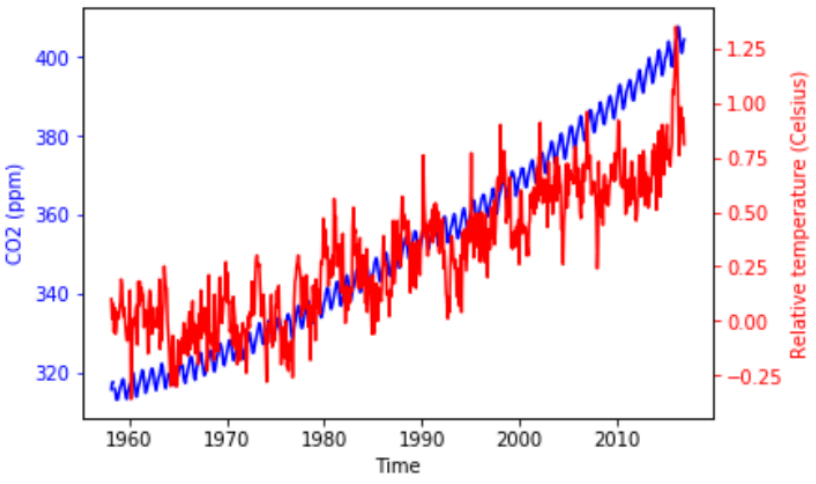
- We can select a period of time like a decade for example, using slicing:

```
sixties=climate_change["1960-01-01":"1969-12-31"]
ax.plot(sixties.index, sixties['CO2']
```

Showing two time series in one figure

- To visualize two figures that share the same y-axis, we use twinx() method:

```
fig, ax = plt.subplots()
ax.plot(climate_change.index, climate_change['CO2'], color = "b")
ax.set_ylabel=('CO2', color ="b")
ax.tick_params('y', color='b')
ax2 = ax.twinx()
ax2.plot(climate_change.index, climate_change['Temp'], color ="r")
ax2.set_ylabel=('Temp', color ="r")
ax2.tick_params('y', color='r')
```




```
# A function to make the process easier
def plot_timeseries(axes, x, y, color, xlabel, ylabel):
    axes.plot(x, y, color=color)
    ax.set_xlabel=(xlabel)
```

```
ax.set_ylabel=(ylabel, color =color)
ax.tick_params('y', color=color)
```

Annotations

- Annotations are usually small pieces of text that refer to a particular part of the visualization
- annotate(text to be added, xy = (x, y))

```
# Add an annotation to a time-series fig
ax.annotate("Birthday", xy=(pd.Timestamp("2003-01-12"), 1))
# We can add the position to place the text
ax.annotate("Birthday", xy=(pd.Timestamp("2003-01-12"), 1), xytext = (pd.
Timestamp("2003-01-12"), -0.2))
# We can also add an arrow to point to the data point we want to annotate
ax.annotate("Birthday", xy=(pd.Timestamp("2003-01-12"), 1), xytext = (pd.
Timestamp("2003-01-12"), -0.2)
arrowprops={"arrowstyle":"→", "color":"grey"}) # default is arrowprops =
{}
```

Quantitative Comparisons

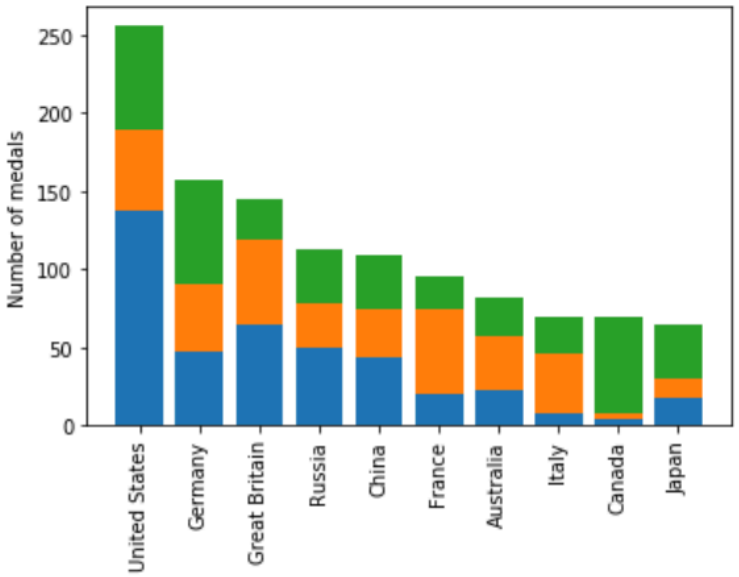
Bar Charts

```
ax, fig=plt.subplots()
ax.bar(medals.index, medals['Gold'])
ax.set_xticklables(medals.index, rotation = 90) # rotate thr data points' nam
es to avoid overlaping
```

- To compare data, we can stack them on each other:

```
ax.bar(medals.index, medals['Silver'], bottom = medals['Gold'])
ax.bar(medals.index, medals['Bronze'], bottom = medals['Gold'] + bottom =
```

```
medals['Silver'])
```



- If we want a map for each color, we add a label argument, then **legend()** method:

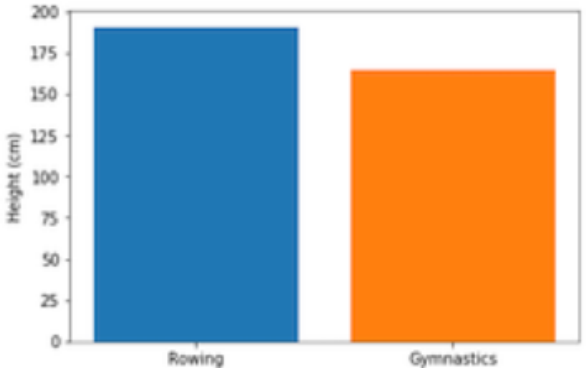
```
ax.bar(medals.index, medals['Gold'], label="Gold")
ax.bar(medals.index, medals['Silver'], bottom = medals['Gold'], lable="Silv
er")
ax.bar(medals.index, medals['Bronze'], bottom = medals['Gold'] + bottom =
medals['Silver']
label="Bronze")
plt.legend()
```

Histograms

- Bar charts **VS** Histogram in comparing two quantitative data:

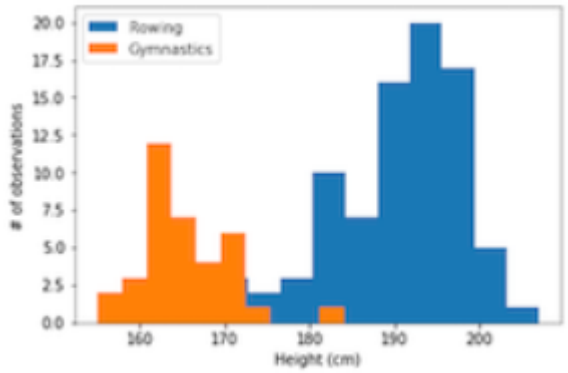
  1- Bar Charts:

```
ax.bar("Rowing", mens_rowing['Heights'].mean())
ax.bar("Gymnatsic", mens_gymnatsic['Heights'].mean())
plt.show()
```



2- Histograms:

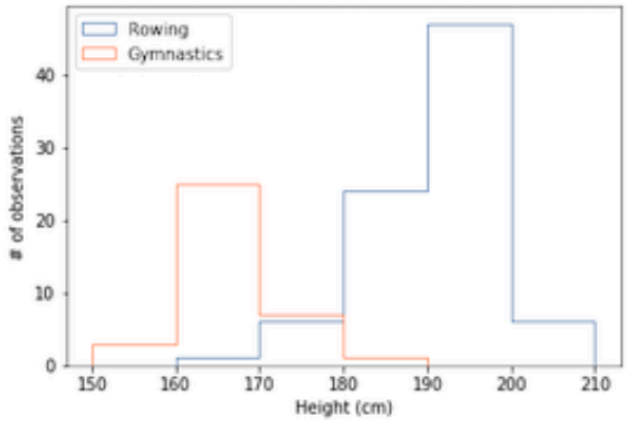
```
ax.hist(mens_rowing['Heights'], label="Rowing")
ax.hist(mens_gymnatsic['Heights'], label="Gymnatsic")
plt.legend()
plt.show()
```



- *Notice1:* we can change the type of the histogram if the current is not clear enough, like in this example we cannot observe data clearly in the intersection between them.

```
ax.hist(mens_rowing['Heights'], label="Rowing", histtype="step")
ax.hist(mens_gymnatsic['Heights'], label="Gymnatsic", histtype="step")
```

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

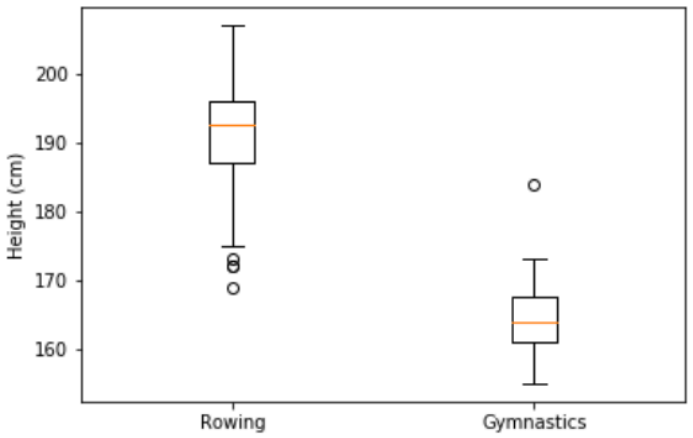


- *Notice2:* if we passed a list of values to the **bins** argument, it will be treated as the boundaries between the bins.

Boxplot

- compare two datasets using boxplot:

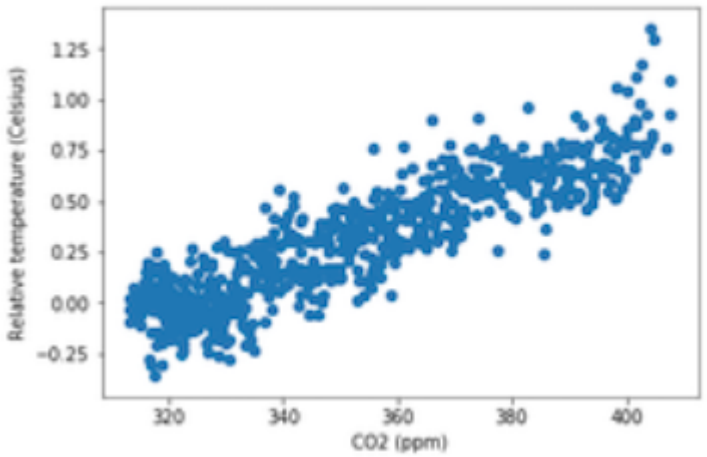
```
ax.boxplot([mens_rowing['Height'], mens_gymnastic['Height']])
ax.set_xticklabels(["Rowing", "Gymnatsic"])
ax.set_ylabel("Heights (cm)")
plt.show()
```



Scatter Plot

- So simple..

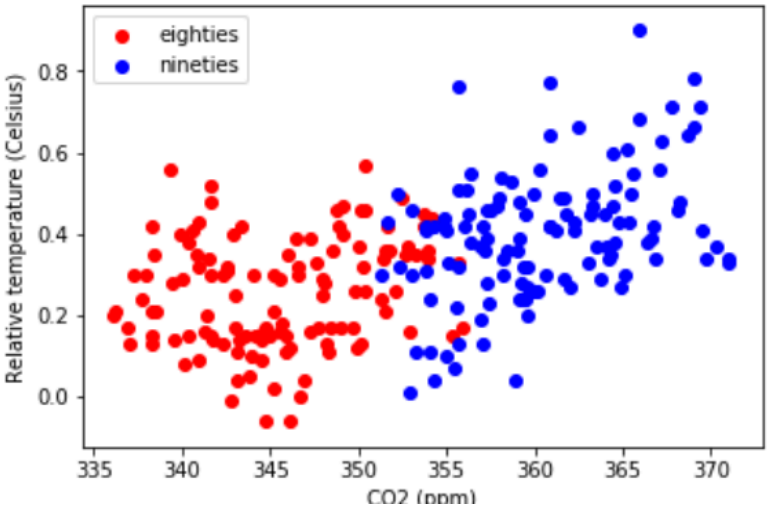
```
ax.scatter(climate_change['co2'], climate_change['temp'])
```



- We can benefit more from the scatter plot when comparing the relationship between CO2 and the temperature in different periods with the help of slicing:

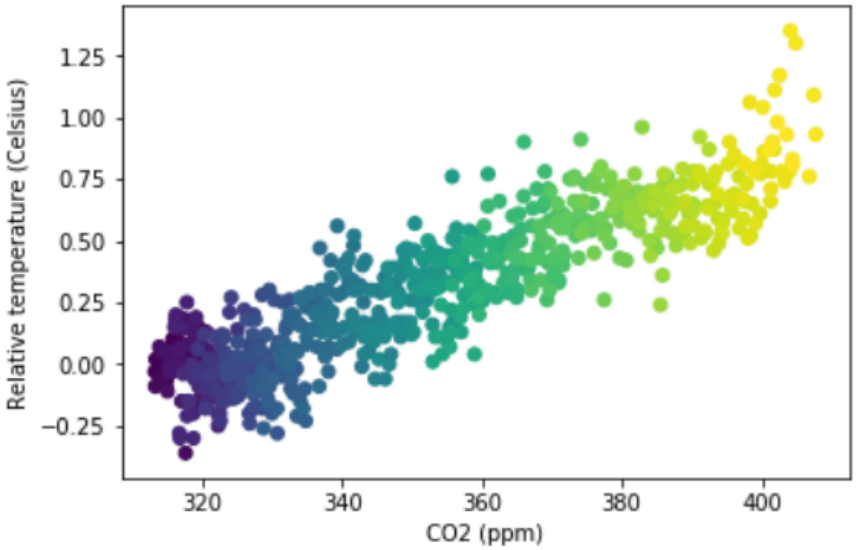
```
eighties= climate_change["1980-01-01":"1989-12-31"]
nineties= climate_change["1990-01-01":"1999-12-31"]
ax.scatter(eighties['co2'], eighties['temp'], color="red", label="eighties")
ax.scatter(nineties['co2'], nineties['temp'], color="blue", label="nineties")
ax.legend()
plt.show()
```

Notice: before comparing two data point groups, we have to make sure that they are distinct!



= We can see that the relationship between CO2 and temperature didn't change over the years, and both levels of CO2 and temperature continued to rise in the 1990s.

- We can add another variable to the comparison with **CO2** and the **temperature**, such as the **time** which represents the index in our example, if we pass the time to the "c" argument (which refers to the color), the early points will be darker than the later ones as shown below:




```
ax.scatter(climate_change['co2'], climate_change['temp'], c= climate_change.index)
```

```
plt.show()
```

Tips to share visualizations with others

- Using existing styles for the chart based on its purpose or place to be shared in:

```
plt.style.use('style-name-ex:ggplot')
```

- Saving the figures into your file system:

```
fig.savefig('medals.png')
```

= To know which files existing in the file system, use this:

```
ls
```

- To control the figure size:

```
fig.set_size_inches([weidth, height])
```

Data visualization with Seaborn

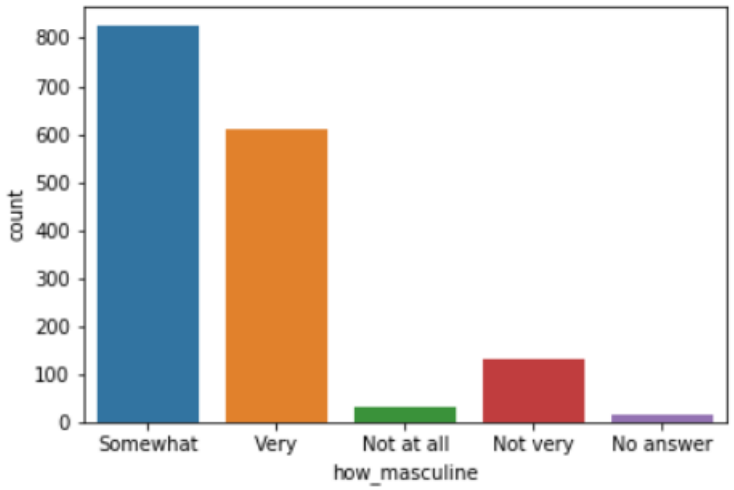
How does Seaborn work well with Pandas?

- In this example, we will use the countplot method, which takes a list of variables and plot each variable's count:

```
genders=["male", "male", "female", "female", "female"]
sns.countplot(x=genders)
```

- If we have a dataset and want to pass a column of men's replies on "How masculine are you?" to know the frequency of each answer, we can approach that with Seaborn:

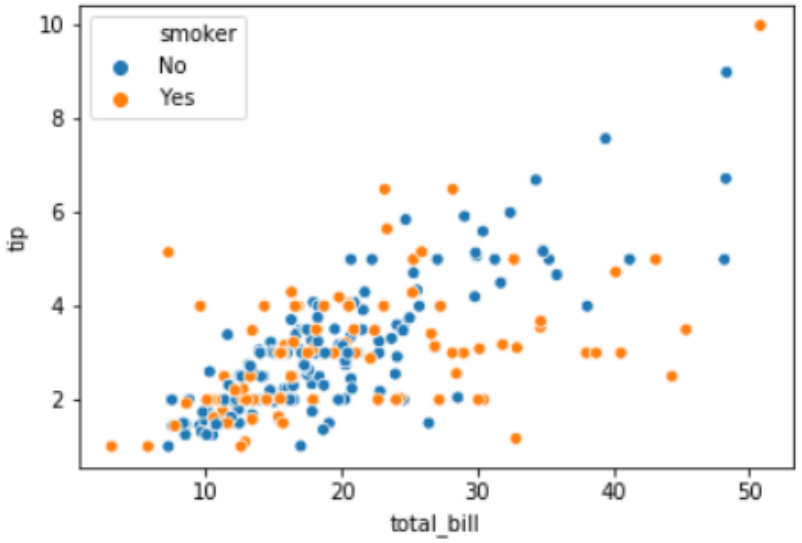
```
df = pd.read_csv('masculinty.csv')
sns.countplot(x="how_masculine", data=df) #(col_name, data_frame)
plt.show()
```



No need to use the legend method and the color parameter!

- Say we want to scatter the relationship between cafe tips and bills with a third variable, such as whether the customers are smokers or not. Previously, we would use slicing and define a color for each class, then create a legend, but with Seaborn, you only have to pass the smokers column to the `hue` argument and everything in done.

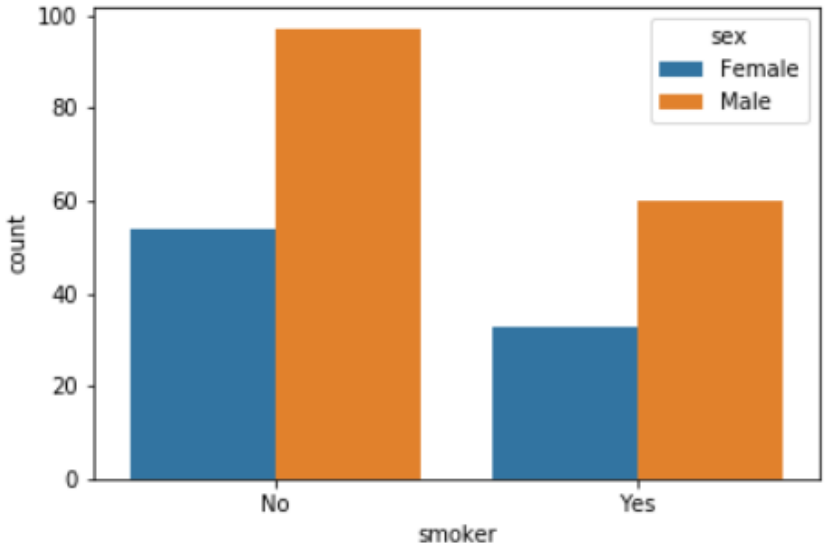
```
sns.scatterplot(x="total_bills", y="tips", data=df, hue="smokers")
```



- You want to customize it by yourself? no problem, pass a dictionary that maps each variable to a color to the `palette` argument:

```
hue_colors={"yes":"black",
                    "no":"red"  }
   sns.scatterplot(x="total_bills", y="tips", data=df, hue="smokers", palette
=hue_colors)
```

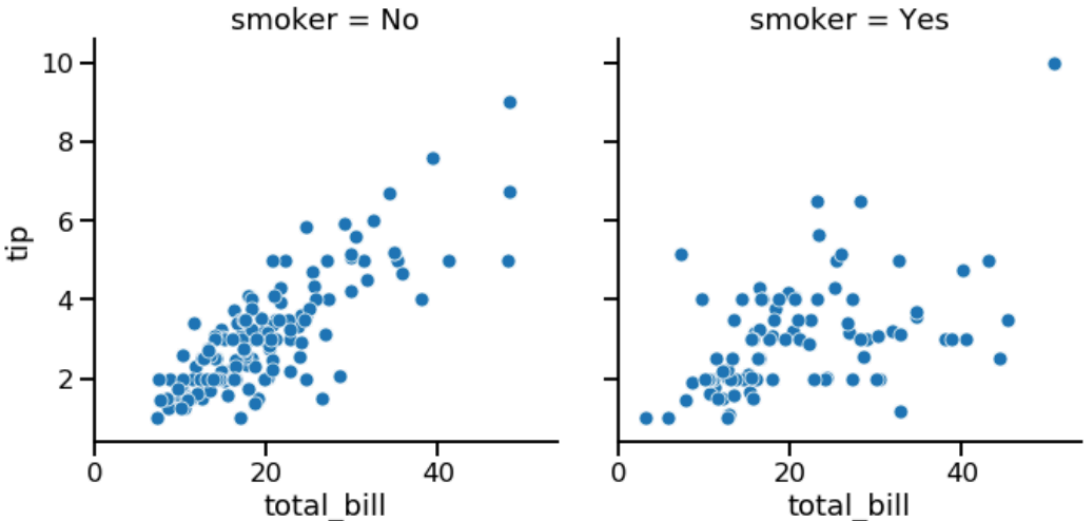
- `hue` works with other plot types such as bar plot:



Visualizing Two Quantitative Variables

Relational Plots

- There are 2 types of relation plots: Scatter plot an Line plot.
- In the cafe bills example, the scatter plot showed us the relation between bills and tips for smokers and non-smokers customers, but what if we want to see that relation separately? like if non-smokers have no relationships with the tips, we only need to visualize it with smokers, and to achieve this, we use **relplot** method



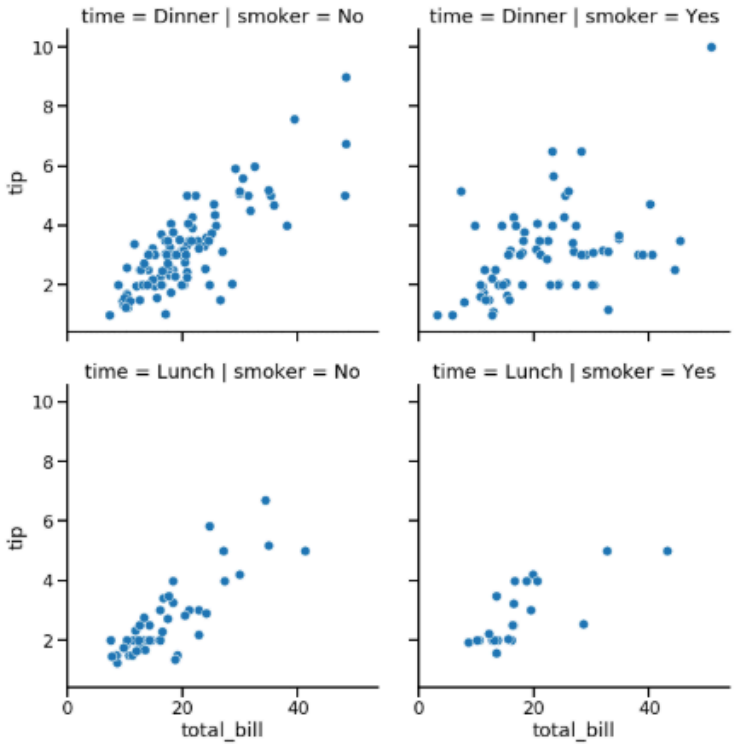
- The relplot() method creates separate plots per sub-groups.

```
#without separating
sns.relplot(x="total_bills", y="tips", data=df, kind="scatter")
#col argument separate the figures based on the column passed(row is eq
uilavent)
sns.relplot(x="total_bills", y="tips", data=df, kind="scatter", col="smokers")
```

- If we want to add a fourth variable like the time of the meal, we can use both col and row argument like this:

```
sns.relplot(x="total_bills", y="tips", data=df, kind="scatter", col="smokers",
```

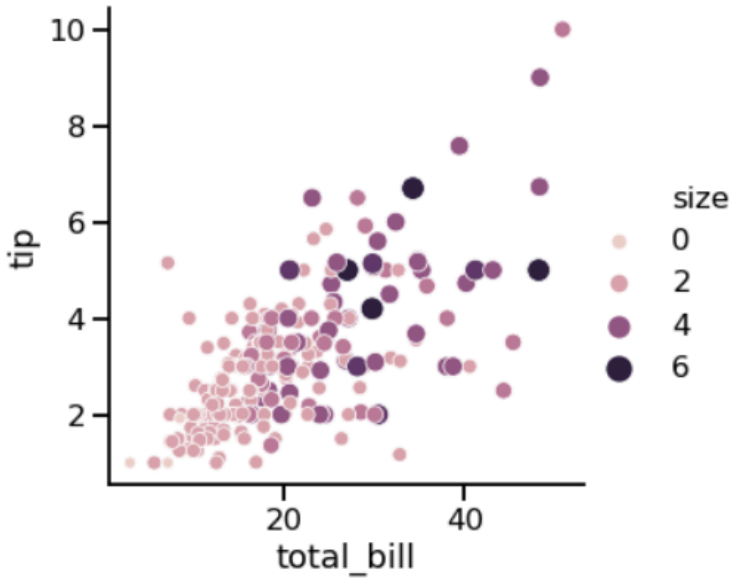
```
row="time")
```



= Notice: if you want to change the order of them, use col_order or row_order arguments.

Customizing Scatter Plots

- If we want to highlight the data points with high values for a third variable, we can use the size argument, which sets the point size based on its value. We can also use the **size** argument with the **color** to differentiate them more.




```
sns.relplot(x="total_bill", y="tip", data=df, kind="scatter", size="size", hue="size")
```

- **Style** argument works as the **color** argument, but instead of assigning different colors for each sub-group, it assign different style for each.
- Instead of **size** and **color** arguments, we can use the **alpha** argument which changes the transparency based on the value of the third variable, so that points with high values will be clearer than the lower ones with lower transparency. (we set alpha between 0 and 0.1)

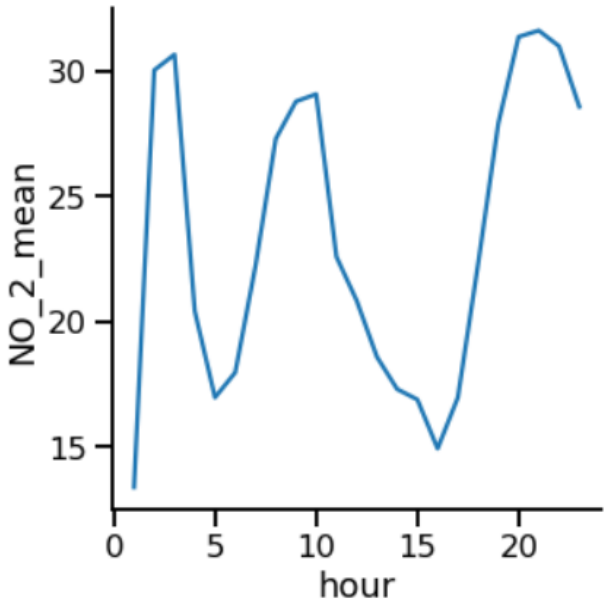
Line Plot

- The second type of the relational plots.
- While each point in a scatter plot is assumed to be an independent observation, line plots are the visualization of choice when we need to track the same thing over time.
- *Example:* tracking the value of a company's stock over time.

Case Study:

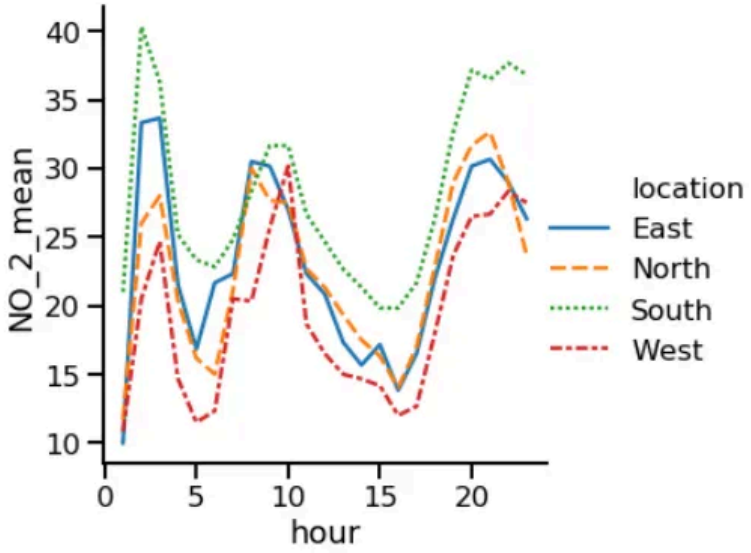
- We want to track the average of NO2 over the hours of the day:

```
sns.relplot(x="hour", y="no2", data=air, kind="line")
```



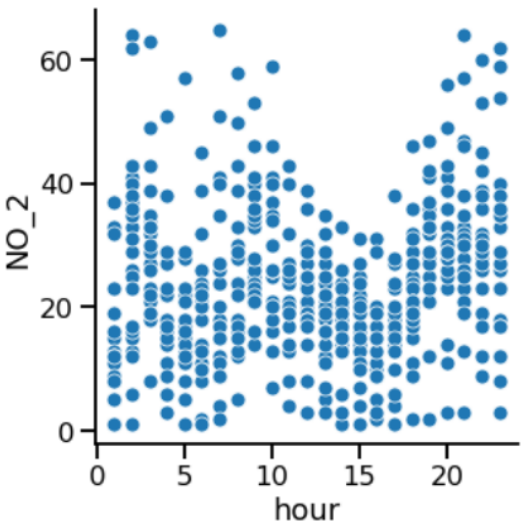
- We can also customize it to track the sub-groups as we have done earlier, like if we want to track the NO2 levels in each location:

```
sns.relplot(x="hour", y="no2", data=air, kind="line", style="location", hue
="location")
```

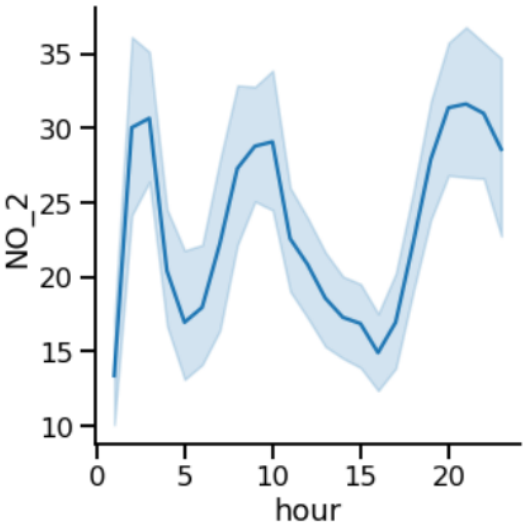


Multi observation per x-axis:

- If we have more than one observation for each hour in this example, plotting them using a scatter plot will be kinda gross, it does suit the **line plot** better, also the seaborn handles these cases by calculating the mean by default and representing it in only one line surrounded by a shaded region which is a confident interval for the mean .



sns.relplot(x="hour", y="no2", data=air, kind="**scatter**")



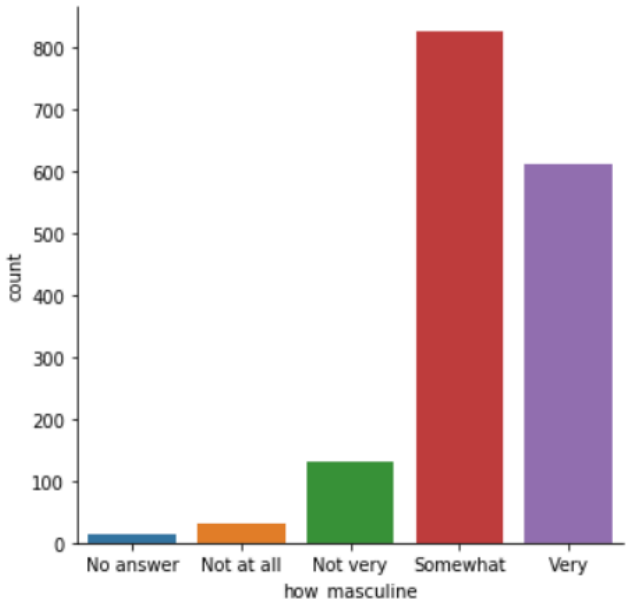
sns.relplot(x="hour", y="no2", data=air, kind="**line**")

- Note: In the line plot, if we want to replace the mean by standard deviation, set the **ci** argument to **sd**, and it we want to remove the shaded region, we can set the **ci** argument to **None**.

Visualizing a Categorical and a Quantitative Variable

- Categorical plots involve a categorical variable (*ex:* count plot and bar plot).
- In this section, we will use **catplot()** to create categorical plots, it offers the same flexibility as **relplot().**

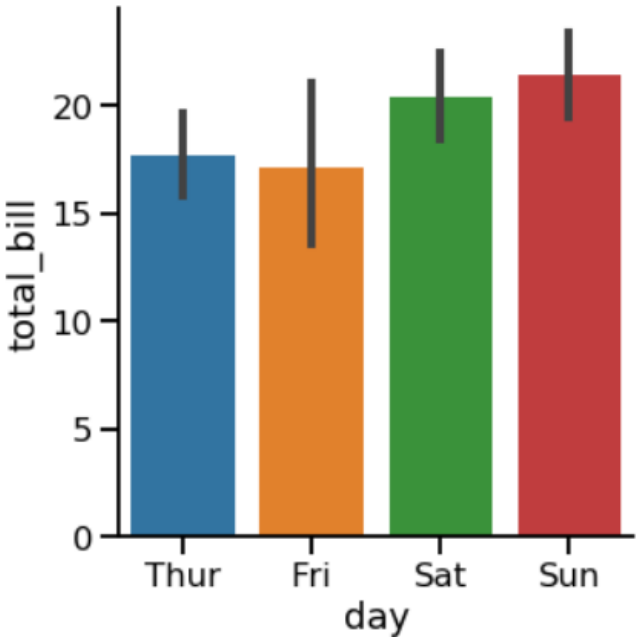
- Example: Do you remember the count plot visualization of the masculinity question? we can do it with the **catplot** and take benefits of it such as ordering the categories in ascending way:




```
category_order= ["No answer", "Not at all", "Not very", "Somewhat", "Ver
y"]
sns.catplot(x="how_masculine", data=how_masculinity, kind="count", orde
r=category_order)
```

Bar plots

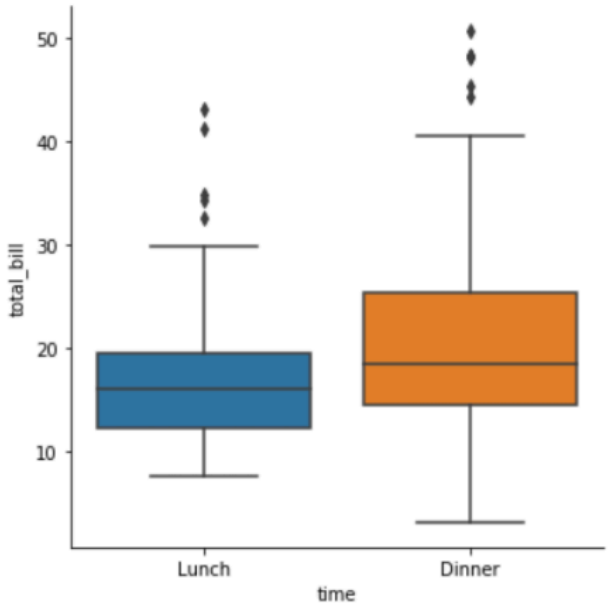
- Bar plots for categorical variables is similar to count plot, but instead of showing the frequency of each category, it shows the mean of a quantitative variable in each category:




```
sns.catplot(x="day", y="total_bill",data=tips ,kind="bar")
```

Box Plot

- Bos plot shows the distribution of quantitative data.
- Here we use it to compare the distributions of quantitative variable across different groups of categorical variable.)

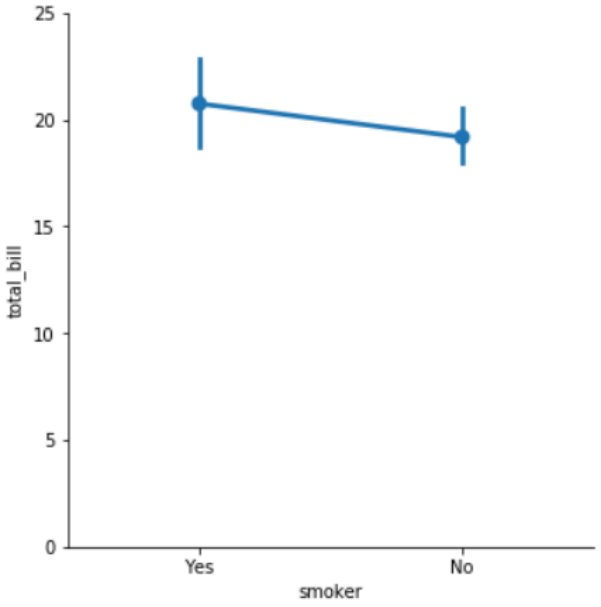



```
sns.catplot(x="time", y="total_bill", data=tips, kind="box")
```

- We can change the whiskers values by passing the min and the max percentage in a list to **whis** argument, we also can change the value of 1.5 by passing a single value to the same argument.

Point Plots

- Point plots show the mean of a quantitative variable for the observations in each category, plotted as a single point.



- This point plot uses the tips dataset and shows the average bill among smokers vs non-smokers.

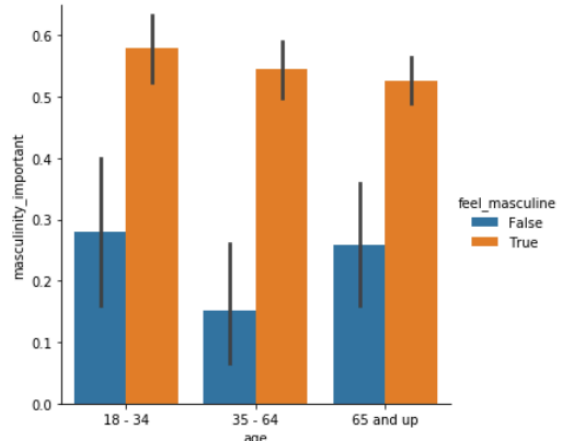
  (The vertical bars extending above and below the mean represent the 95% confidence intervals for that mean, as we use a random data sample of a population we can be 95% sure that the true population mean in each group lies within the confidence interval shown).

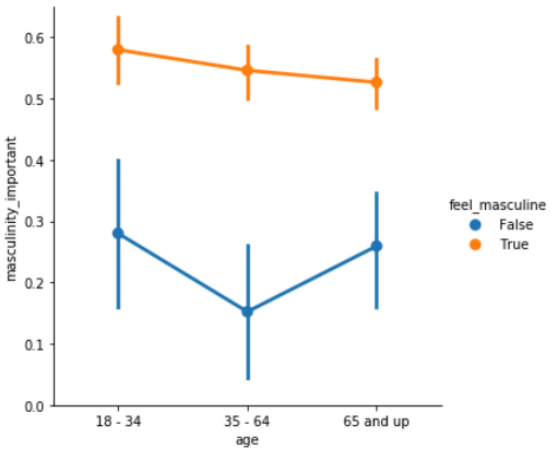
Point plots VS. Line Plots:

- **Line plots** are relational plots where both the x-axis and the y-axis are quantitative, while the **point plot** shows the mean of a quantitative variable across different categories (They do not need to be rational).

Point plots VS. Bar Plots:

- Both point plot and bar plot show the mean of a quantitative variable across different categories, and both show the confidence intervals, we choose one of them based on the data we want to plot.
- *Example:* using the masculinity dataset, we want to display the answers of different age groups to this question: "Is it important to feel like others see you masculine?":





- See that the point plot figure is easier to compare the mean for groups when they are stacked above each other.

```
sns.catplot(x="age", y="masculinity_importance", data=masculine, kind="point")
```

= If we want to remove the lines between the points, set the **join** parameter to False.

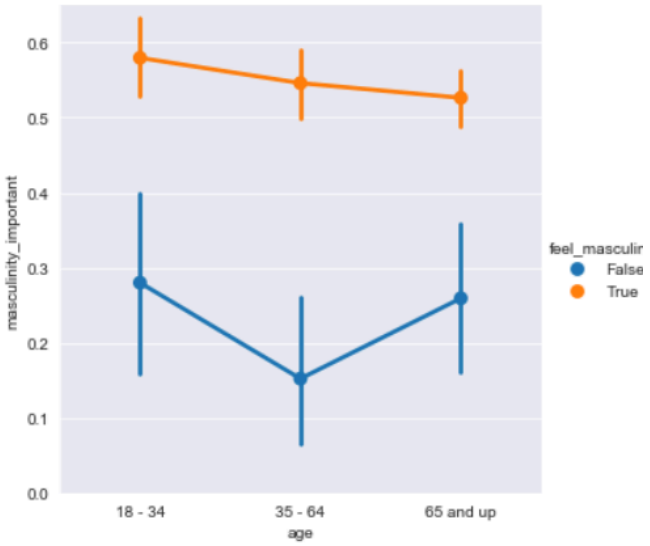
= To calculate the median instead of the mean, use the estimator argument to median.

Customizing Seaborn Plots

Changing figure style

Background and axes:

- Preset Options: white, dark, white grid, dark grid, and ticks.
- sns.set_style() is called to apply those styles.




```
sns.set_style("darkgrid")
sns.catplot(x="age", y="masculinity_importance", data=masculine, kind="p
oint")
```

Changing the main colors of the figure

- `sns.set_palette()` is used to change the colors of the plots.
- There are many existing palettes in Seaborn such as **“sequential palettes”** which contain each color in saturation from darker to lighter.
- We can customize it and create our own color palette this way:

```
my_palette=["black", "grey", "yellow"] # hex is also fine
sns.set_palette(my_palette)
```

Scaling:

- `sns.set_context()`: to change the scale.
- Options from smaller to larger: paper, notebook, talk, and poster. (`default is paper`)

Titles and labels

- There are two types of objects in Seaborn: FacetGrid and AxesSubplot.

FacetGrid	Support subplots	relplot(), catplot()
AxesSubplot	Only support single plot	scatterplot(), countplot(), etc.

= You can print the type of the plot after assigning it to a variable, then using type function.

Adding Title to Facet grid

```
g= sns.catplot(x="Region", y="Birthrate", data=gdp, kind="box")
g.fig.suptitle("New Title", y=0.5) # y is optional to control the title height
```

- If we have more than one figure (sub-plots), we use **set_titles,** and pass the column of the categories to it in curly brackets:

```
g= sns.catplot(x="Region", y="Birthrate", data=gdp, kind="box")
g.fig.suptitle("New Title", y=0.5)
g.set_title("This is {col_name}")
```

Adding Title to Axes Subplot

```
g= sns.boxplot(x="Region", y="Birthrate", data=gdp])
g.set_title("New Title", y=0.5) # y is optional to control the title height
```

Adding Axis labels

```
g.set(xlabel="X", ylabel="Y")
```

Rotating tick labels

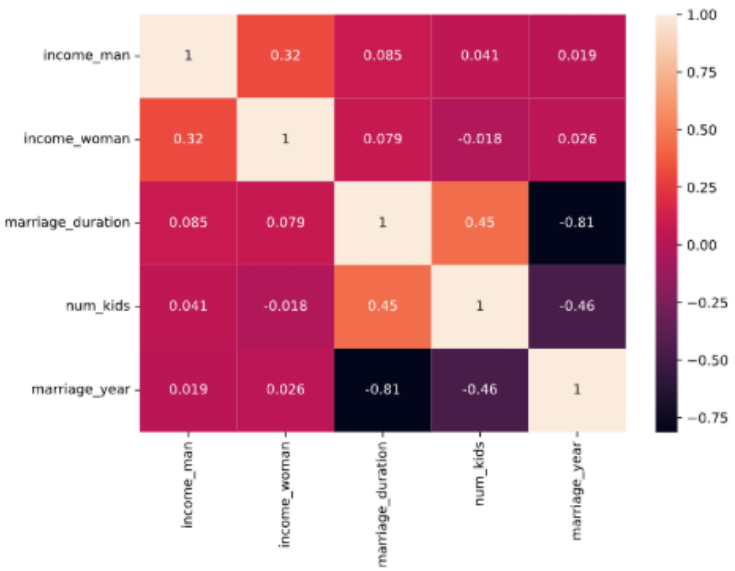
- There is no a function to do that in Seaborn, instead, we use matplotlib as we used to:

```
plt.xticks(rotation= 90)
```

Heatmap

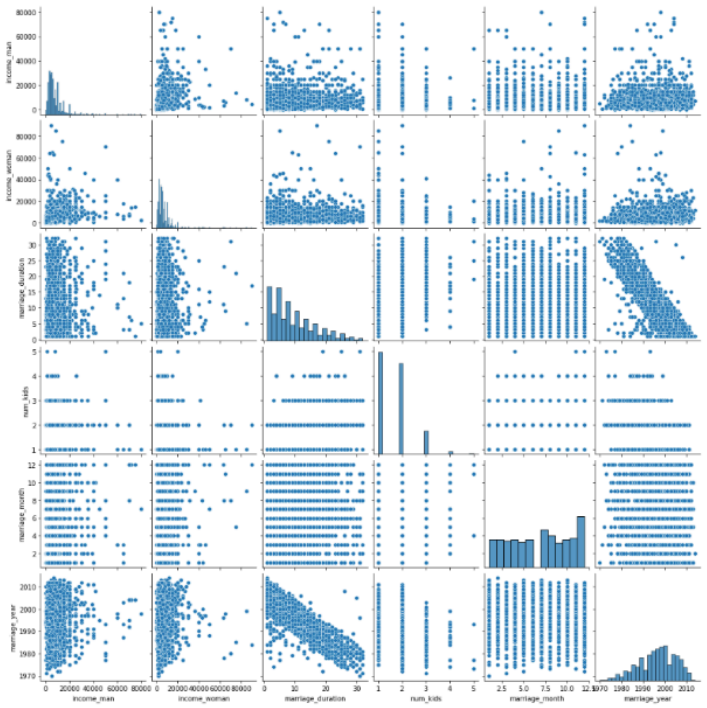
Visualizing correlation using heatmaps

```
sns.heatmap(divorce.corr(), annot=True)
plt.show()
```



Pairplots

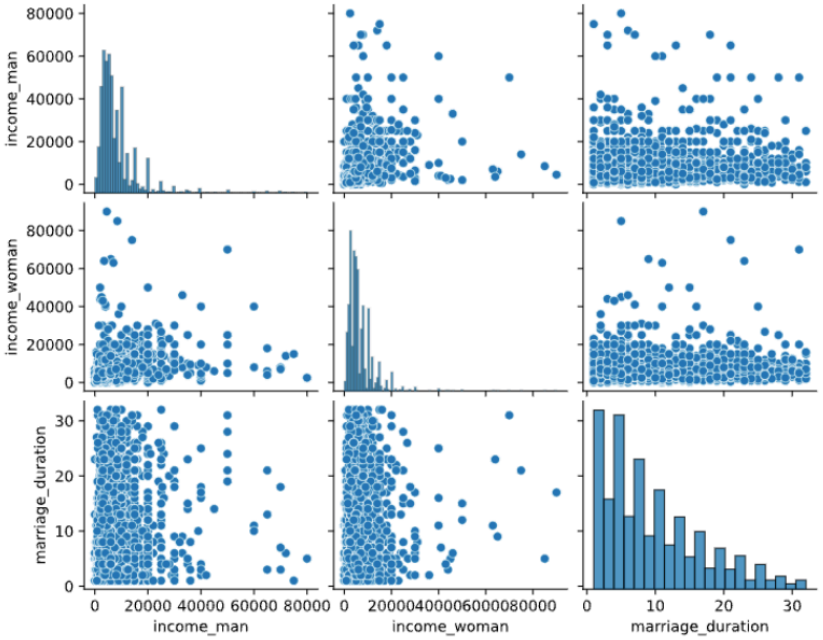
- Pair plots visualize every relationship between numeric variables in one, it is useful to take a quick look into the relationships and correlations between variables:




```
sns.pairplot(data = divorce)
plt.show()
```

- This method does not really fit the large datasets as it will be difficult to read, so we can specify the columns that we want to visualize the relationships between them:

```
sns.pairplot(data = divorce, vars=['income_man', 'income_women', 'marrig
e_duration'])
plt.show()
```


income_man
income_woman
marriage_duration
0
20000
40000
60000
80000
0
10
20
30