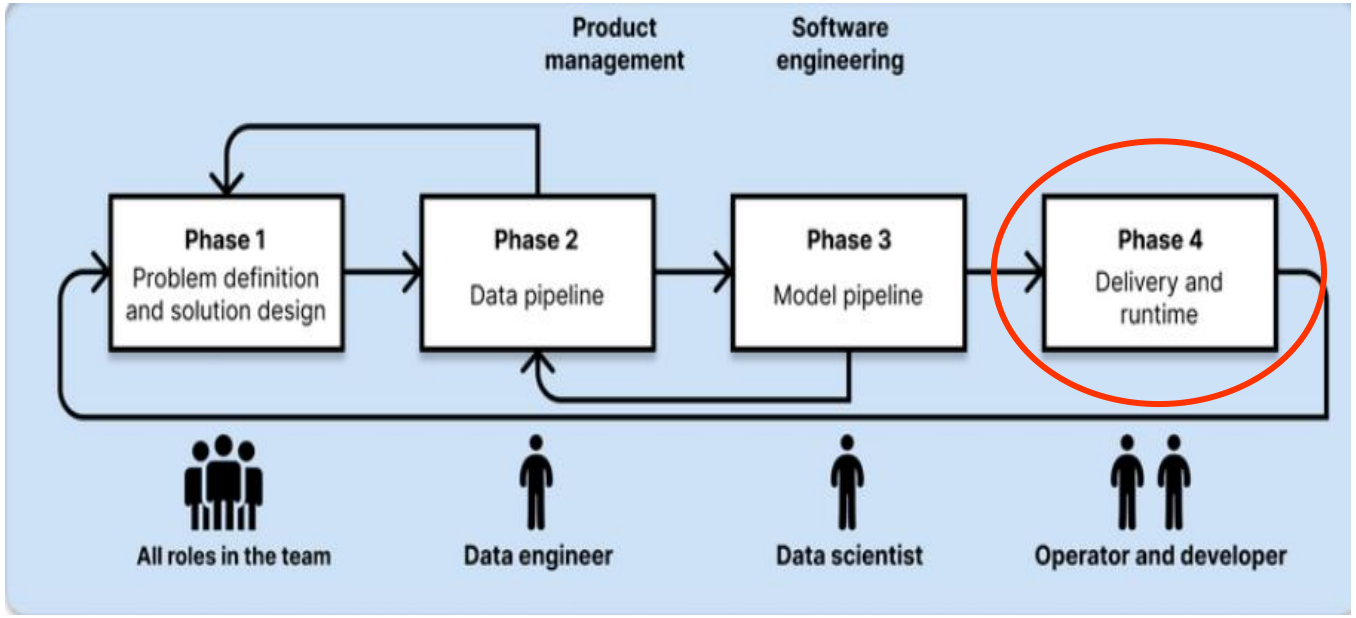
Python for Data Sciences

Data Visualization: Matplotlib, Seaborn

- Armando Camerlingo

IR0000032 – ITINERIS, Italian Integrated Environmental Research Infrastructures System
(D.D. n. 130/2022 - CUP B53C22002150006) Funded by EU - Next Generation EU PNRR-Mission 4 “Education and Research” - Component 2: “From research to business” - Investment 3.1: “Fund for the realisation of an integrated system of research and innovation infrastructures”

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani



- Where do we want to run our model?
- CPU or GPU?
- Can parallelize some parts of our code?
- Plots and metrics about our model

Data Visualization

ITINERIS

Matplotlib

Seaborn

Matplotlib

ITINERIS

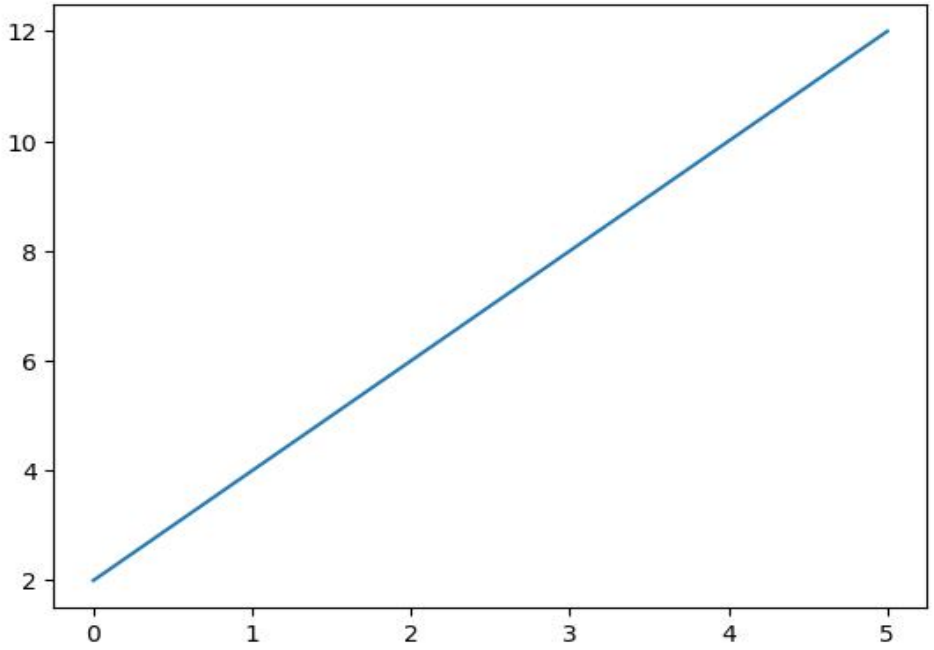
Matplotlib is an open source library based on Numpy to visualize data with multiple choices of graphs and customizations.

Matplotlib: lineplot

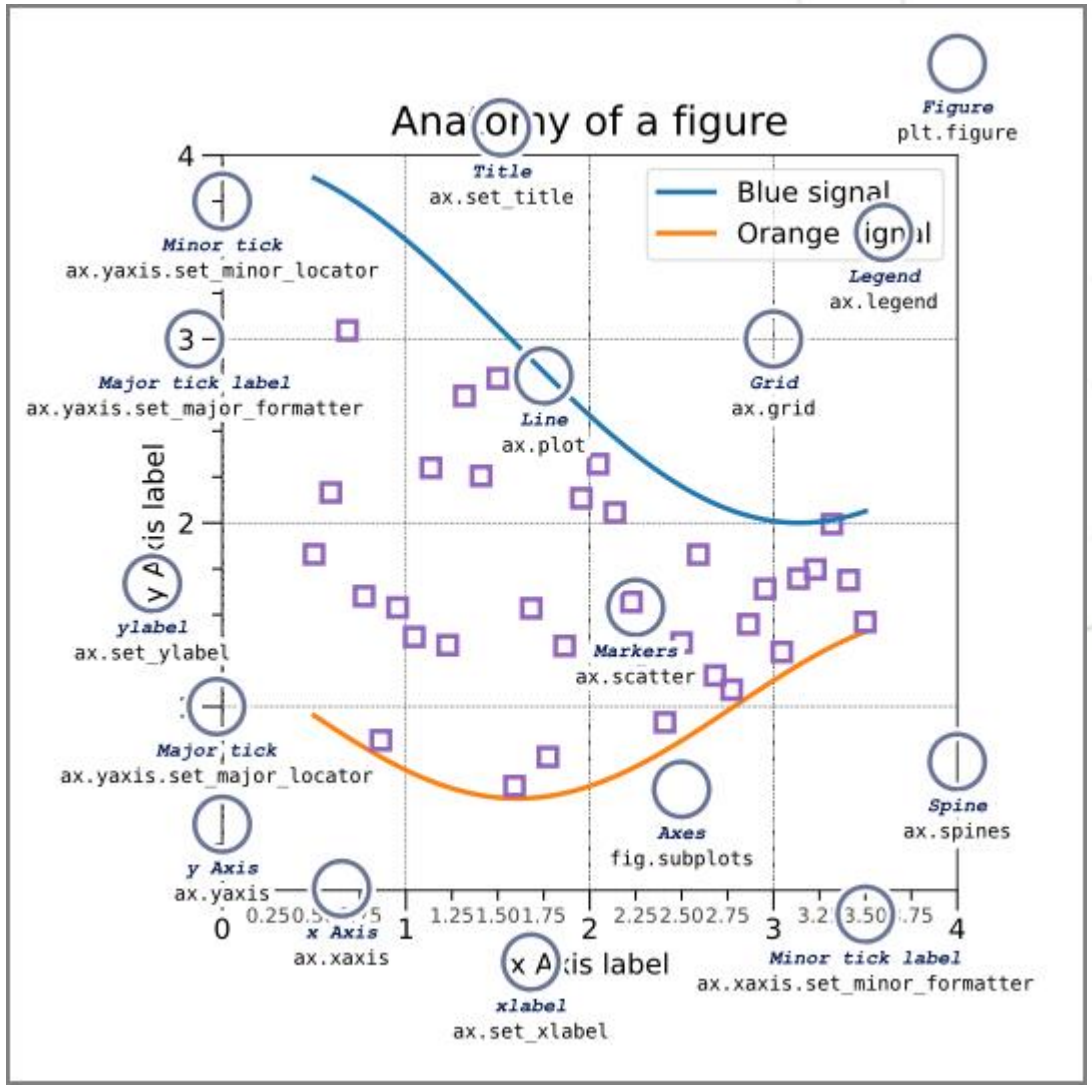
```
import matplotlib.pyplot as plt
import numpy as np
```

```
X = [0, 1, 2, 3, 4, 5]
Y = [2, 4, 6, 8, 10, 12]
```

```
plt.plot(X,Y)
```



Matplotlib: elements of a figure

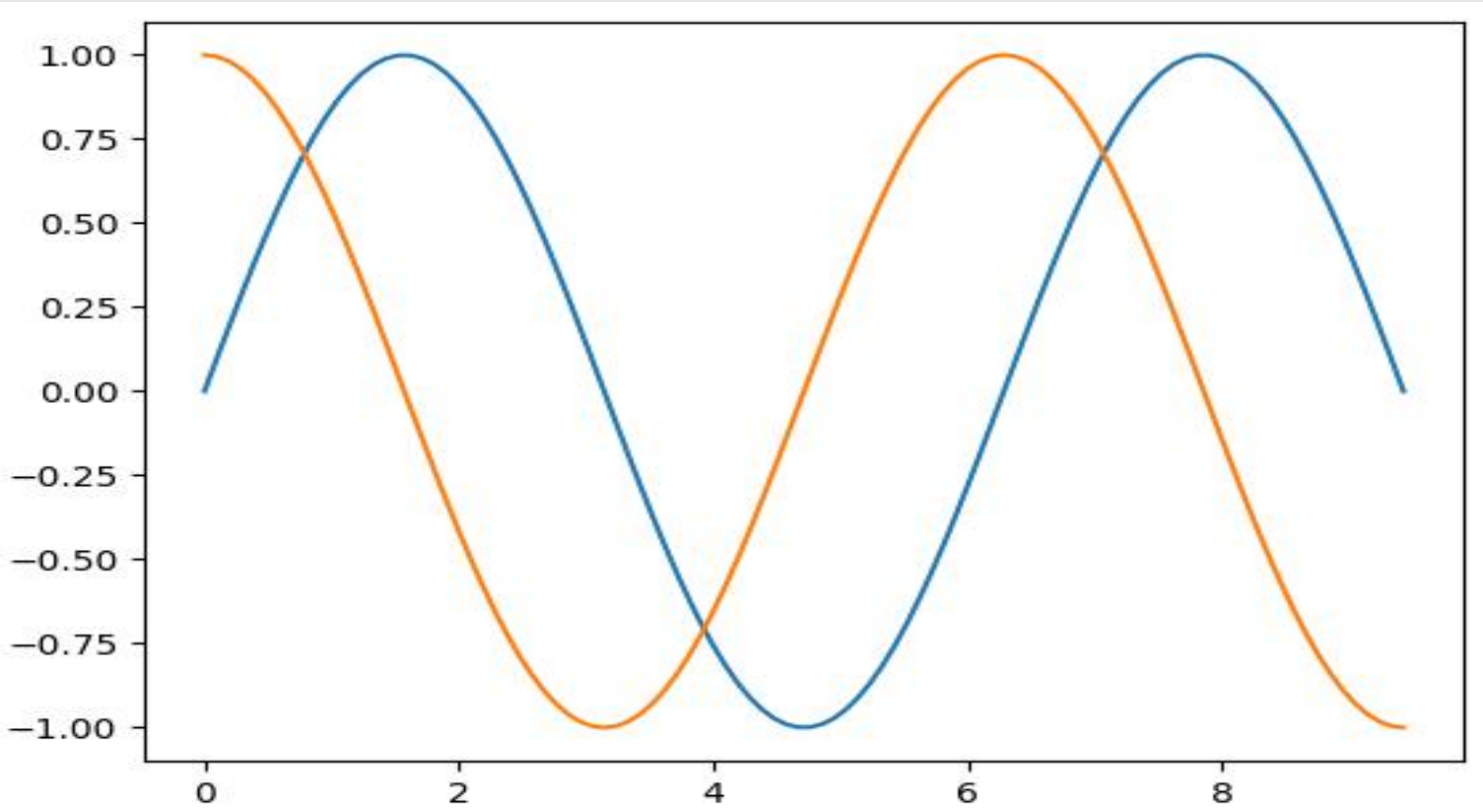


Matplotlib: multiple plot in 1 figure

```
X = np.linspace(0, 3*np.pi, 100)
Y1, Y2, Y3, Y4 = np.sin(X), np.sin(X+np.pi/2), 2*np.sin(X), np.sin(2*X)
```

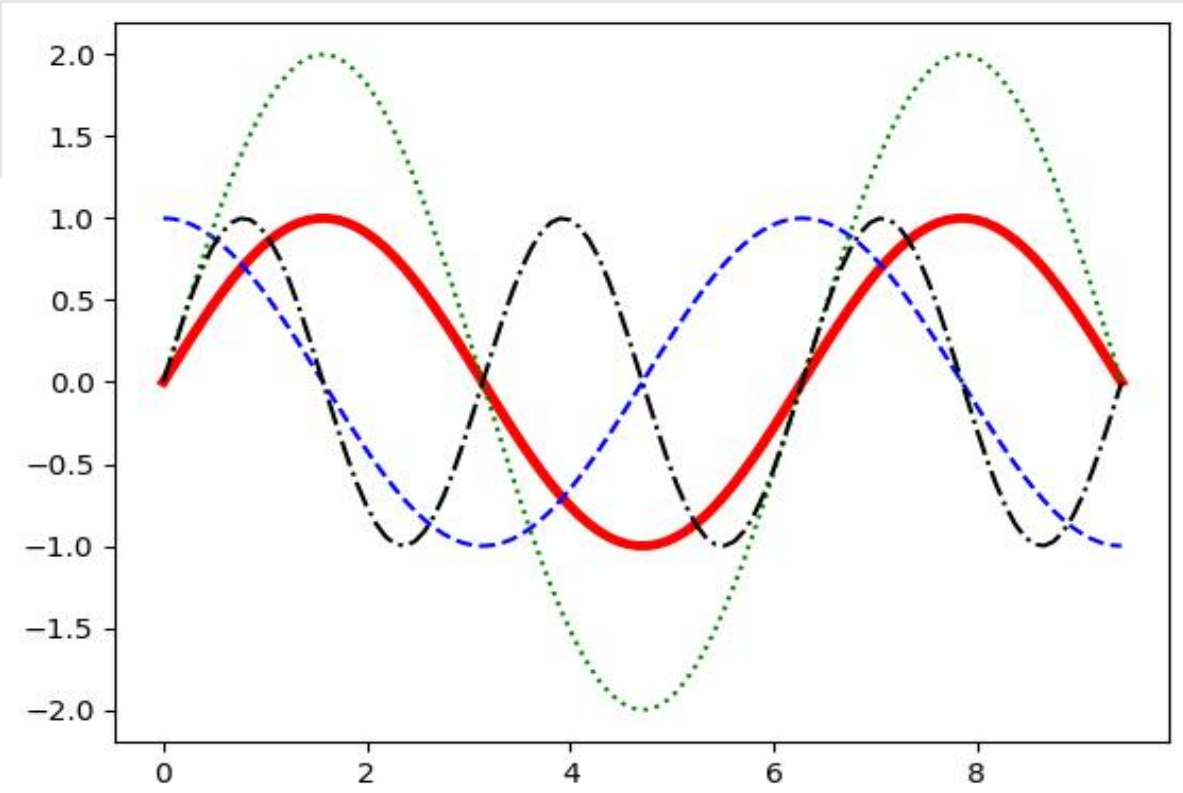
```
plt.figure()

plt.plot(X, Y1)
plt.plot(X, Y2)

plt.show()
```

If we don't call a new figure, all plots will be drawn in the same axes.



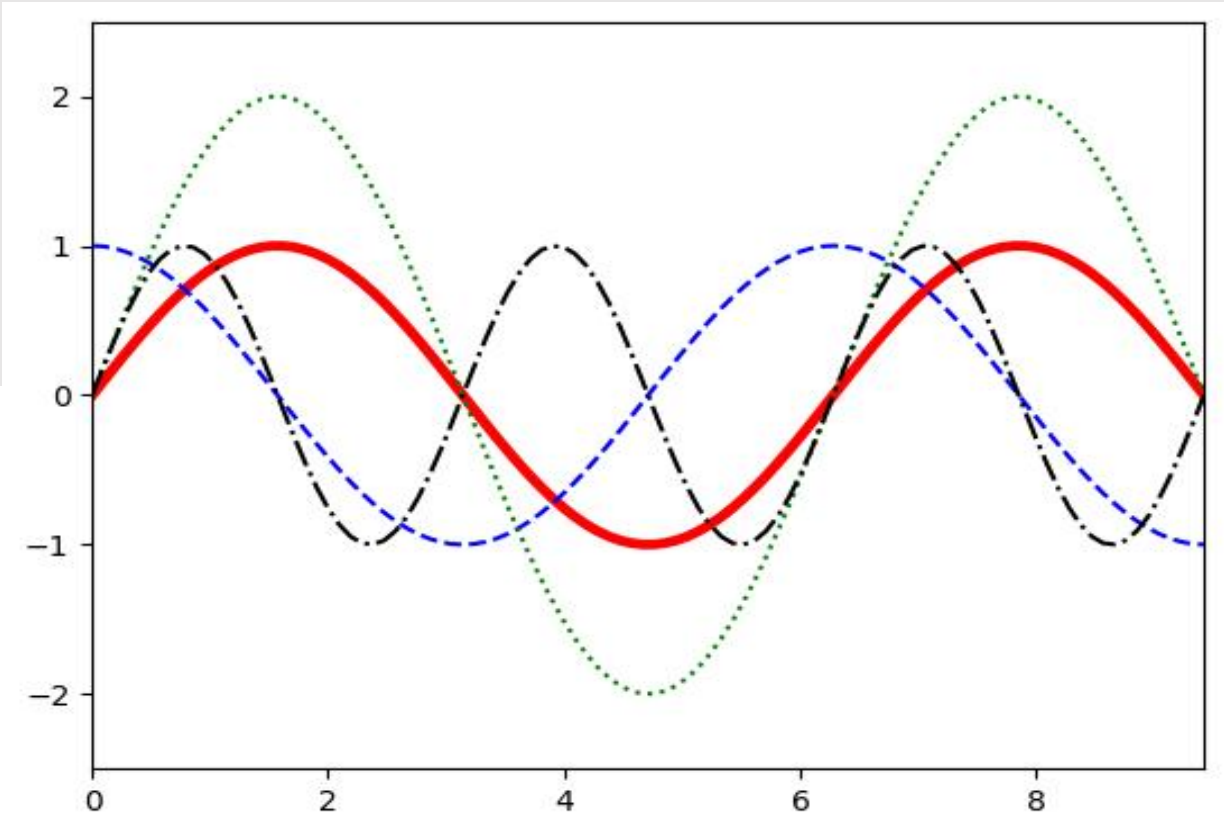
Matplotlib: line attributes

```
plt.figure()

plt.plot(X, Y1, color = "red", linewidth = 3.5) #  linewidth default is 3
plt.plot(X, Y2, color = "blue", linestyle = "--")
plt.plot(X, Y3, color = "green", linestyle = ":")
plt.plot(X, Y4, color = "black", linestyle = "-.")


plt.show()
```



Matplotlib: figuresize and axes limits

```
plt.figure(figsize = (8,5)) # Define the size of your figure

plt.plot(X, Y1, color = "red", linewidth = 3.5,)
plt.plot(X, Y2, color = "blue", linestyle = "--",)
plt.plot(X, Y3, color = "green", linestyle = ":",)
plt.plot(X, Y4, color = "black", linestyle = "-.",)

#set x and y axis limits
plt.xlim(0, 3*np.pi)
plt.ylim(-2.5, 2.5)

plt.show()
```

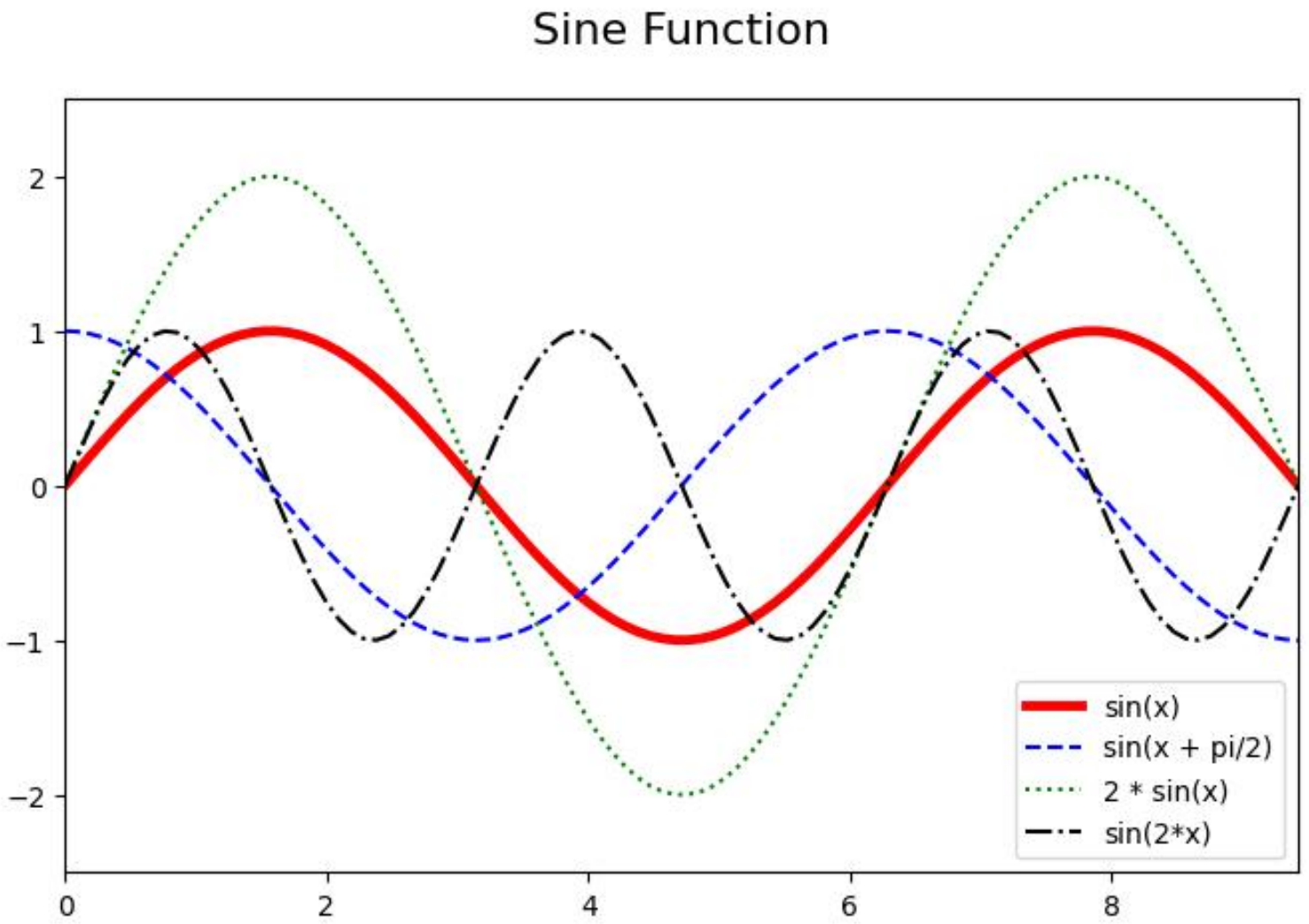


Matplotlib: titles and legends

ITINERIS

```
plt.figure(figsize = (8,5)) # Define the size of your figure

plt.plot(X, Y1, color = "red", linewidth = 3.5, label = "sin(x)")
plt.plot(X, Y2, color = "blue", linestyle = "--", label = "sin(x + pi/2)")
plt.plot(X, Y3, color = "green", linestyle = ":", label = "2 * sin(x)")
plt.plot(X, Y4, color = "black", linestyle = "-.", label = "sin(2*x)")

#set x and y axis limits
plt.xlim(0, 3*np.pi)
plt.ylim(-2.5, 2.5)

plt.title("Sine Function", fontsize = 16, pad = 20)

plt.legend(loc="lower right")

plt.show()
```

Matplotlib: titles and legends

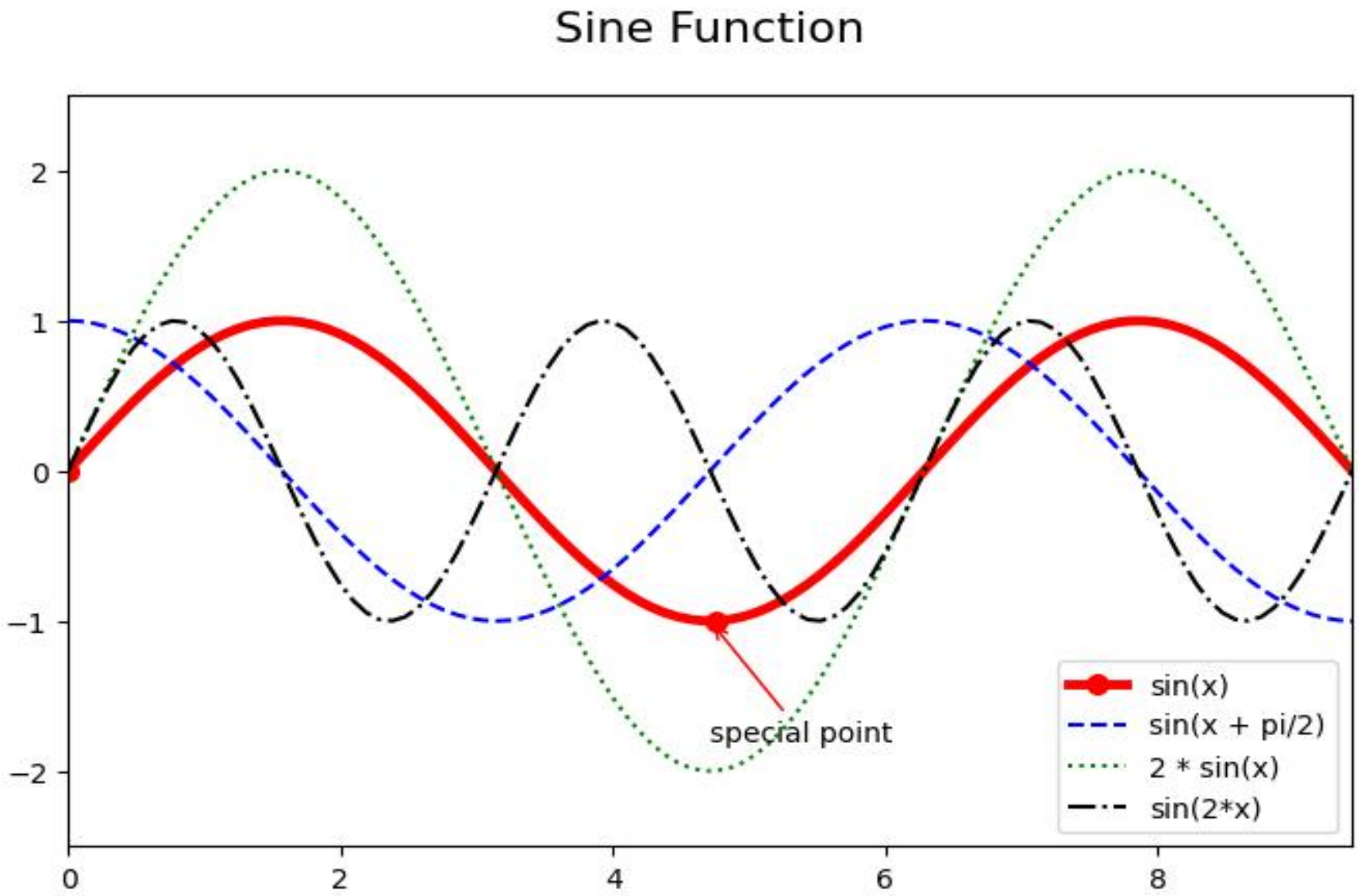


Matplotlib: annotations

ITINERIS

```
plt.figure(figsize = (8,5))

# Let's add a special marker and annotate it
plt.plot(X, Y1, color = "red", linewidth = 3.5, label = "sin(x)", marker = "o",
markersize = 7, markevery=50)
plt.plot(X, Y2, color = "blue", linestyle = "--", label = "sin(x + pi/2)")
plt.plot(X, Y3, color = "green", linestyle = ":", label = "2 * sin(x)")
plt.plot(X, Y4, color = "black", linestyle = "-.", label = "sin(2*x)")

plt.xlim(0, 3*np.pi)
plt.ylim(-2.5, 2.5)

# Here we add the annotation
plt.annotate("special point", xy=(1.5*np.pi, -1), xytext=(1.5*np.pi, -1.8),
arrowprops = {"arrowstyle":"->", "color":"red"})
plt.title("Sine Function", fontsize = 16, pad = 20)
plt.legend(loc="lower right")

plt.show()
```

Matplotlib: Annotations

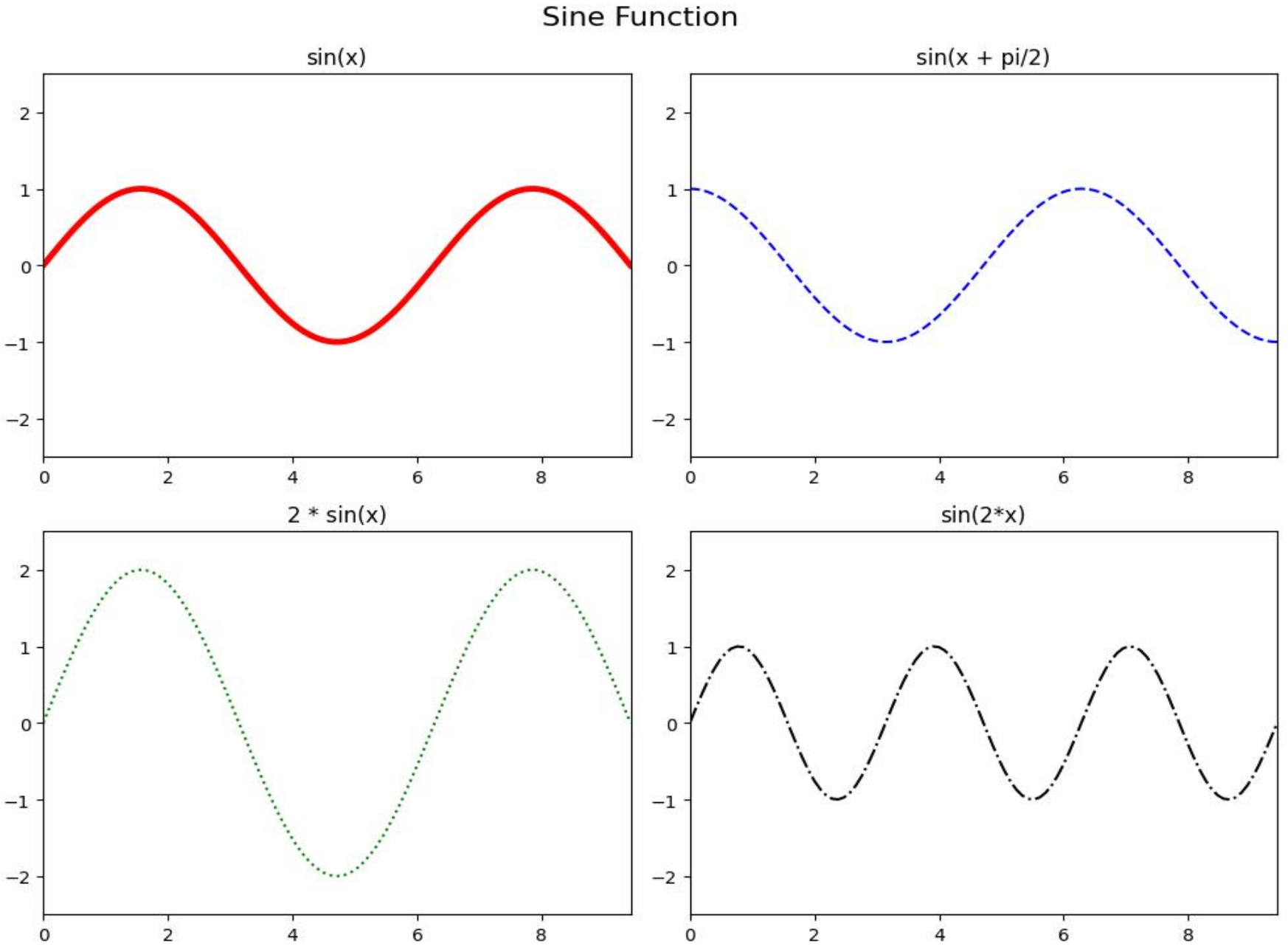


Matplotlib: subplots

ITINERIS

```
plt.figure(figsize = (10,8))
plt.suptitle("Sine Function", fontsize = 16)

ax_1 = plt.subplot(2,2,1, xlim=(0, 3*np.pi), ylim=(-2.5, 2.5))
plt.plot(X, Y1, color = "red", linewidth = 3.5, label = "sin(x)")
plt.title("sin(x)")

ax_2 = plt.subplot(2,2,2, sharex=ax_1, sharey=ax_1)
plt.plot(X, Y2, color = "blue", linestyle = "--", label = "sin(x + pi/2)")
plt.title("sin(x + pi/2)")

ax_3 = plt.subplot(2,2,3,  sharex=ax_1, sharey=ax_1)
plt.plot(X, Y3, color = "green", linestyle = ":", label = "2 * sin(x)")
plt.title("2 * sin(x)")

ax_4 = plt.subplot(2,2,4,  sharex=ax_1, sharey=ax_1)
plt.plot(X, Y4, color = "black", linestyle = "-.", label = "sin(2*x)")
plt.title("sin(2*x)")

plt.tight_layout()
plt.show()
```

Matplotlib: subplots



Matplotlib: saving plots


```
plt.figure(figsize = (10,8))
plt.suptitle("Sine Function", fontsize = 16)

ax_1 = plt.subplot(2,2,1, xlim=(0, 3*np.pi), ylim=(-2.5, 2.5))
#RGB code
plt.plot(X, Y1, color = (184/255,134/255,11/255), linewidth = 3, label = "sin(x)")
plt.title("sin(x)")
# Let's set ticks as multiples of pi
ticks = np.arange(0,3.5*np.pi,np.pi/2)
tick_labels = ["0", "0.5*pi", "pi", "1.5*pi", "2*pi", "2.5*pi", "3*pi"]
plt.xticks(ticks, labels=tick_labels)
ax_1.set_facecolor((0,0,0)) # Backgroundcolour of each axes

ax_2 = plt.subplot(2,2,2, sharex=ax_1, sharey=ax_1)
plt.plot(X, Y2, color = (1,1,1), linestyle = "--", linewidth = 3, label = "sin(x +
pi/2)")
plt.title("sin(x + pi/2)")
ax_2.set_facecolor((0,0,0))
```

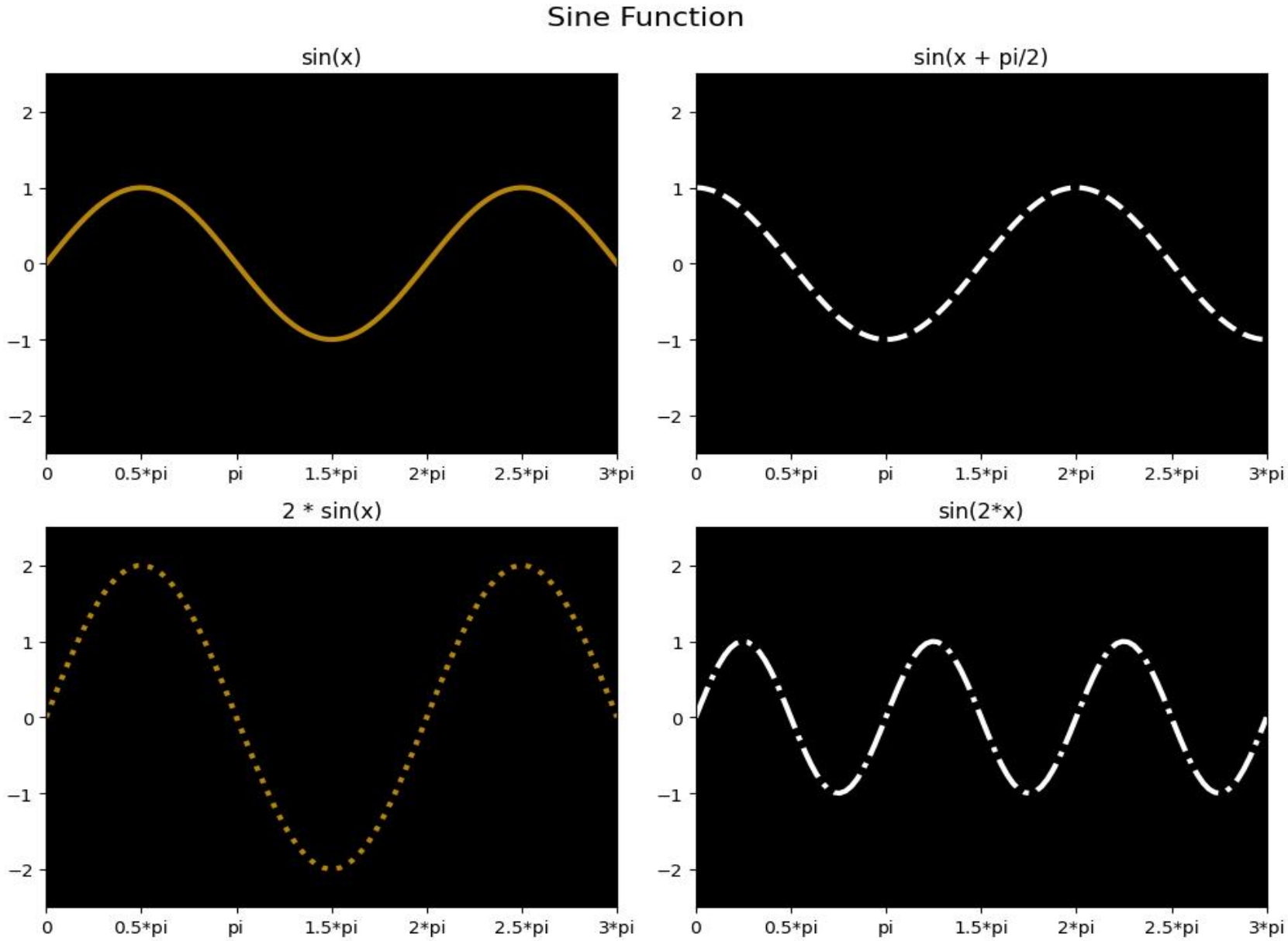
• • •

Matplotlib: saving plots

. . .

```
ax_3 = plt.subplot(2,2,3,  sharex=ax_1, sharey=ax_1)
#Hex code
plt.plot(X, Y3, color = "#B8860B", linestyle = ":", linewidth = 3, label = "2 *
sin(x)")

plt.title("2 * sin(x)")
ax_3.set_facecolor("#000000")

ax_4 = plt.subplot(2,2,4,  sharex=ax_1, sharey=ax_1)
plt.plot(X, Y4, color = "white", linestyle = "-.", linewidth = 3, label =
"sin(2*x)") #Colour name
plt.title("sin(2*x)")
ax_4.set_facecolor("black")

plt.tight_layout() # Matplotlib makes sure your spaces are not to big and tidies our
plot a bit

plt.show()
```

Matplotlib: saving plots



Matplotlib: saving plots

```
plt.savefig("firstmatplotlibfig.png", dpi=150) # Let's save this figure as a png
```

```
<Figure size 640x480 with 0 Axes>
```

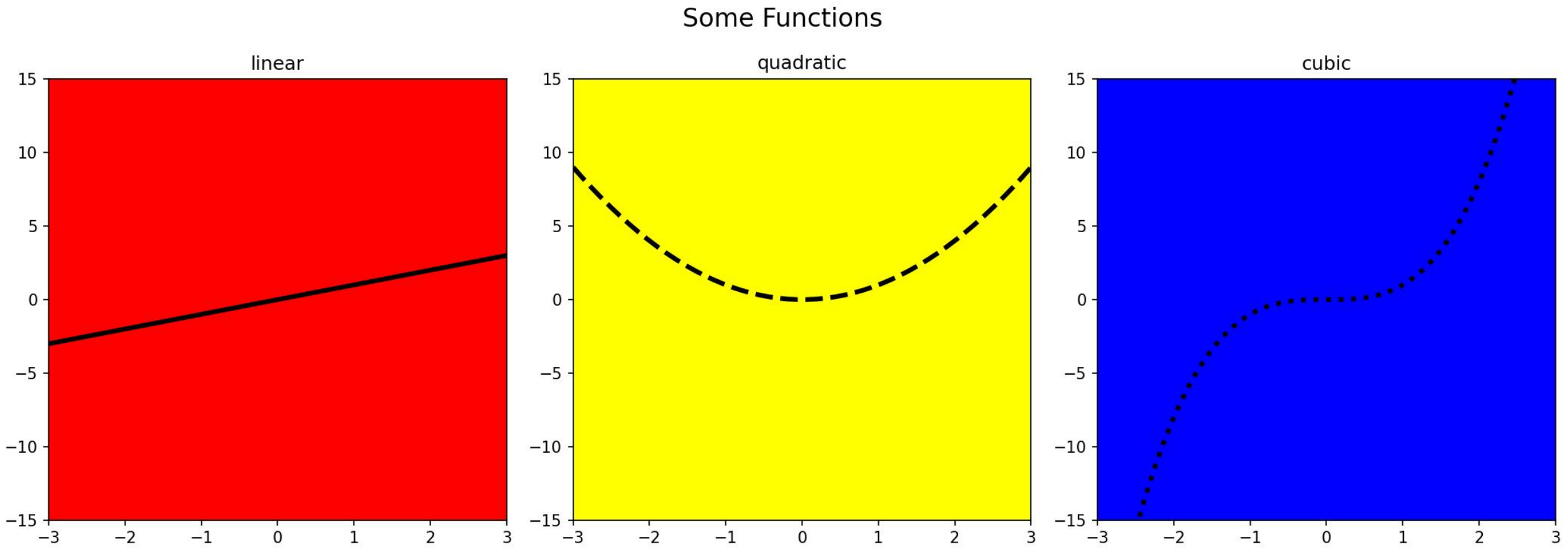
```
!ls
```

```
firstmatplotlibfig.png sample_data
```

Exercise 1

Given the data, replicate this plot:

```
X = np.linspace(-3, 3)
F1, F2, F3 = X, X**2, X**3
```

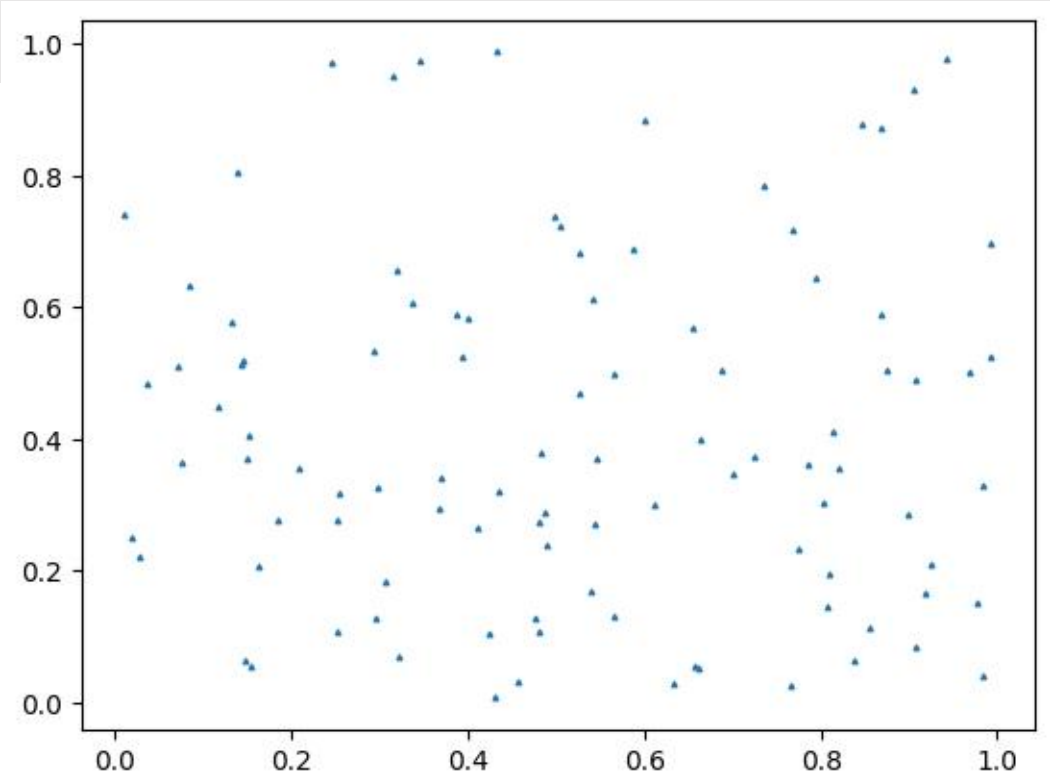


Solution

```
plt.figure(figsize = (14,5))
plt.suptitle("Some Functions", fontsize = 16)

ax_1 = plt.subplot(1,3,1, xlim=(-3, 3), ylim=(-15, 15))
plt.plot(X, F1, color = "black", linewidth = 3)
plt.title("linear")
ax_1.set_facecolor("red")

ax_2 = plt.subplot(1,3,2, sharex=ax_1, sharey=ax_1)
plt.plot(X, F2, color = "black", linestyle = "--", linewidth = 3)
plt.title("quadratic")
ax_2.set_facecolor("yellow")

ax_3 = plt.subplot(1,3,3,  sharex=ax_1, sharey=ax_1)
plt.plot(X, F3, color = "black", linestyle = ":", linewidth = 3)
plt.title("cubic")
ax_3.set_facecolor("blue")


plt.tight_layout()
plt.show()
```

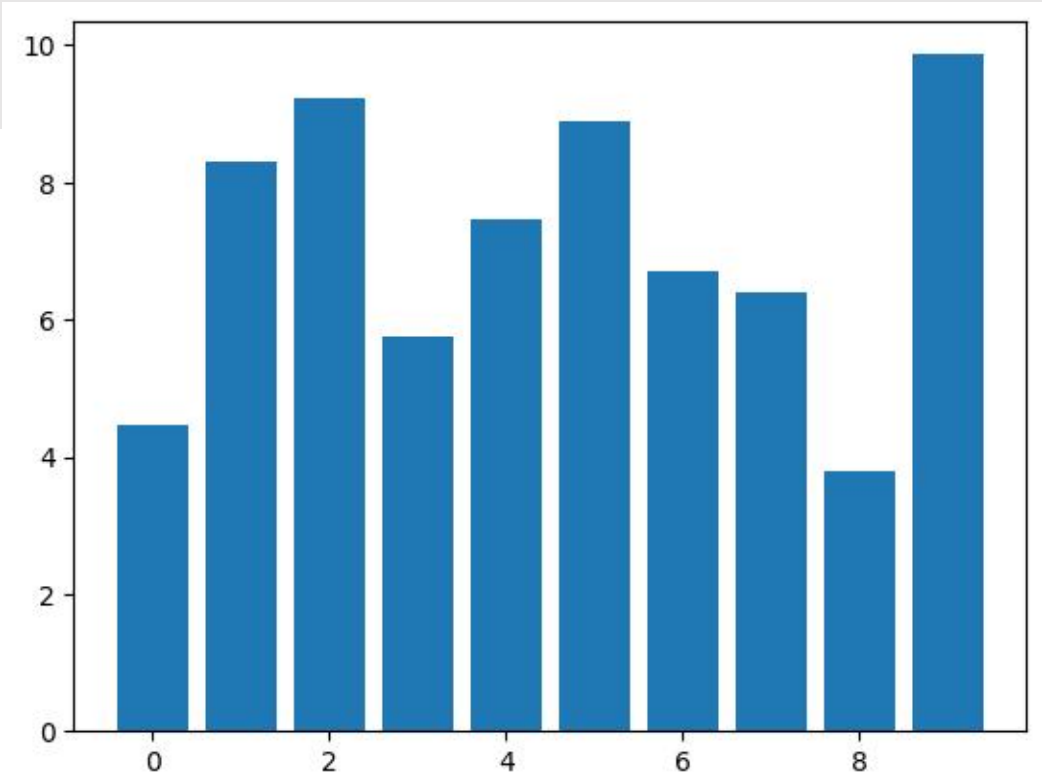
Matplotlib: scatterplots

```
A = np.random.uniform(0 ,1, 100)
B = np.random.uniform(0, 1, 100)

plt.figure()
plt.scatter(A, B, s = 3, marker = "^")
plt.show()
```



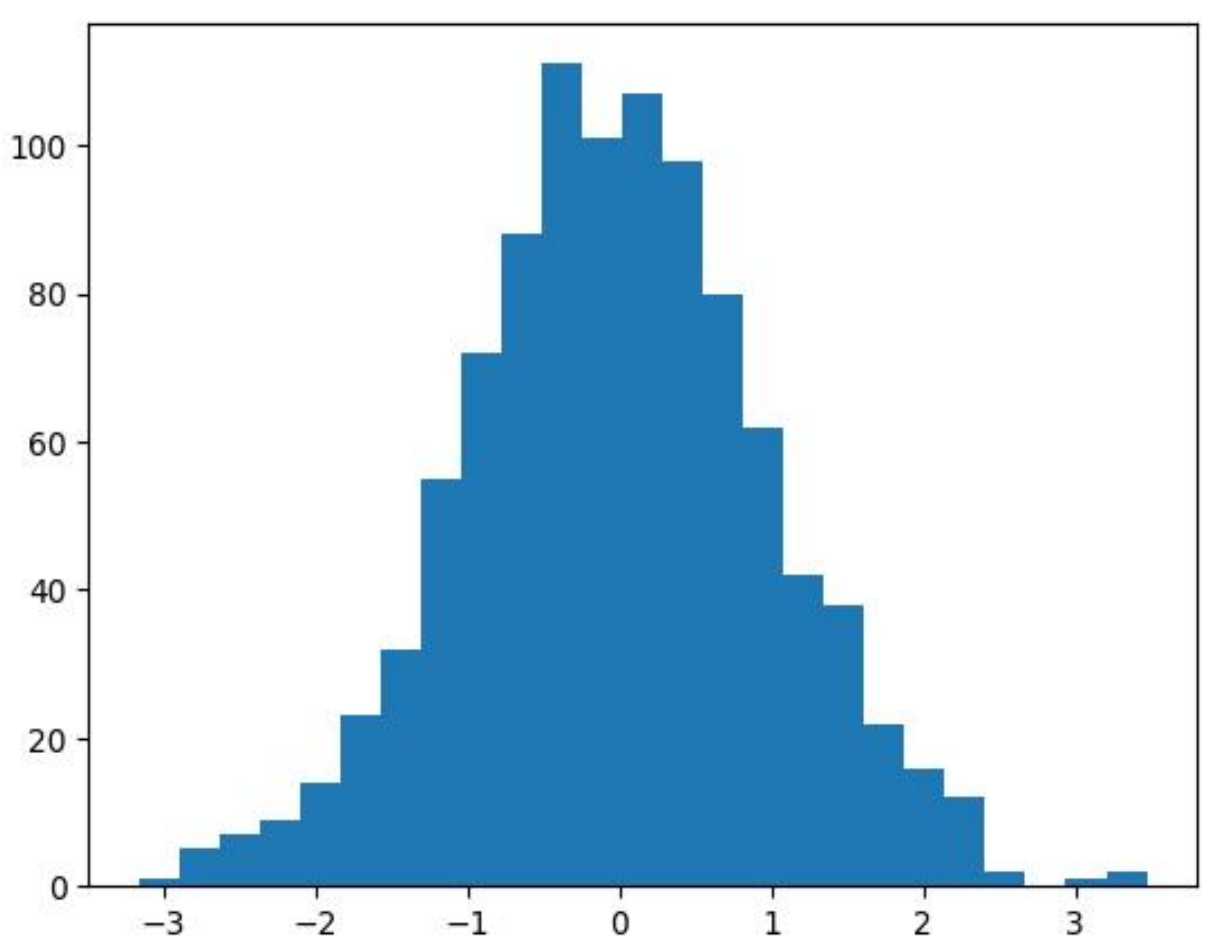
Matplotlib: bar charts

```
C = np.arange(10)
D = np.random.uniform(1, 10, 10)

plt.figure()
plt.bar(C, D)

plt.show()
```



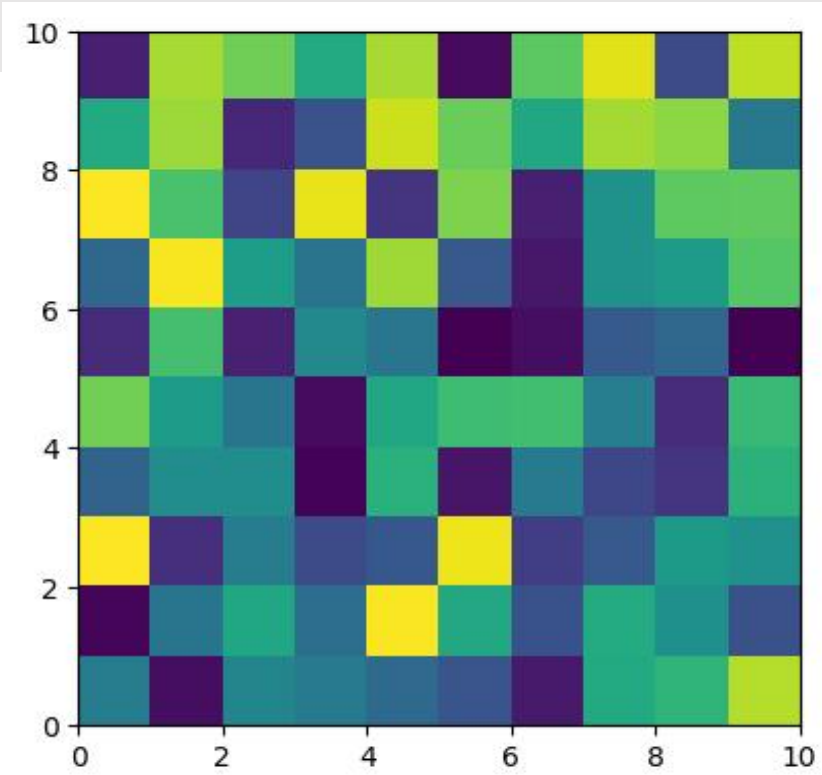
Matplotlib: histograms

```
E = np.random.normal(0, 1, 1000)
plt.figure()
plt.hist(E, bins=25)
plt.show()
```



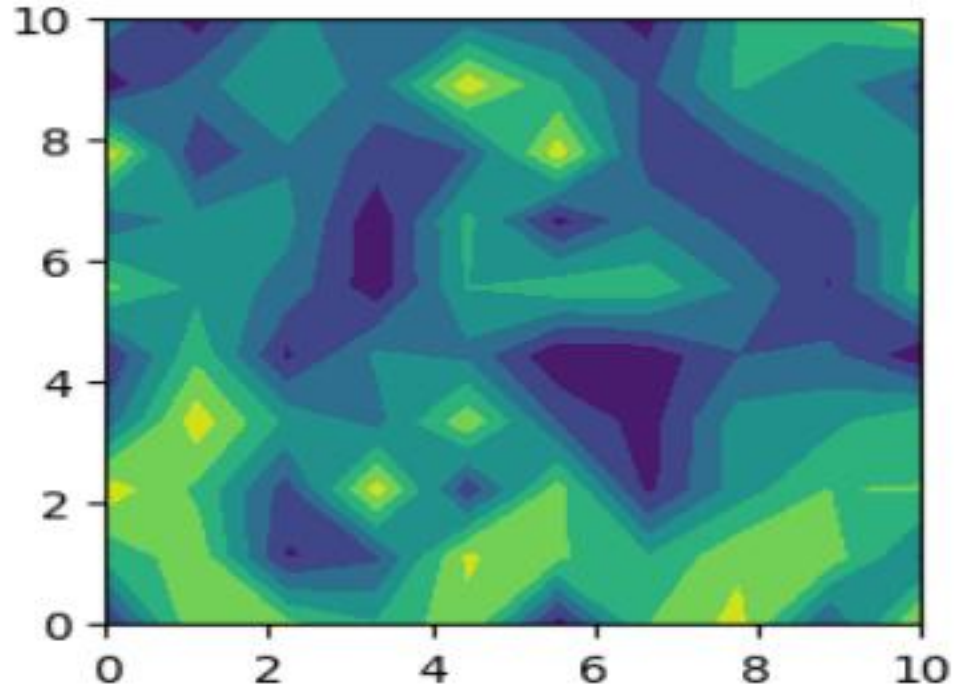
Matplotlib: heatmap

```
F = np.random.uniform(0, 1, (10, 10))

plt.figure(figsize=(10,5))

ax1 = plt.subplot(1,2,1)
plt.imshow(F, extent=[0, len(F), 0, len(F)])
ax1.set_aspect("equal", "box")
```



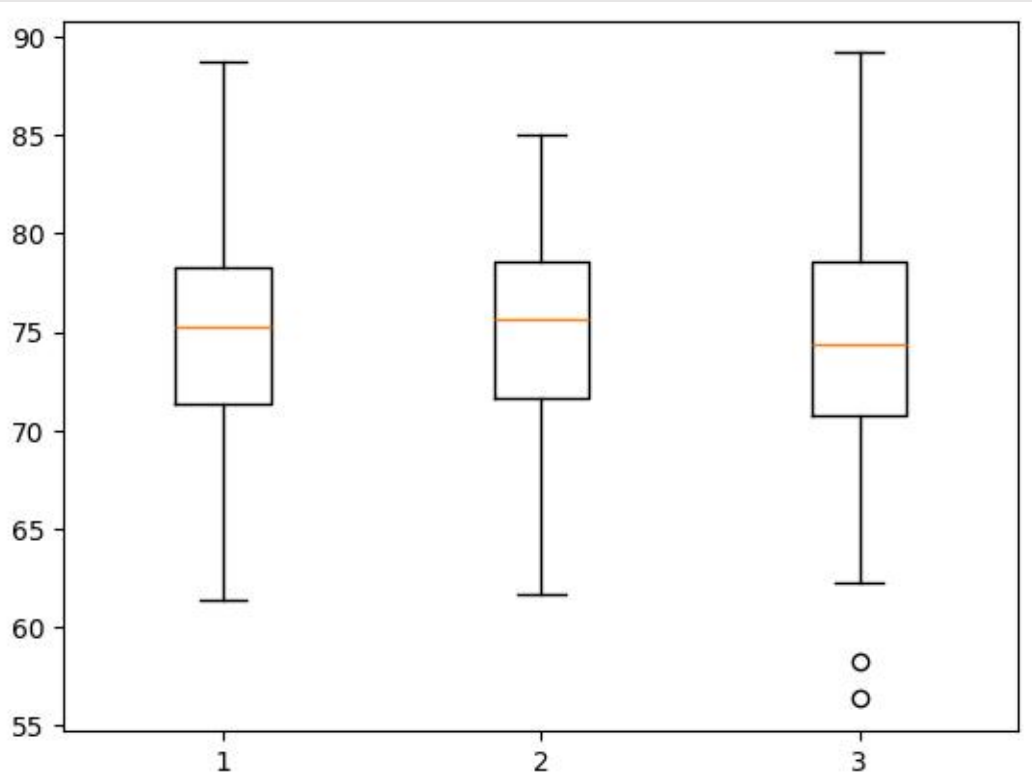
Matplotlib: Contour Plot

```
F = np.random.uniform(0, 1, (10, 10))

plt.figure(figsize=(10,5))

ax2 = plt.subplot(1,2,2, sharey=ax1)
plt.contourf(F, extent=[0, len(F), 0, len(F)])
ax2.set_aspect("equal", "box")
plt.show()
```



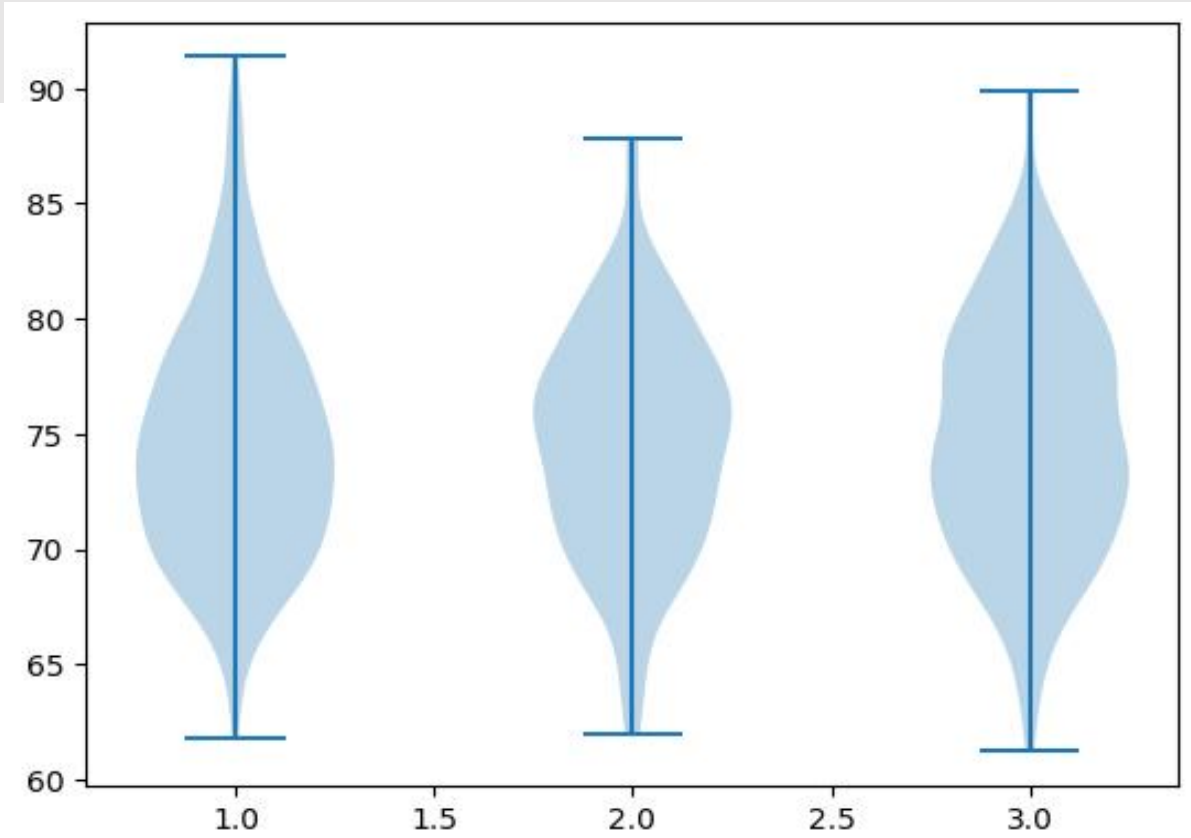
Matplotlib: box-plot

```
G = np.random.normal(75,5, (200, 3))

plt.figure()
plt.boxplot(G)
plt.show()
```



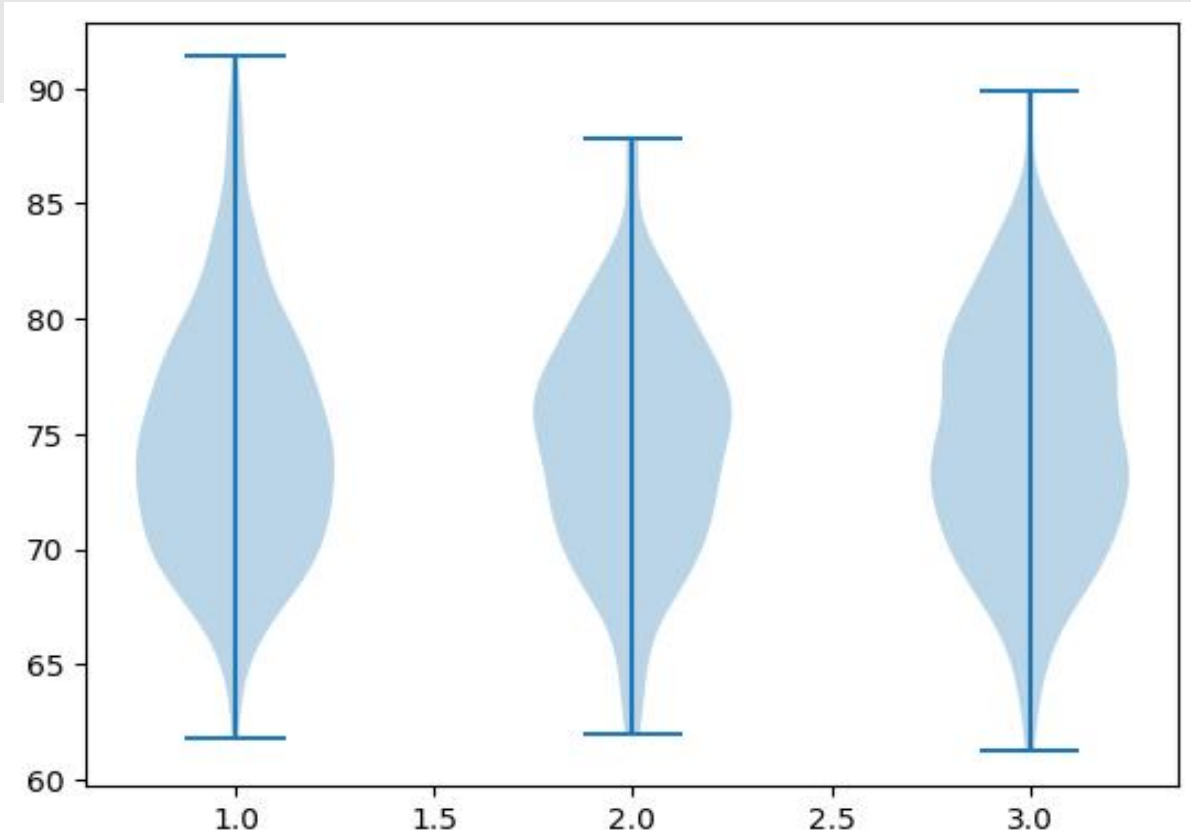
Matplotlib: violin-plot

```
G = np.random.normal(75,5, (200, 3))

plt.figure()
plt.violinplot(G)
plt.show()
```



Matplotlib: violin-plot

```
G = np.random.normal(75,5, (200, 3))

plt.figure()
plt.violinplot(G)
plt.show()
```

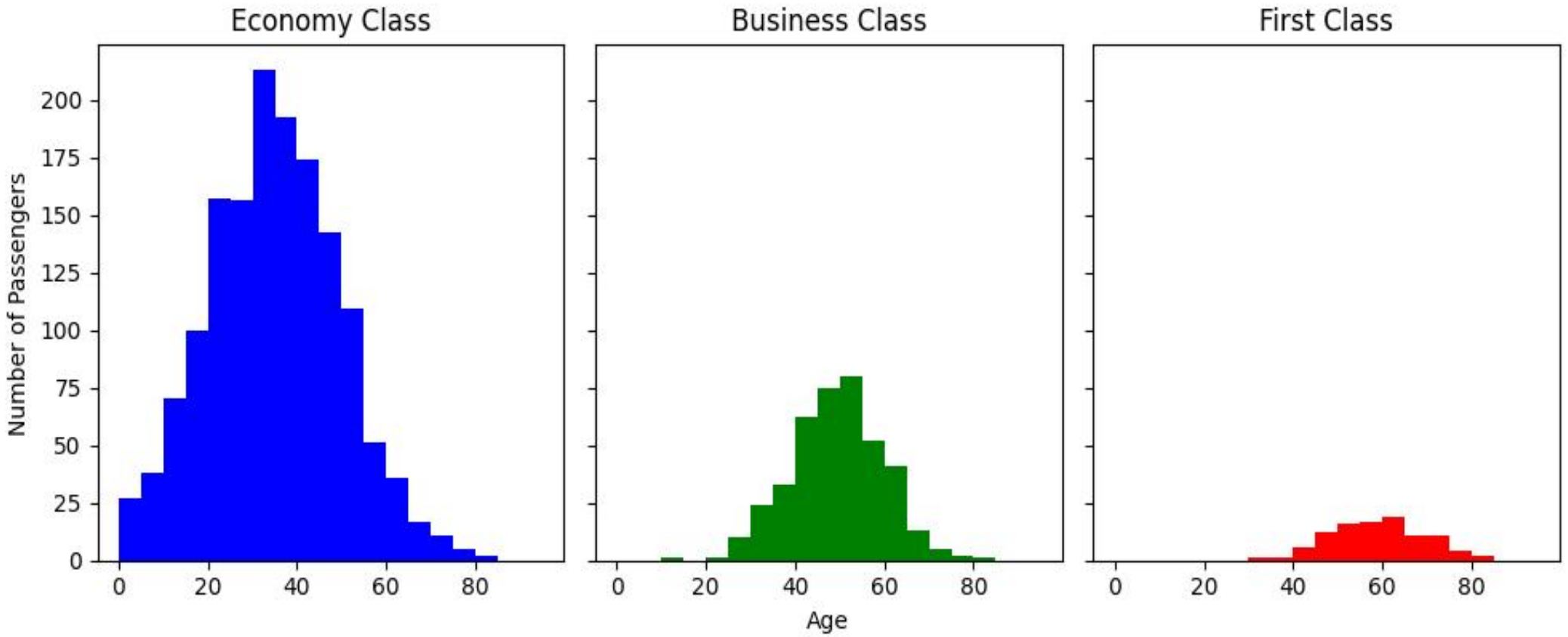


Matplotlib: object-oriented approach


```
First_Class = abs(np.random.normal(58, 9, 100))
Business_Class = abs(np.random.normal(50, 10, 400))
Economy_Class = abs(np.random.normal(35, 15, 1500))
Bins = np.arange(0,100,5)
```

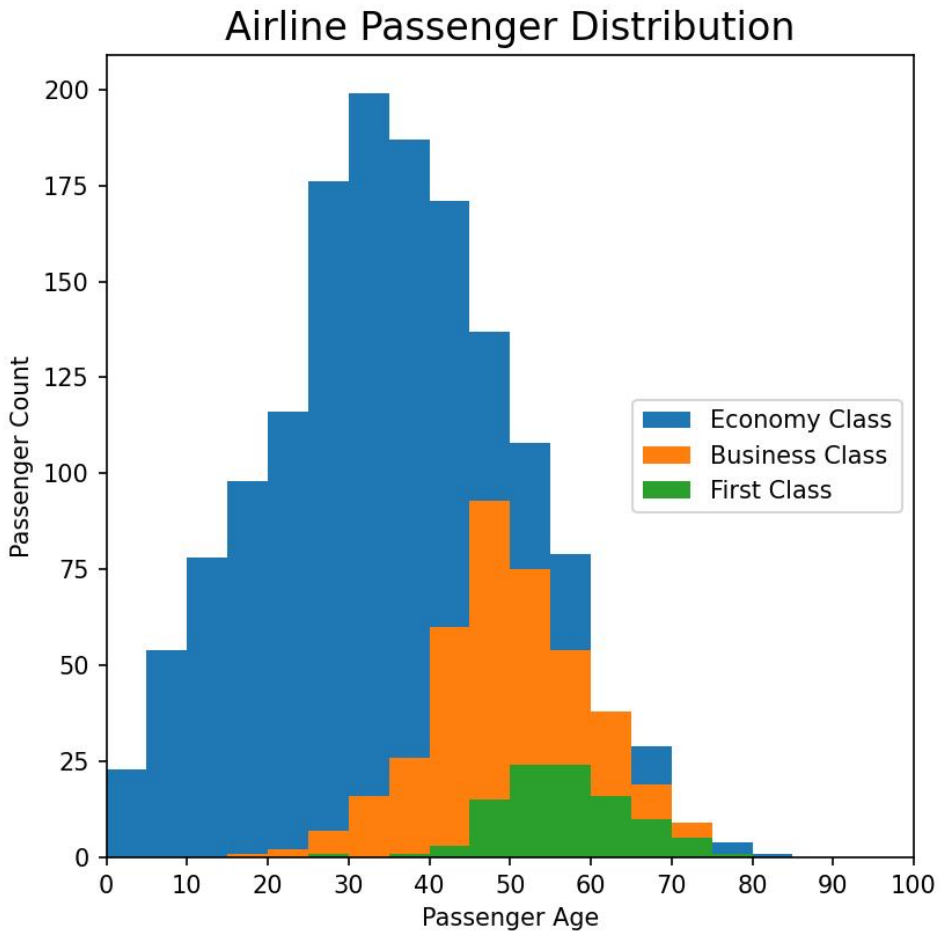
```
fig, axs = plt.subplots(1,3, figsize = (10,4), sharex=True, sharey=True)

axs[0].hist(Economy_Class, bins=Bins, color="blue")
axs[1].hist(Business_Class, bins=Bins, color="green")
axs[2].hist(First_Class, bins=Bins, color="red")

axs[0].set_title("Economy Class")
axs[1].set_title("Business Class")
axs[2].set_title("First Class")

axs[1].set_xlabel("Age")
axs[0].set_ylabel("Number of Passengers")

plt.tight_layout()
plt.show()
```

Matplotlib: object-oriented approach



Exercise 2

Given the data of the last example, replicate this plot:



Solution

```
plt.figure(figsize=(6,6))

plt.hist(Economy_Class, bins=Bins, label="Economy Class")
plt.hist(Business_Class, bins=Bins, label="Business Class")
plt.hist(First_Class, bins=Bins, label="First Class")

plt.xticks([0,10,20,30,40,50,60,70,80,90,100])
plt.xlim(0,100)
plt.xlabel("Passenger Age")
plt.ylabel("Passenger Count")

plt.title("Airline Passenger Distribution", fontsize=16)
plt.legend(loc="center right")

plt.show()
```

Seaborn

Seaborn is another Python package to plot your data and is ideal for Data Science. Built upon Matplotlib, it is made to work with Pandas Dataframes.

Seaborn

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

Seaborn offers also a quick way to download some toy datasets, for example:

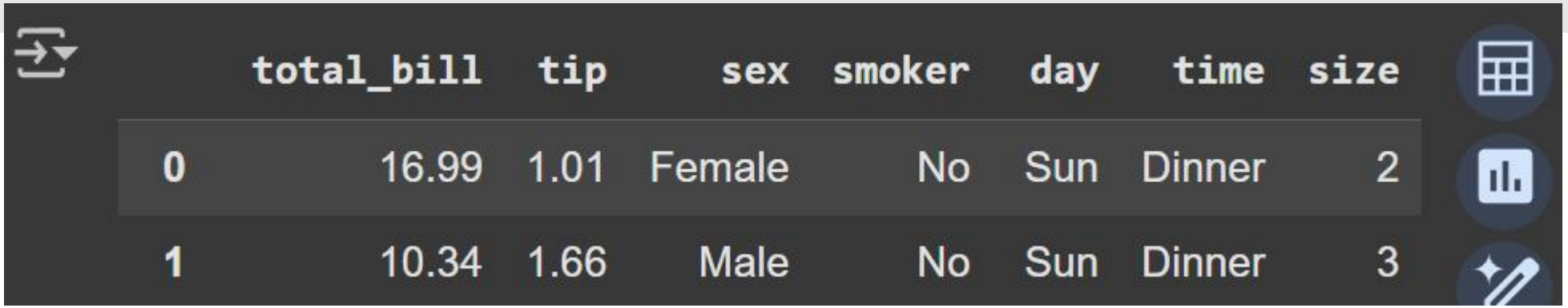
```
#This is a dataset tipping behaviour of diners

tips = sns.load_dataset("tips")
```

Seaborn

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

Seaborn offers also a quick way to download some toy datasets, for example:

```
#This is a dataset tipping behaviour of diners

tips = sns.load_dataset("tips")
tips
```



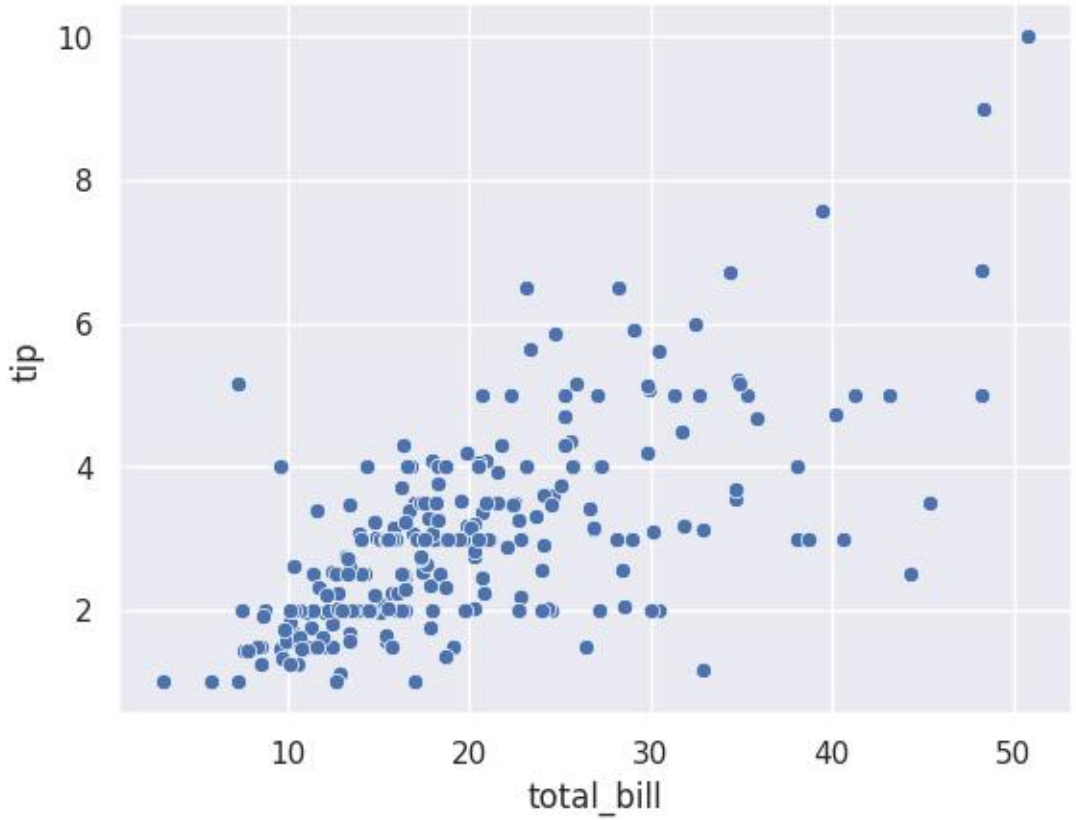
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3

Seaborn: scatterplots

For now, let's use the default theme

```
sns.set_theme()
```

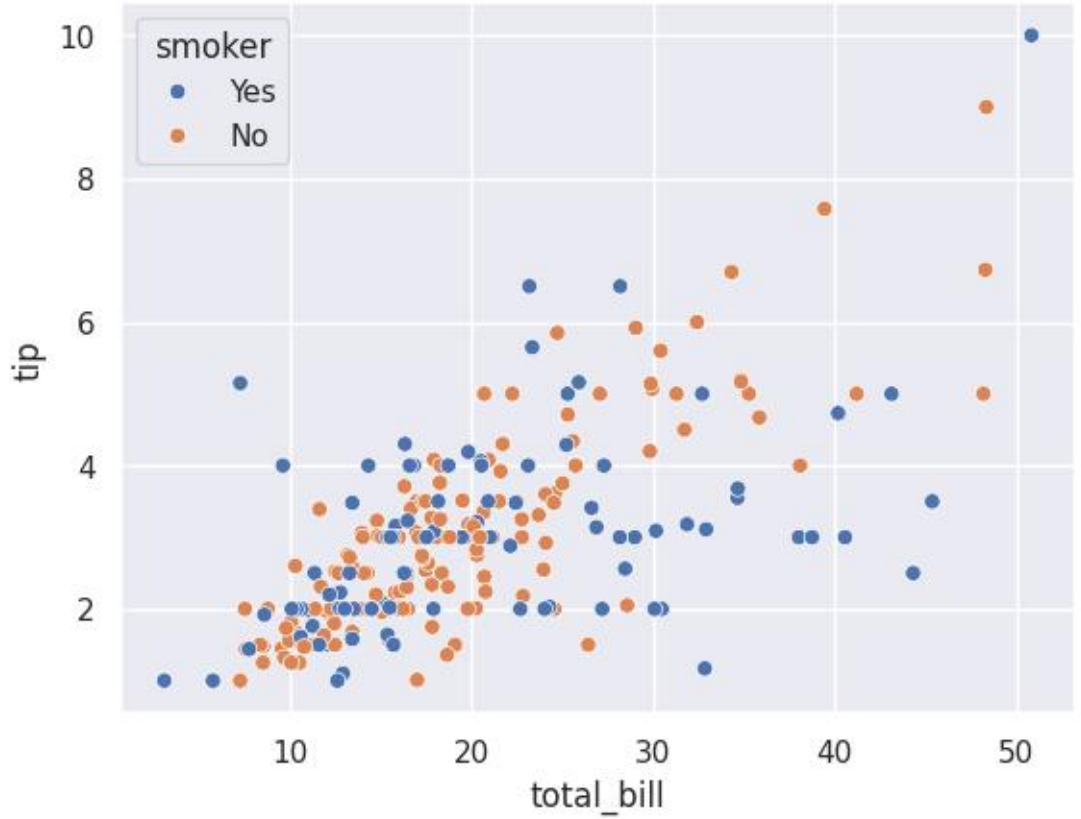
```
sns.scatterplot(data=tips, x="total_bill", y="tip")
```



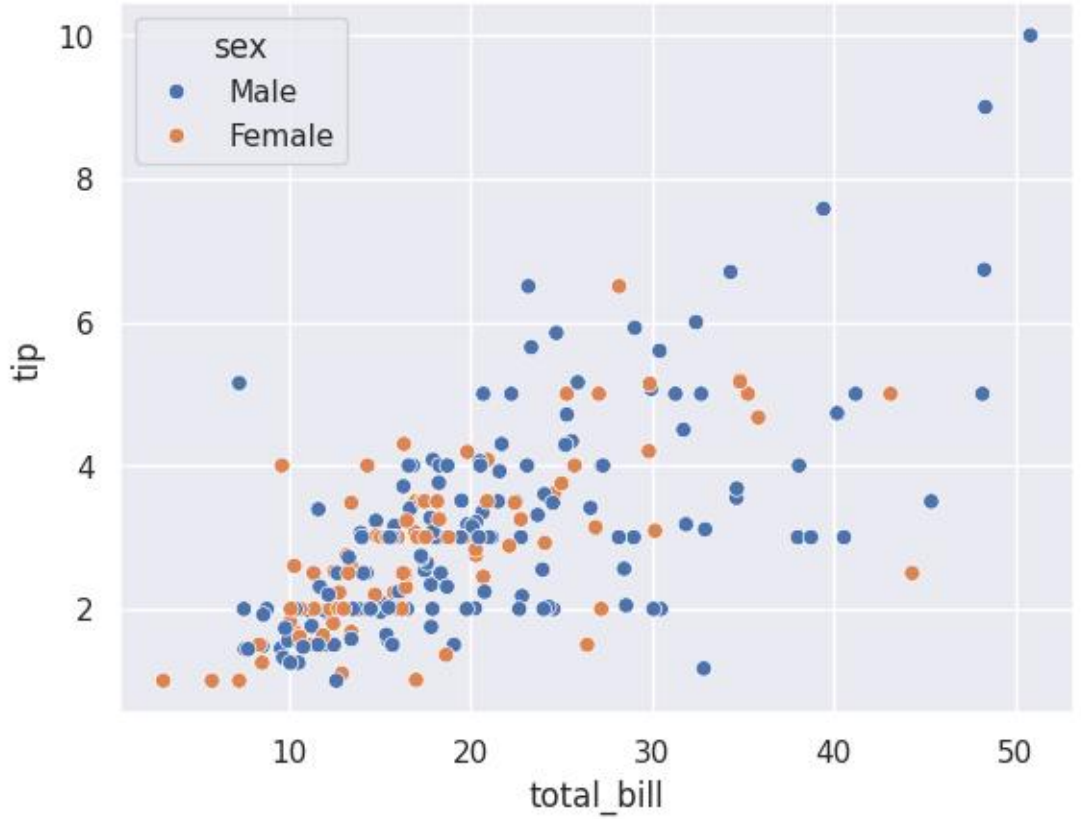
Seaborn: scatterplots

Seaborn offers the possibility to identify features with different options:

```
sns.scatterplot(data=tips,
x="total_bill", y="tip",
hue="smoker")
```



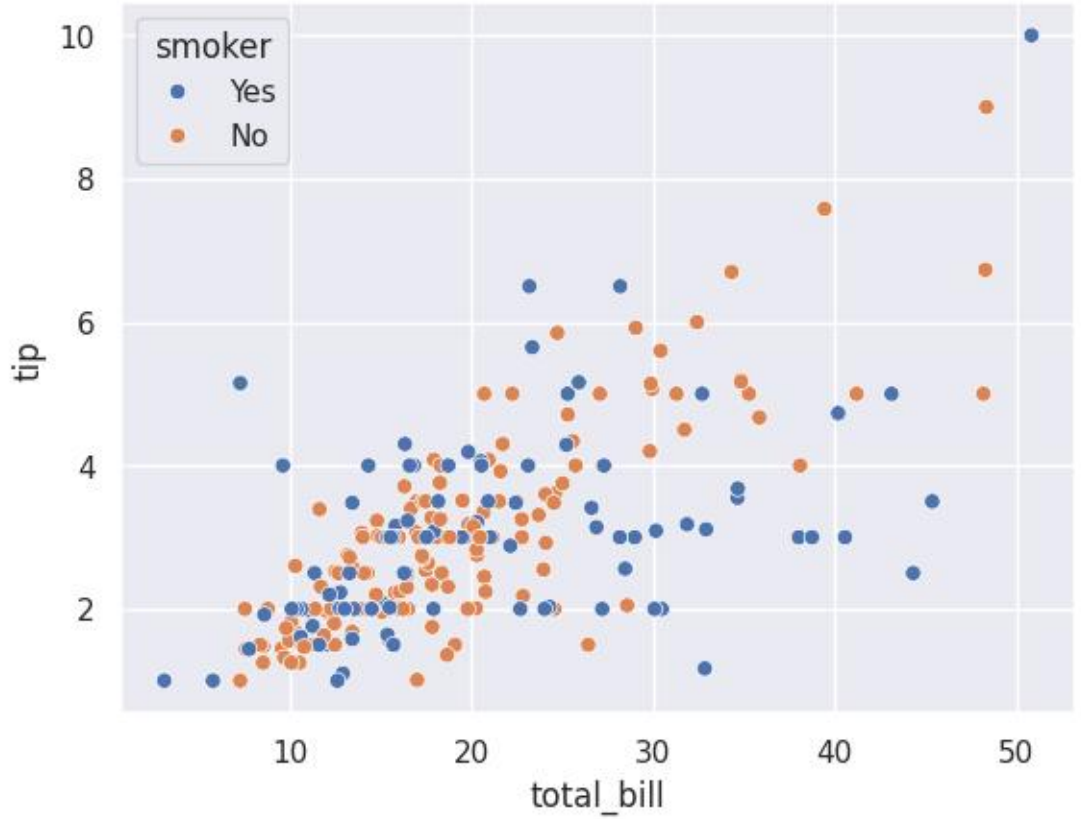

```
sns.scatterplot(data=tips,
x="total_bill", y="tip",
hue="sex")
```



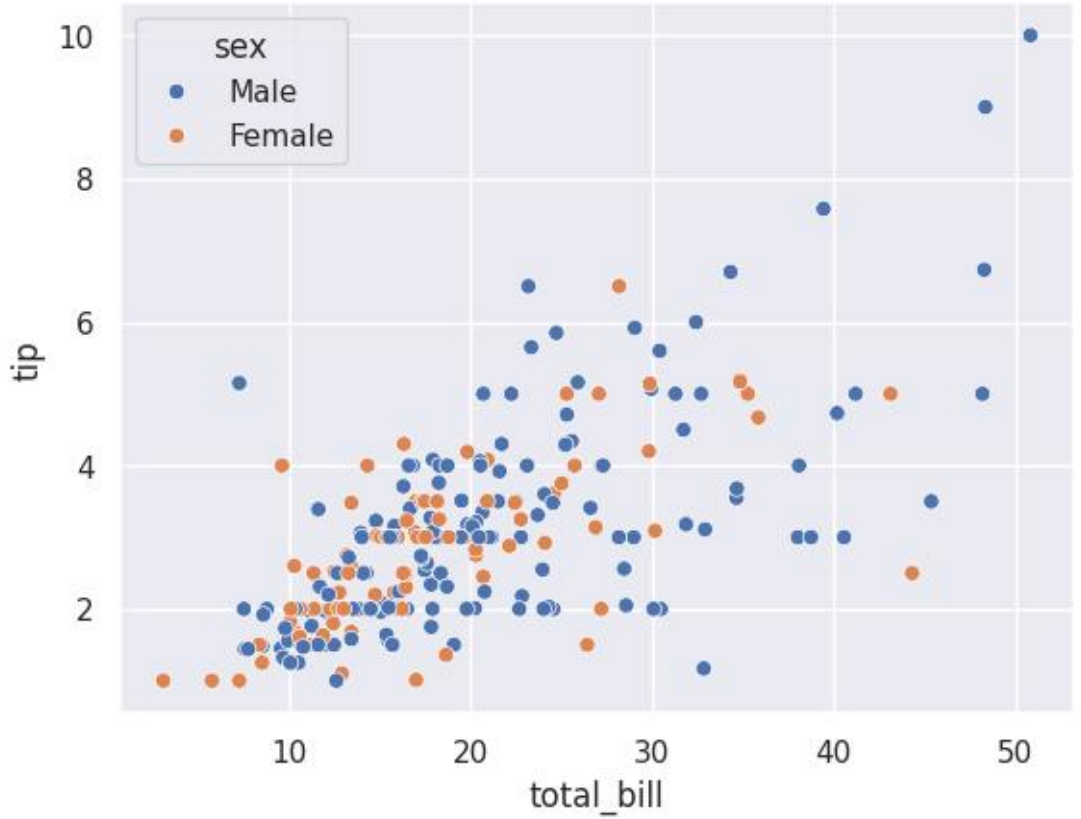
Seaborn: scatterplots

Seaborn offers the possibility to identify features with different options:

```
sns.scatterplot(data=tips,
x="total_bill", y="tip",
hue="smoker")
```



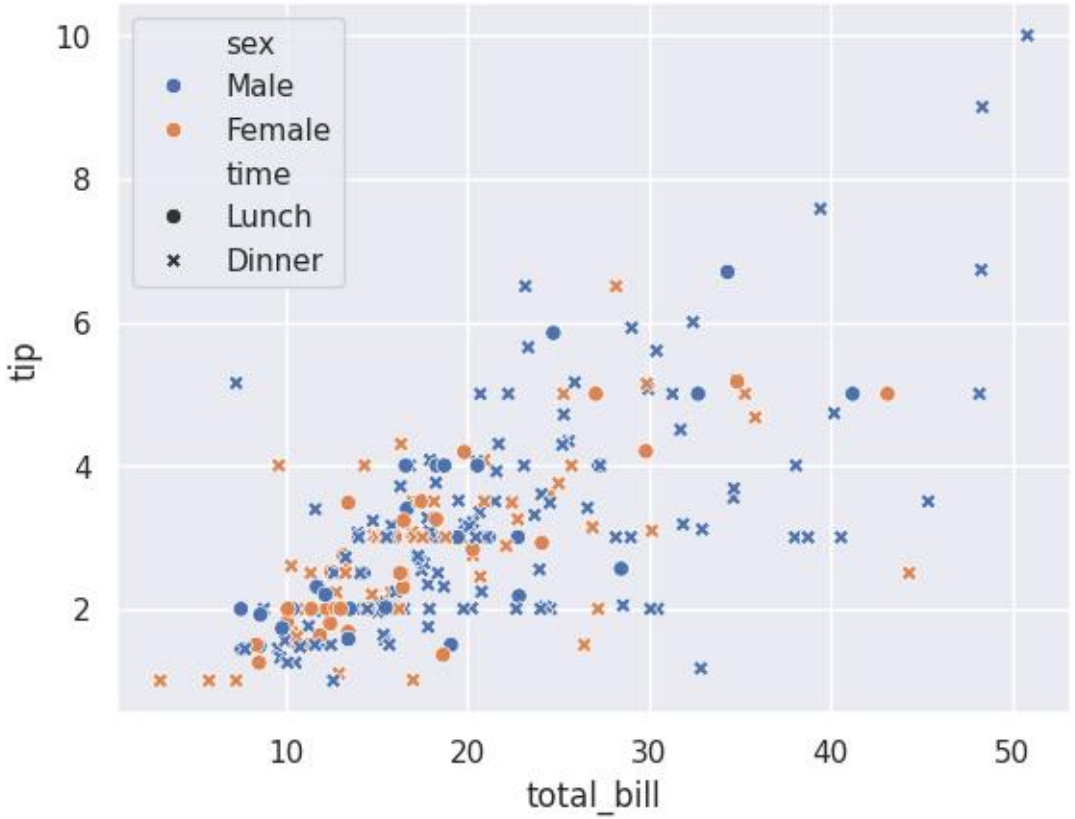

```
sns.scatterplot(data=tips,
x="total_bill", y="tip",
hue="sex")
```



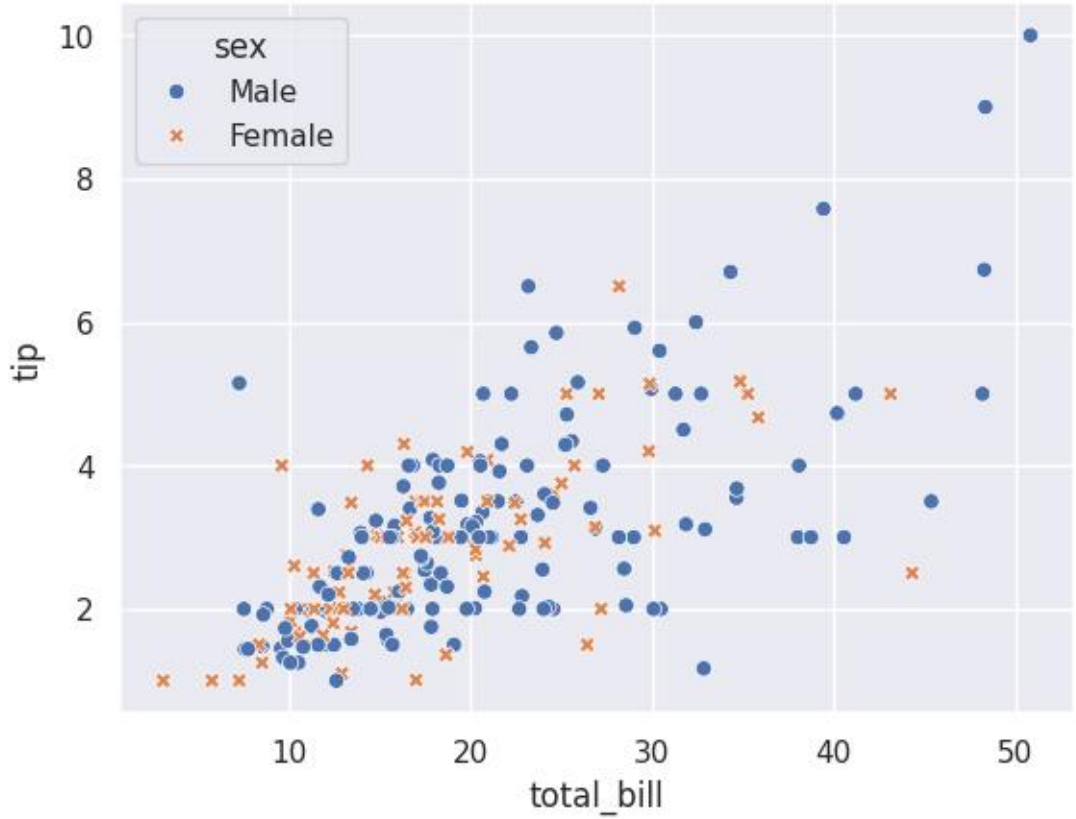
Seaborn: scatterplots

Seaborn offers the possibility to identify features with different options:

```
sns.scatterplot(data=tips,
x="total_bill", y="tip",
hue="sex", style="time")
```



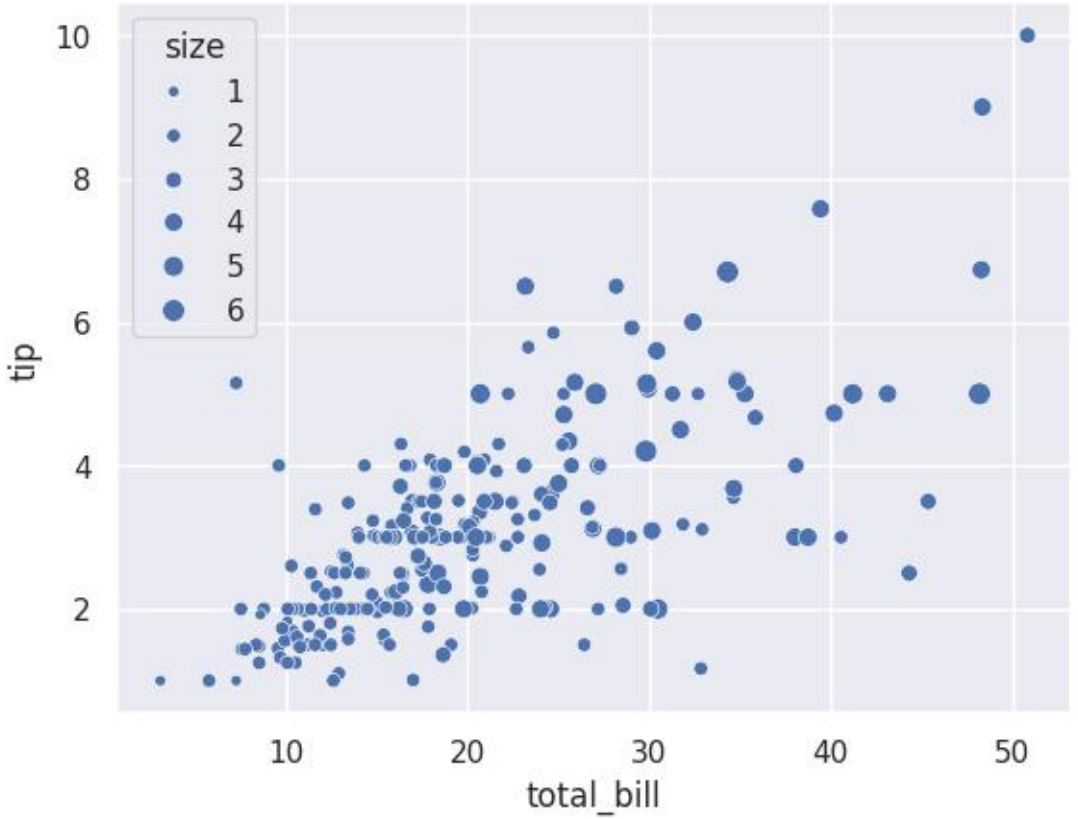

```
sns.scatterplot(data=tips,
x="total_bill", y="tip",
hue="sex", style="sex")
```



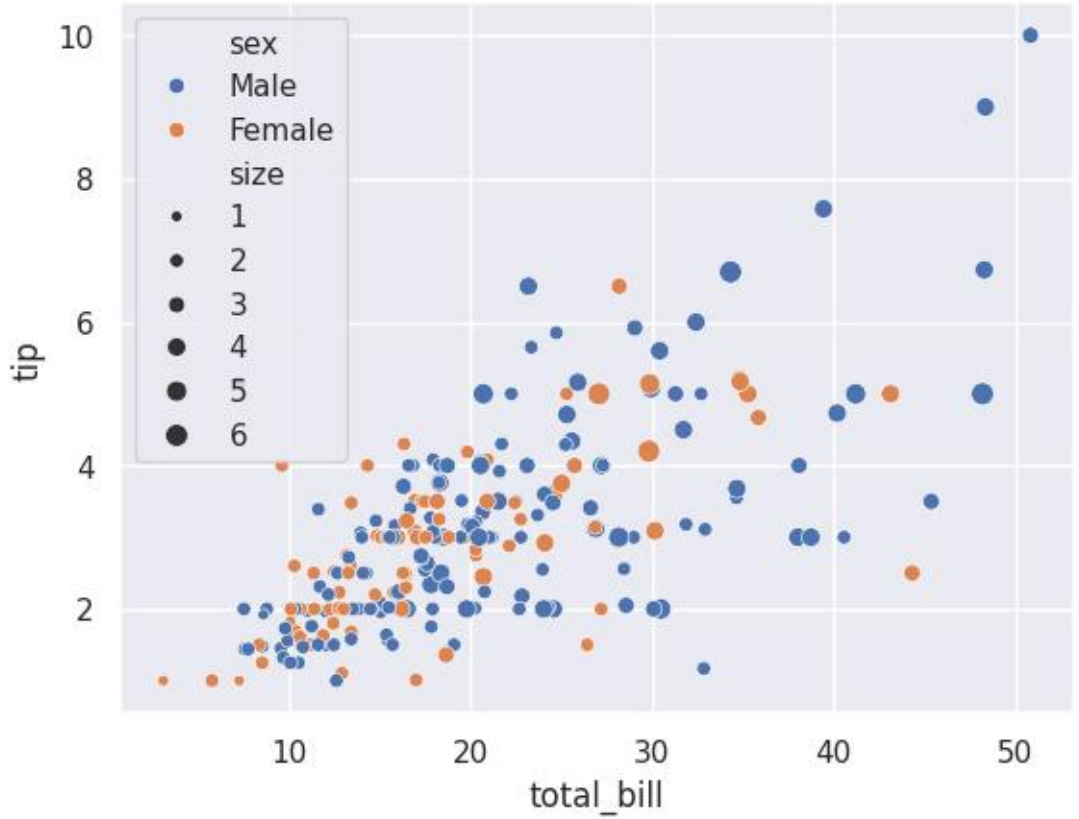
Seaborn: scatterplots

Seaborn offers the possibility to identify features with different options:

```
sns.scatterplot(data=tips,
x="total_bill", y="tip",
size="size")
```




```
sns.scatterplot(data=tips,
x="total_bill", y="tip",
size="size", hue="sex")
```



Seaborn: lineplots

In matplotlib it is quite annoying to do a lineplot from a dataframe. You have to do a groupby before and still you can't control how to plot it.

```
#This is a dataset about the number of passengers on every month of the
#year
flights = sns.load_dataset("flights")
```

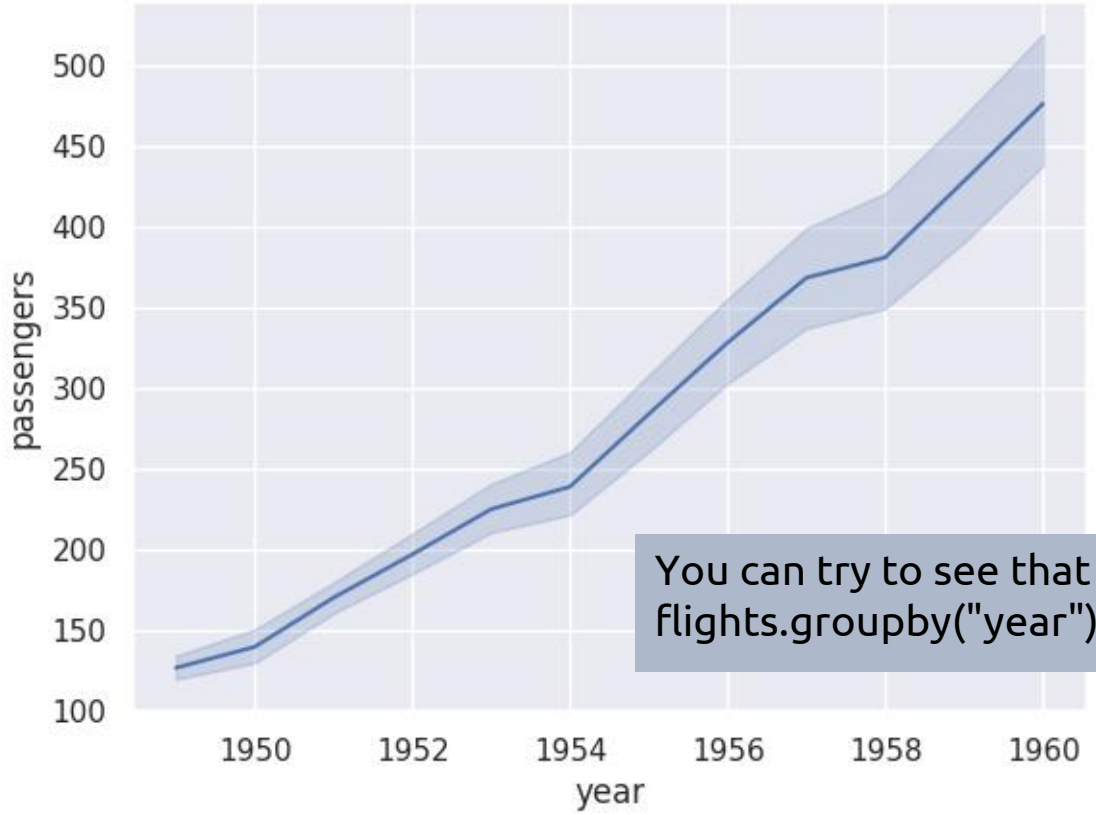
```
flights.head()
```

	year	month	passengers
0	1949	Jan	112
1	1949	Feb	118
2	1949	Mar	132
3	1949	Apr	129
4	1949	May	121

Seaborn: lineplots

Seaborn performs the aggregation and reduction by itself. By default, the mean is computed together with the 95% confidence interval.

```
sns.lineplot(data=flights, x="year", y="passengers")
```

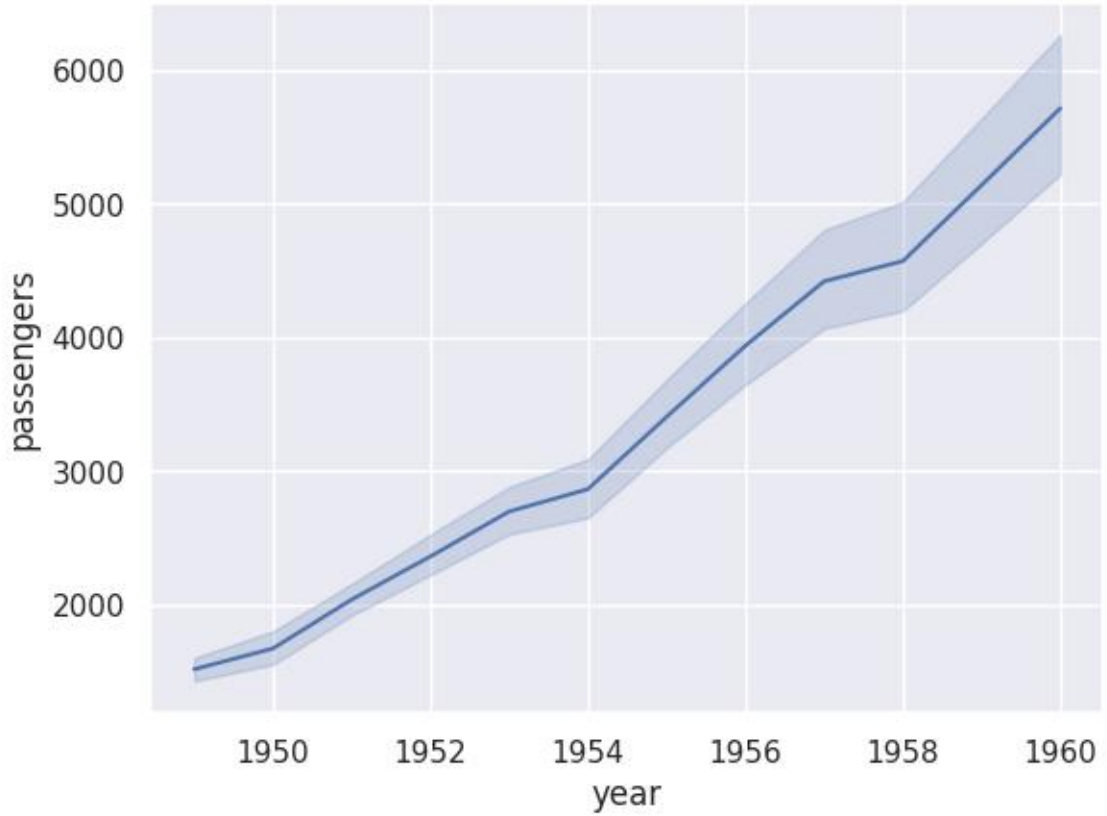


You can try to see that it's the same as running flights.groupby("year")["passengers"].mean().plot()

Seaborn: lineplots

You can use other estimators:

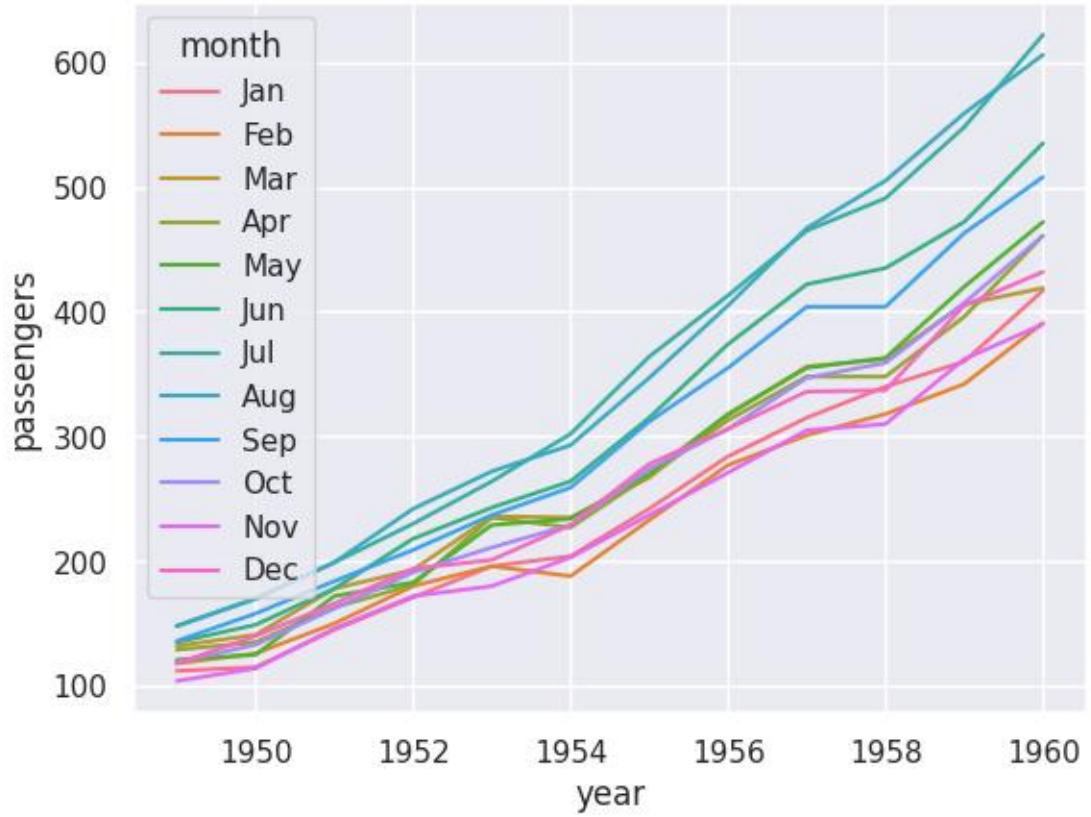
```
sns.lineplot(data=flights, x="year", y="passengers", estimator="sum")
```



Seaborn: lineplots

You can use all the options we have seen before:

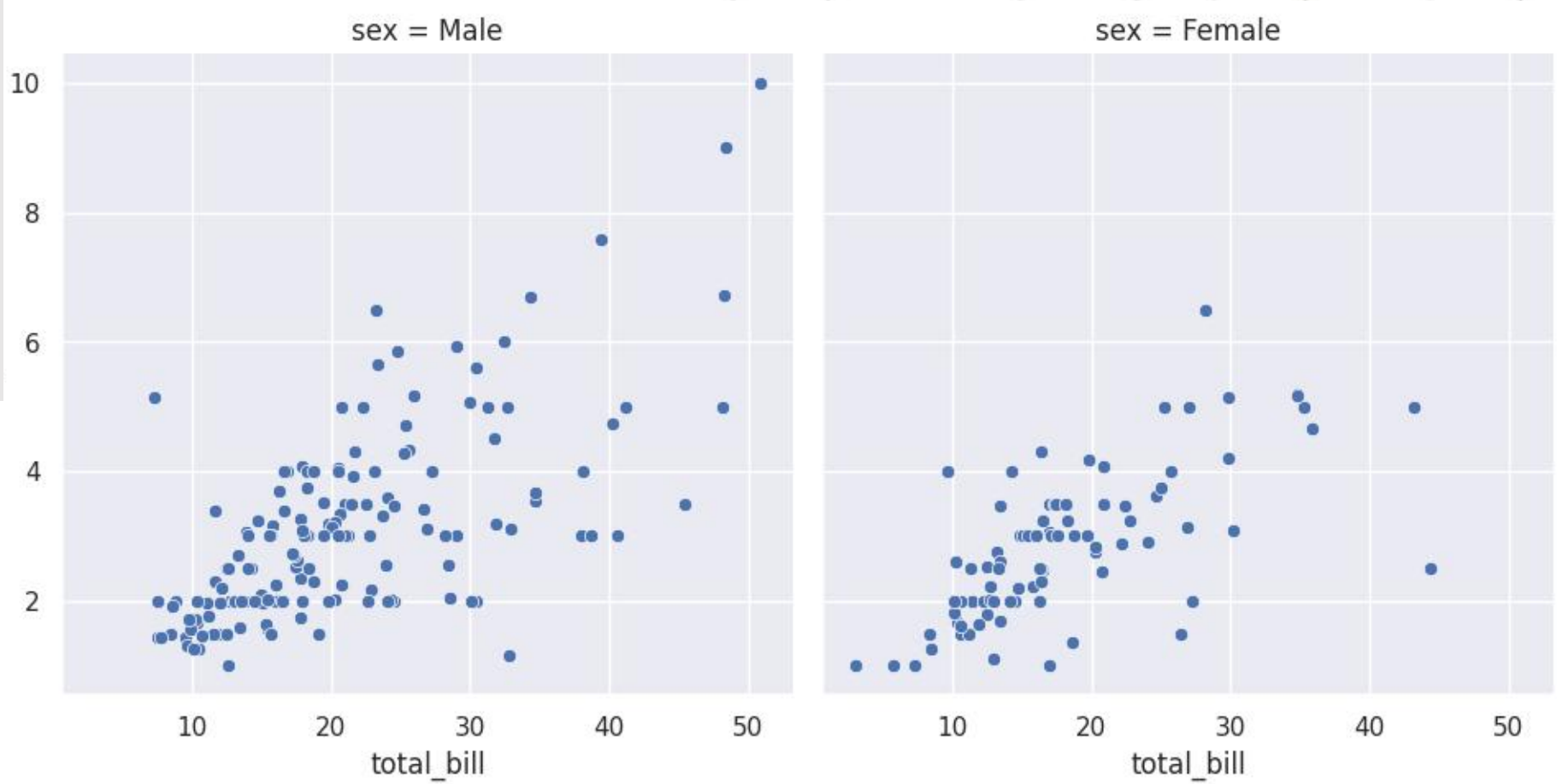
```
sns.lineplot(data=flights, x="year", y="passengers", hue="month")
```



Seaborn: relplots

Different relational plots are available in Seaborn to give you insight into your data.

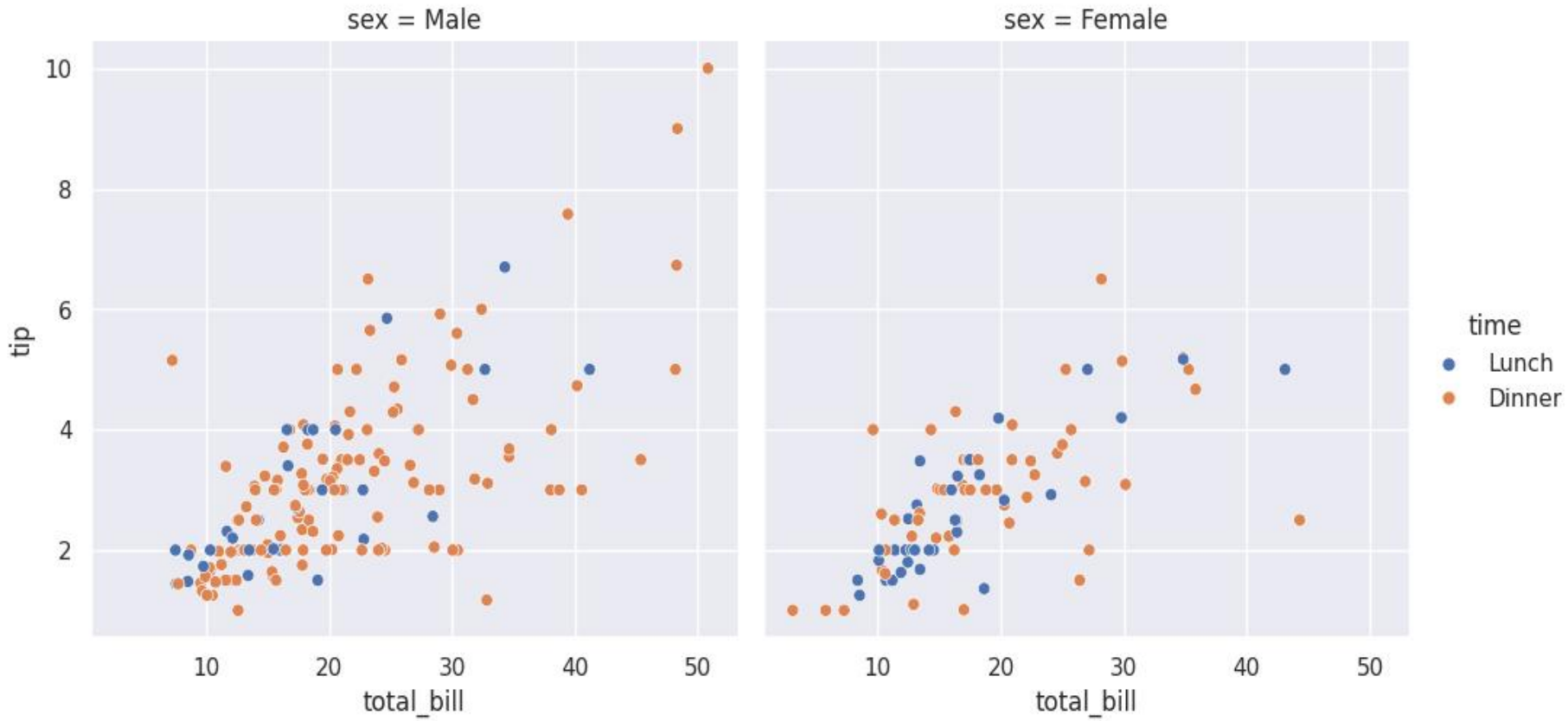
```
sns.relplot(
    data=tips,
    x="total_bill",
    y="tip",
    kind="scatter",
    col="sex",
)
```



Seaborn: relplots

Different relational plots are available in Seaborn to give you insight into your data.

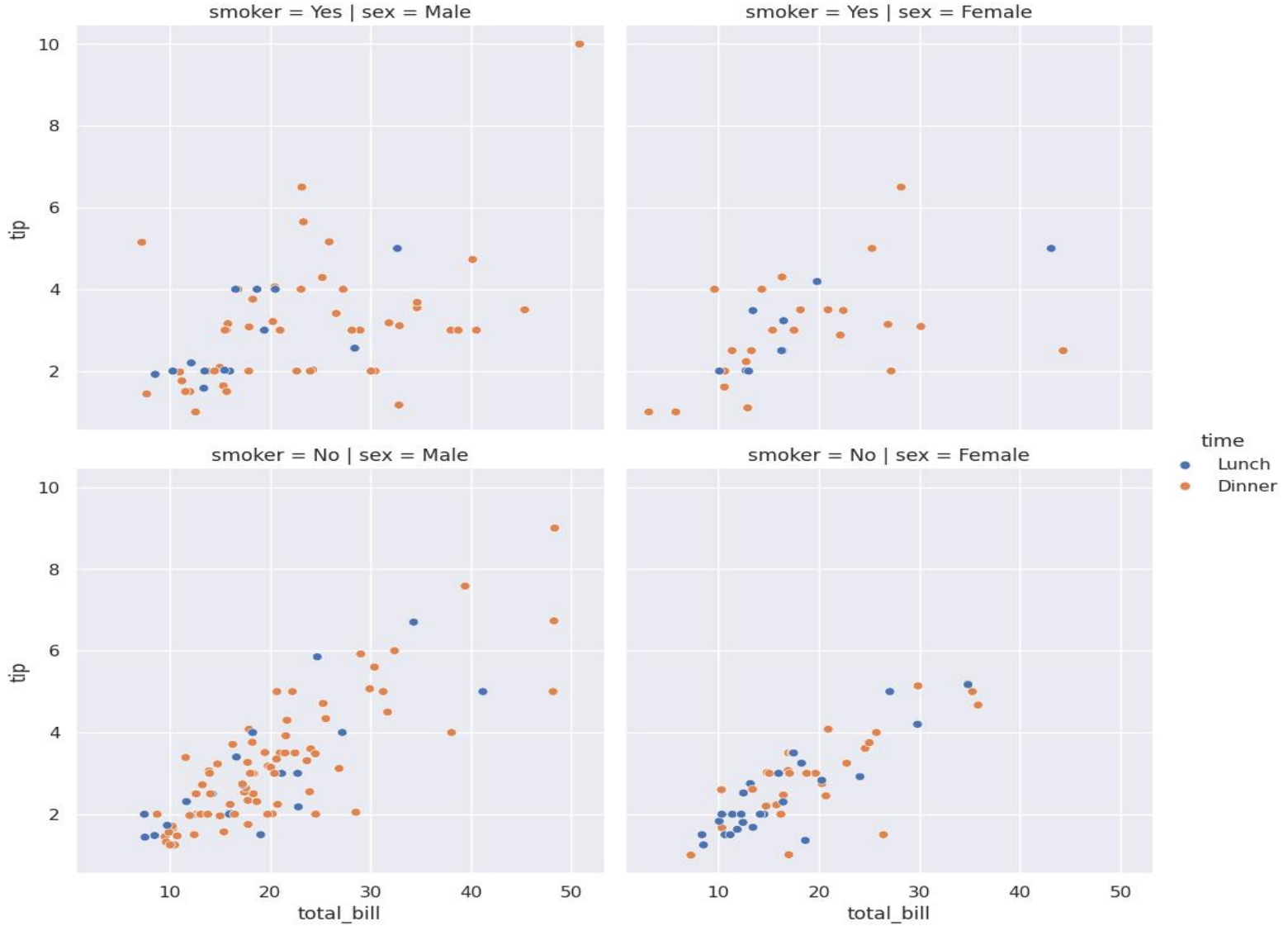
```
sns.relplot(
    data=tips,
    x="total_bill",
    y="tip",
    kind="scatter",
    col="sex",
    hue="time",
)
```



Seaborn: relplots

Different relational plots are available in Seaborn to give you insight into your data.

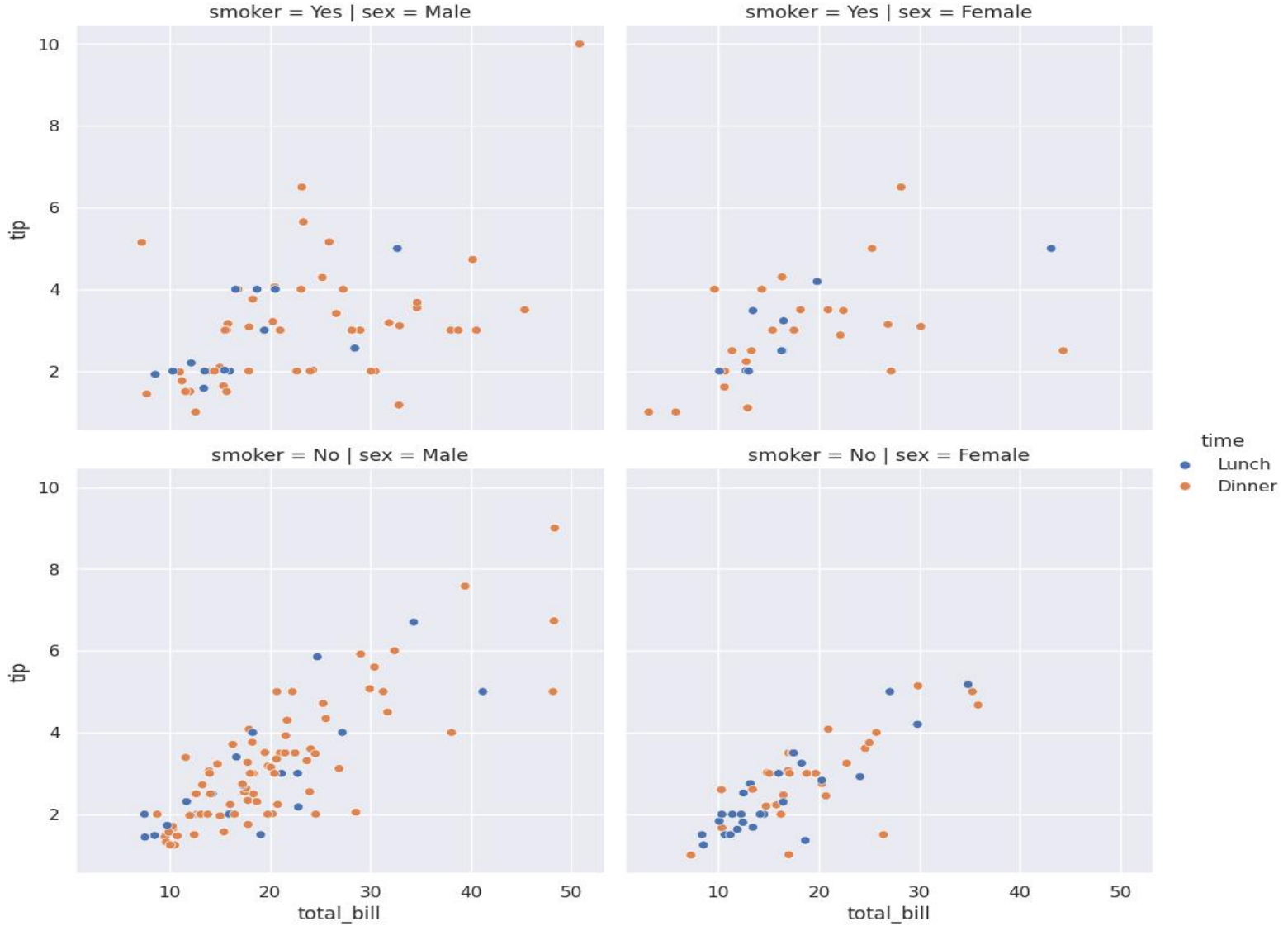
```
sns.relplot(
    data=tips,
    x="total_bill",
    y="tip",
    kind="scatter",
    col="sex",
    hue="time",
    row="smoker"
)
```



Seaborn: relplots

Different relational plots are available in Seaborn to give you insight into your data.

```
sns.relplot(
    data=tips,
    x="total_bill",
    y="tip",
    kind="scatter",
    col="sex",
    hue="time",
    row="smoker"
)
```



Exercise

Create a relplot describing how taxi fares depends on distance. Split the data according to the pick-up borough and how they were paid. Point must be of the same color of the taxis (yellow, green).

```
trips = sns.load_dataset("taxis")
```

```
trips.head(3)
```

	pickup	dropoff	passengers	distance	fare	tip	tolls	total	color	payment	pickup_zone	dropoff_zone	pi
0	2019-03-23 20:21:09	2019-03-23 20:27:24	1	1.60	7.0	2.15	0.0	12.95	yellow	credit card	Lenox Hill West	UN/Turtle Bay South	

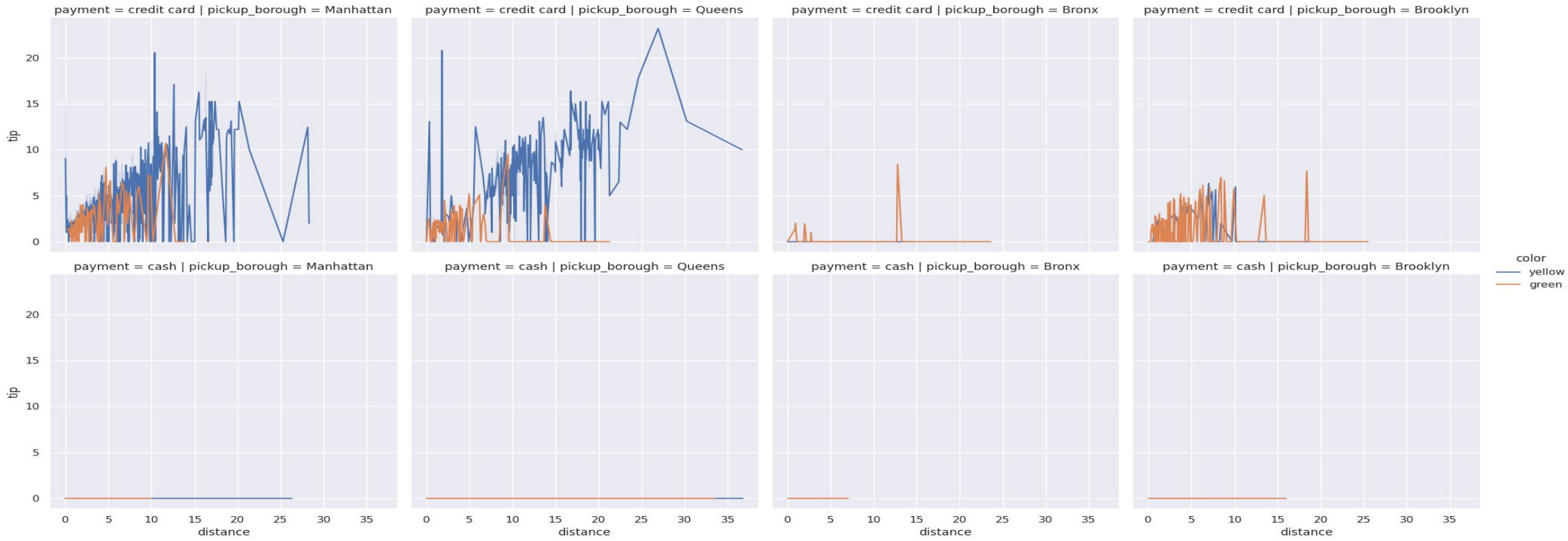
Then create a new column with the hour and do a second replot to check if the tip is linked to the hour

```
trips["hour"] = trips["pickup"].dt.hour
```

Solution

```
trips = sns.load_dataset("taxis")
```

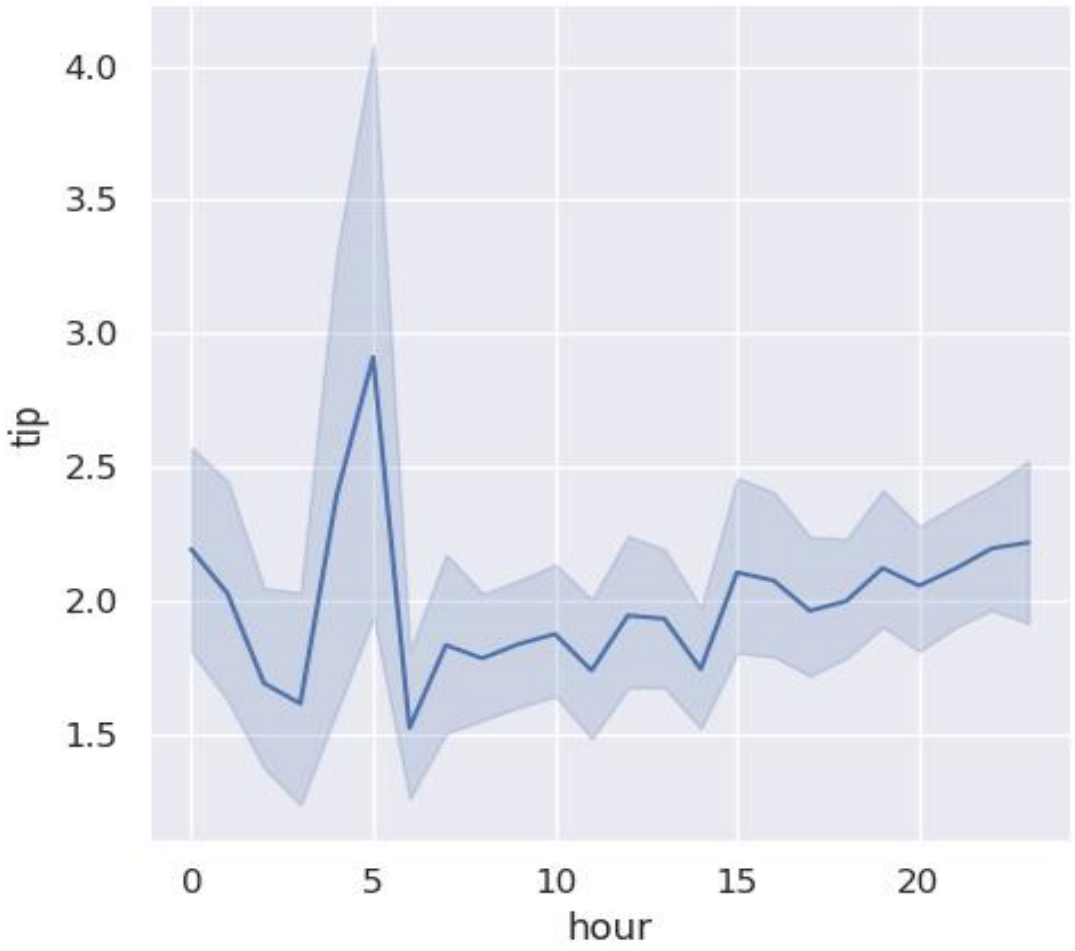
```
sns.relplot(data=trips,kind="line",x="distance",y="tip",
            col="pickup_borough",row="payment",hue="color")
```



Solution

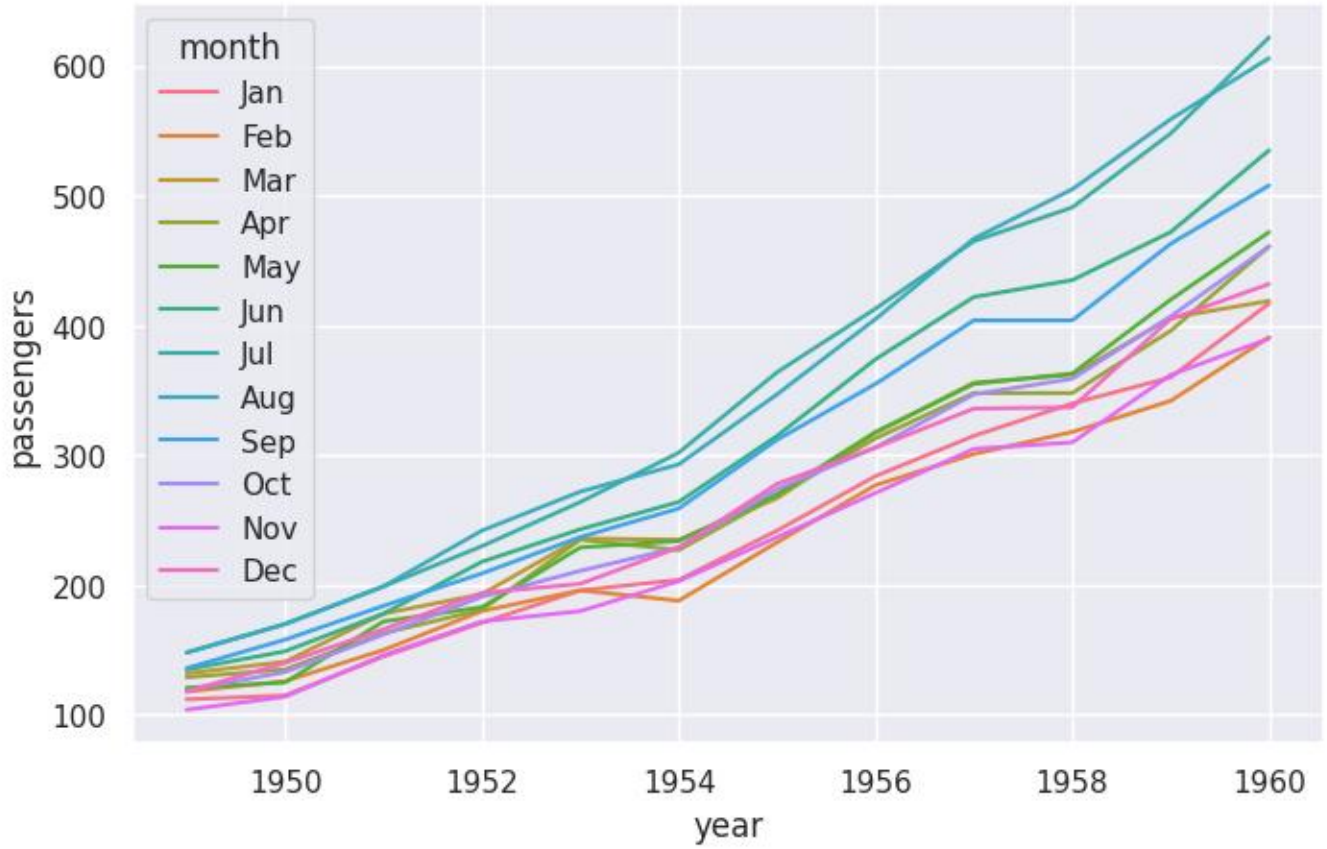
```
trips = sns.load_dataset("taxis")
```

```
sns.relplot(
    data=trips,
    kind="line",
    x="hour",
    y="tip",
)
```



Seaborn: change plot size

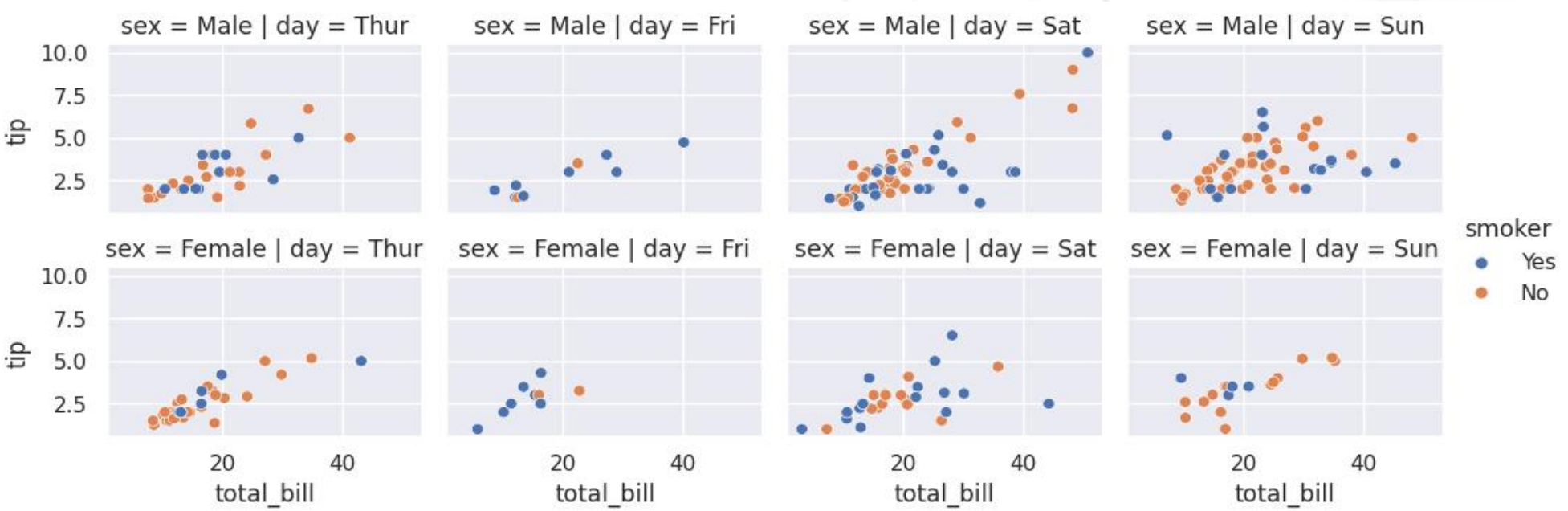
```
plt.figure(figsize=(8,5))
sns.lineplot(data=flights, x="year", y="passengers", hue="month")
```



Seaborn: change plot size

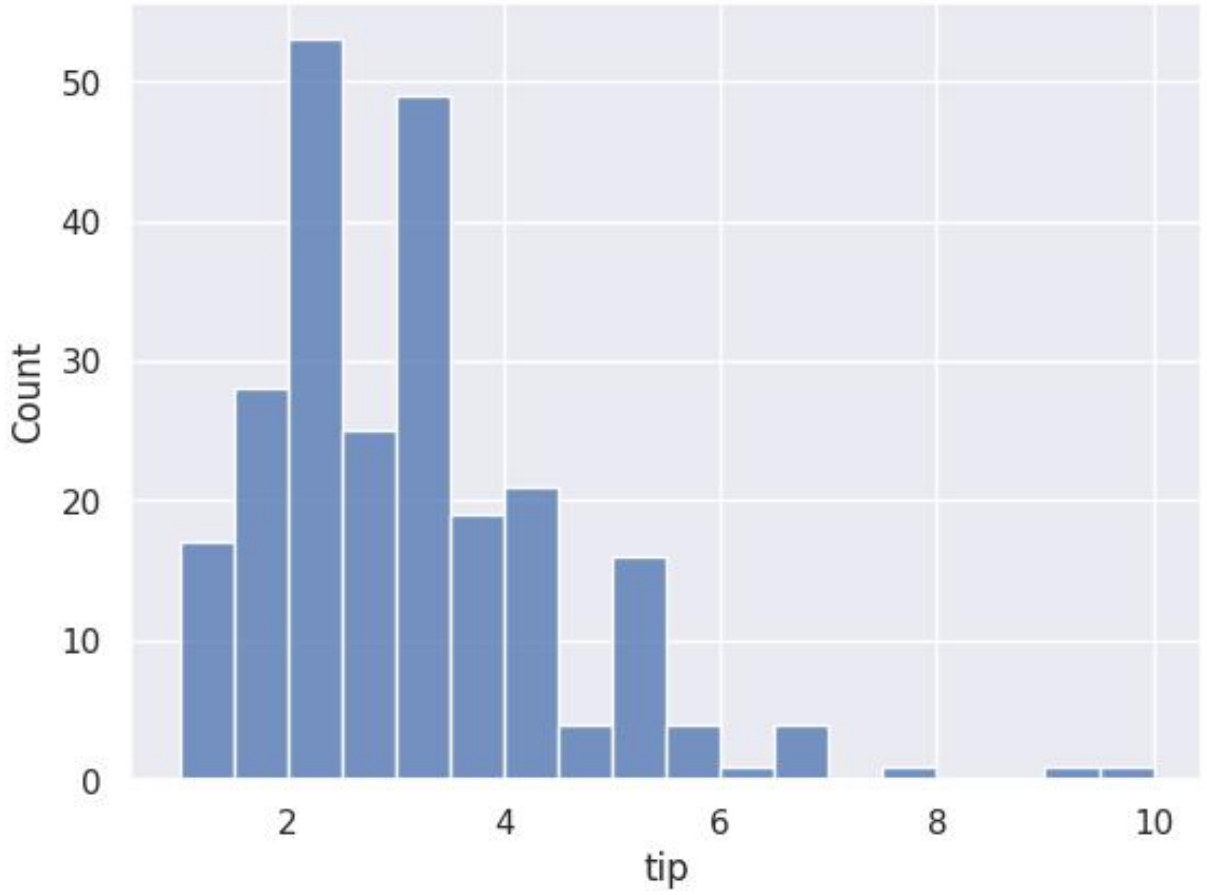
When using relplots, it is better to use height and aspect to set the weigth:

```
sns.relplot(
    data=tips,

x="total_bill",
    y="tip",

kind="scatter",
    hue="smoker",
    col="day",
    row="sex",
    height=2,
    aspect=1.3
)
```



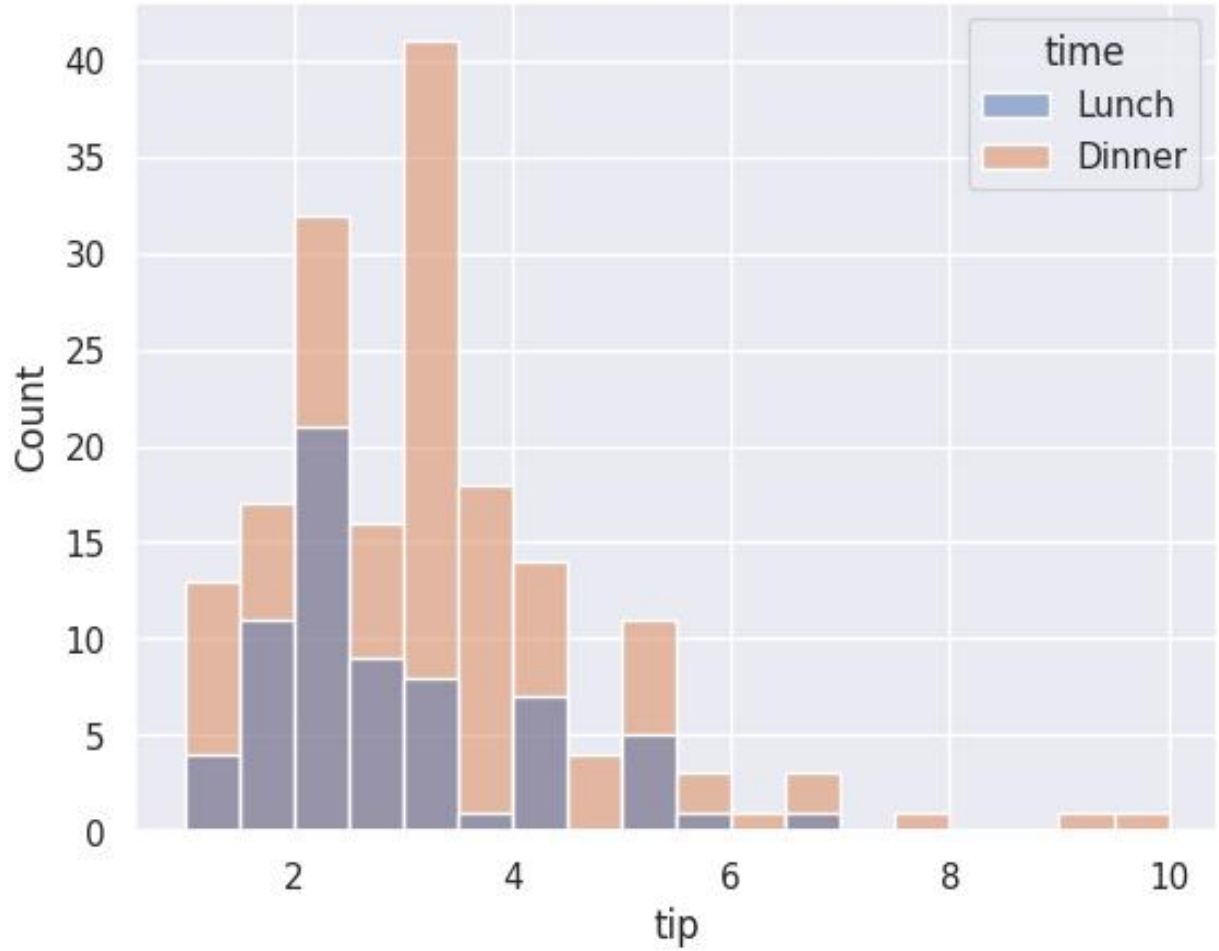
Seaborn: histograms

```
sns.histplot(data=tips, x="tip")
```



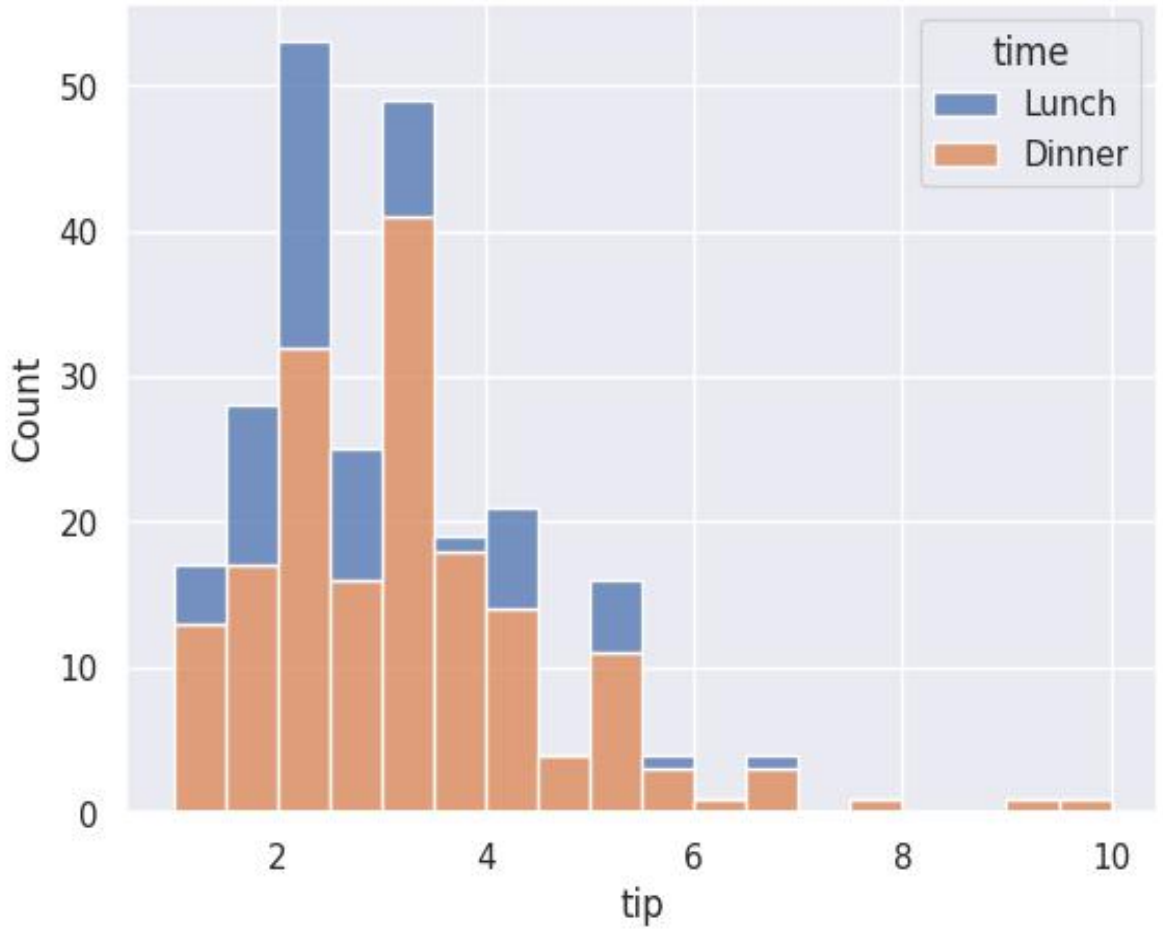
Seaborn: histograms

```
sns.histplot(data=tips, x="tip", hue="time")
```



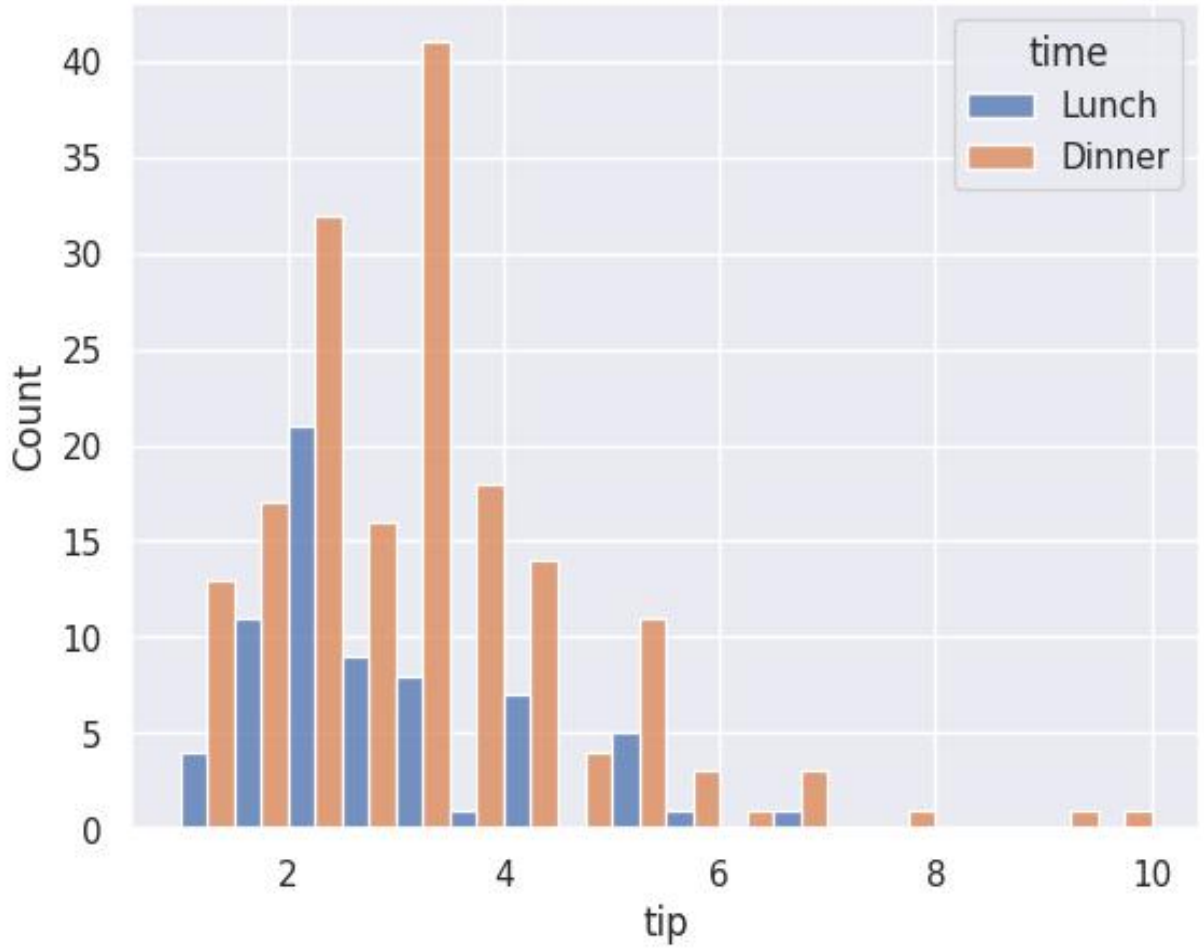
Seaborn: histograms

```
sns.histplot(data=tips, x="tip", hue="time", multiple="stack")
```



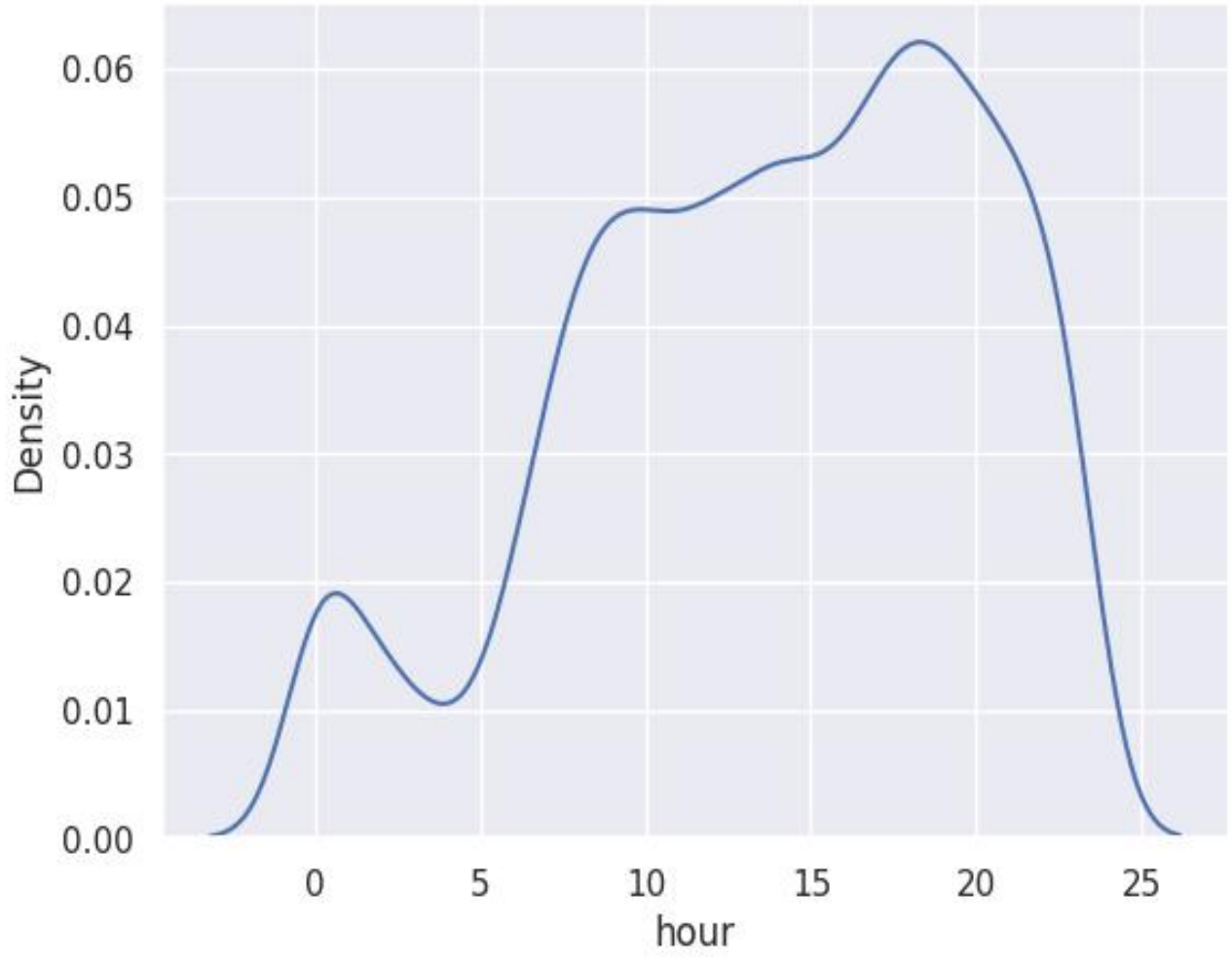
Seaborn: histograms

```
sns.histplot(data=tips, x="tip", hue="time", multiple="dodge")
```



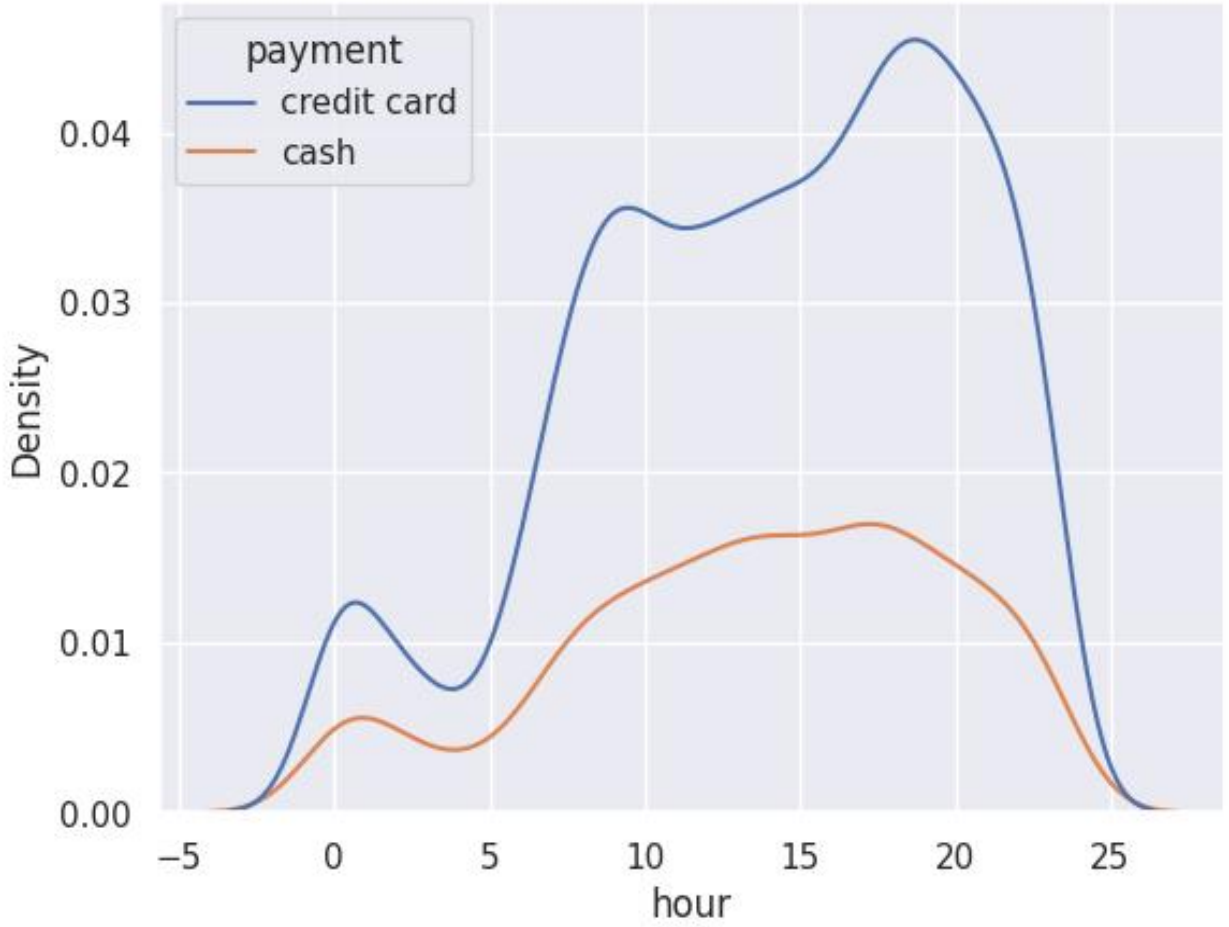
Seaborn: KDE plots

```
sns.kdeplot(data=trips, x="hour")
```



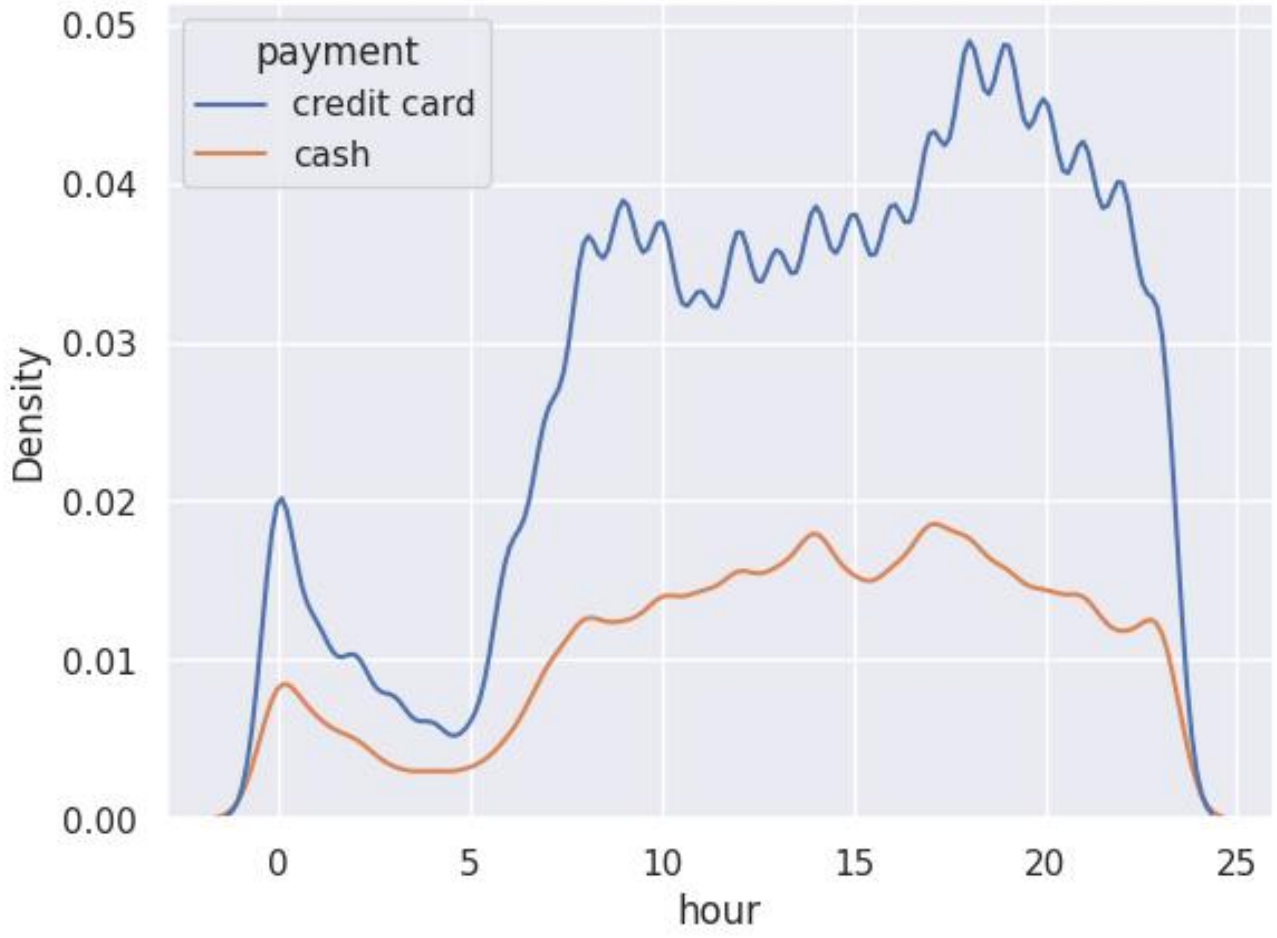
Seaborn: KDE plots

```
sns.kdeplot(data=trips, x="hour", hue="payment")
```



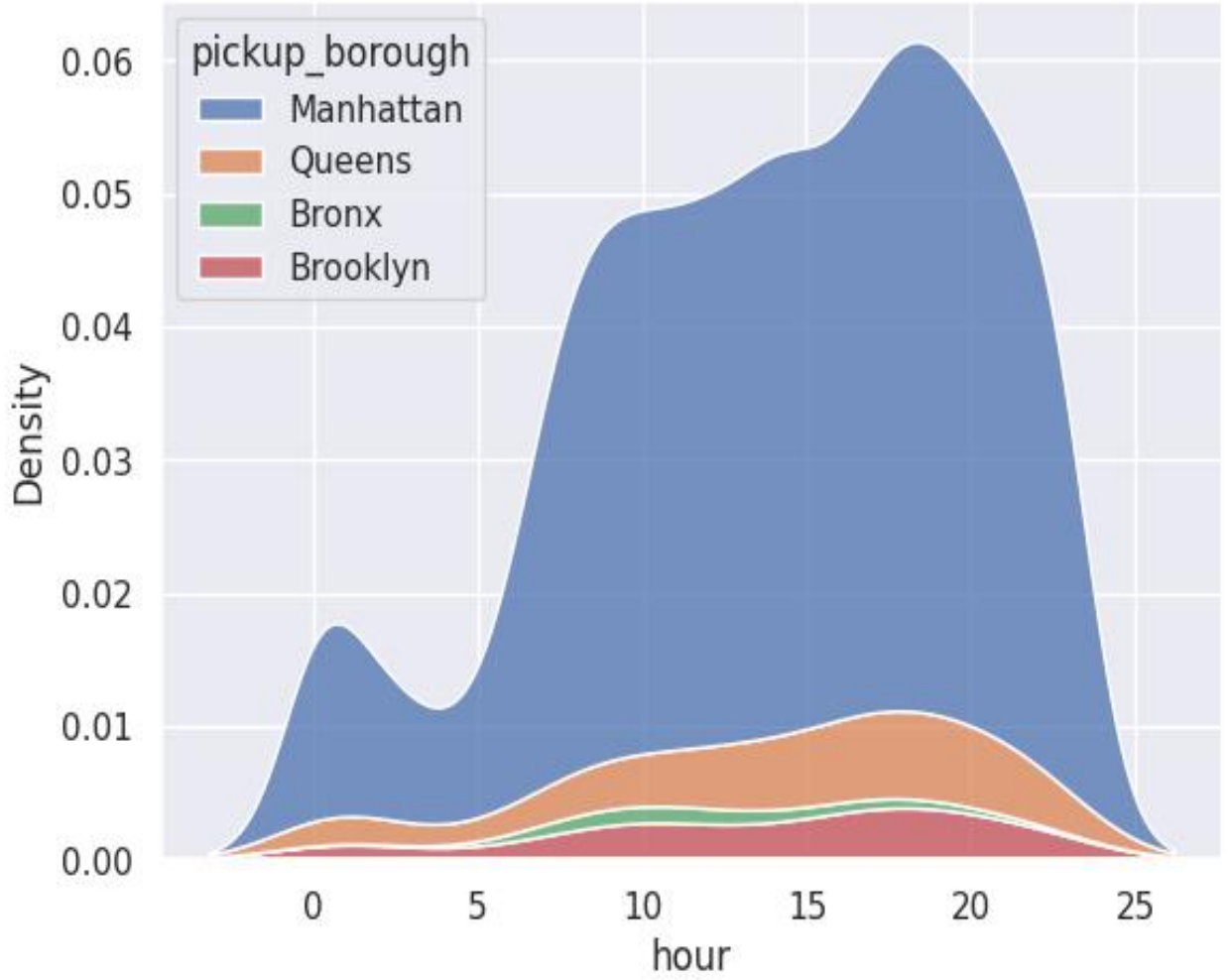
Seaborn: KDE plots

```
sns.kdeplot(data=trips, x="hour", hue="payment", bw_adjust=0.4)
```



Seaborn: KDE plots

```
sns.kdeplot(data=trips, x="hour", hue="pickup_borough", multiple="stack")
```



Seaborn: multivariate distribution

Let’s use a different datasets on penguins:

```
#This loads a dataset about a penguin population
penguins = sns.load_dataset("penguins")
```

```
penguins.head()
```

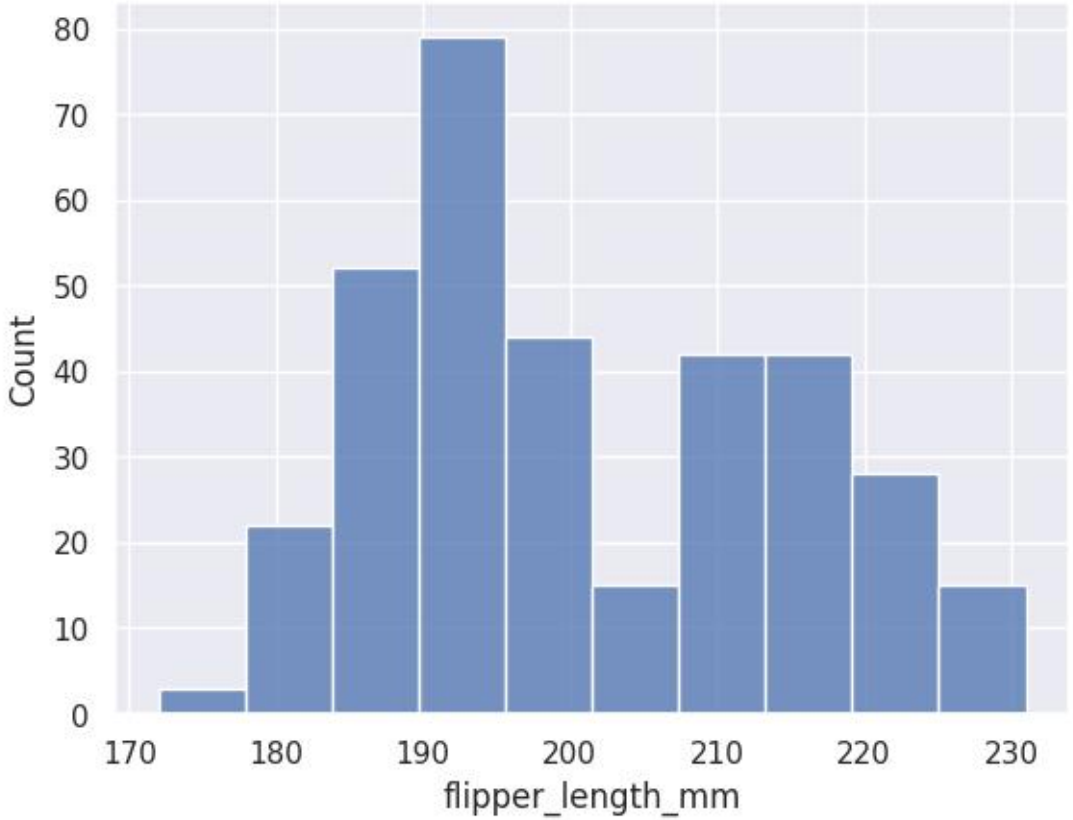
	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female

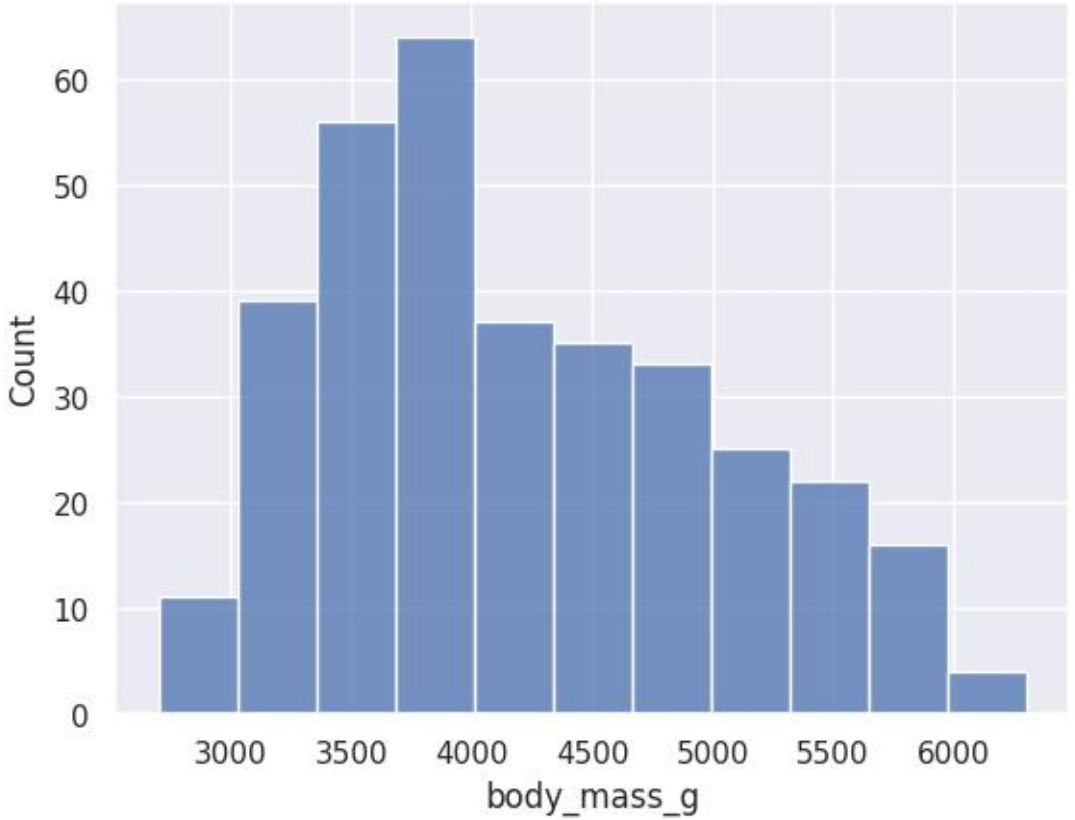
```
penguins.species.unique()
```

```
array(['Adelie', 'Chinstrap', 'Gentoo'], dtype=object)
```

Seaborn: multivariate distribution

```
sns.histplot(data=penguins, x="flipper_length_mm")
```

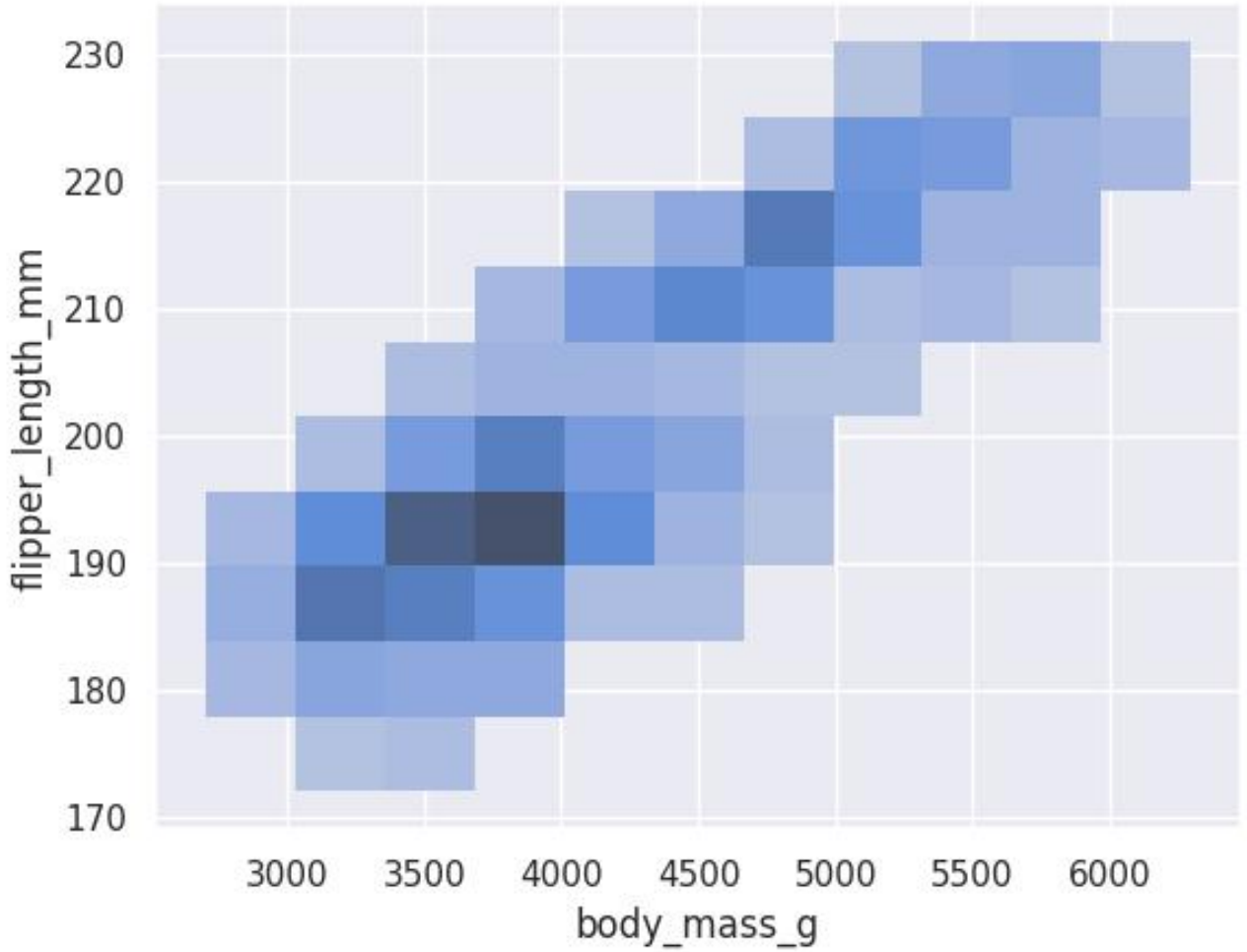





```
sns.histplot(data=penguins, x="body_mass_g")
```

Seaborn: multivariate distribution

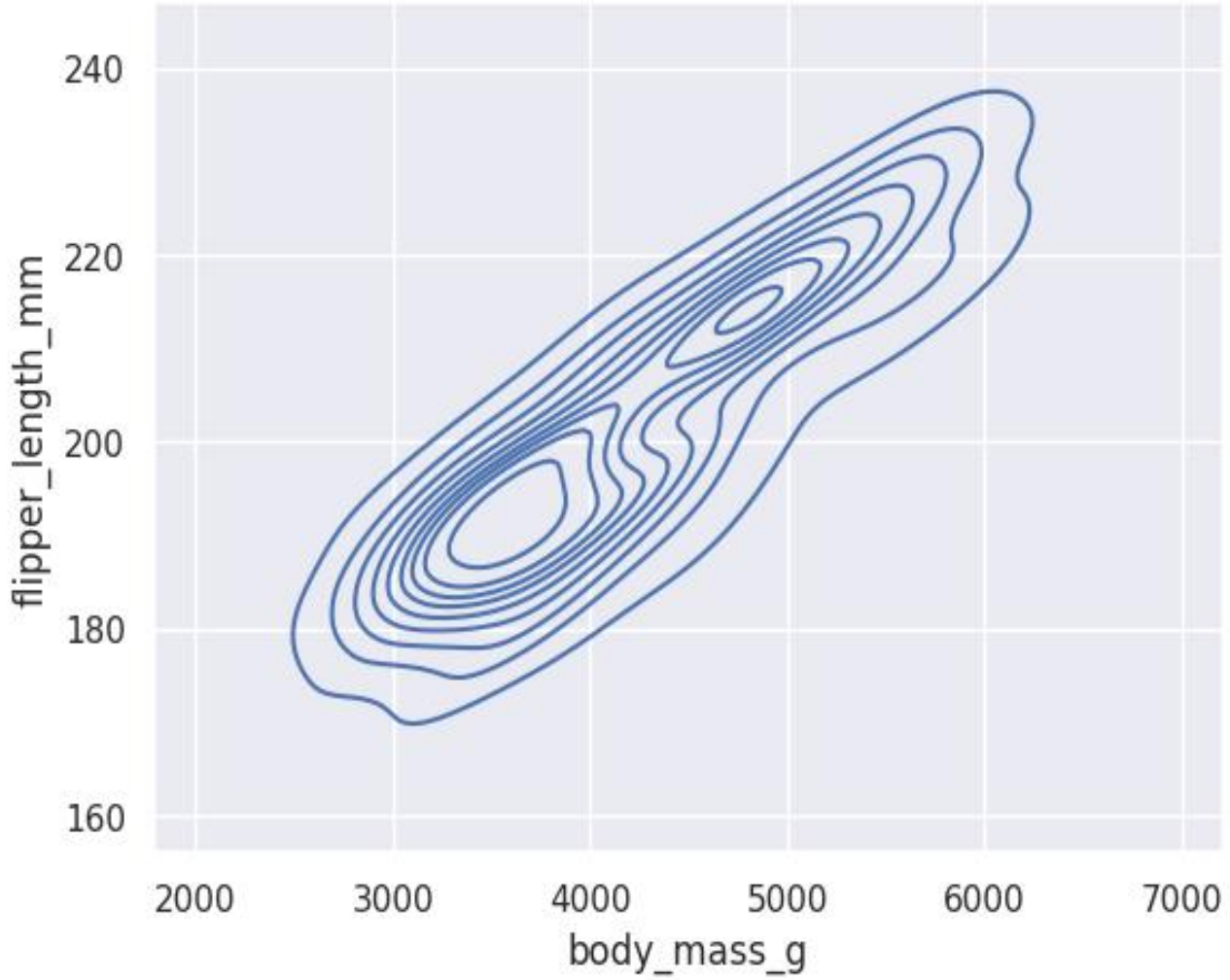
```
sns.histplot(data=penguins, x="body_mass_g", y="flipper_length_mm")
```



Seaborn: multivariate distribution

ITINERIS

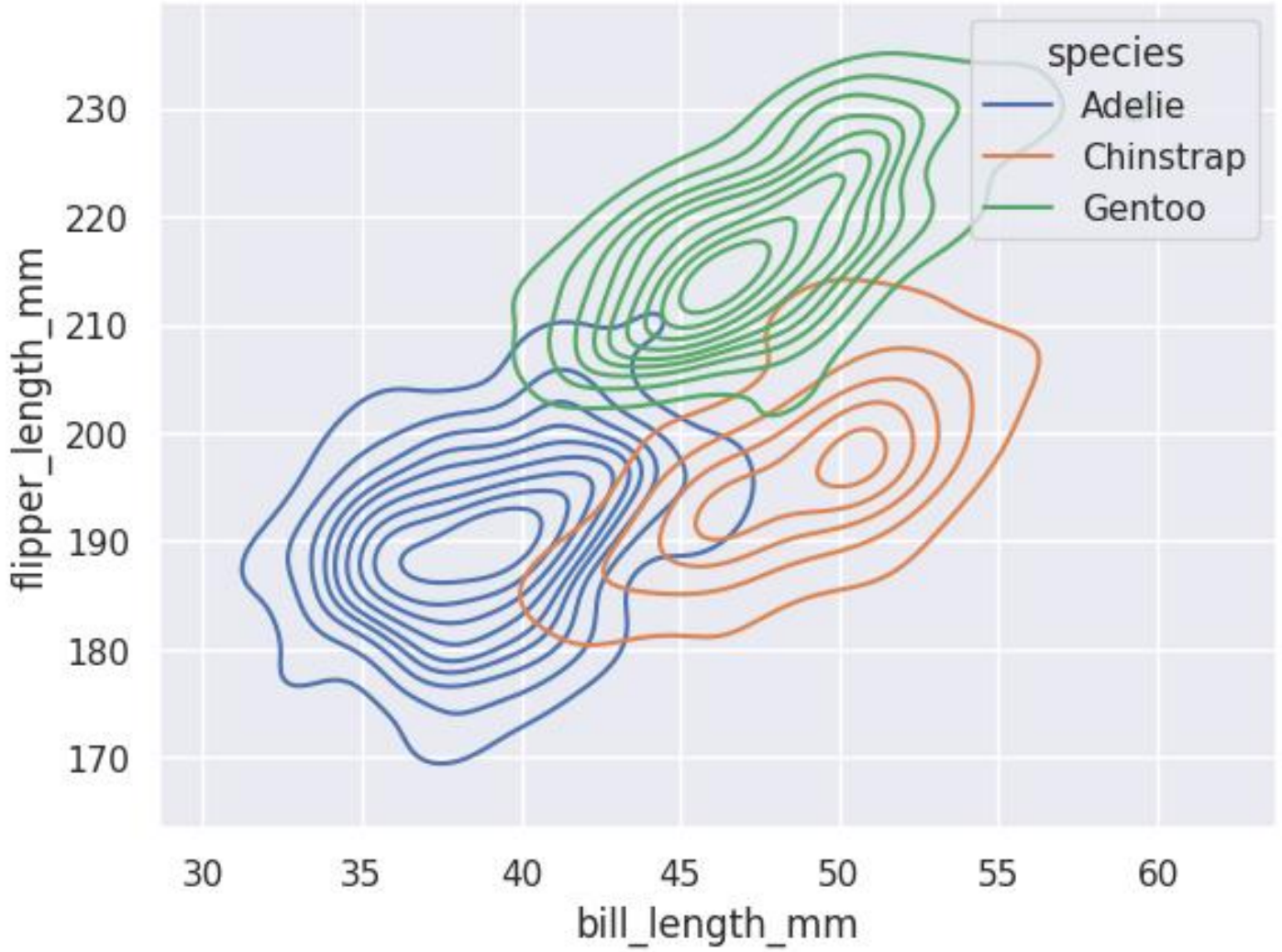
```
sns.kdeplot(data=penguins, x="body_mass_g", y="flipper_length_mm")
```



Seaborn: multivariate distribution

ITINERIS

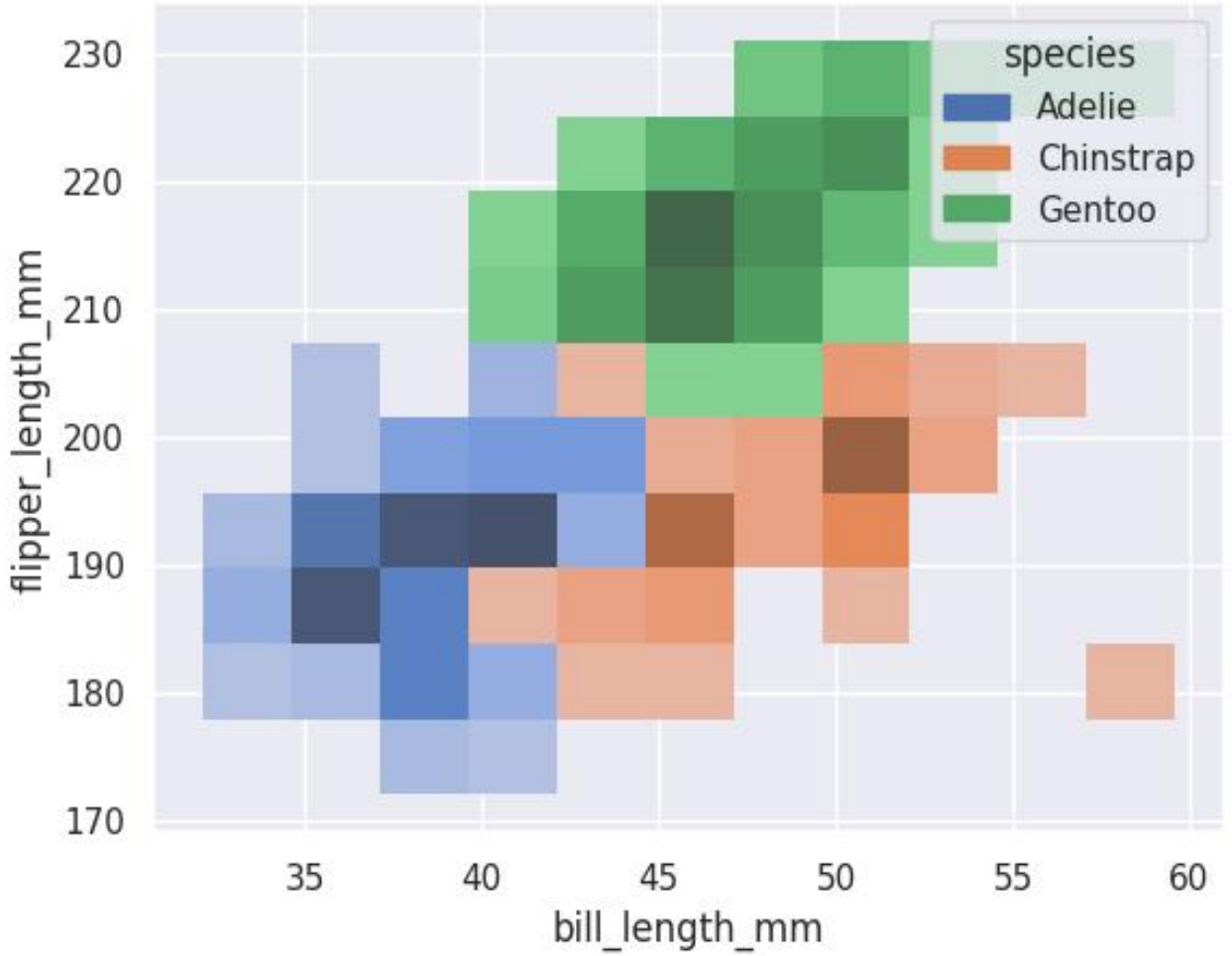
```
sns.kdeplot(data=penguins, x="bill_length_mm", y="flipper_length_mm", hue="species")
```



Seaborn: multivariate distribution

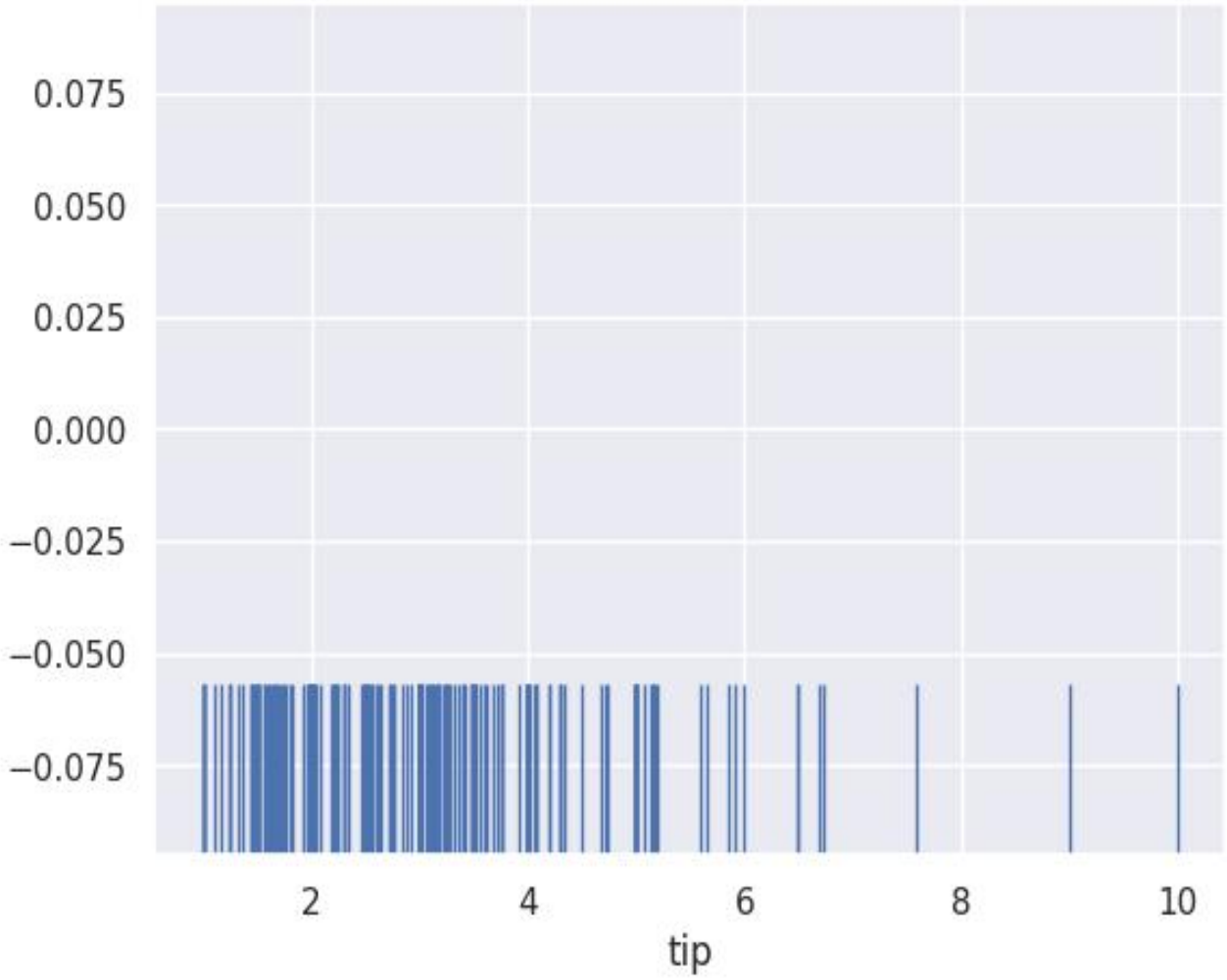
ITINERIS

```
sns.histplot(data=penguins, x="bill_length_mm", y="flipper_length_mm", hue="species")
```



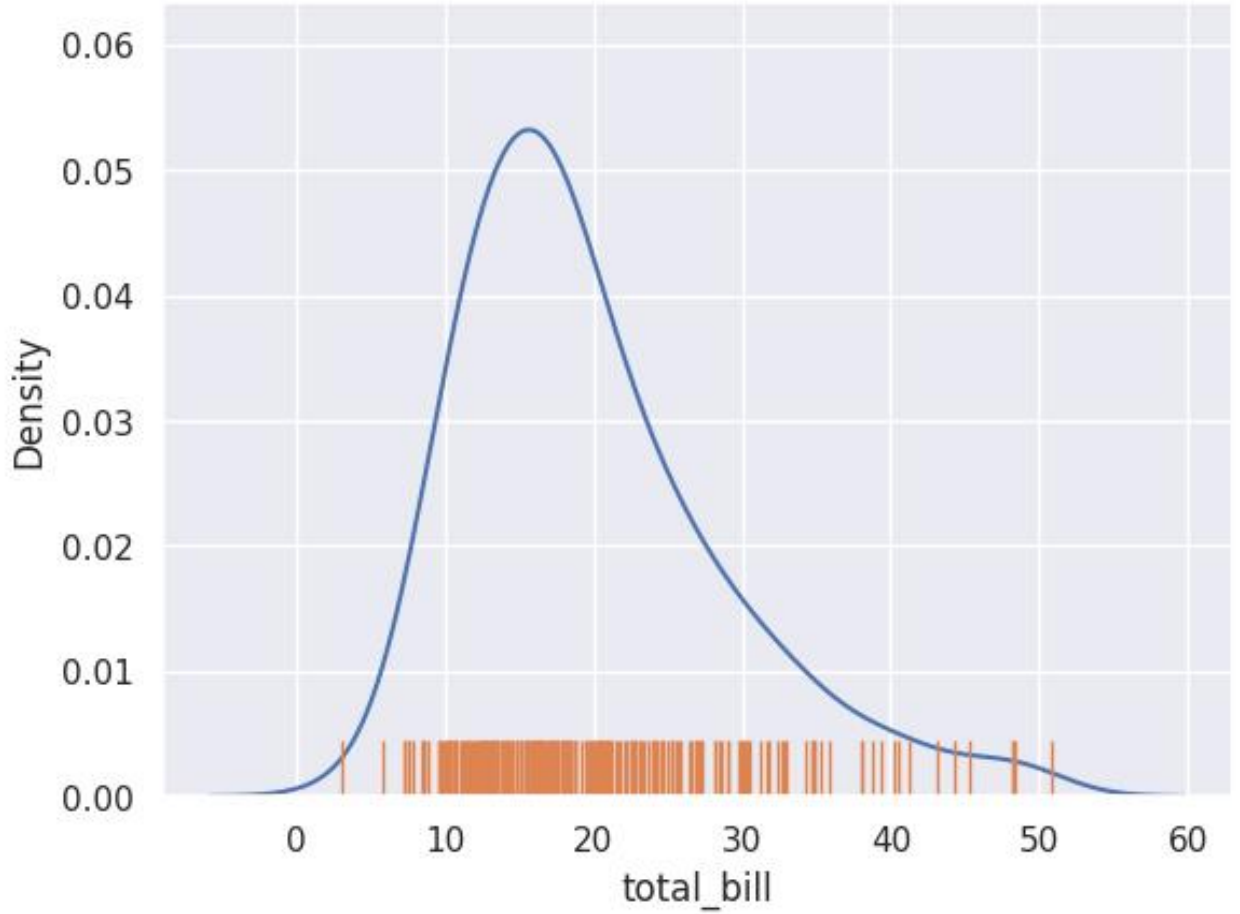
Seaborn: rugplots

```
sns.histplot(data=penguins, x="bill_length_mm", y="flipper_length_mm", hue="species")
```



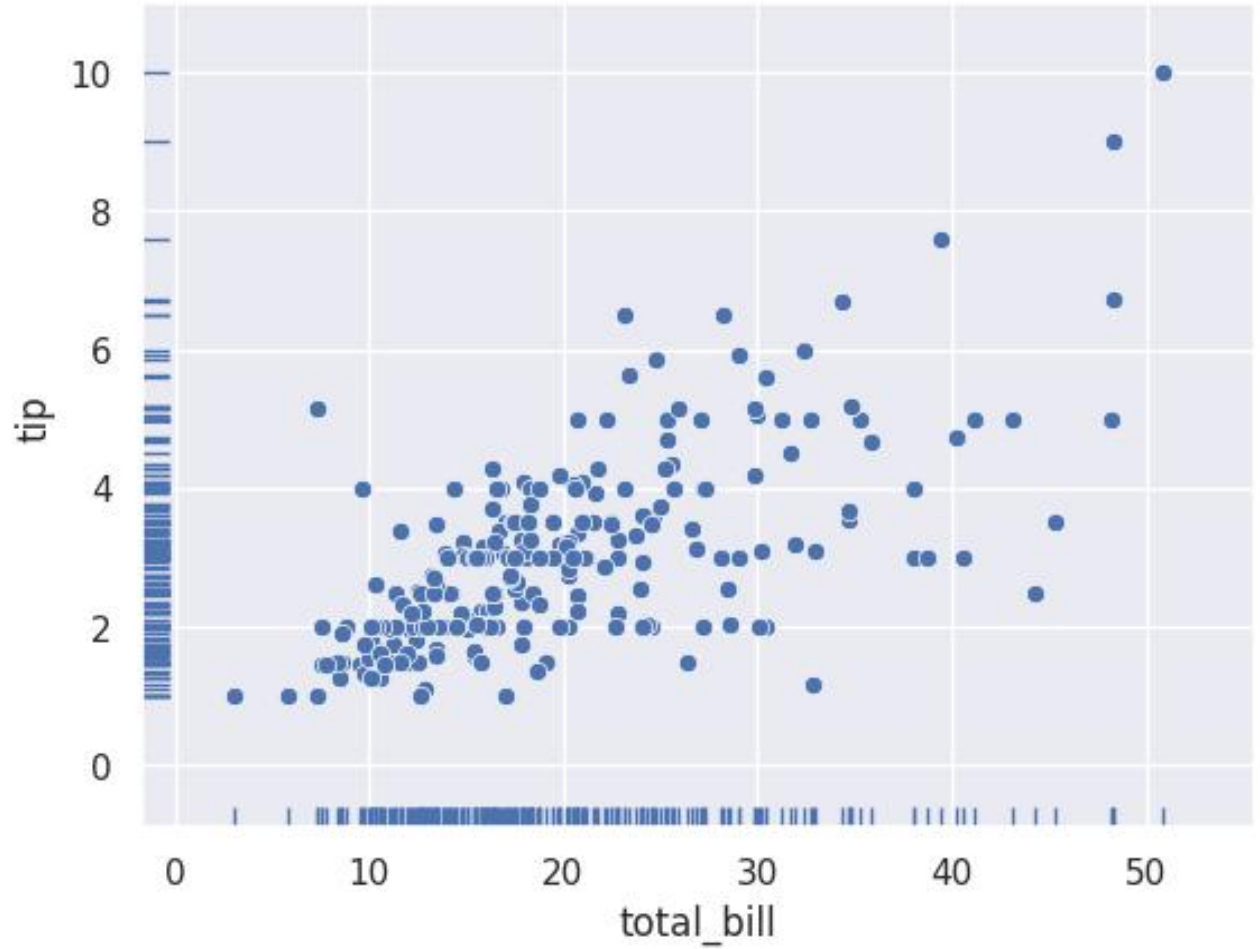
Seaborn: rugplots

```
sns.kdeplot(data=tips, x="total_bill")
sns.rugplot(data=tips, x="total_bill", height=0.07)
```



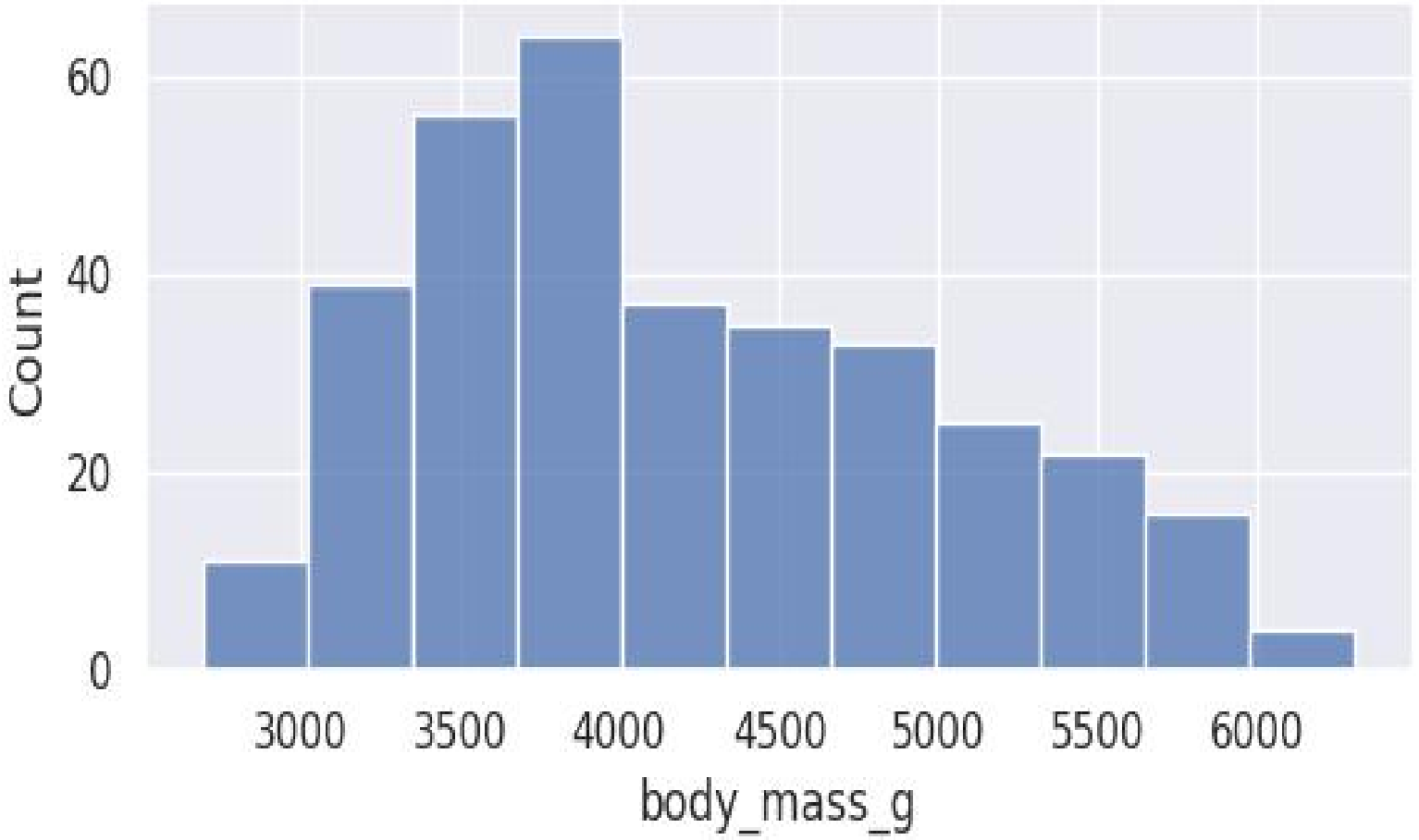
Seaborn: rugplots

```
sns.scatterplot(data=tips, x="total_bill", y="tip")
sns.rugplot(data=tips,x="total_bill", y="tip")
```



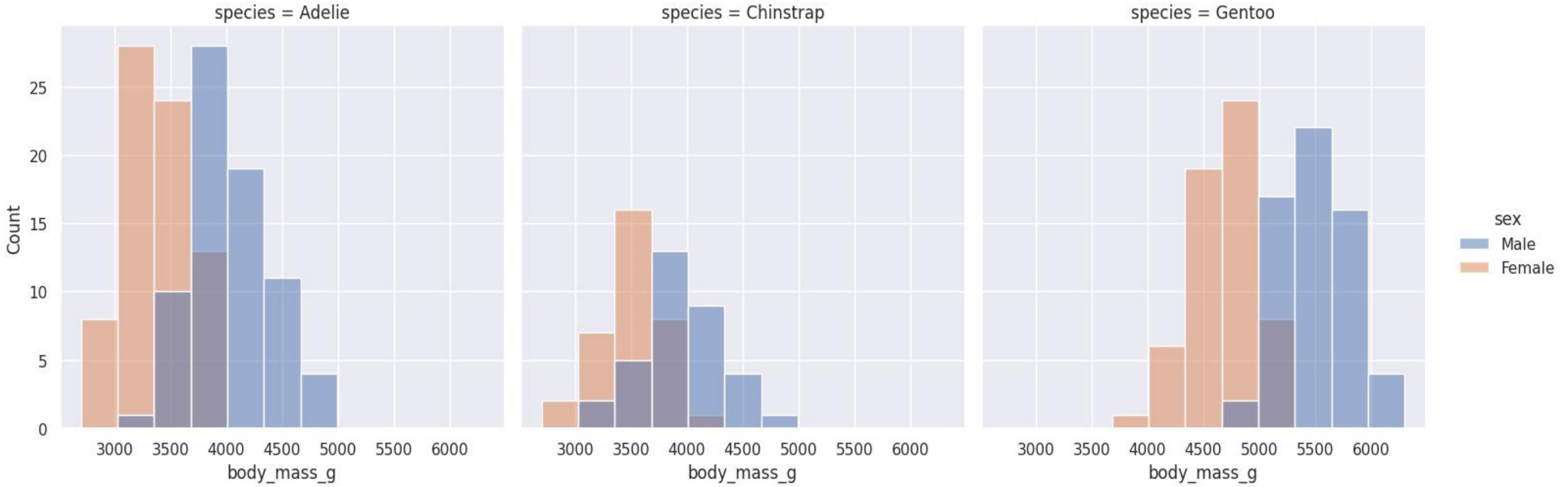
Seaborn: distribution plots

```
sns.displot(kind="hist", data=penguins, x="body_mass_g", height=3, aspect=2)
```



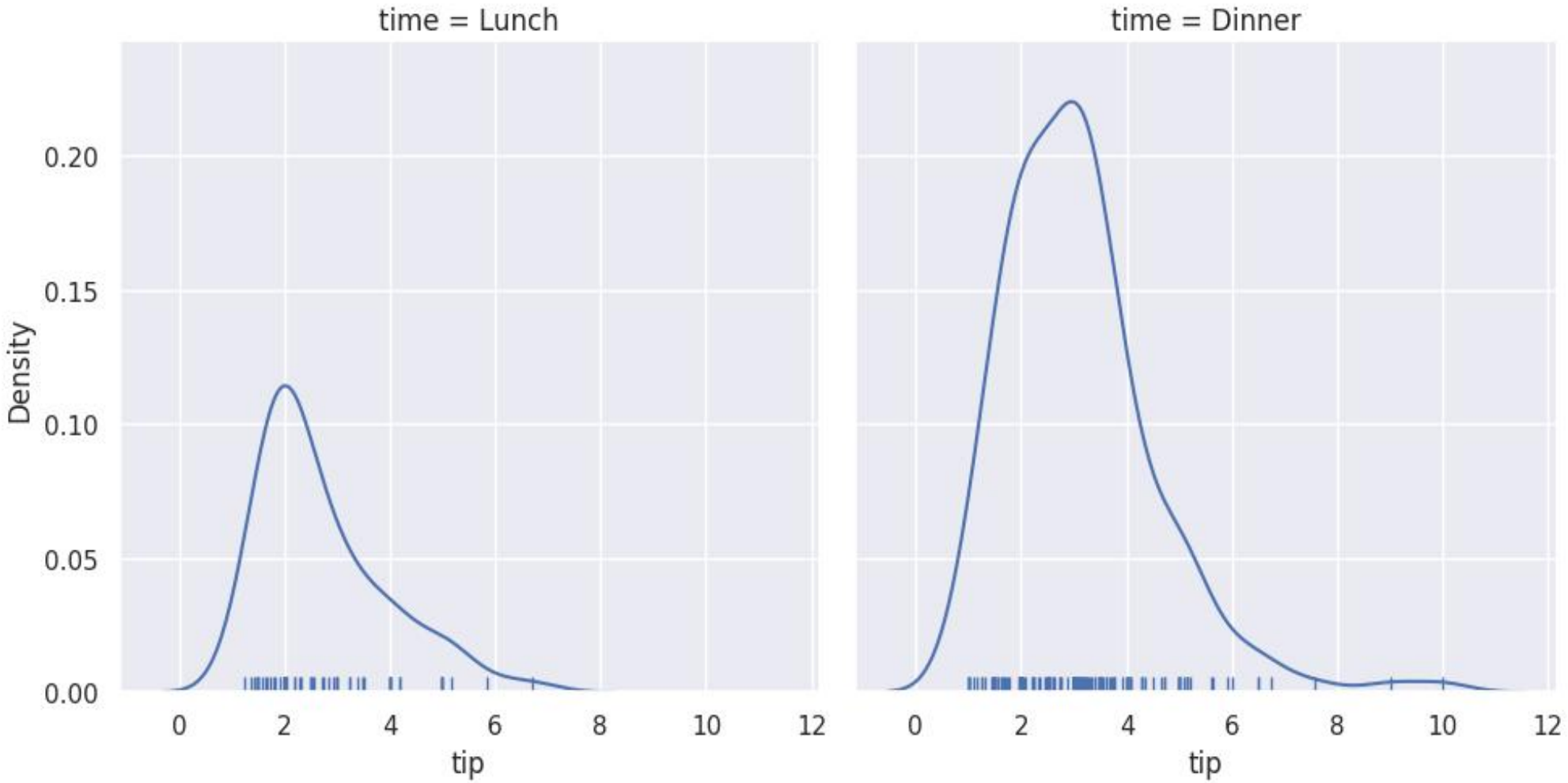
Seaborn: displots

```
sns.displot(
    kind="hist",
    data=penguins,
    hue="sex",
     x="body_mass_g",
    col="species"
)
```



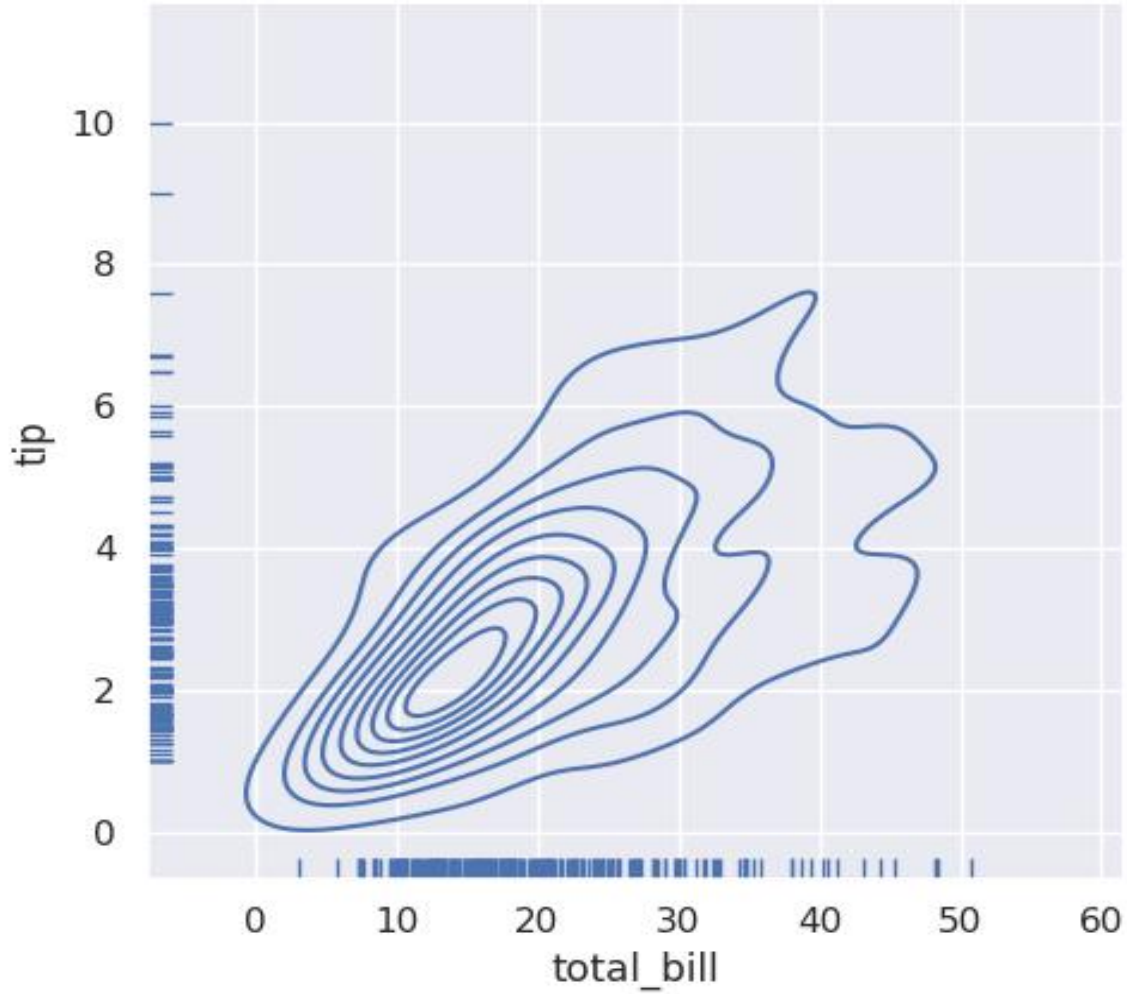
Seaborn: distribution plots

```
sns.displot(data=tips, kind="kde", x="tip", col="time", rug=True)
```



Seaborn: distribution plots

```
sns.displot(data=tips, kind="kde", x="total_bill", y="tip", rug=True)
```

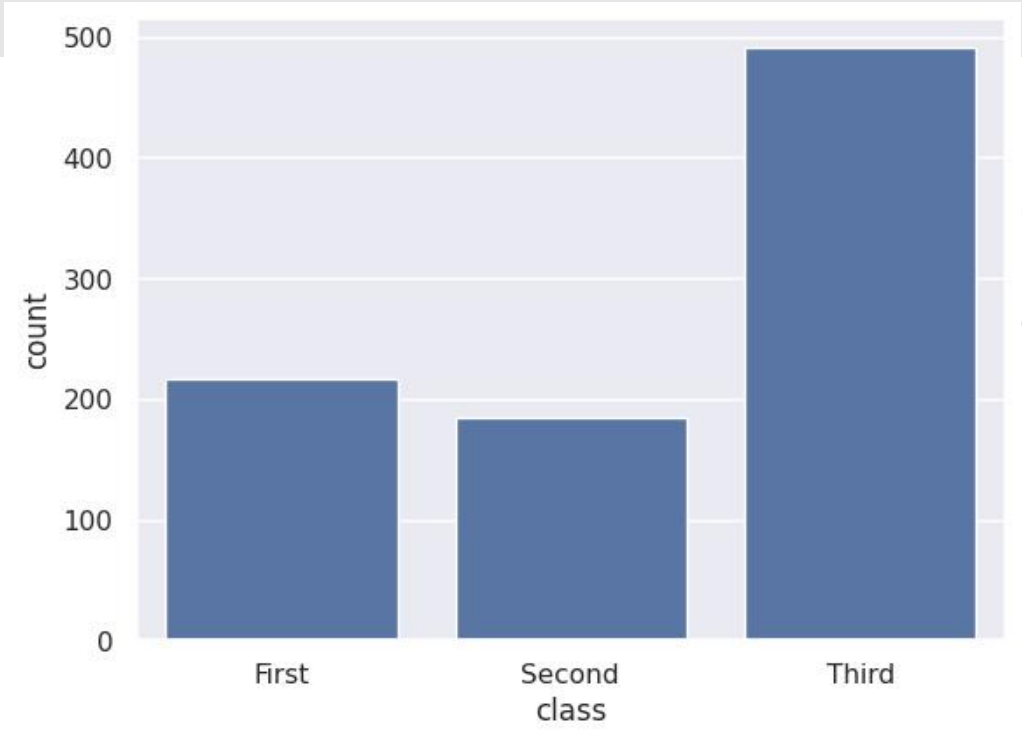


Seaborn: countplots

```
titanic = sns.load_dataset("titanic")
titanic.head()
```

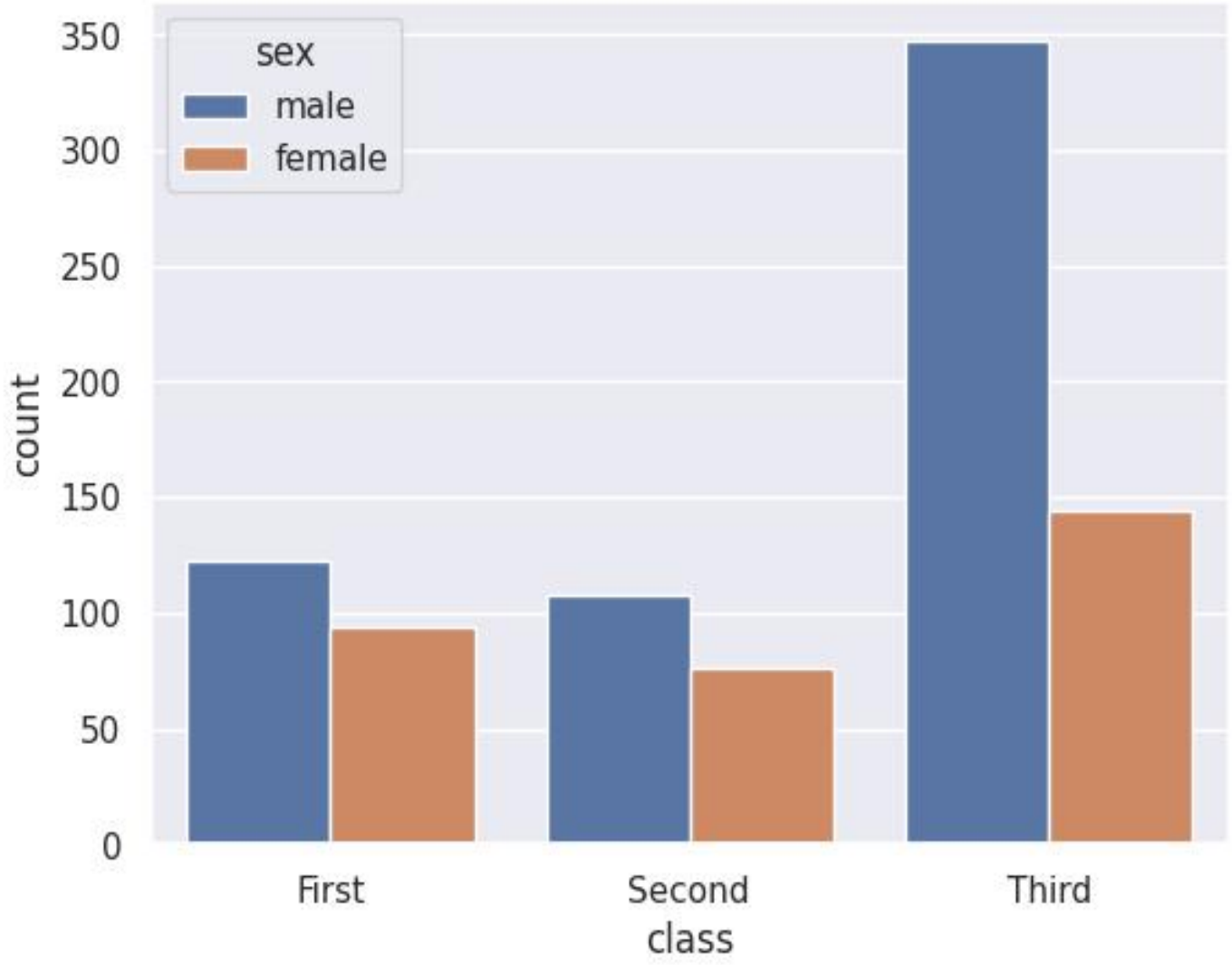
	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	al
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	

```
sns.countplot(data=titanic, x="class")
```



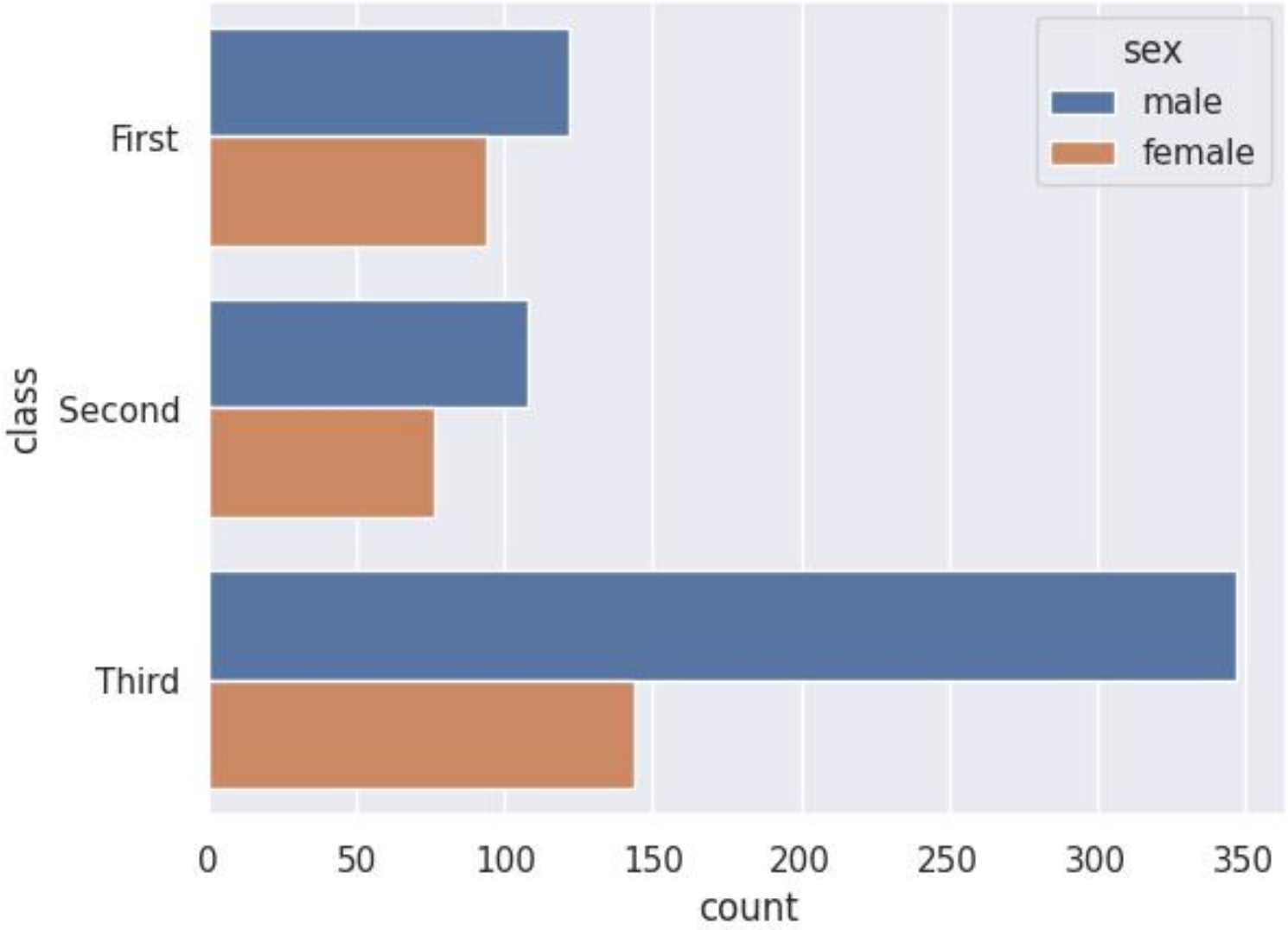
Seaborn: countplots

```
sns.countplot(data=titanic, x="class", hue="sex")
```



Seaborn: countplots

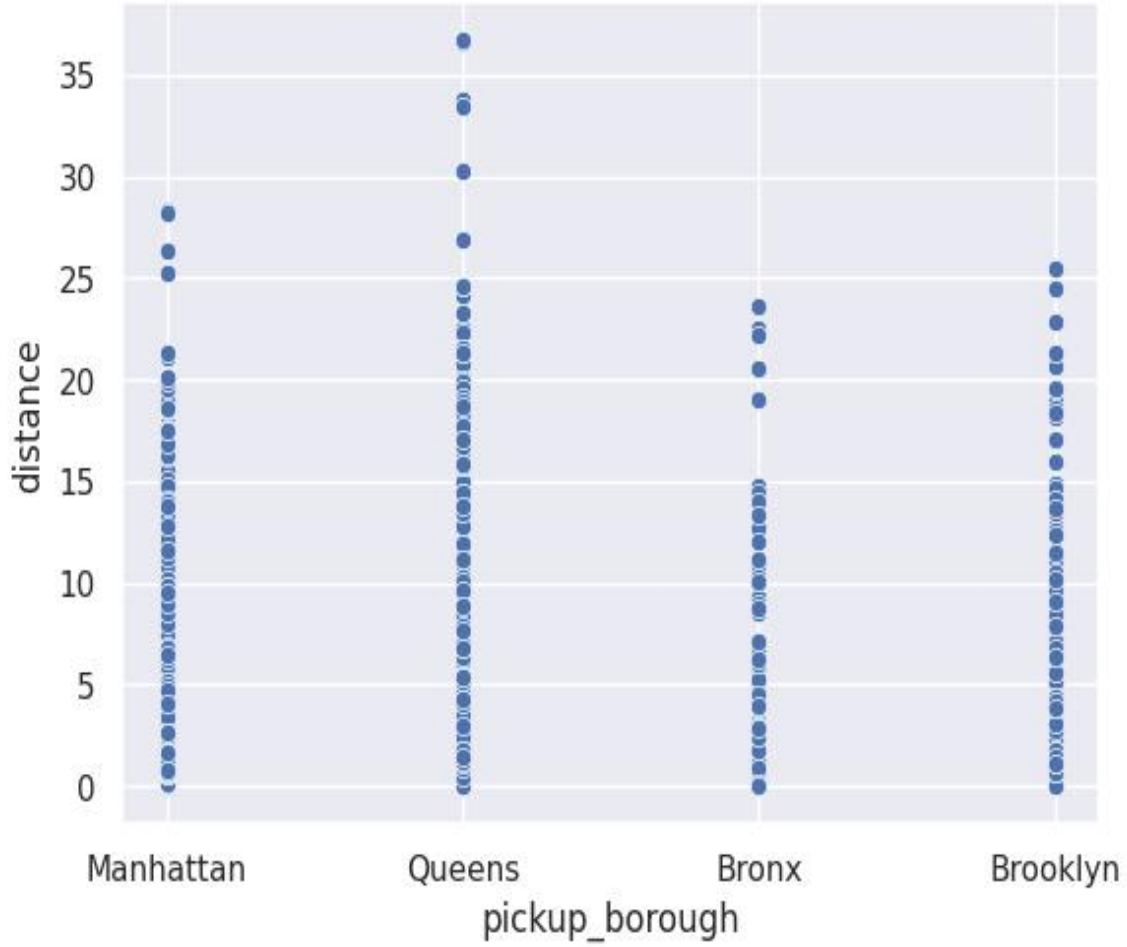
```
sns.countplot(data=titanic, y="class", hue="sex")
```



Seaborn: stripplots

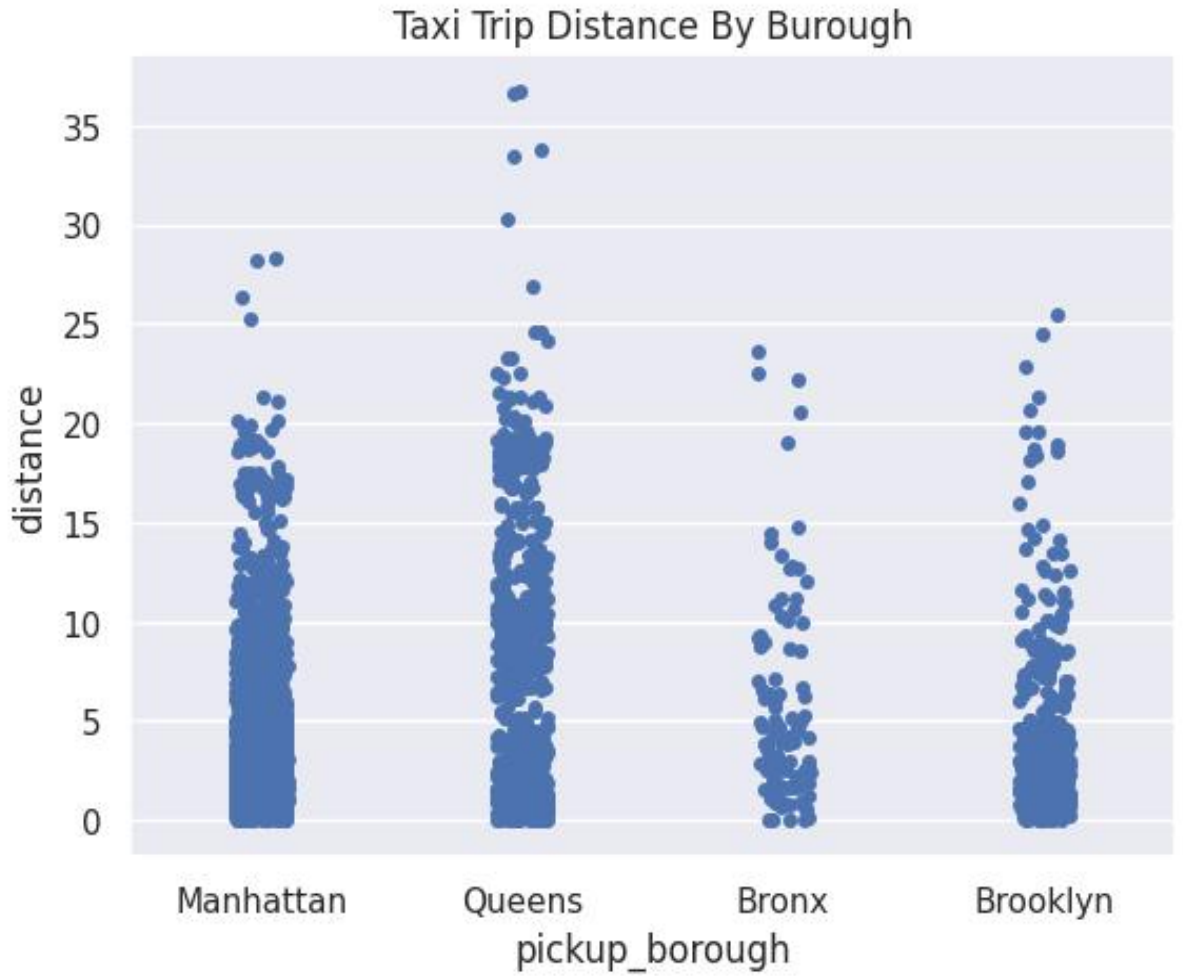
Scatterplot are not quite useful when treating categorical data:

```
sns.scatterplot(data=trips, x="pickup_borough", y="distance")
```



Seaborn: stripplots

```
plt.figure(dpi=100)
sns.stripplot(data=trips, x="pickup_borough", y="distance")
plt.title("Taxi Trip Distance By Burough")
```



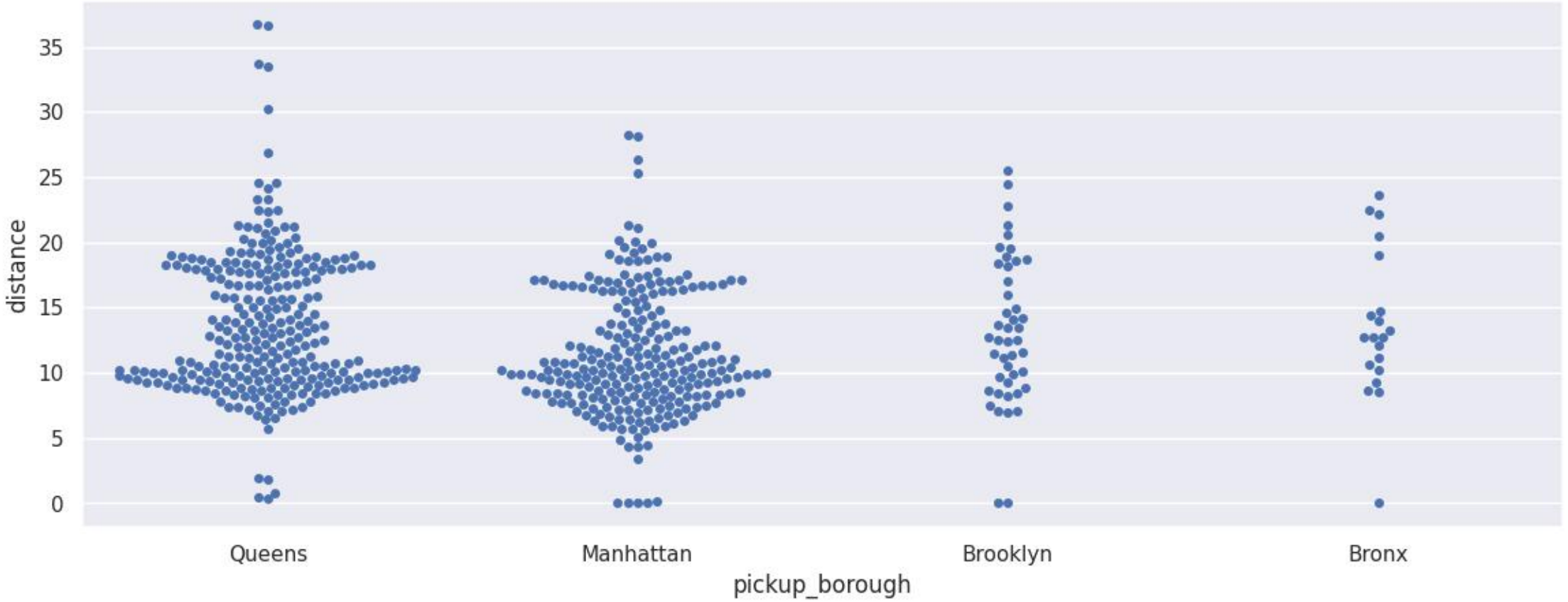
Seaborn: stripplots

```
trips_sample = trips.nlargest(600, "total")
```

```
plt.figure(figsize=(12,5))
sns.stripplot(data=trips_sample, x="pickup_borough", y="distance")
plt.title("Taxi Trips By Borough")
```

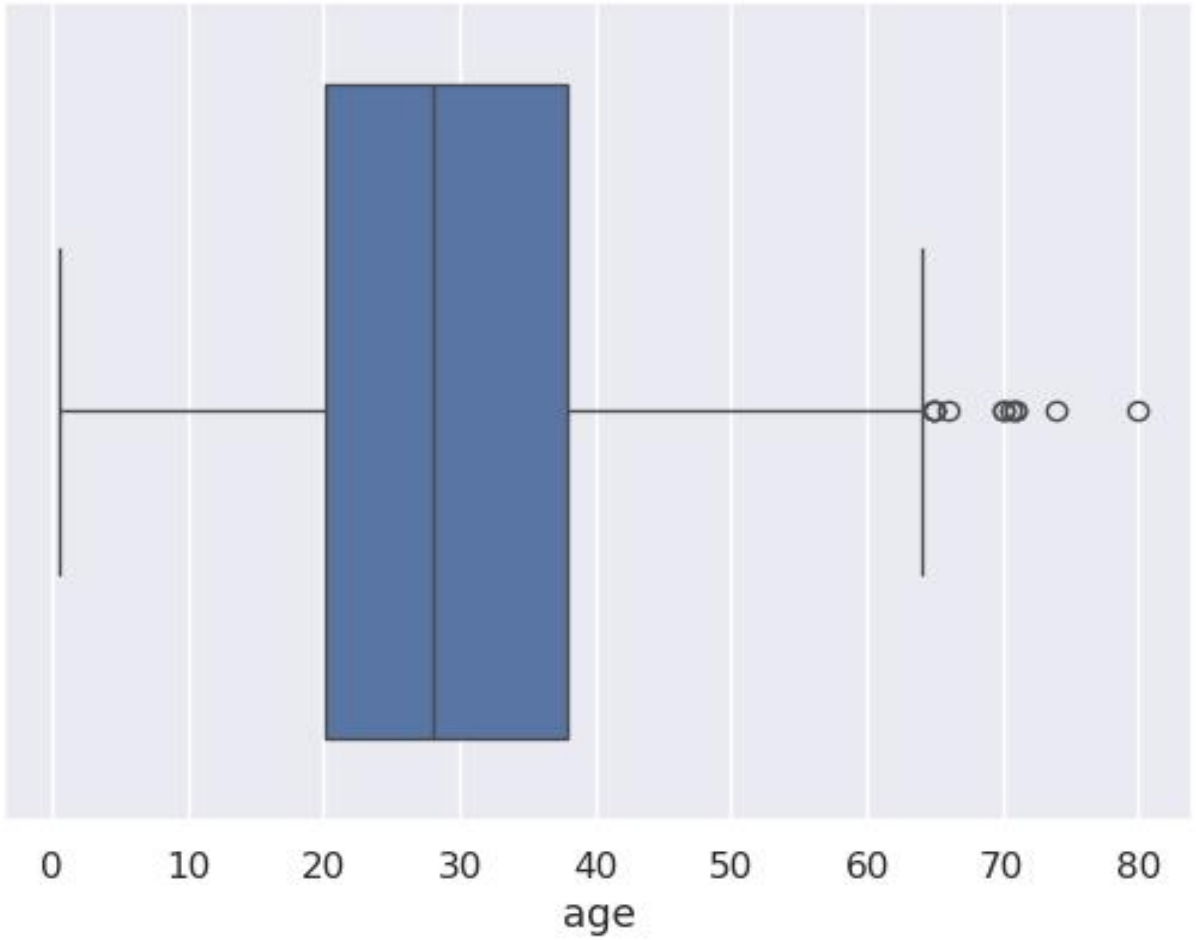
Seaborn: swarmplots

```
plt.figure(figsize=(14,5))
sns.swarmplot(data=trips_sample, x="pickup_borough", y="distance")
```



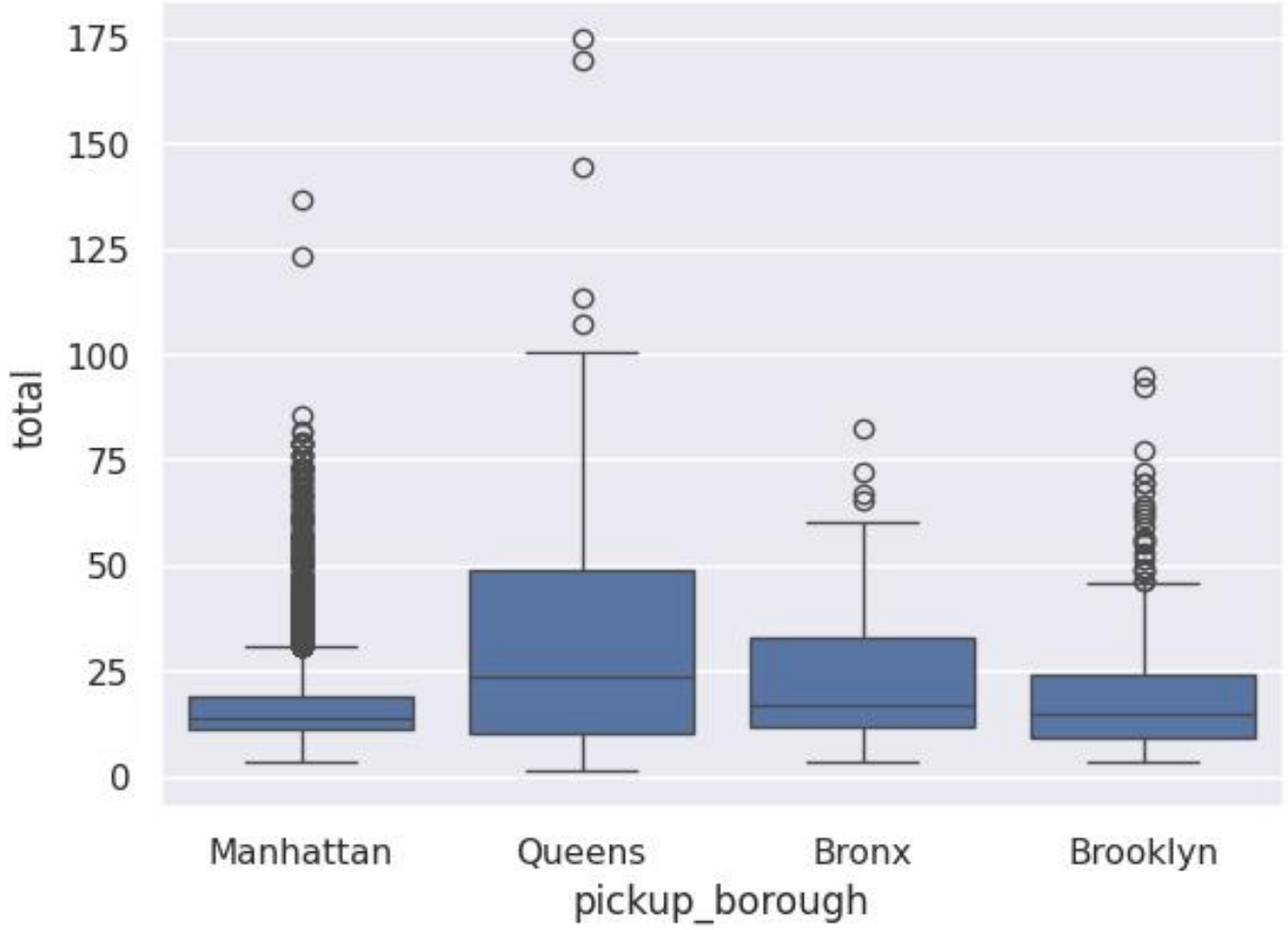
Seaborn: boxplots

```
sns.boxplot(data=titanic, x="age")
```



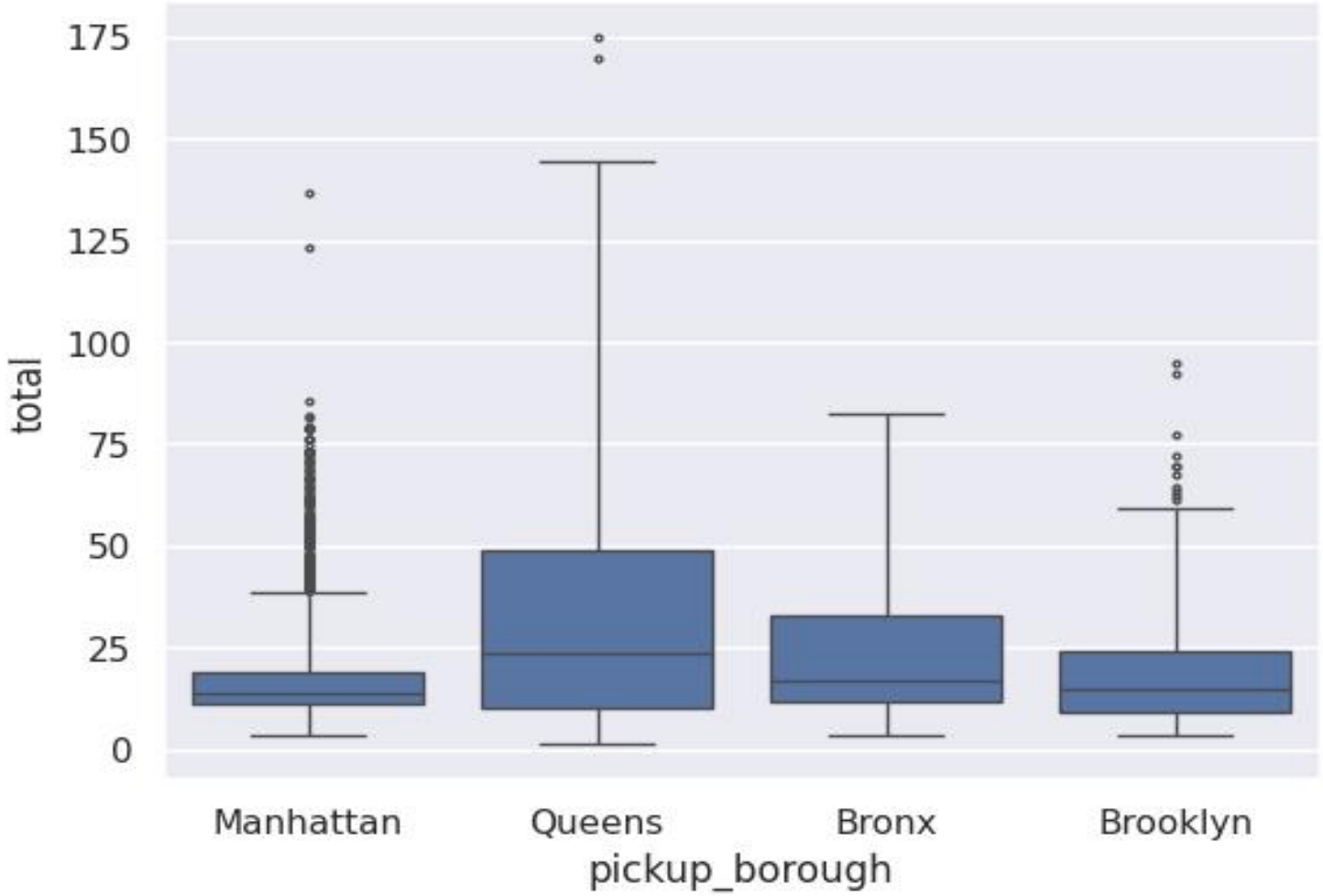
Seaborn: boxplots

```
sns.boxplot(data=trips, x="pickup_borough", y="total")
```



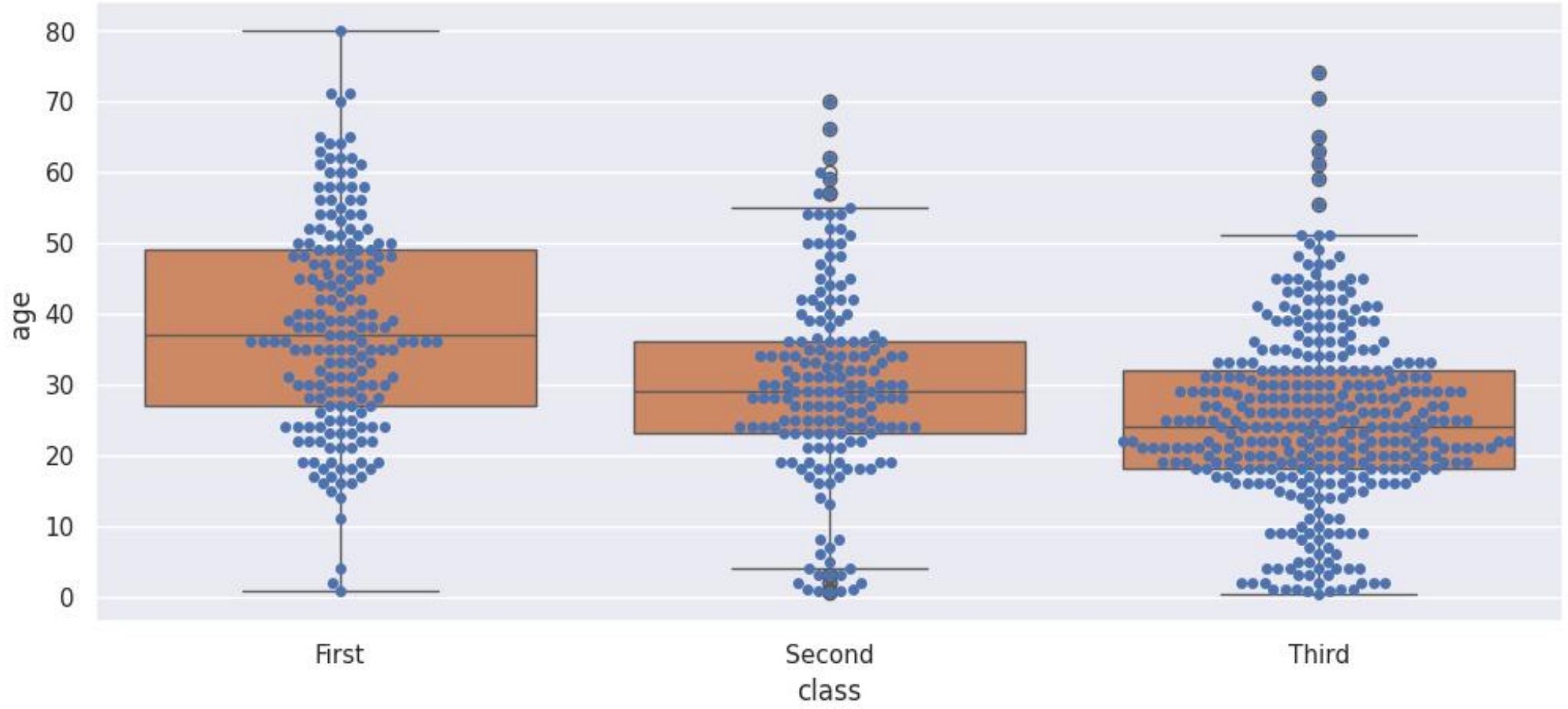
Seaborn: boxplots

```
sns.boxplot(data=trips, x="pickup_borough", y="total", whis=2.5, fliersize=2)
```



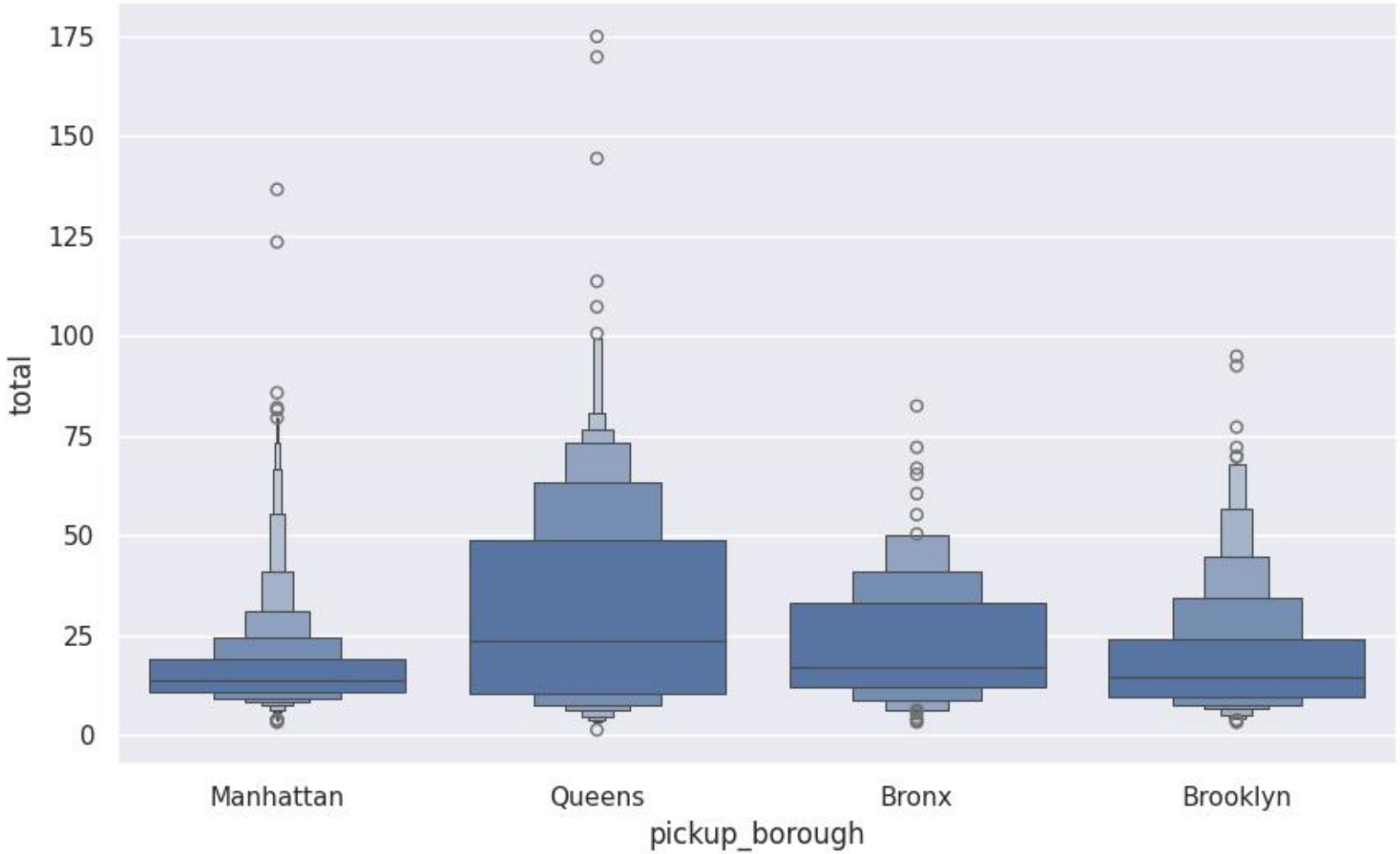
Seaborn: boxplots

```
plt.figure(figsize=(12,5))
sns.swarmplot(data=titanic, x="class", y="age")
sns.boxplot(data=titanic, x="class", y="age")
```



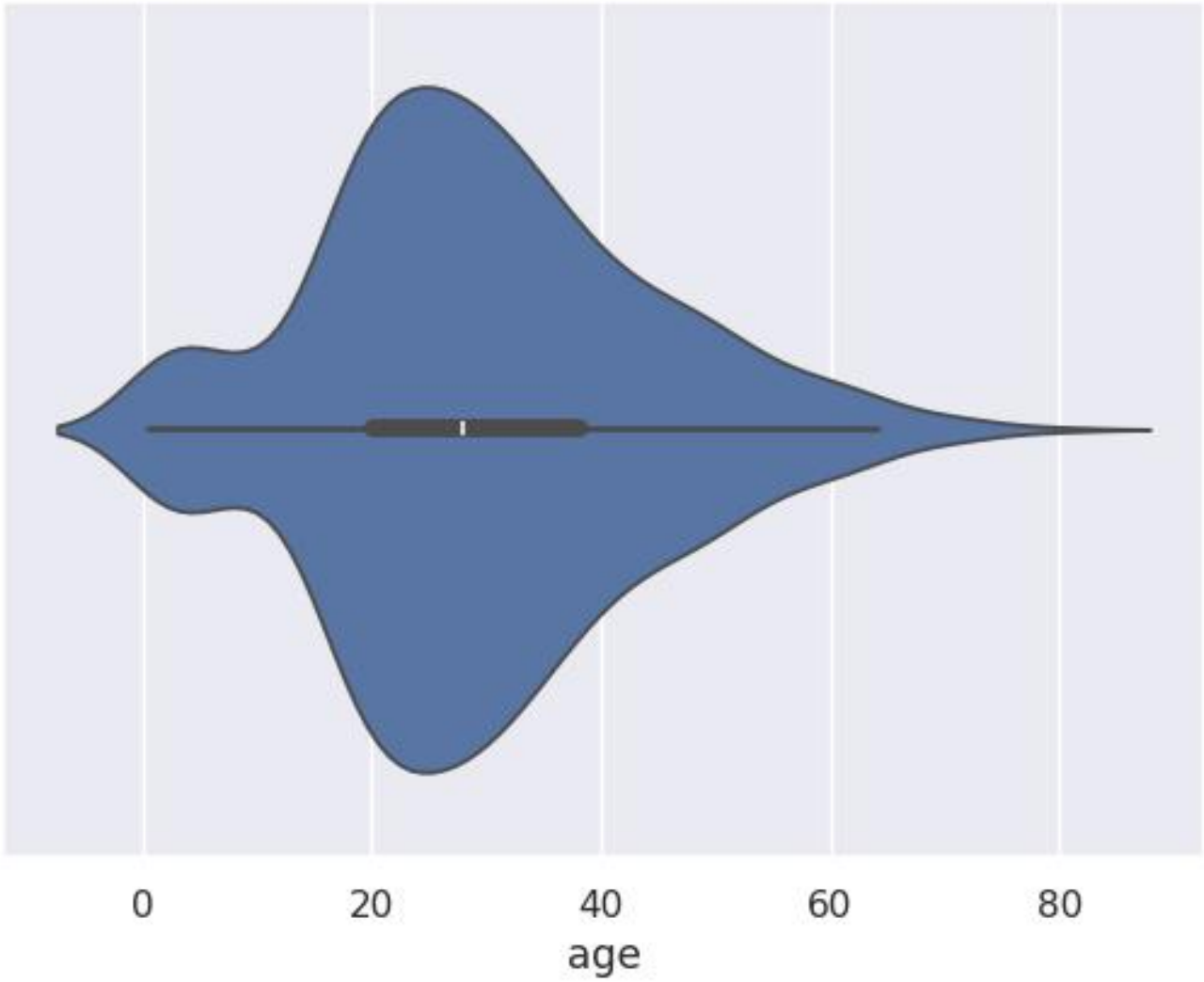
Seaborn: boxenplots

```
plt.figure(figsize=(10,6))
sns.boxenplot(data=trips, x="pickup_borough", y="total")
```



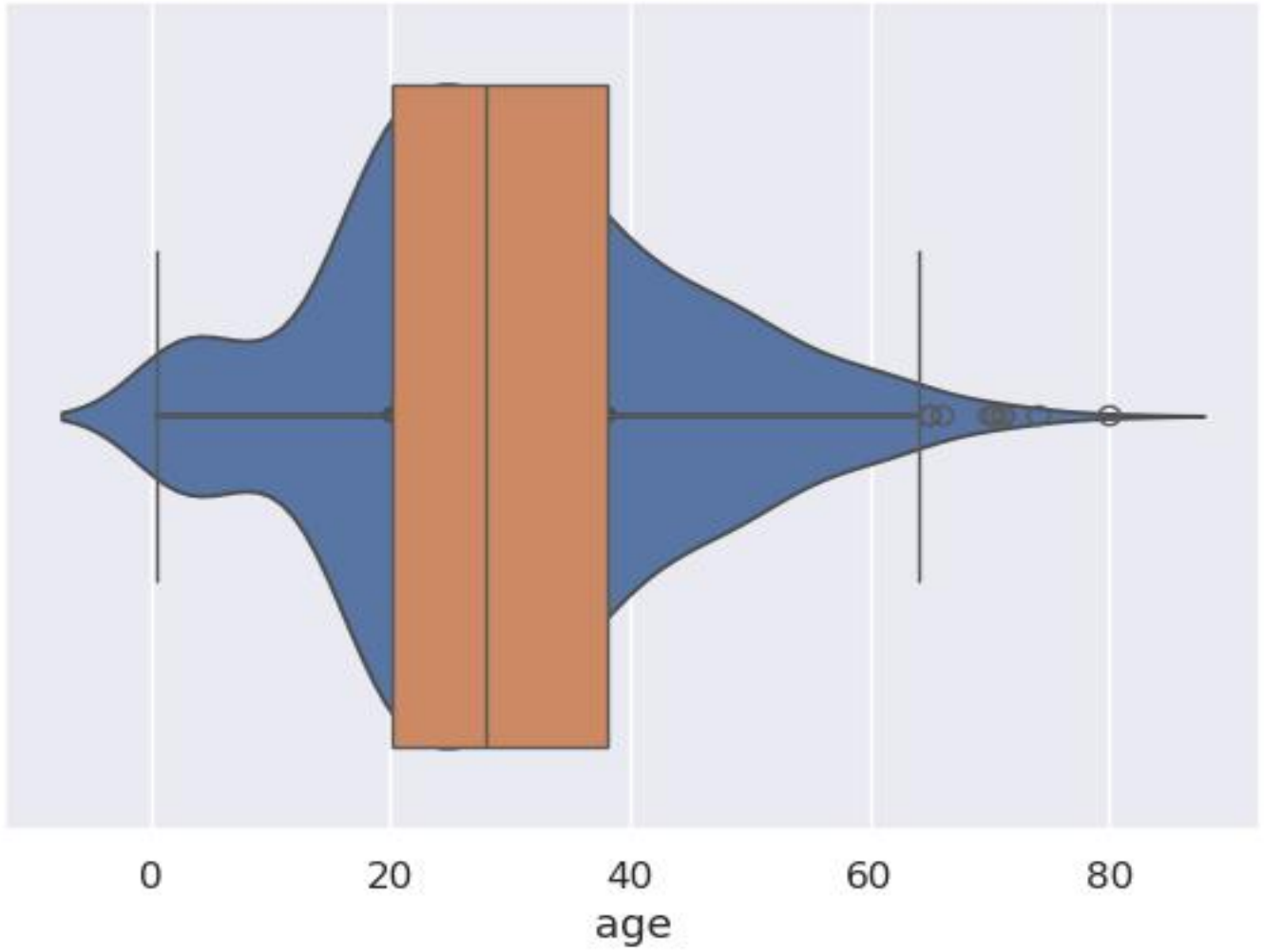
Seaborn: violinplots

```
sns.violinplot(data=titanic, x="age")
```



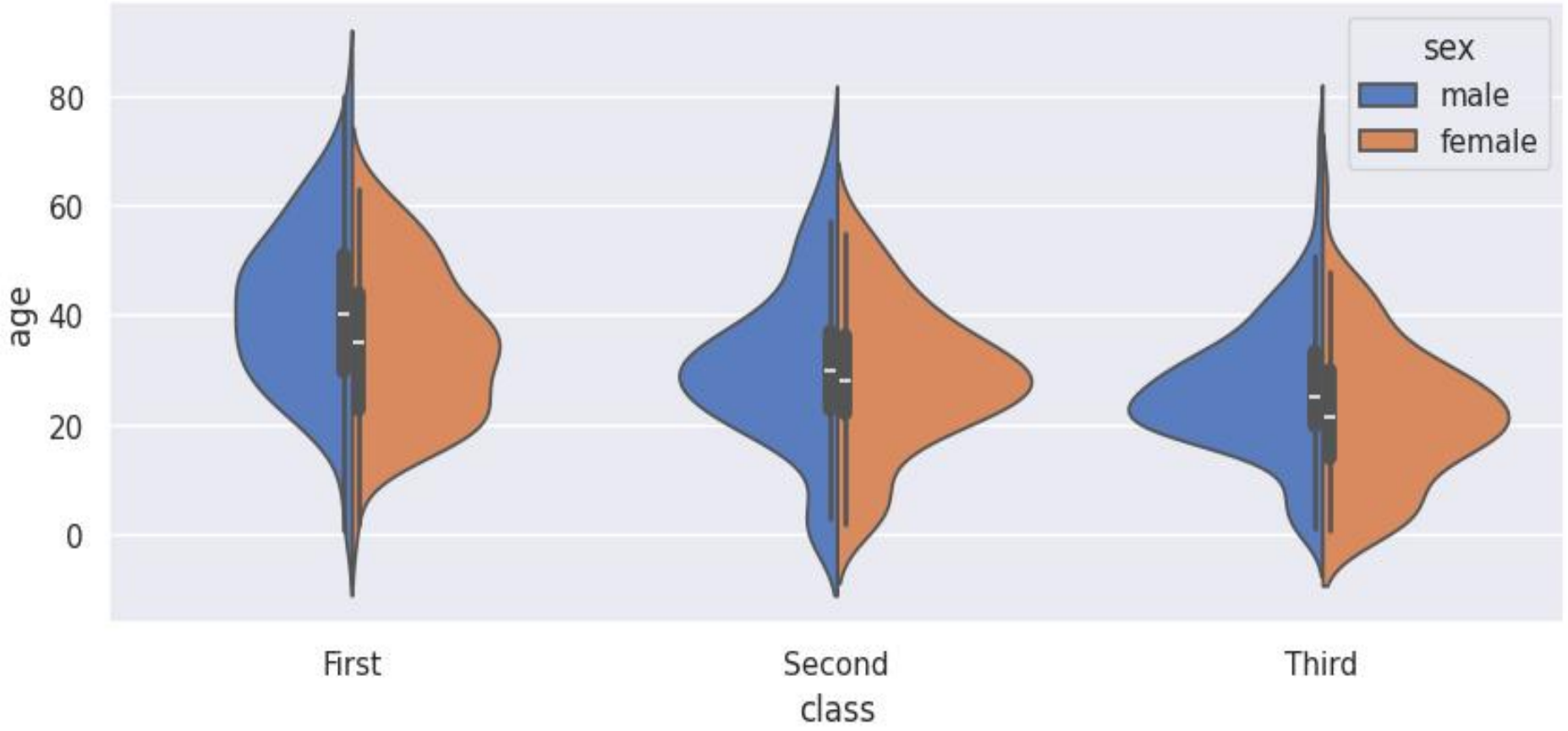
Seaborn: violinplots

```
sns.violinplot(data=titanic, x="age")
sns.boxplot(data=titanic, x="age")
```



Seaborn: violinplots

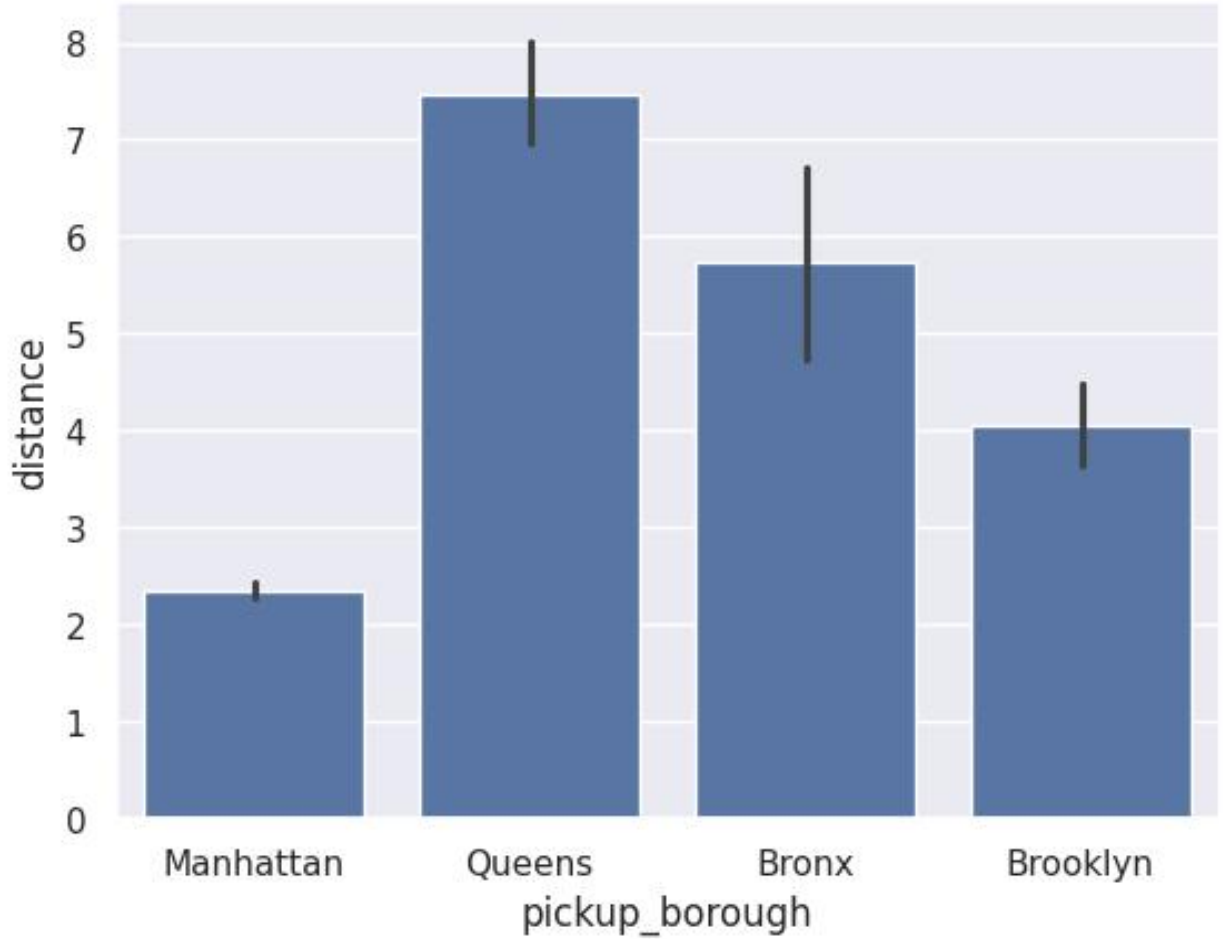
```
plt.figure(figsize=(10,4))
sns.violinplot(data=titanic, x="class", y="age", hue="sex", split=True, palette="muted")
```



Seaborn: barplots

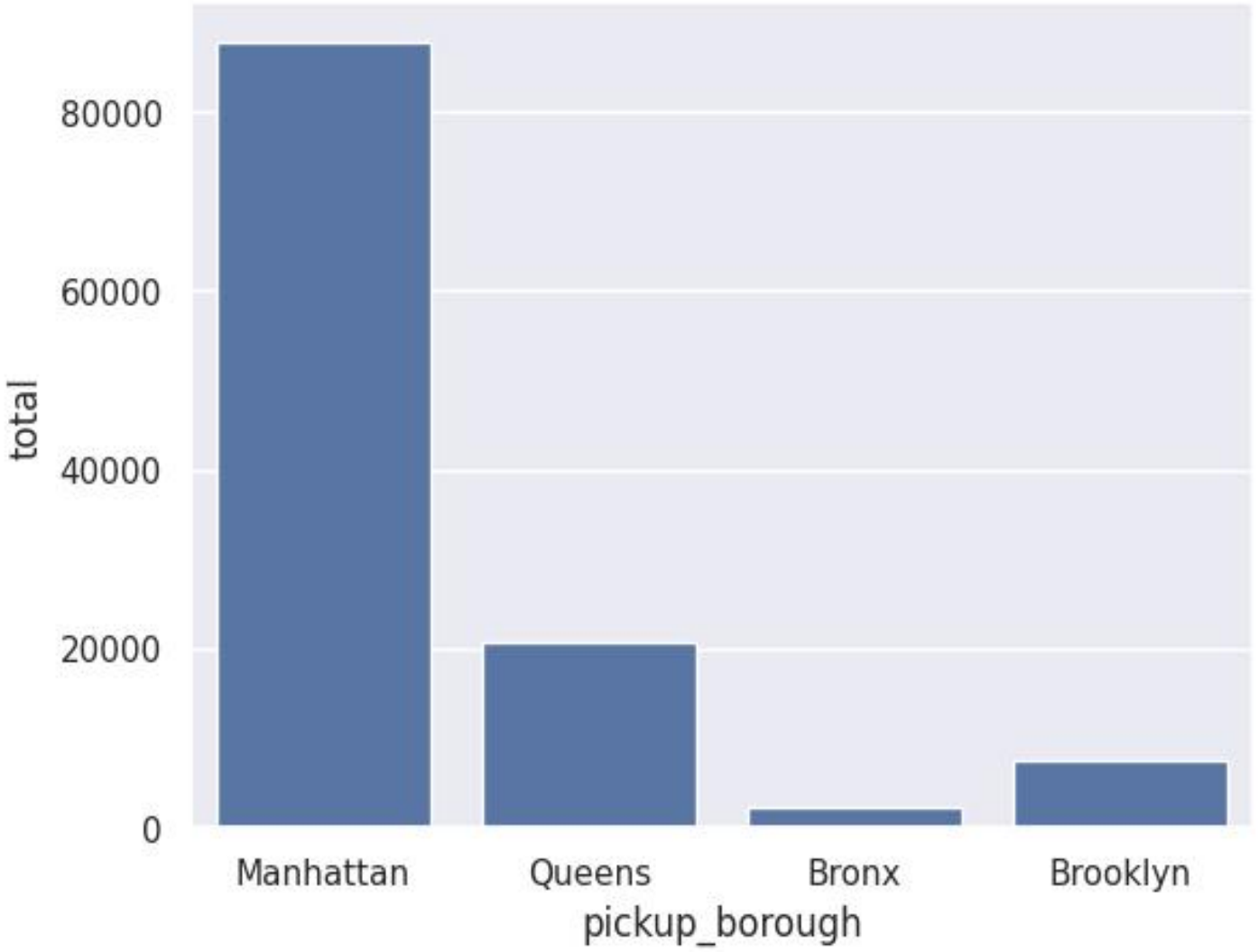
Barplot give a different output in respect to other packages. By default, seaborn aggregates data using the mean.

```
sns.barplot(data=trips, x="pickup_borough", y="distance")
```



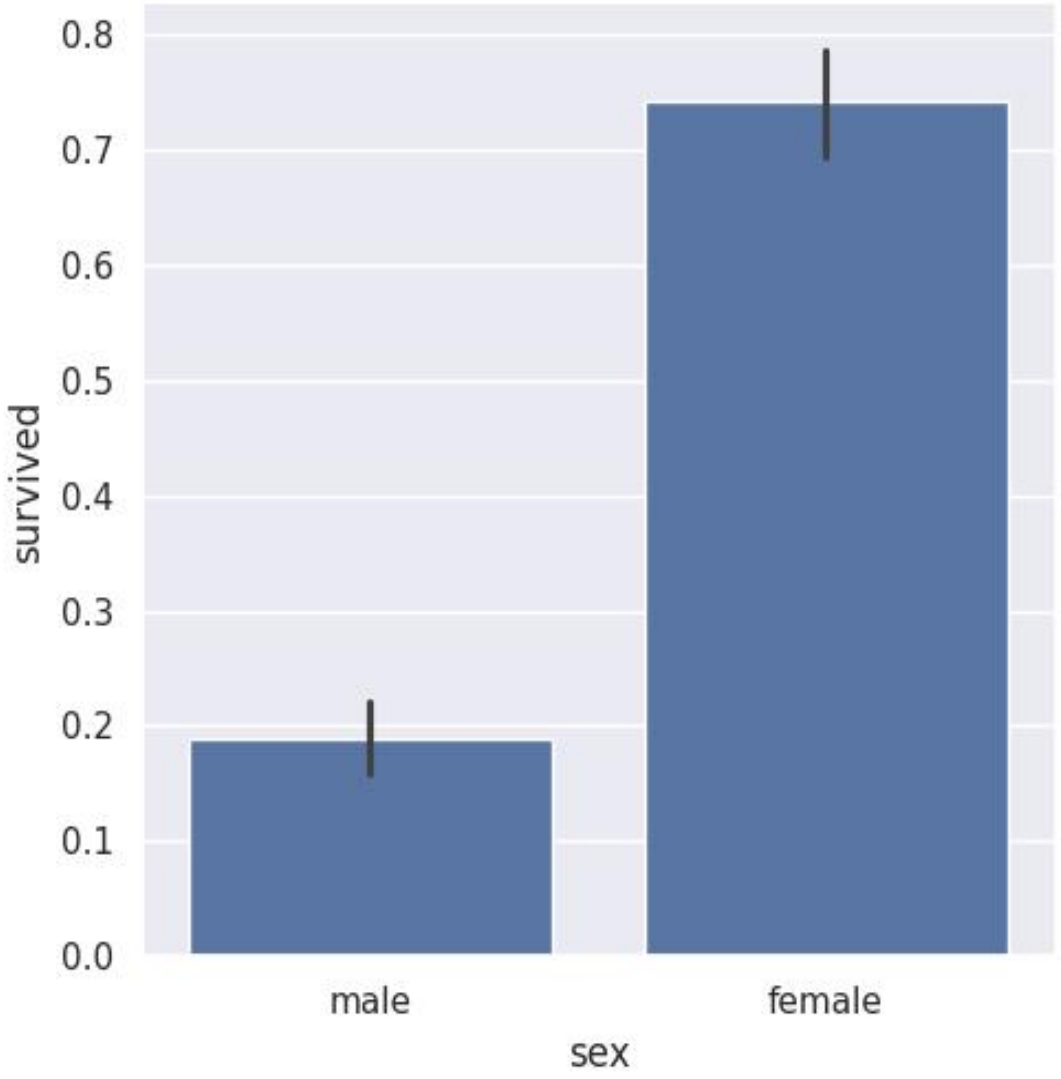
Seaborn: barplots

```
sns.barplot(data=trips, x="pickup_borough", y="total", estimator=sum, ci=None)
```



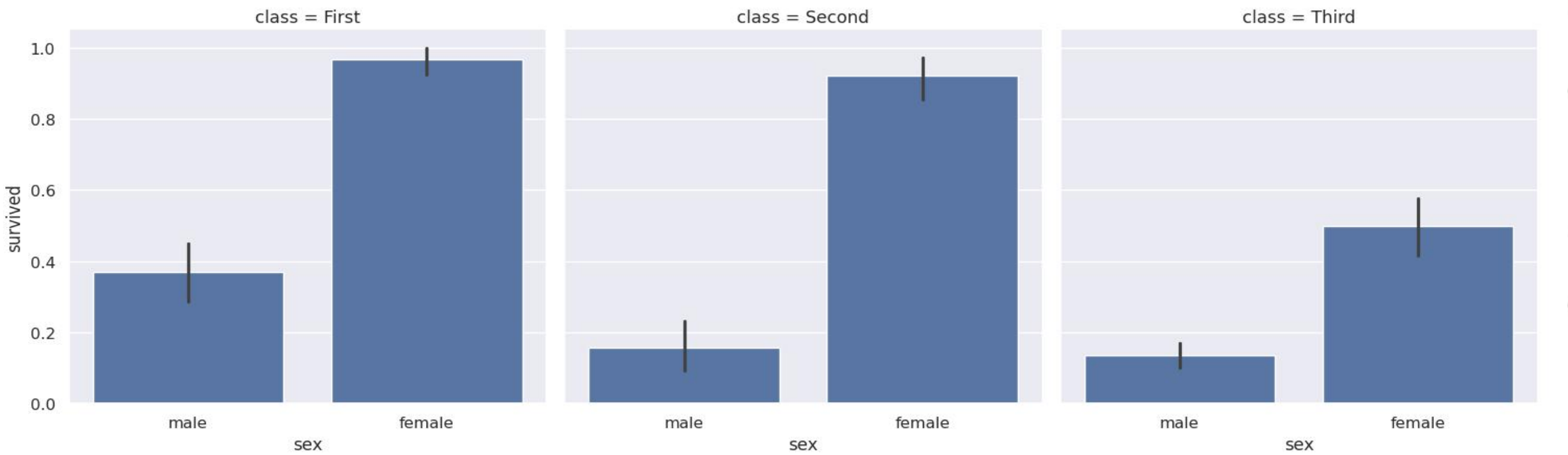
Seaborn: categorical plots

```
sns.catplot(data=titanic, x="sex", y="survived", kind="bar")
```



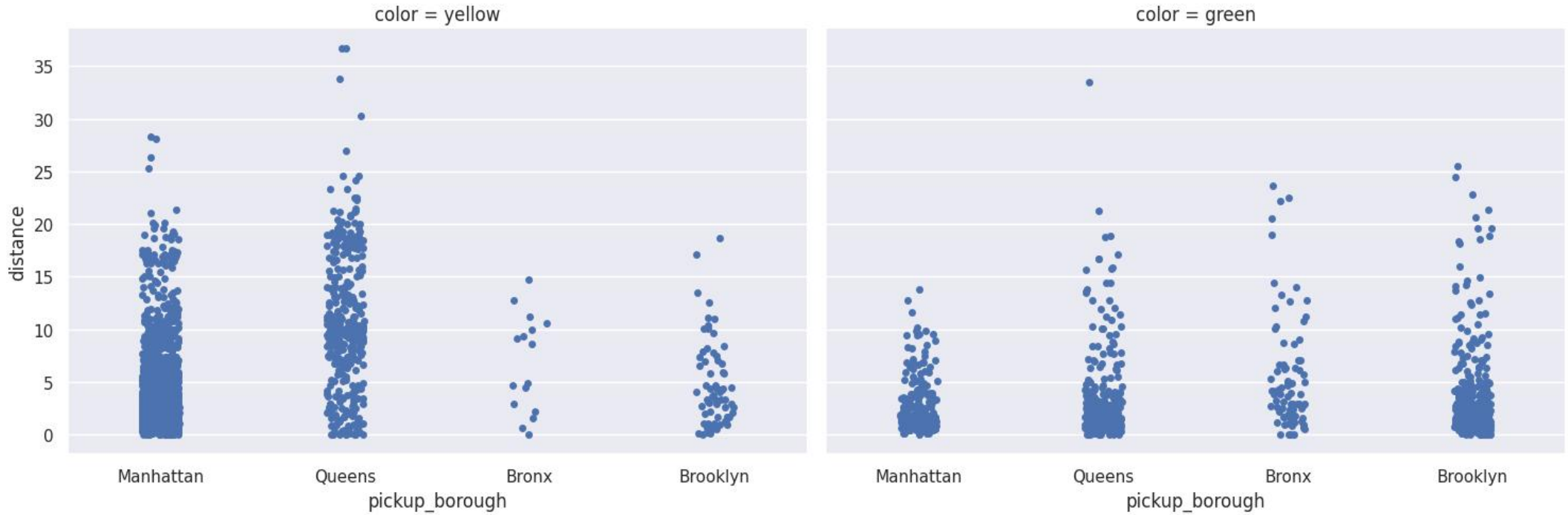
Seaborn: catplots

```
sns.catplot(data=titanic, x="sex", y="survived", kind="bar", col="class")
```



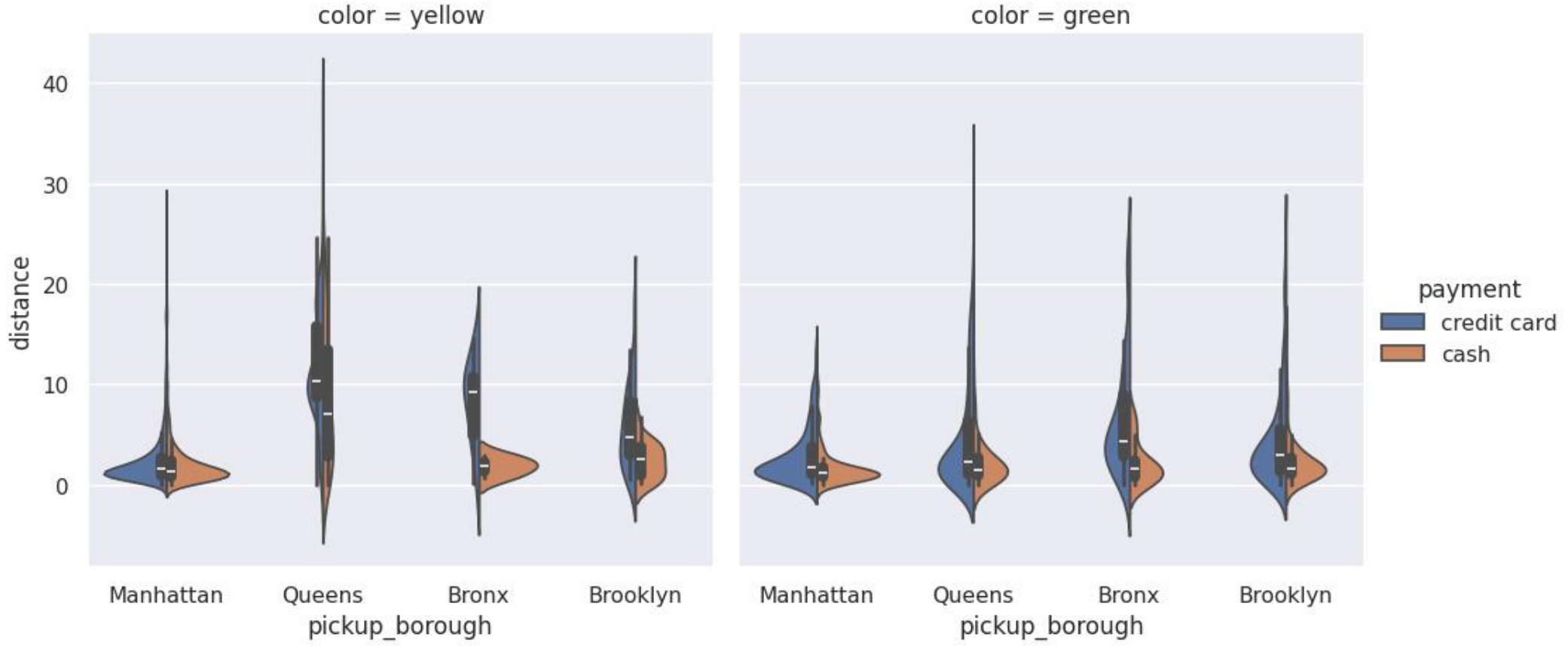
Seaborn: catplots

```
sns.catplot(
    data=trips,
    kind="strip",
    x="pickup_borough",
    y="distance",
    col="color",
    height=5,
    aspect=1.5
)
```



Seaborn: catplots

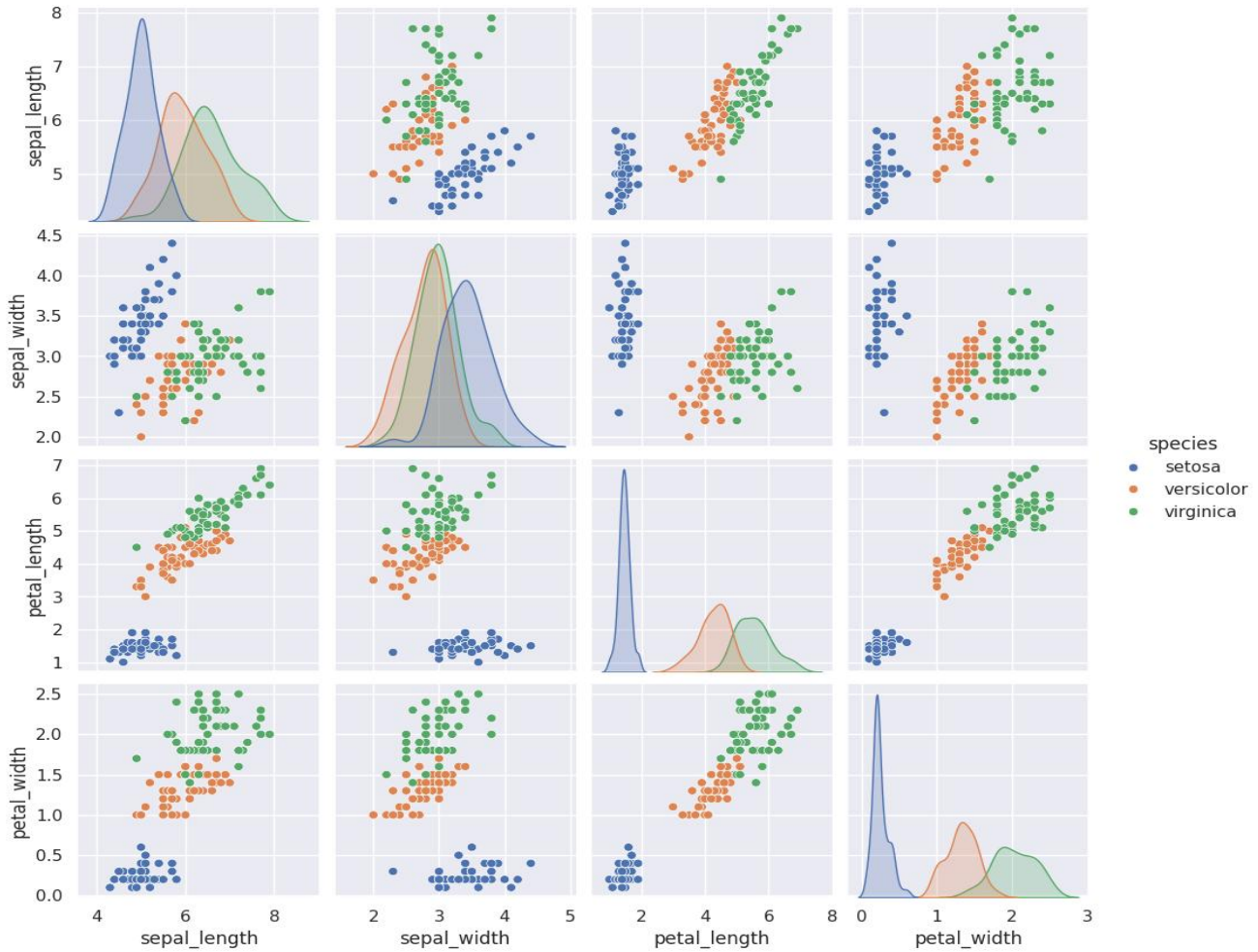
```
sns.catplot(data=trips,
            kind="violin",
            x="pickup_borough",
            y="distance",
            hue="payment",
            split=True,
            col="color")
```



Seaborn: pairplots

```
iris = sns.load_dataset("iris")
```

```
sns.pairplot(data=iris, hue="species")
```



Exercise

Load the Penguins dataset using Seaborn.

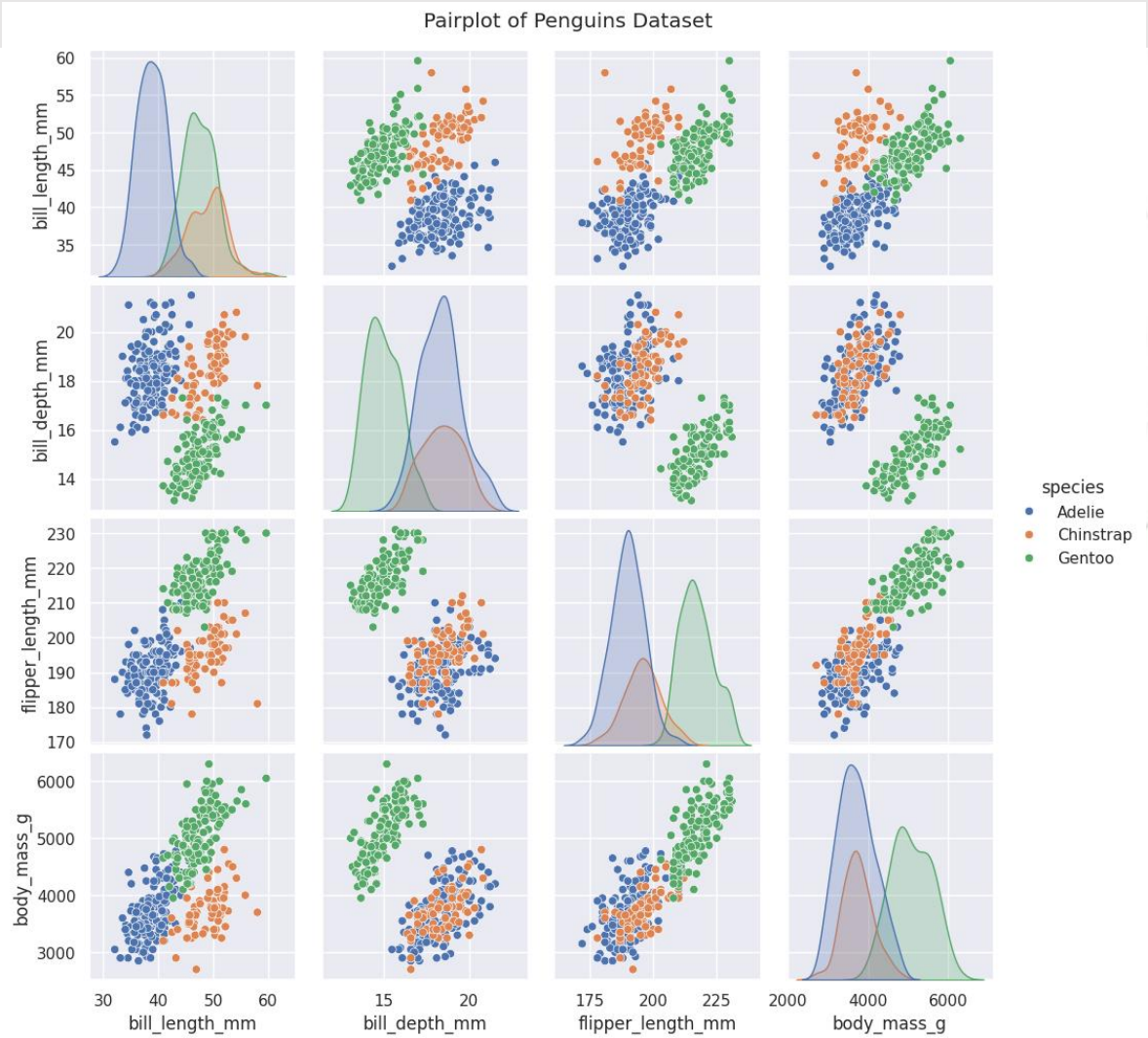
1. Create a pairplot to visualize the relationships between the numerical variables in the dataset, differentiating between species. Use the Seaborn pairplot function. Set the hue parameter to differentiate the species.

2. Create a swarmplot to visualize the distribution of body mass across different species and overlay it with a boxplot for additional statistical insights. Use the Seaborn swarmplot function to create the swarmplot. Overlay a boxplot on top of the swarmplot using the boxplot function. Ensure that the plots are clearly distinguishable by adjusting the transparency (alpha) of the boxplot.

3. If you had to build a classifier, on which features would you focus?

Solution

1.
```
# Create the pairplot
sns.pairplot(penguins, hue='species')
plt.suptitle('Pairplot of Penguins Dataset', y=1.02)
plt.show()
```



Solution

2.

```
# Create the swarmplot
sns.swarmplot(x='species', y='body_mass_g', data=penguins, color='blue', alpha=0.6)

# Overlay the boxplot
sns.boxplot(x='species', y='body_mass_g', data=penguins)

plt.title('Body Mass by Species')
plt.show()
```

THANKS!

IR0000032 – ITINERIS, Italian Integrated Environmental Research Infrastructures System
(D.D. n. 130/2022 - CUP B53C22002150006) Funded by EU - Next Generation EU PNRR-Mission 4 “Education and Research” - Component 2: “From research to business” - Investment 3.1: “Fund for the realisation of an integrated system of research and innovation infrastructures”

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani