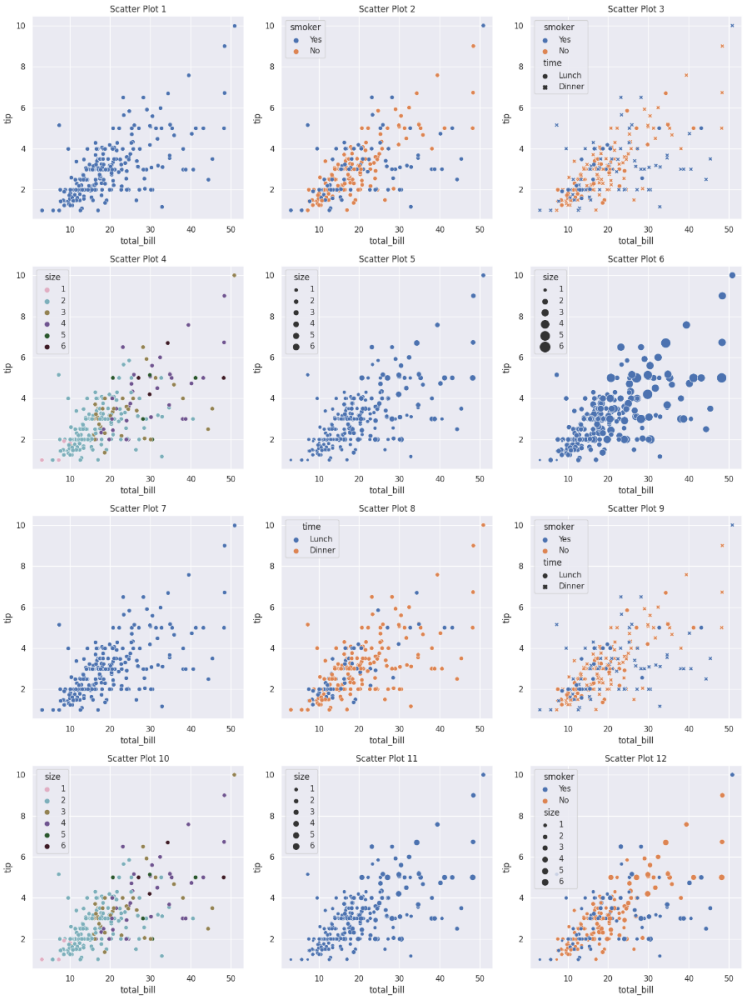
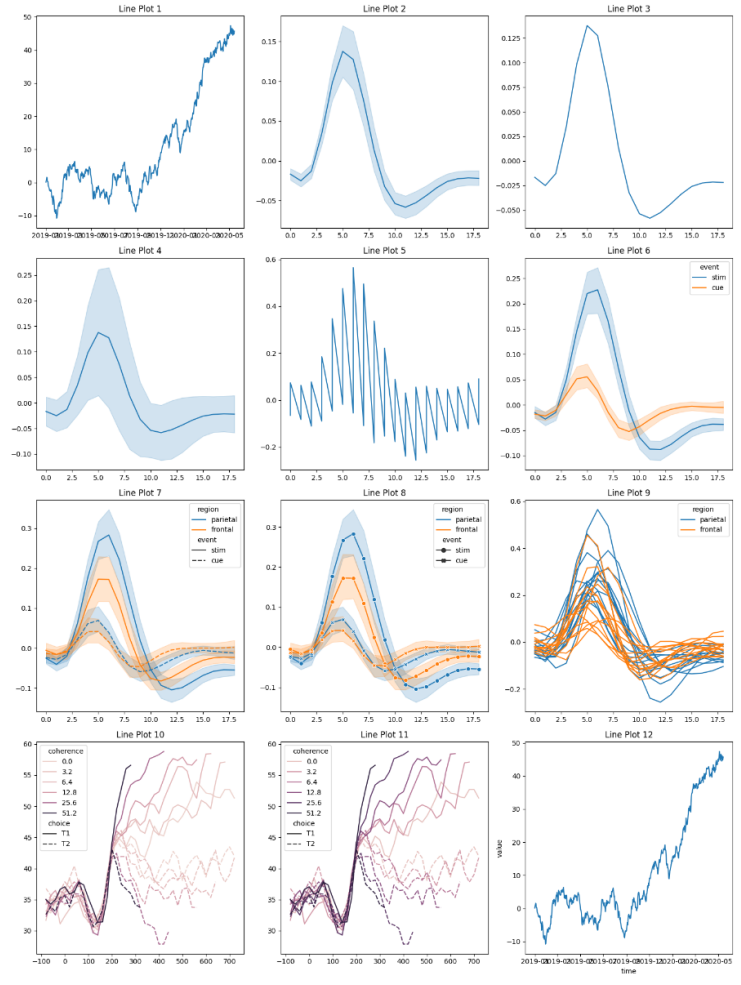
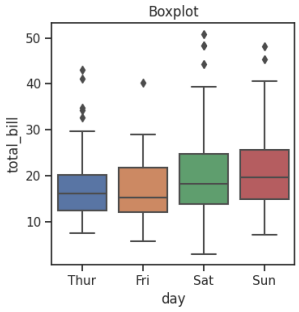
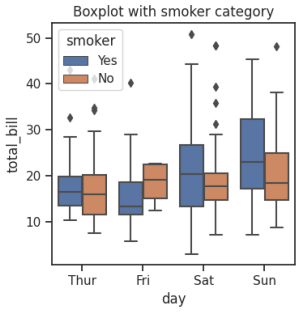
PYTHON SEABORN PART-1

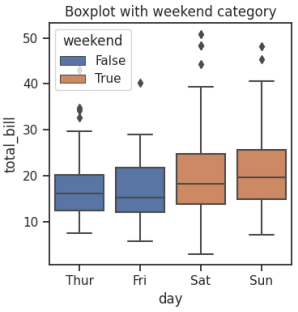
Seaborn is a Python data visualization library built on Matplotlib. It provides a high-level interface for creating visually appealing and informative statistical graphics. Seaborn simplifies the process of creating complex visualizations and offers various customization options. It is particularly useful for exploring datasets with multiple variables and complex relationships.

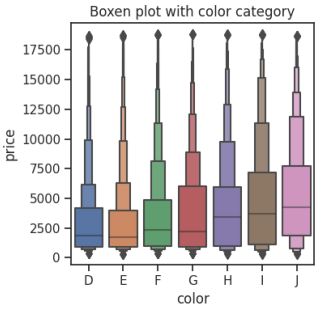


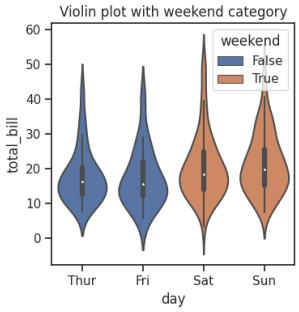


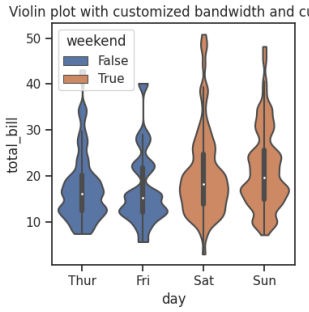


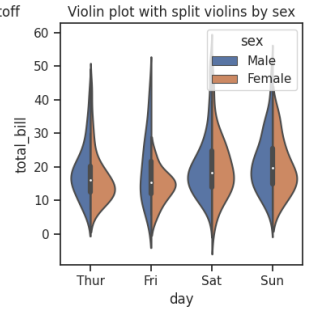


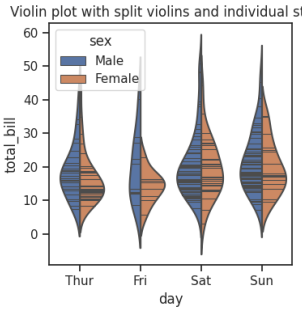












Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
# Ignore all warnings
warnings.filterwarnings("ignore")
```

Scatter Plots

A scatter plot is a type of plot used to visualize the relationship between two numerical variables. It displays individual data points as dots or markers on a two-dimensional plane, with one variable represented on the x-axis and the other variable represented on the y-axis. The position of each data point on the plot corresponds to its values for the respective variables.

Scatter plots are useful for understanding patterns, trends, and correlations between variables. They allow us to identify the presence or absence of any relationship between the variables. When the points on the scatter plot form a discernible pattern, it suggests a relationship between the variables. Different patterns can indicate positive or negative correlations, linear or nonlinear relationships, clusters or groups of data, or the absence of any apparent relationship.

In addition to displaying the data points, scatter plots can be enhanced with additional visual cues. These include adding a regression line to show the overall trend of the data, coloring the data points based on a categorical variable to differentiate groups or categories, and varying the size of the markers to represent a third numerical variable.

Seaborn, a popular Python data visualization library, provides various functions, such as relplot and scatterplot, for creating scatter plots. These functions offer options to customize the appearance of scatter plots, such as changing marker styles, adding trend lines, adjusting the color palette, and incorporating additional variables through the use of hue, style, and size parameters.

Overall, scatter plots are a valuable tool for visualizing the relationship between two numerical variables and gaining insights into the underlying data patterns and trends.

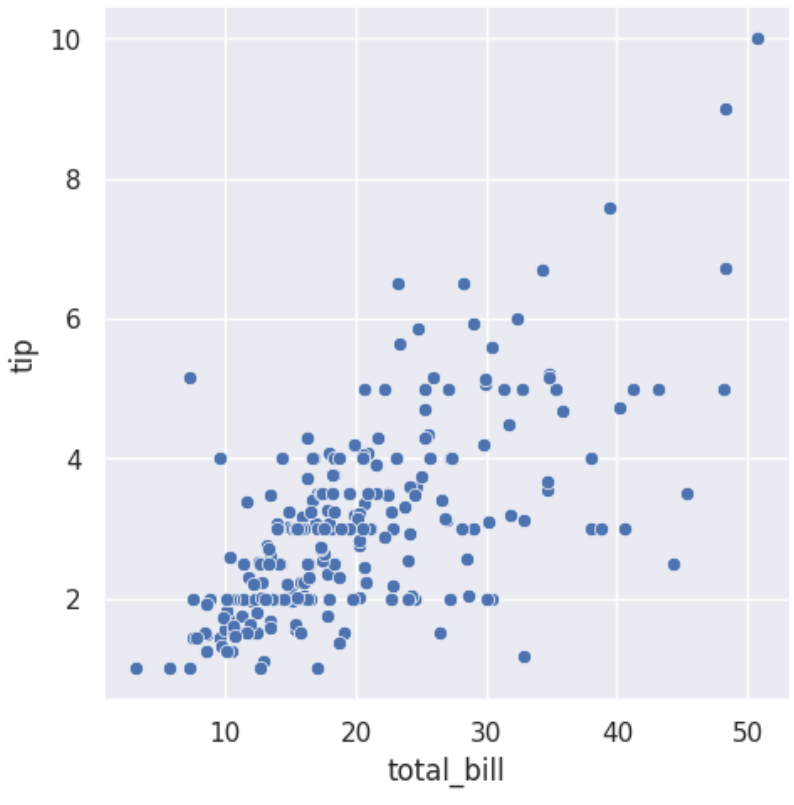
```
# Set the style of the plots to "darkgrid"
sns.set(style="darkgrid")
```

```
# Load the 'tips' dataset from Seaborn
tips = sns.load_dataset("tips")
```

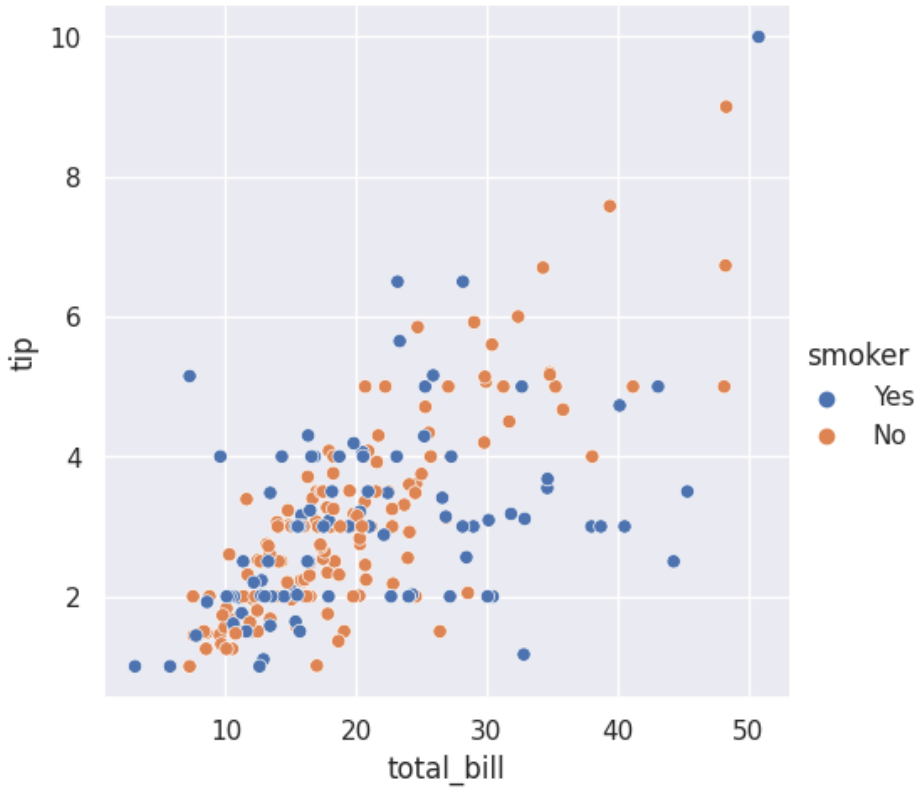
```
# Display the first few rows of the dataset
tips.head()
```

	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
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

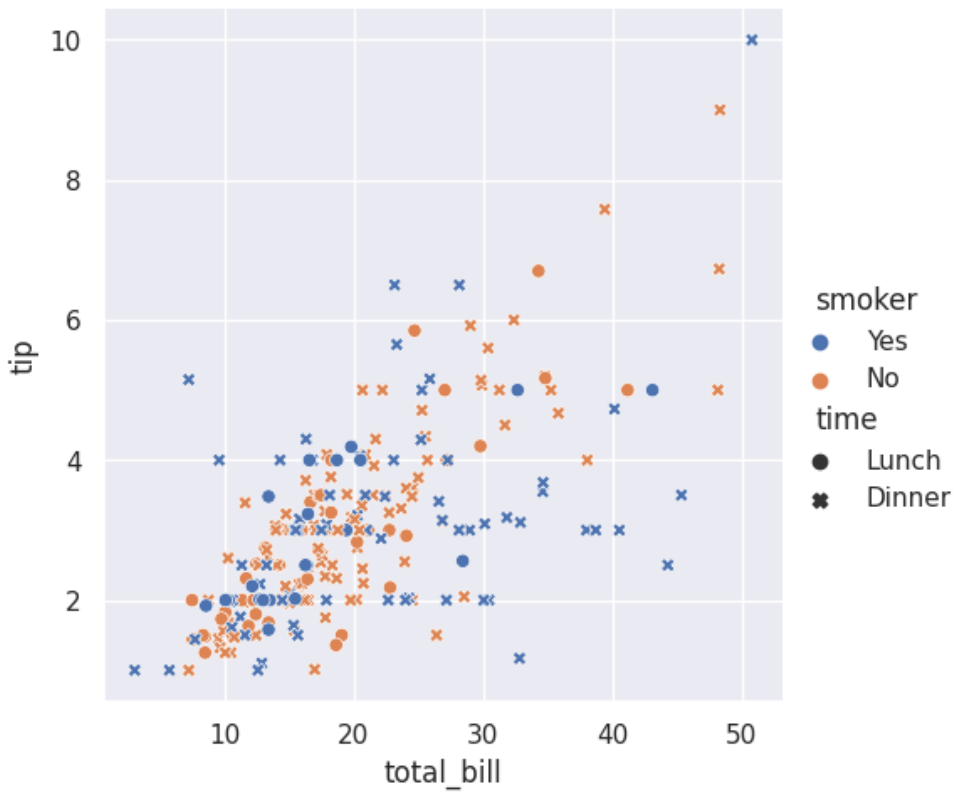
```
# Basic scatter plot using 'relplot' without specifying any additional parameters
sns.relplot(x="total_bill", y="tip", data=tips)
plt.show()
```



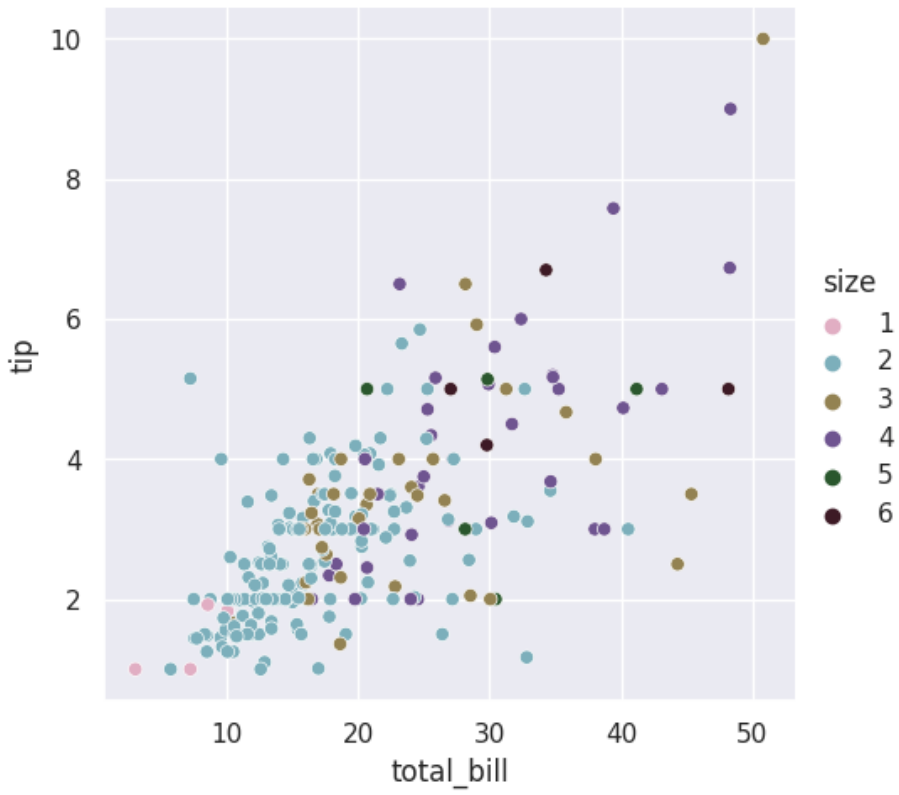
```
# Scatter plot with hue parameter to differentiate points by 'smoker' category
sns.relplot(x="total_bill", y="tip", data=tips, hue="smoker")
plt.show()
```



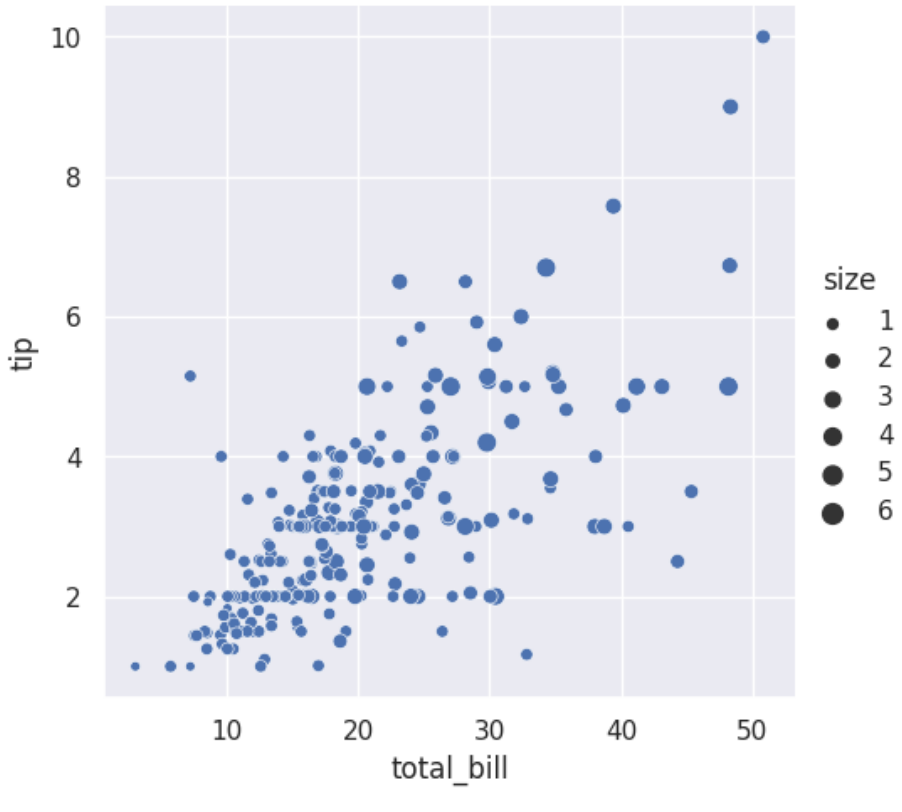

```
# Scatter plot with hue and style parameters to differentiate points by 'smoker' and 'time' categories respectively
sns.relplot(x="total_bill", y="tip", data=tips, hue="smoker", style="time")
plt.show()
```



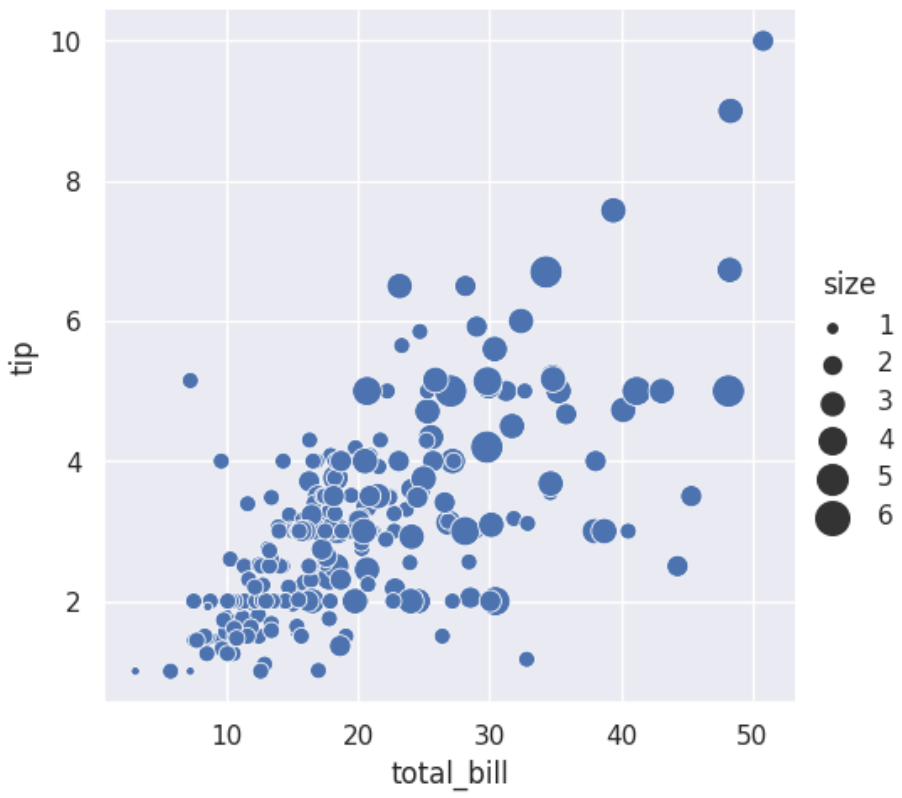

```
# Scatter plot with hue parameter to map the 'size' category to different colors using a specified palette
sns.relplot(x="total_bill", y="tip", data=tips, hue="size", palette="ch:r=-5,l=.75")
plt.show()
```



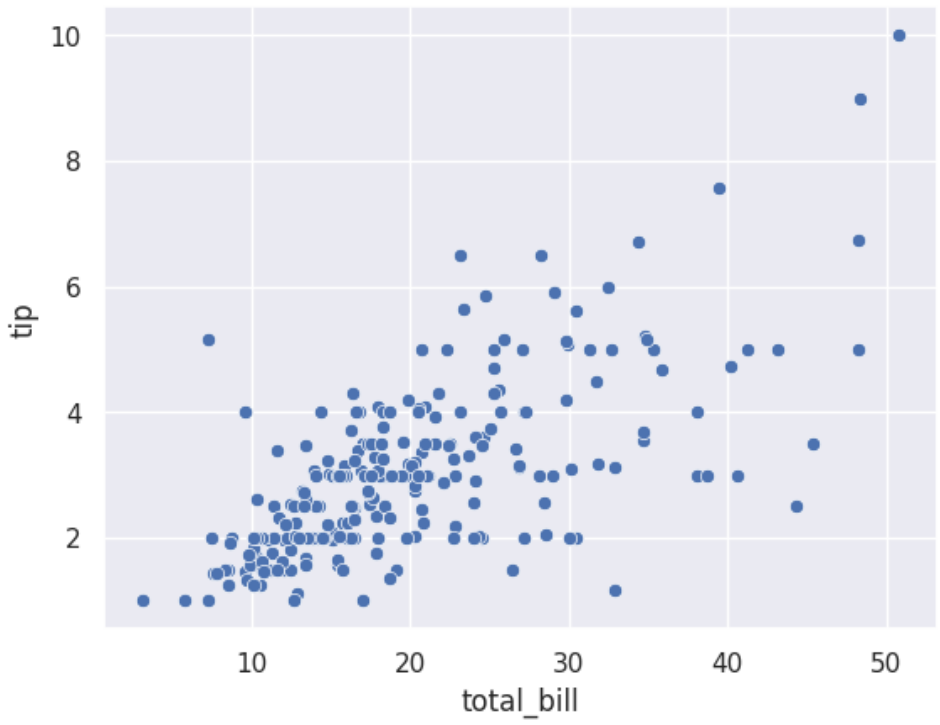
```
# Scatter plot with size parameter to vary the size of the points based on the 'size' category
sns.relplot(x="total_bill", y="tip", data=tips, size="size")
plt.show()
```



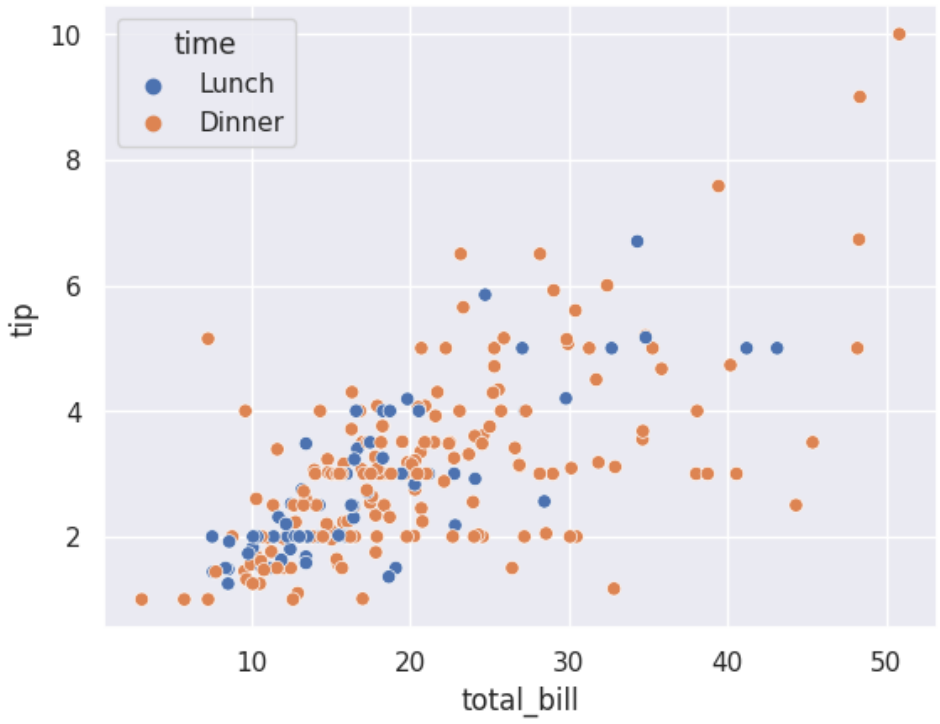

```
# Scatter plot with size parameter and custom range for sizes
sns.relplot(x="total_bill", y="tip", data=tips, size="size", sizes=(15, 200))
plt.show()
```



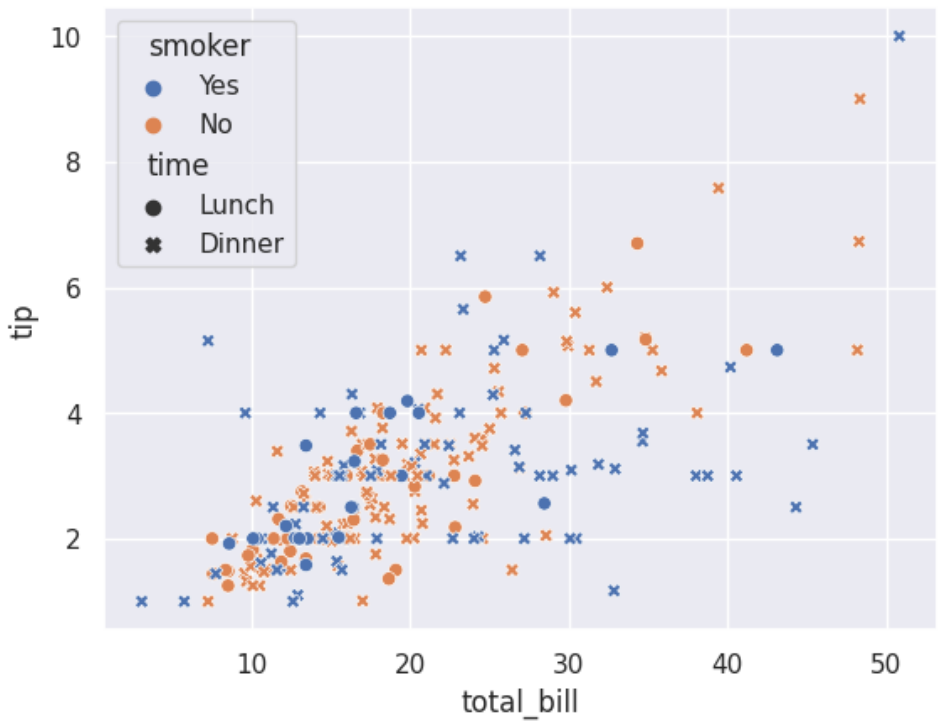

```
# Scatter plot using 'scatterplot' function with explicit axes object
ax = sns.scatterplot(x="total_bill", y="tip", data=tips)
plt.show()
```



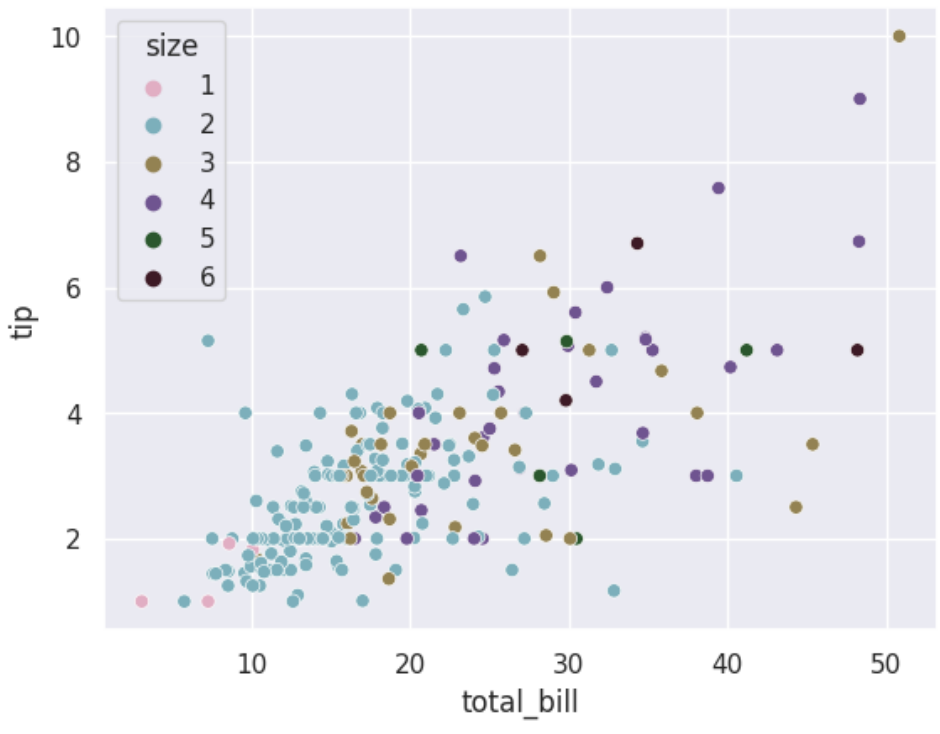

```
# Scatter plot with hue parameter using explicit axes object
ax = sns.scatterplot(x="total_bill", y="tip", data=tips, hue="time")
plt.show()
```



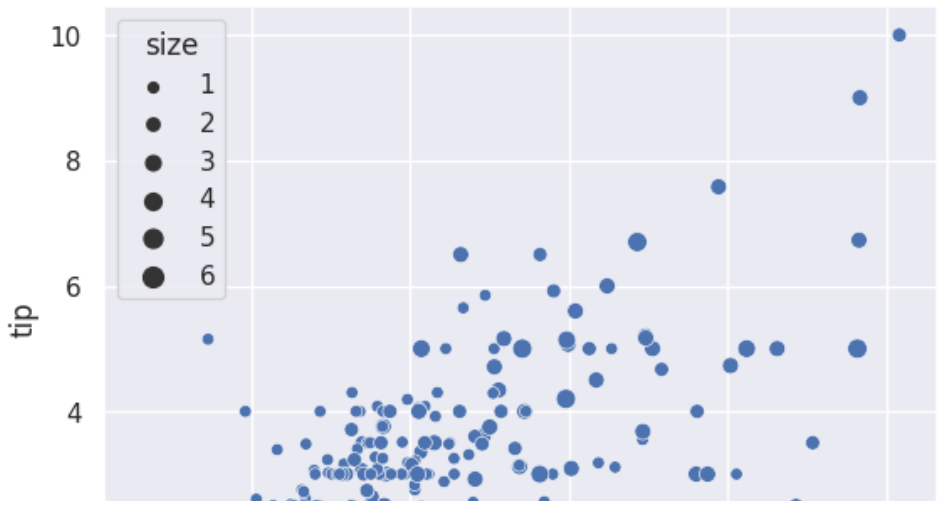

```
# Scatter plot with hue and style parameters using explicit axes object
ax = sns.scatterplot(x="total_bill", y="tip", data=tips, hue="smoker", style="time")
plt.show()
```



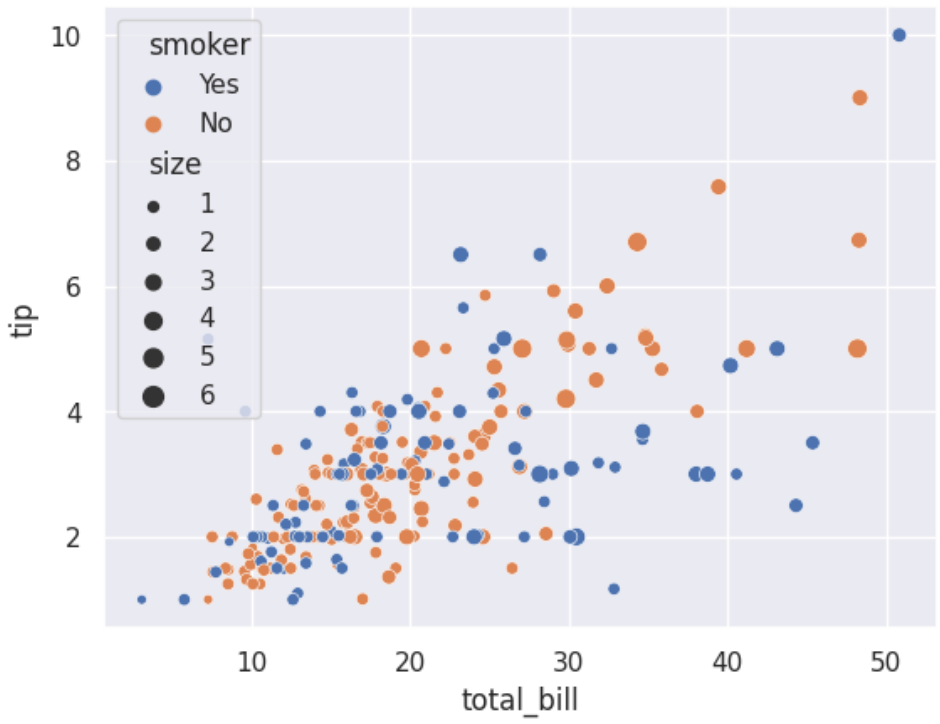

```
# Scatter plot with hue parameter and custom palette
sns.scatterplot(x="total_bill", y="tip", data=tips, hue="size", palette="ch:r=-5,l=.75")
plt.show()
```



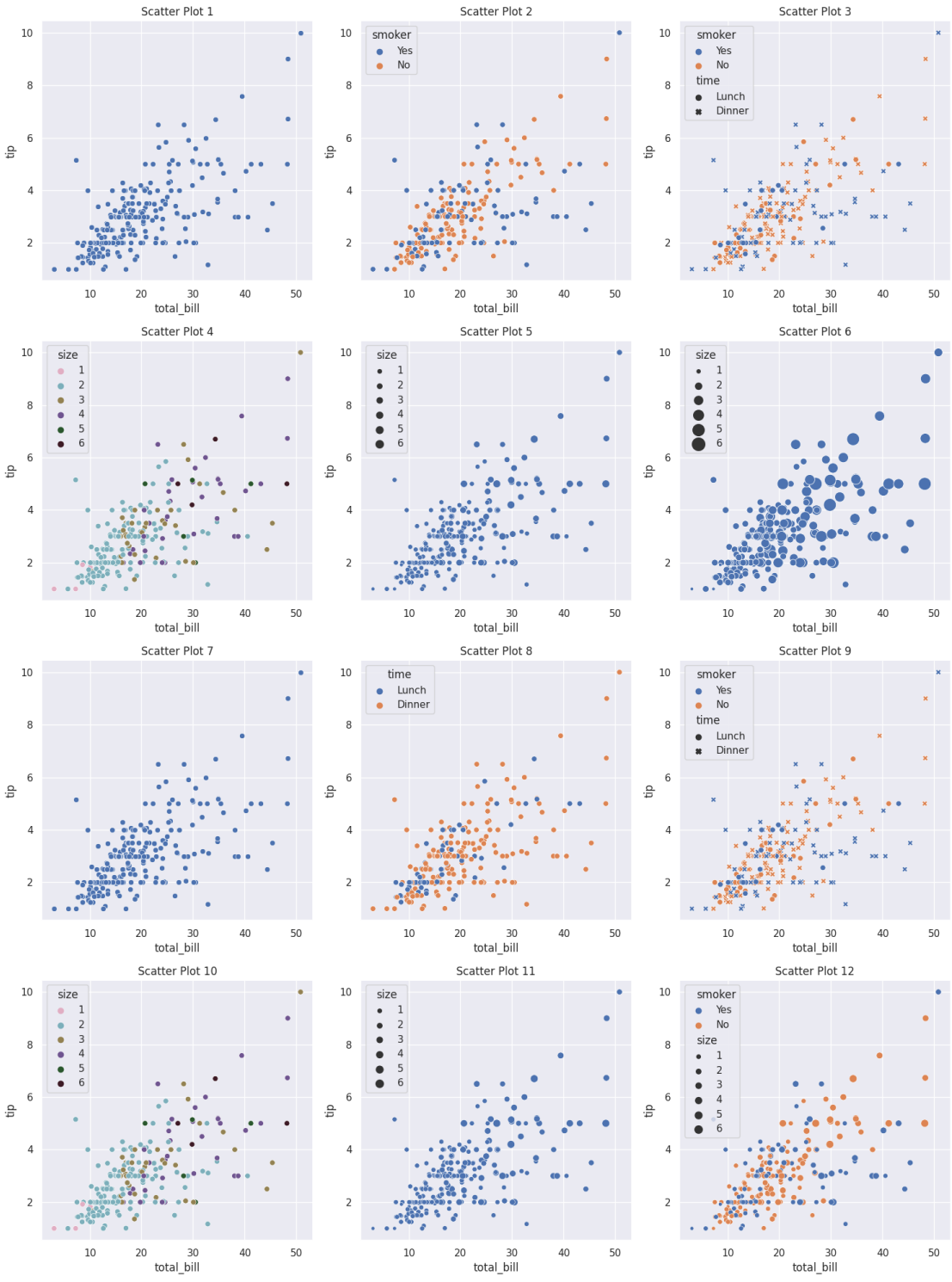

```
# Scatter plot with size parameter
sns.scatterplot(x="total_bill", y="tip", data=tips, size="size")
plt.show()
```




```
# Scatter plot with size and hue parameters
sns.scatterplot(x="total_bill", y="tip", data=tips, size="size", hue="smoker")
plt.show()
```




```
# Create a figure and axes using subplots with 4 rows and 3 columns
fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(15, 20))

# Flatten the axes array for easy indexing
axes = axes.flatten()

# Scatter plots with different parameters
sns.scatterplot(x="total_bill", y="tip", data=tips, ax=axes[0])
sns.scatterplot(x="total_bill", y="tip", data=tips, hue="smoker", ax=axes[1])
sns.scatterplot(x="total_bill", y="tip", data=tips, hue="smoker", style="time", ax=axes[2])
sns.scatterplot(x="total_bill", y="tip", data=tips, hue="size", palette="ch:r=-5,l=.75", ax=axes[3])
sns.scatterplot(x="total_bill", y="tip", data=tips, size="size", ax=axes[4])
sns.scatterplot(x="total_bill", y="tip", data=tips, size="size", sizes=(15, 200), ax=axes[5])
sns.scatterplot(x="total_bill", y="tip", data=tips, ax=axes[6])
sns.scatterplot(x="total_bill", y="tip", data=tips, hue="time", ax=axes[7])
sns.scatterplot(x="total_bill", y="tip", data=tips, hue="smoker", style="time", ax=axes[8])
sns.scatterplot(x="total_bill", y="tip", data=tips, hue="size", palette="ch:r=-5,l=.75", ax=axes[9])
sns.scatterplot(x="total_bill", y="tip", data=tips, size="size", ax=axes[10])
sns.scatterplot(x="total_bill", y="tip", data=tips, size="size", hue="smoker", ax=axes[11])

# Remove empty subplots
for i in range(12, len(axes)):
    fig.delaxes(axes[i])

# Set titles for each plot
axes[0].set_title("Scatter Plot 1")
axes[1].set_title("Scatter Plot 2")
axes[2].set_title("Scatter Plot 3")
axes[3].set_title("Scatter Plot 4")
axes[4].set_title("Scatter Plot 5")
axes[5].set_title("Scatter Plot 6")
axes[6].set_title("Scatter Plot 7")
axes[7].set_title("Scatter Plot 8")
axes[8].set_title("Scatter Plot 9")
axes[9].set_title("Scatter Plot 10")
axes[10].set_title("Scatter Plot 11")
axes[11].set_title("Scatter Plot 12")

# Adjust the layout to avoid overlapping labels
plt.tight_layout()

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



Line Plot

A line plot is a type of graph that represents data points connected by straight lines. It is commonly used to display the trend or progression of a variable over a continuous range, such as time or a numerical sequence. In a line plot, the x-axis typically represents the independent variable (e.g., time) and the y-axis represents the dependent variable (e.g., measurement or value).

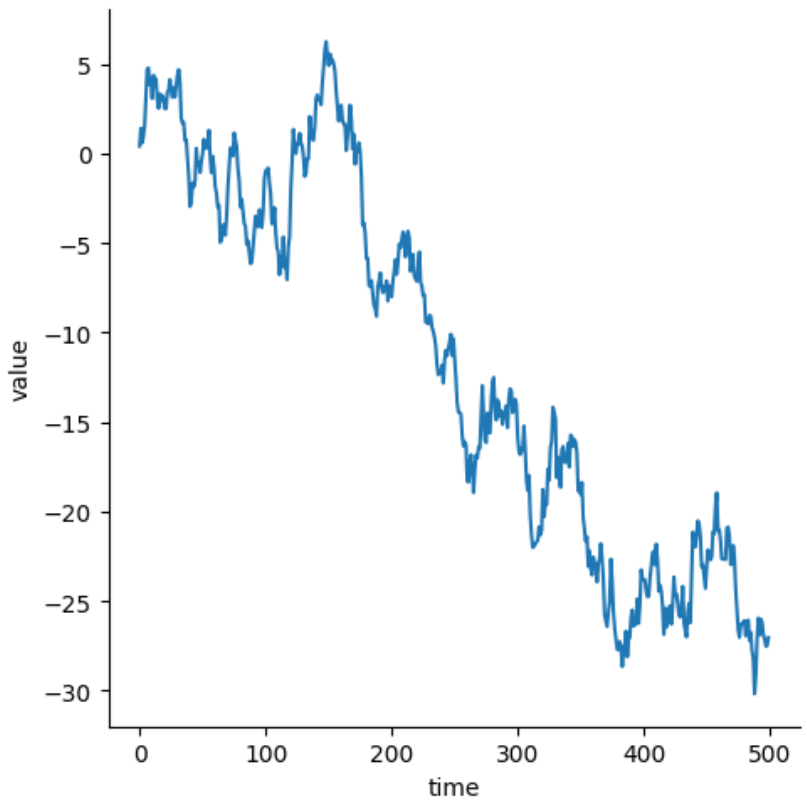
Line plots are effective for visualizing the overall pattern, trend, and variability of a variable. They allow us to observe how the values change over the range of the independent variable and identify any relationships, trends, or patterns that may exist. Line plots can reveal information such as upward or downward trends, fluctuations, seasonality, or the absence of any noticeable trend.

In addition to displaying the main line connecting the data points, line plots can include additional elements to enhance the visualization. These elements may include markers at each data point to highlight individual values, error bars to represent uncertainty or variability, shading or bands to indicate confidence intervals, and labels or annotations to provide additional information.

Seaborn, a popular Python data visualization library, provides functions such as relplot and lineplot for creating line plots. These functions offer various customization options, allowing you to adjust the line style, marker style, color palette, and other visual aspects of the plot. You can also incorporate additional variables using parameters like hue or style to differentiate groups or categories.

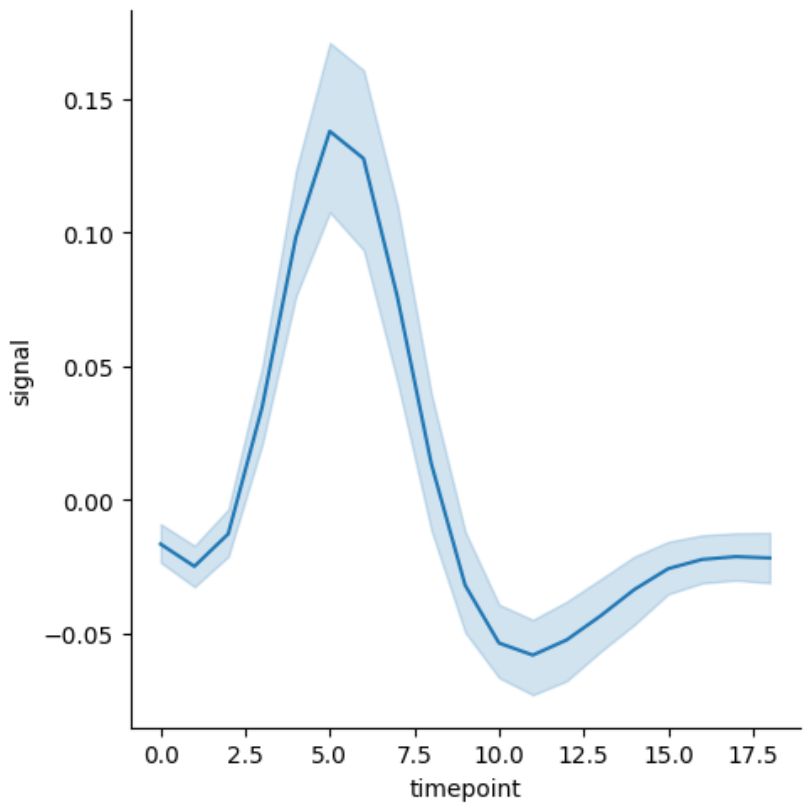
Line plots are widely used in many fields, including scientific research, finance, and data analysis, to visualize time series data, trends, and patterns. They provide a concise and intuitive way to understand and communicate the behavior and evolution of variables over a continuous range.

```
# Create a DataFrame with a time column ranging from 0 to 499 and a value column
df = pd.DataFrame(dict(time=np.arange(500), value=np.random.randn(500).cumsum()))

# Create a line plot using seaborn's relplot with x as "time" and y as "value"
g = sns.relplot(x="time", y="value", kind="line", data=df)
```



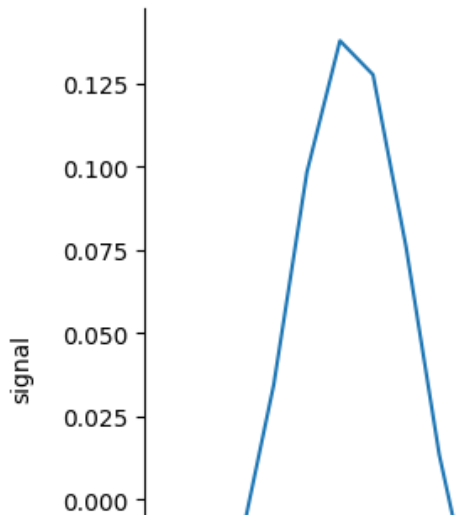

```
fmir= sns.load_dataset("fmri")


# Create a line plot with x as "timepoint" and y as "signal" from the fmri dataset
sns.relplot(x="timepoint", y="signal", data=fmir, kind="line")
```

```
<seaborn.axisgrid.FacetGrid at 0x7f6143f091e0>
```



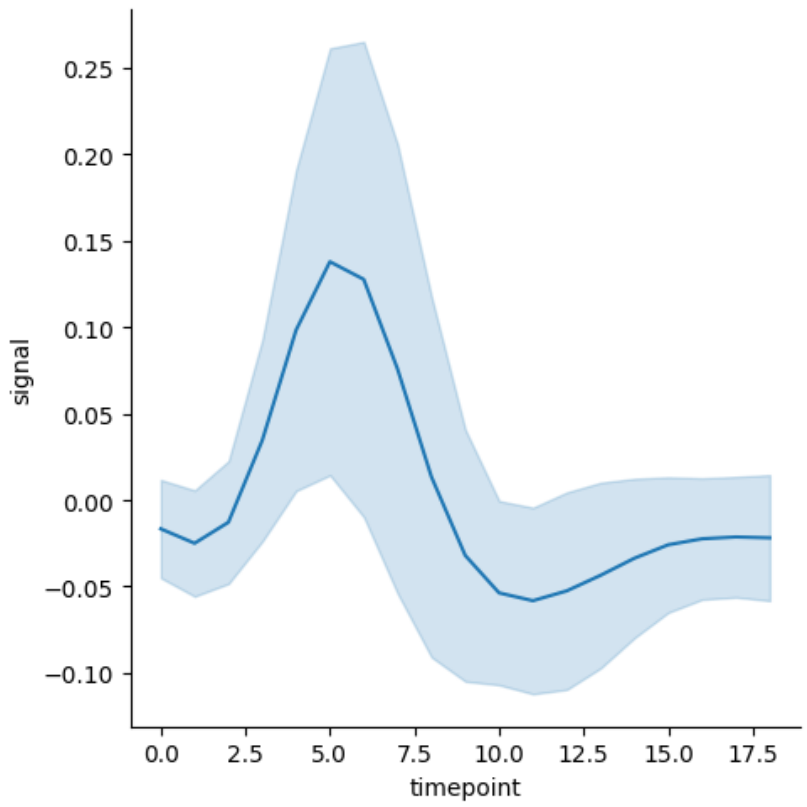

```
# Create a line plot with x as "timepoint" and y as "signal", with no confidence interval (ci=None)
sns.relplot(x="timepoint", y="signal", data=fmir, kind="line", ci=None)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f614267fe50>
```



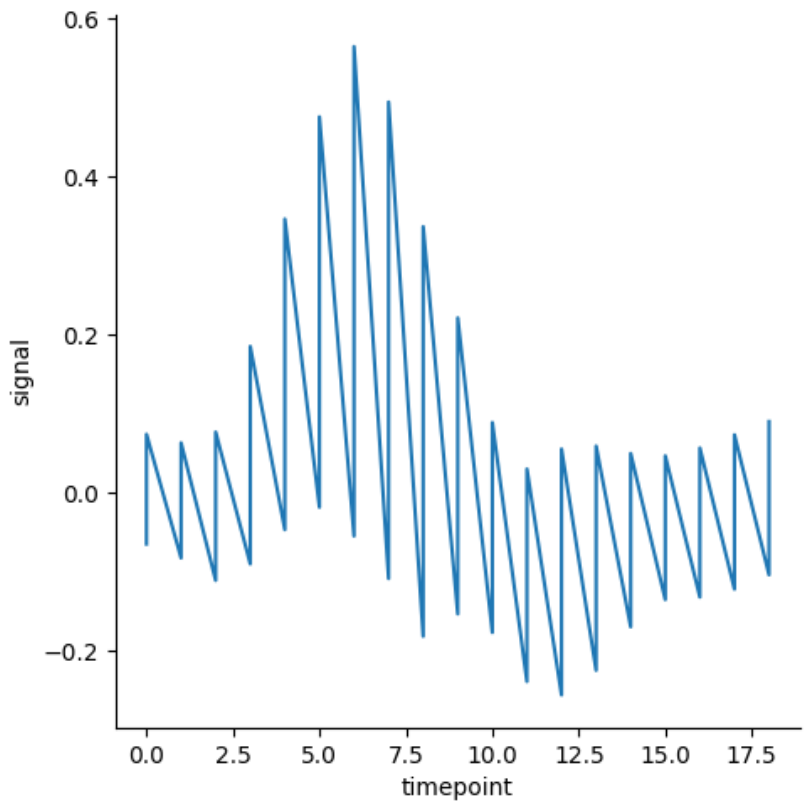

```
# Create a line plot with x as "timepoint" and y as "signal", using the standard deviation as the confidence interval (ci="sd")
sns.relplot(x="timepoint", y="signal", data=fmir, kind="line", ci="sd")
```

```
<seaborn.axisgrid.FacetGrid at 0x7f614340e380>
```




```
# Create a line plot with x as "timepoint" and y as "signal", without using any estimator (estimator=None) and no confidence interval (ci=None)
sns.relplot(x="timepoint", y="signal", estimator=None, data=fmir, kind="line", ci=None)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f6144893a00>
```

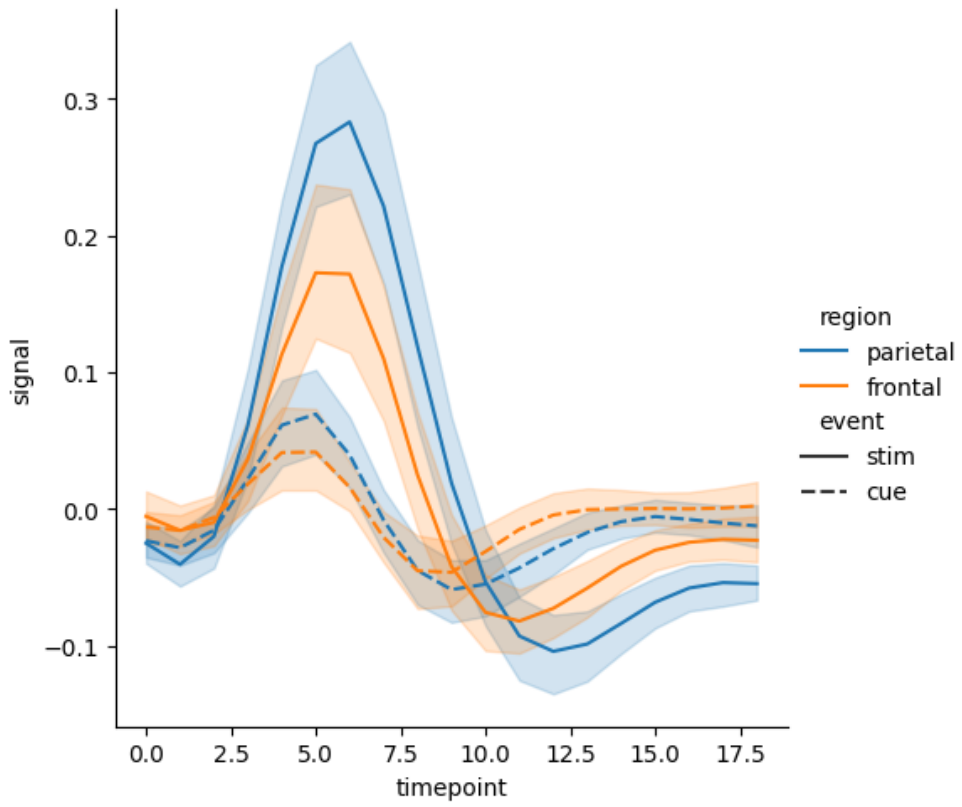



```
# Create a line plot with x as "timepoint" and y as "signal", colored by "event"
sns.relplot(x="timepoint", y="signal", data=fmir, kind="line", hue="event")
```

```
<seaborn.axisgrid.FacetGrid at 0x7f614243ceb0>
```

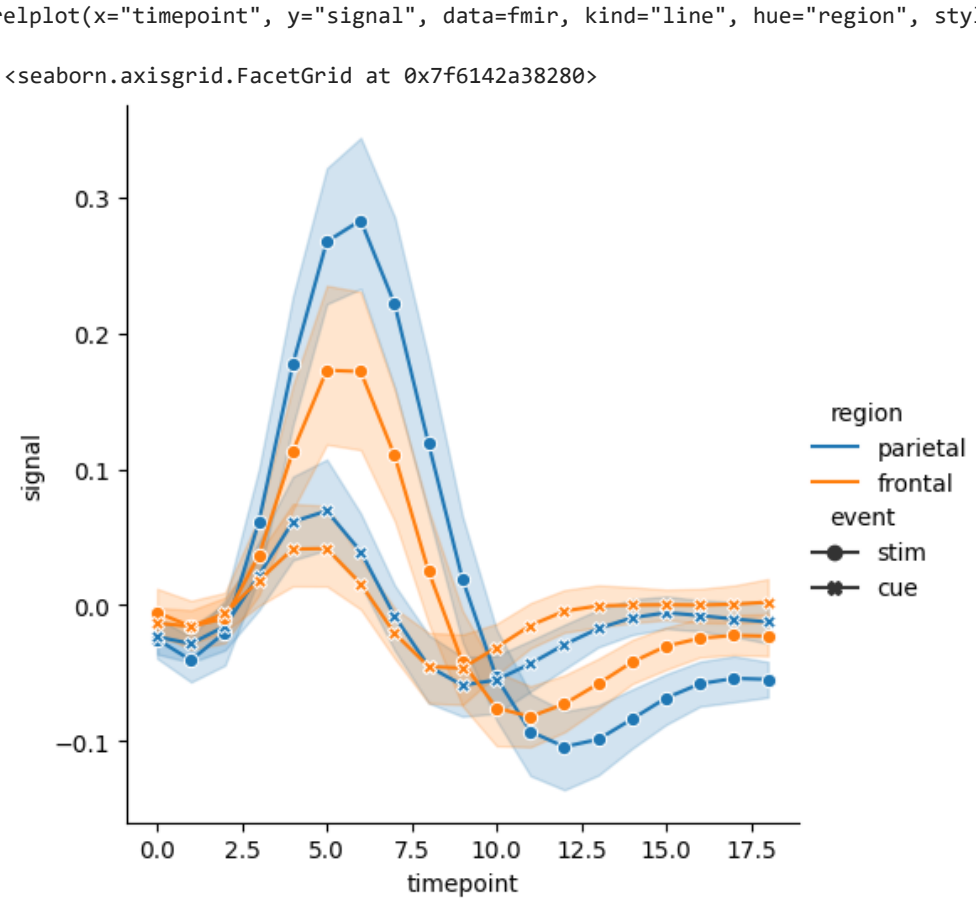

```
# Create a line plot with x as "timepoint" and y as "signal", colored by "region" and styled by "event"
sns.relplot(x="timepoint", y="signal", data=fmir, kind="line", hue="region", style="event")
```

```
<seaborn.axisgrid.FacetGrid at 0x7f6144fe6380>
```




```
# Create a line plot with x as "timepoint" and y as "signal", colored by "region" and styled by "event", without dashes and with markers
sns.relplot(x="timepoint", y="signal", data=fmir, kind="line", hue="region", style="event", dashes=False, markers=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f6142a38280>
```



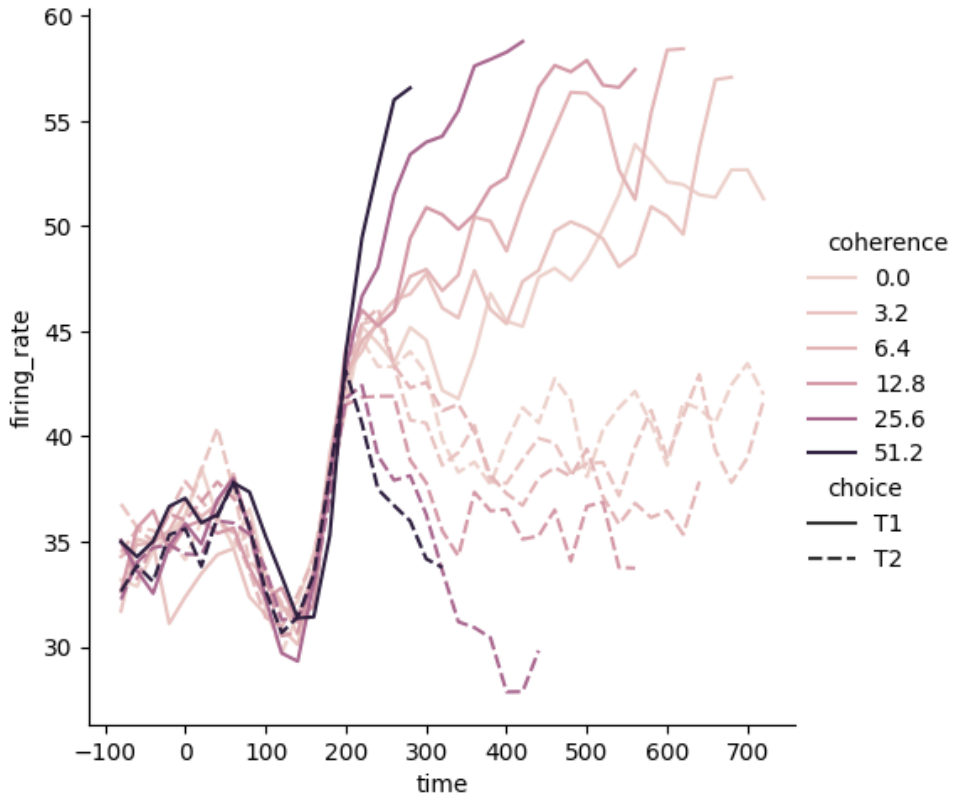

```
# Create a line plot with x as "timepoint" and y as "signal" for the subset of data where event is 'stim',
# colored by "region" and using "subject" as the unit of observation without using any estimator
sns.relplot(x="timepoint", y="signal", data=fmir.query("event == 'stim'"), kind="line", hue="region", units="subject", estimator=None)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f6143172da0>
```


```
# Load the dots dataset from seaborn and filter it for align == 'dots'
dots = sns.load_dataset("dots").query("align == 'dots'")
```

```
# Create a line plot with x as "time" and y as "firing_rate", colored by "coherence" and styled by "choice"
sns.relplot(x="time", y="firing_rate", data=dots, kind="line", hue="coherence", style="choice")
```

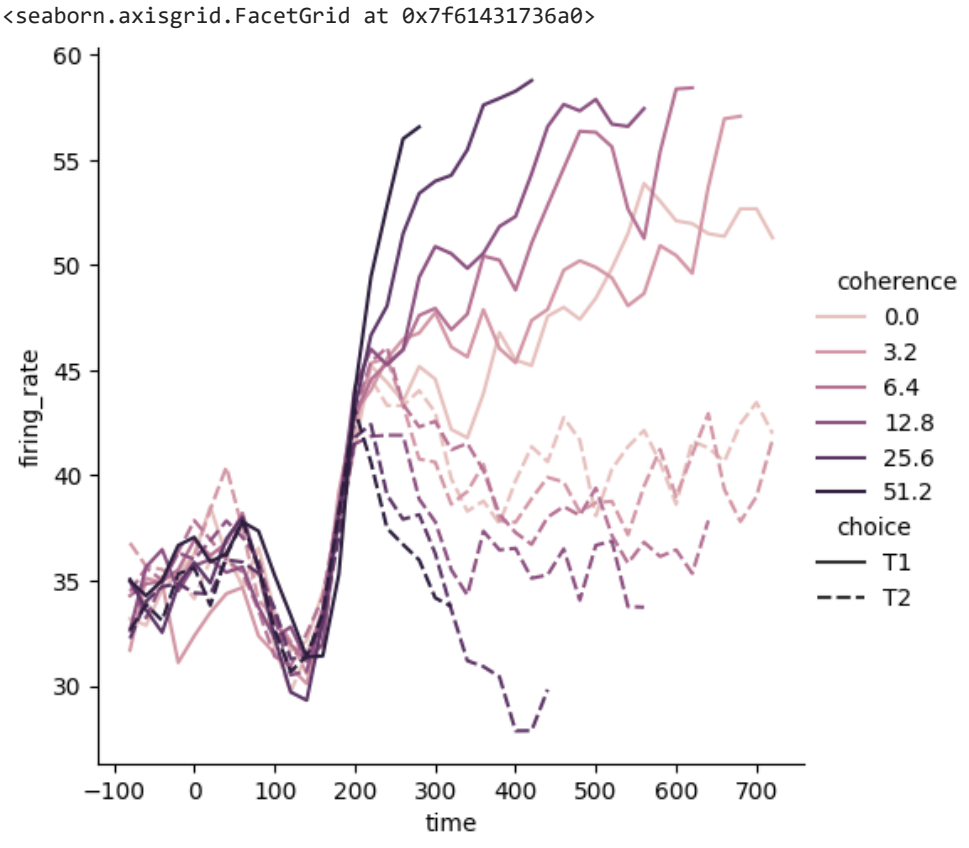
```
<seaborn.axisgrid.FacetGrid at 0x7f6143d13400>
```




```
# Create a custom palette using cubehelix_palette
palette = sns.cubehelix_palette(light=.8, n_colors=6)
```

```
# Create a line plot with x as "time" and y as "firing_rate", colored by "coherence", styled by "choice" and using the custom palette
sns.relplot(x="time", y="firing_rate", data=dots, kind="line", hue="coherence", style="choice", palette=palette)
```

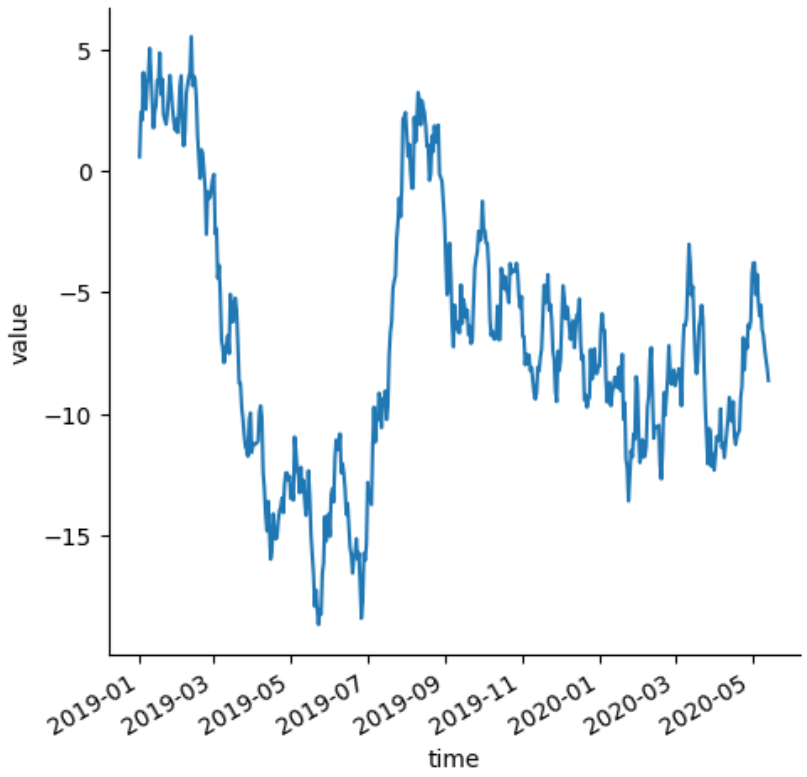
```
<seaborn.axisgrid.FacetGrid at 0x7f61431736a0>
```




```
# Create a DataFrame with a time column ranging from "2019-01-01" to "2019-05-15" and a value column
df = pd.DataFrame(dict(time=pd.date_range("2019-01-01", periods=500), value=np.random.randn(500).cumsum()))
```

```
# Create a line plot with x as "time" and y as "value" for the DataFrame with dates
g = sns.relplot(x="time", y="value", kind="line", data=df)
```

```
# Rotate the x-axis labels for better visibility
g.fig.autofmt_xdate()
```



Certainly! Here's a small description explaining how relplot with kind="line" can be replaced with lineplot:

Both relplot and lineplot are functions provided by the Seaborn library for creating line plots. However, there is a difference in how they are used.

relplot is a figure-level function in Seaborn that is designed to create a variety of plots, including line plots, scatter plots, and others. It provides a convenient interface for creating complex visualizations by specifying the kind parameter. When kind is set to "line", relplot generates a line plot.

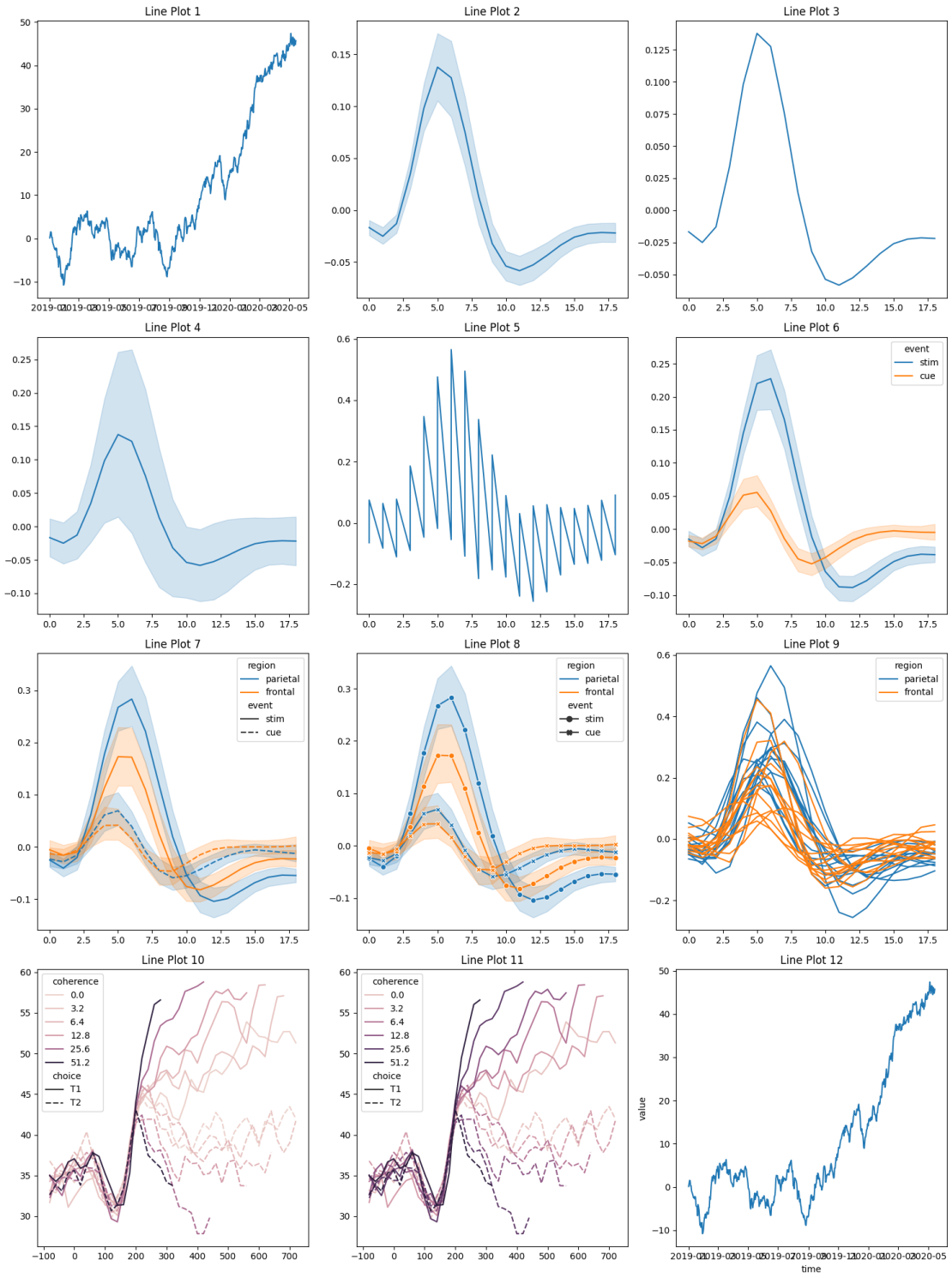
On the other hand, lineplot is an axes-level function in Seaborn specifically designed for creating line plots. It offers more flexibility and control over the plot's customization and behavior.

To replace relplot with lineplot for line plots, you can directly use lineplot to create the desired line plots. Instead of specifying kind="line", you can pass the necessary arguments to lineplot function, such as x, y, and other optional parameters like hue, style, or ci.

By using lineplot, you can have more fine-grained control over the customization of your line plots, such as adjusting the linestyle, marker style, adding error bars, or modifying the color palette.

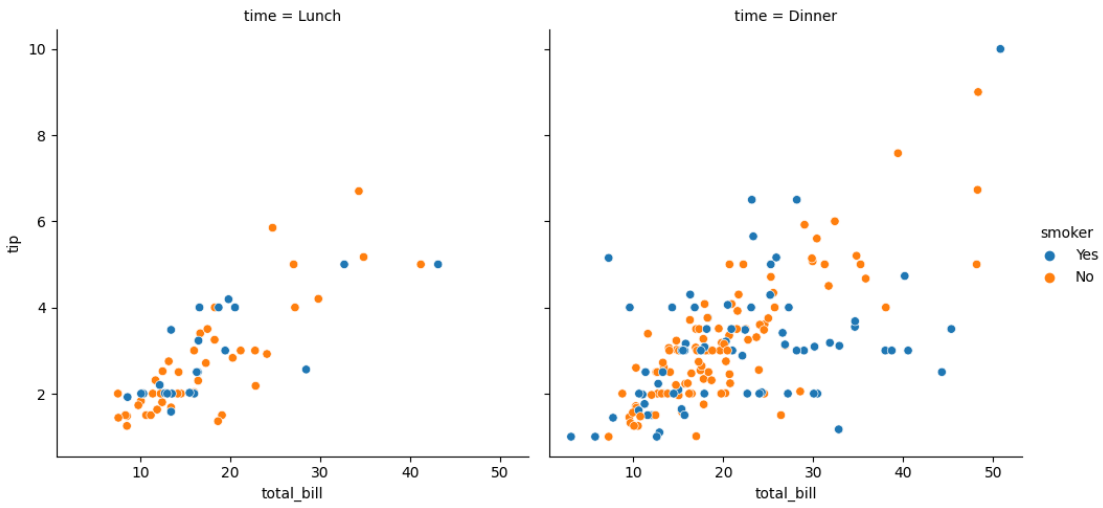
Overall, if you need a more specialized and customizable line plot, it is recommended to use lineplot directly. However, if you prefer a simpler interface and want to create a line plot along with other types of plots, relplot with kind="line" can be a convenient option.

```
# Create a figure and axes using subplots with 4 rows and 3 columns
fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(15, 20))

# Line plots with different parameters
sns.lineplot(x="time", y="value", data=df, ax=axes[0, 0])
sns.lineplot(x="timepoint", y="signal", data=fmir, ax=axes[0, 1])
sns.lineplot(x="timepoint", y="signal", ci=None, data=fmir, ax=axes[0, 2])
sns.lineplot(x="timepoint", y="signal", ci="sd", data=fmir, ax=axes[1, 0])
sns.lineplot(x="timepoint", y="signal", estimator=None, ci=None, data=fmir, ax=axes[1, 1])
sns.lineplot(x="timepoint", y="signal", hue="event", data=fmir, ax=axes[1, 2])
sns.lineplot(x="timepoint", y="signal", hue="region", style="event", data=fmir, ax=axes[2, 0])
sns.lineplot(x="timepoint", y="signal", hue="region", style="event", dashes=False, markers=True, data=fmir, ax=axes[2, 1])
sns.lineplot(x="timepoint", y="signal", hue="region", units="subject", estimator=None, data=fmir.query("event == 'stim'"), ax=axes[2, 2])
sns.lineplot(x="time", y="firing_rate", hue="coherence", style="choice", data=dots, ax=axes[3, 0])
sns.lineplot(x="time", y="firing_rate", hue="coherence", style="choice", palette=palette, data=dots, ax=axes[3, 1])
sns.lineplot(x="time", y="value", data=df, ax=axes[3, 2])

# Remove x-axis and y-axis labels from empty subplots
for i in range(4):
    for j in range(3):
        if i != 3 or j != 2:
            axes[i, j].set(xlabel=None, ylabel=None)

# Set titles for each plot
axes[0, 0].set_title("Line Plot 1")
axes[0, 1].set_title("Line Plot 2")
axes[0, 2].set_title("Line Plot 3")
axes[1, 0].set_title("Line Plot 4")
axes[1, 1].set_title("Line Plot 5")
axes[1, 2].set_title("Line Plot 6")
axes[2, 0].set_title("Line Plot 7")
axes[2, 1].set_title("Line Plot 8")
axes[2, 2].set_title("Line Plot 9")
axes[3, 0].set_title("Line Plot 10")
axes[3, 1].set_title("Line Plot 11")
axes[3, 2].set_title("Line Plot 12")

# Adjust the spacing between subplots
plt.tight_layout()

# Show the plots
plt.show()
```



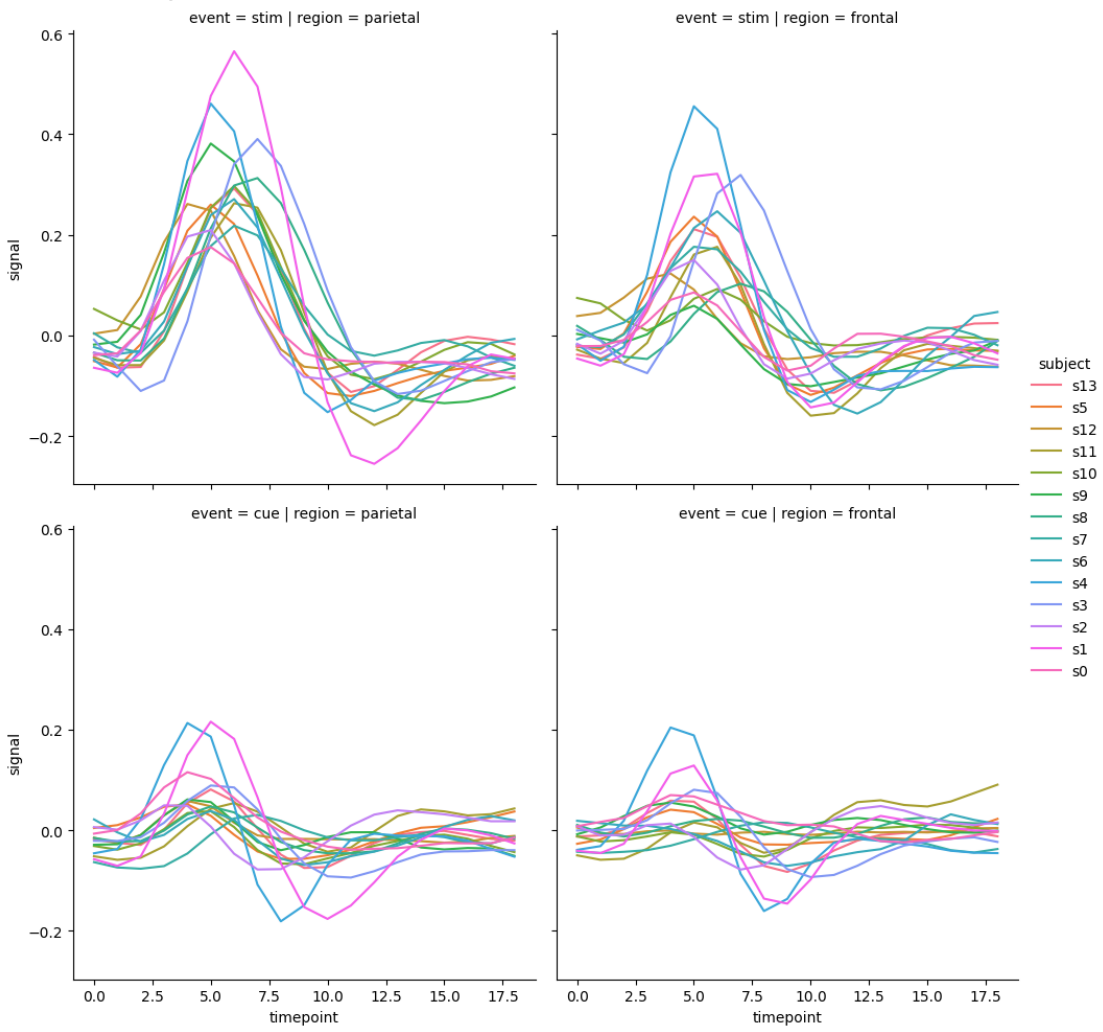
Showing multiple relationship with facts

```
# Scatter plot with multiple relationships based on smoker and time
sns.relplot(x= "total_bill", y= "tip", hue="smoker",col="time",data=tips)
plt.show()
```




```
# Line plot with multiple relationships based on subject, region, and event
sns.relplot(x="timepoint", y="signal", data=fmir, kind="line", hue="subject",col = "region",row="event",estimator = None)
```

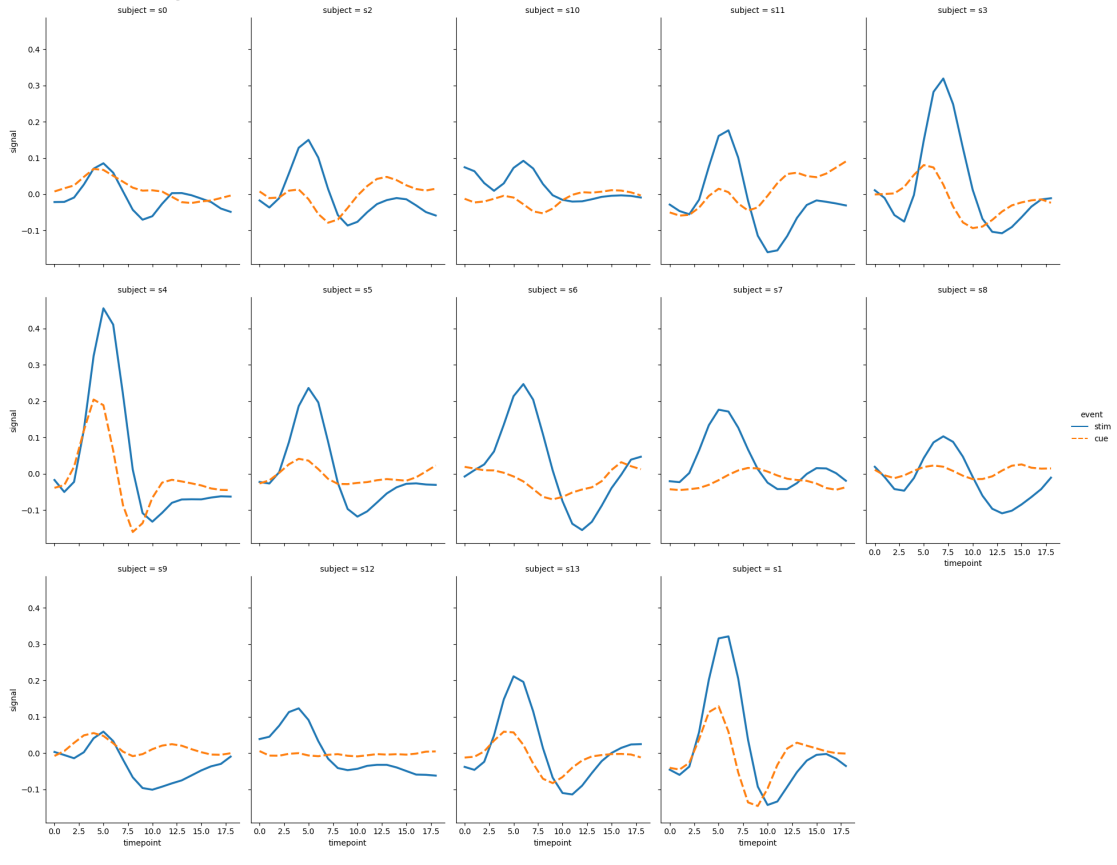
```
<seaborn.axisgrid.FacetGrid at 0x7f614322baf0>
```




```
df= pd.DataFrame(np.random.randn(500,2).cumsum(axis=0),columns=["x","y"])
```

```
# Line plot with multiple relationships based on event, subject, and region
sns.relplot(x="timepoint", y="signal", data=fmir.query("region == 'frontal'"), kind="line", hue="event",style = "event",col = "subject",col_wrap=5,height=5,asp
```

```
<seaborn.axisgrid.FacetGrid at 0x7f61434d2c80>
```

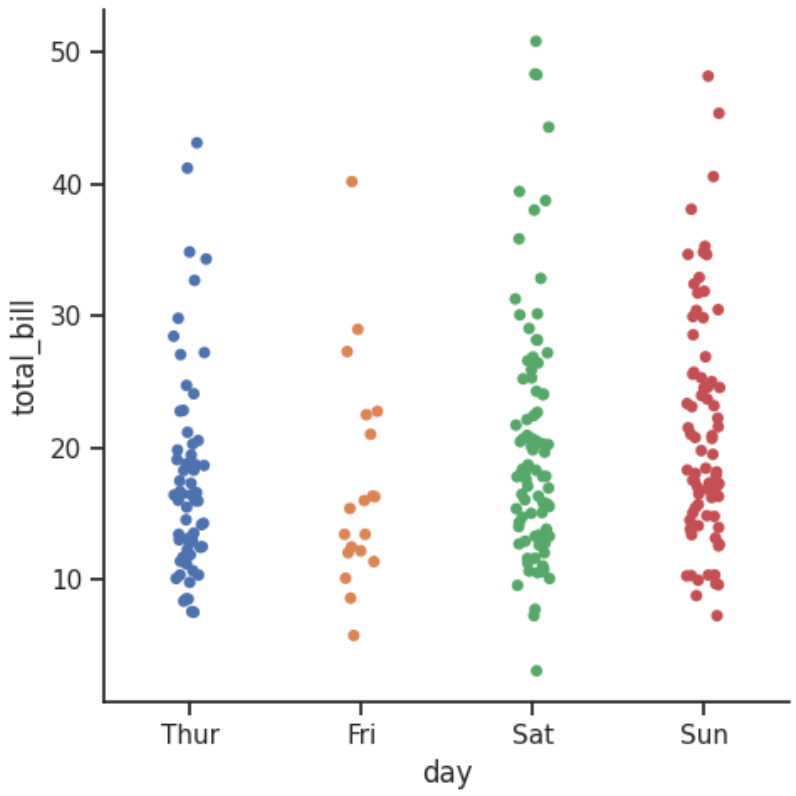


Categorical Scatterplot

```
sns.set(style="ticks", color_codes=True)
```

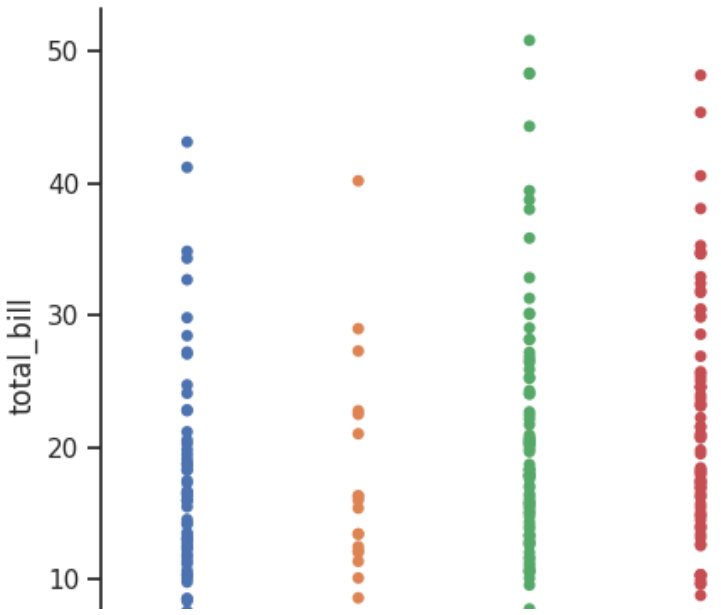
```
# Scatterplot colored by "day"
sns.catplot(x="day", y="total_bill", data=tips, hue="day")  # Scatterplot of total_bill by day
```

```
<seaborn.axisgrid.FacetGrid at 0x7f417f2cba00>
```



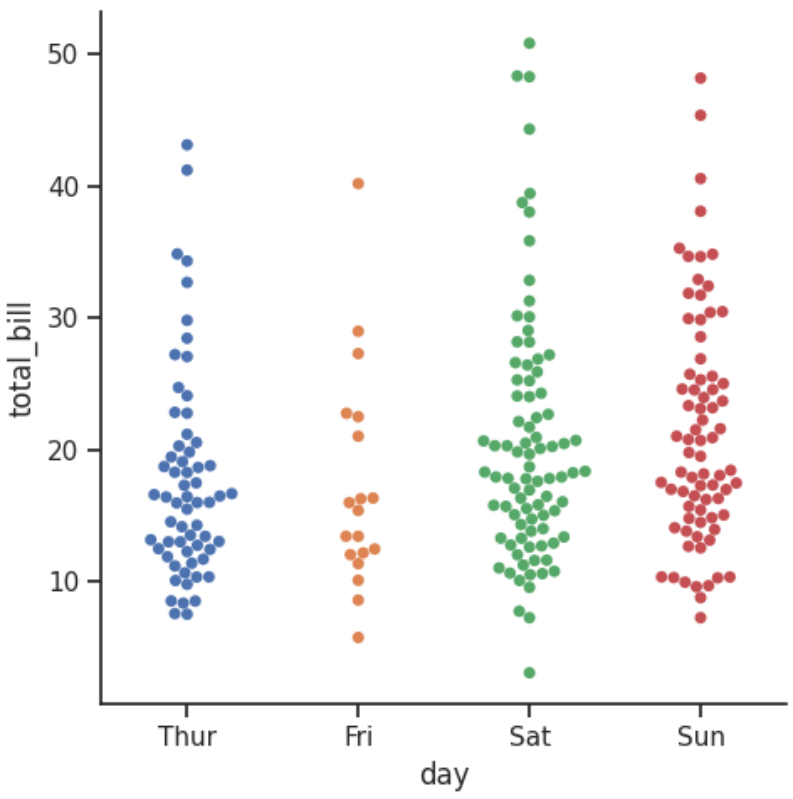

```
# Scatterplot without jitter effect
sns.catplot(x="day", y="total_bill", data=tips, hue="day", jitter=False)  # Scatterplot of total_bill by day (without jitter)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f417f237880>
```



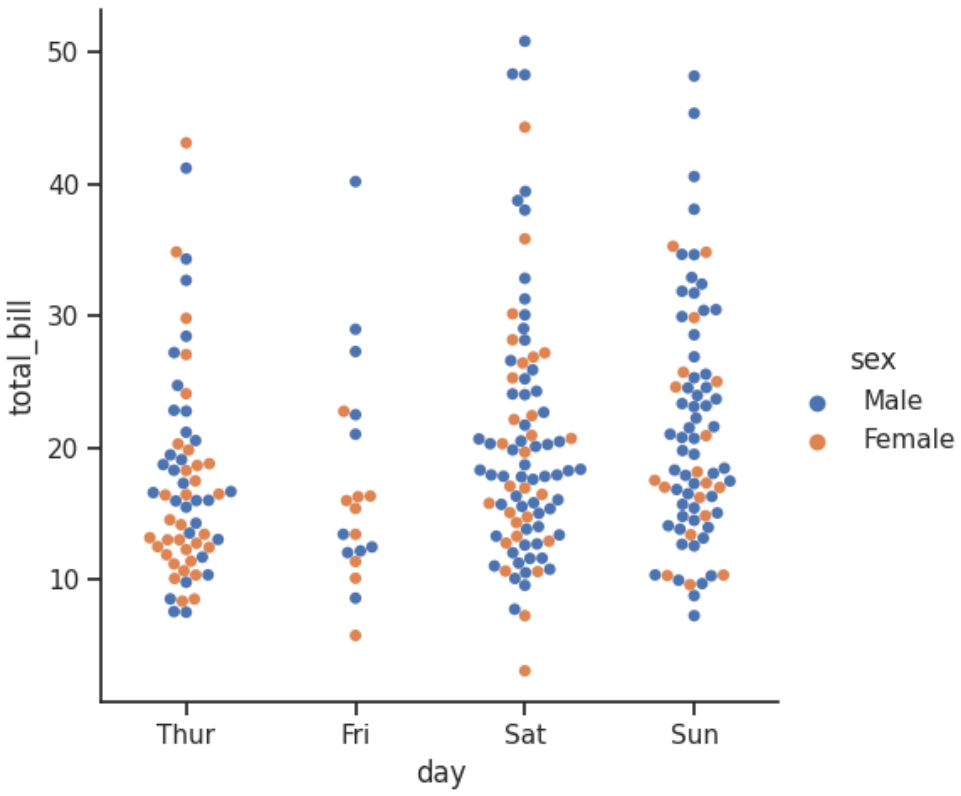

```
# Swarm plot
sns.catplot(x="day", y="total_bill", data=tips, hue="day", kind="swarm")  # Swarm plot of total_bill by day
```

```
<seaborn.axisgrid.FacetGrid at 0x7f417cfa3520>
```



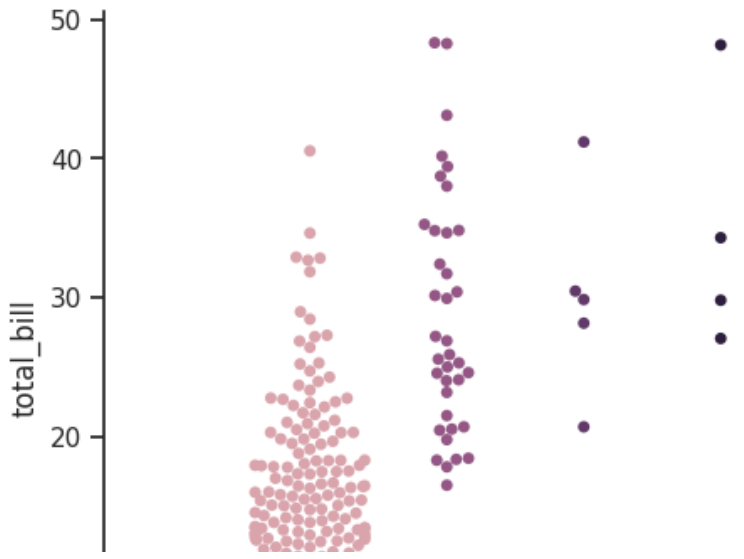

```
# Swarm plot with colors based on "sex"
sns.catplot(x="day", y="total_bill", data=tips, hue="sex", kind="swarm")  # Swarm plot of total_bill by day (colored by sex)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f417cfd48b0>
```



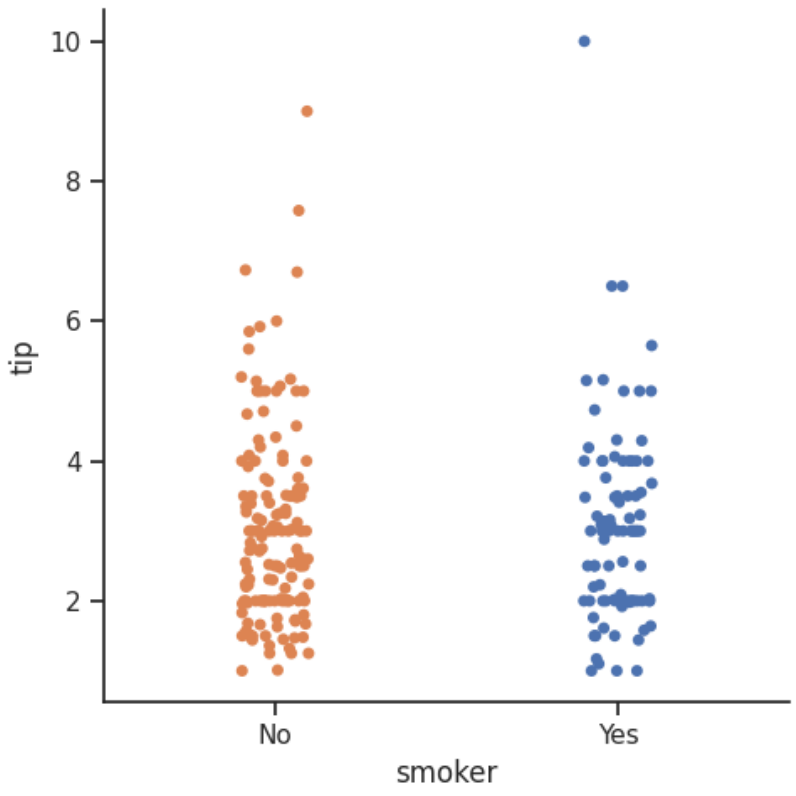

```
# Swarm plot with filtered data and colors based on "size"
sns.catplot(x="size", y="total_bill", data=tips.query("size!=3"), hue="size", kind="swarm")  # Swarm plot of total_bill by size (excluding size 3)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f417d040730>
```



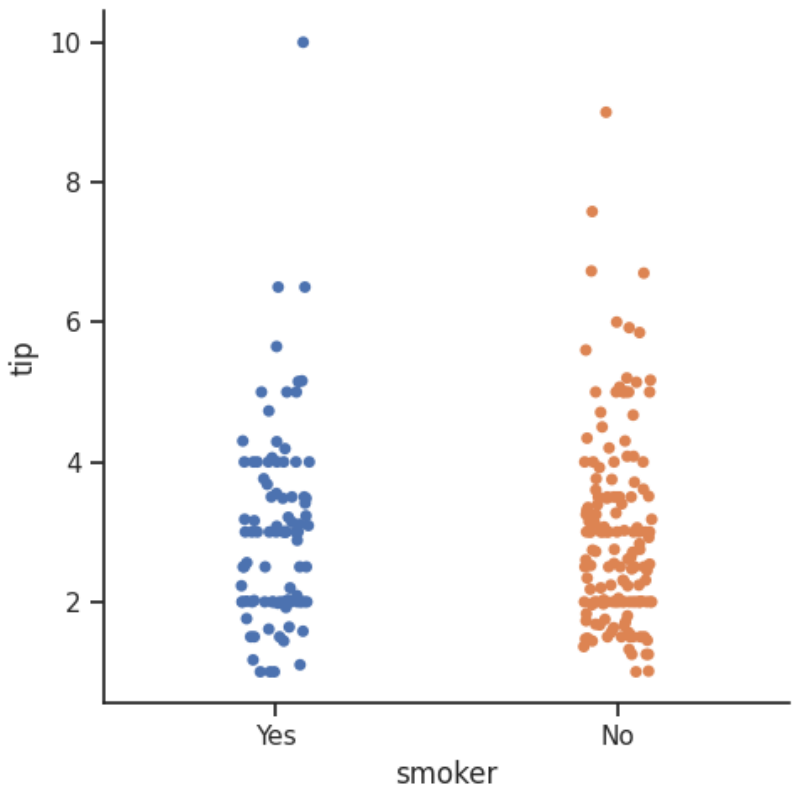

```
# Scatterplot with "smoker" category and ordered hue
sns.catplot(x="smoker", y="tip", data=tips, hue="smoker", order=['No','Yes'])  # Scatterplot of tip by smoker status (ordered hue)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f417ceea170>
```



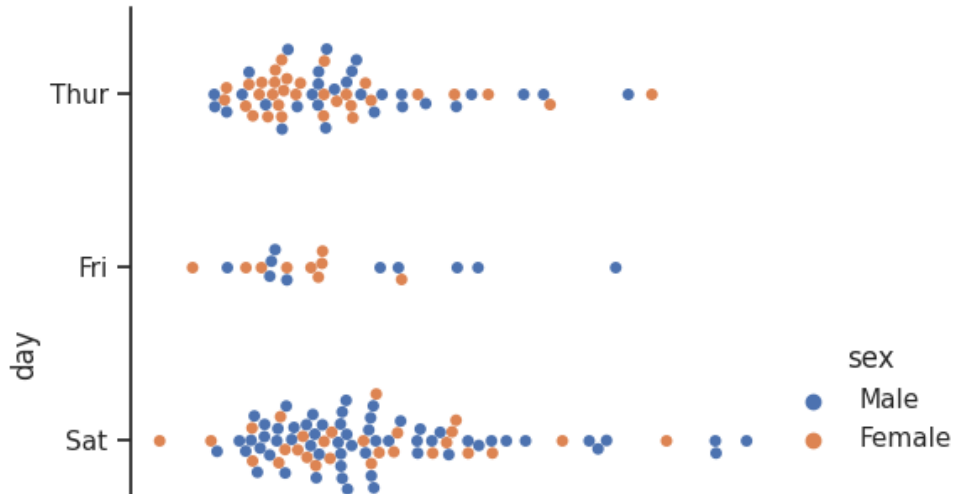

```
# Scatterplot with "smoker" category and reversed hue order
sns.catplot(x="smoker", y="tip", data=tips, hue="smoker", order=['Yes','No'])  # Scatterplot of tip by smoker status (reversed hue order)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f417ce0fcd0>
```

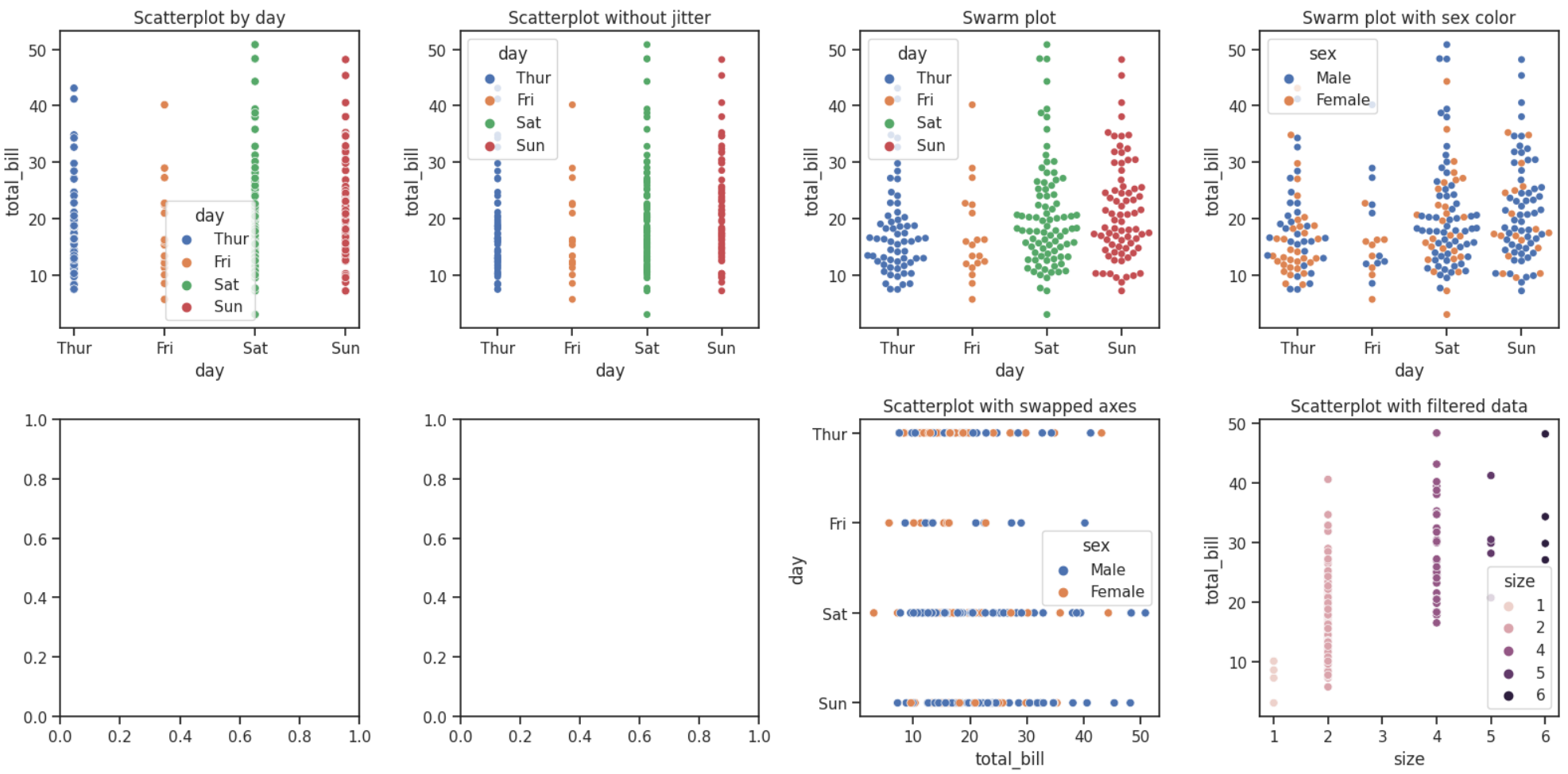



```
# Scatterplot with "sex" category and swapped axes
sns.catplot(y="day", x="total_bill", data=tips, hue="sex", kind="swarm")  # Swarm plot of total_bill by day (swapped axes)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f417ce41060>
```



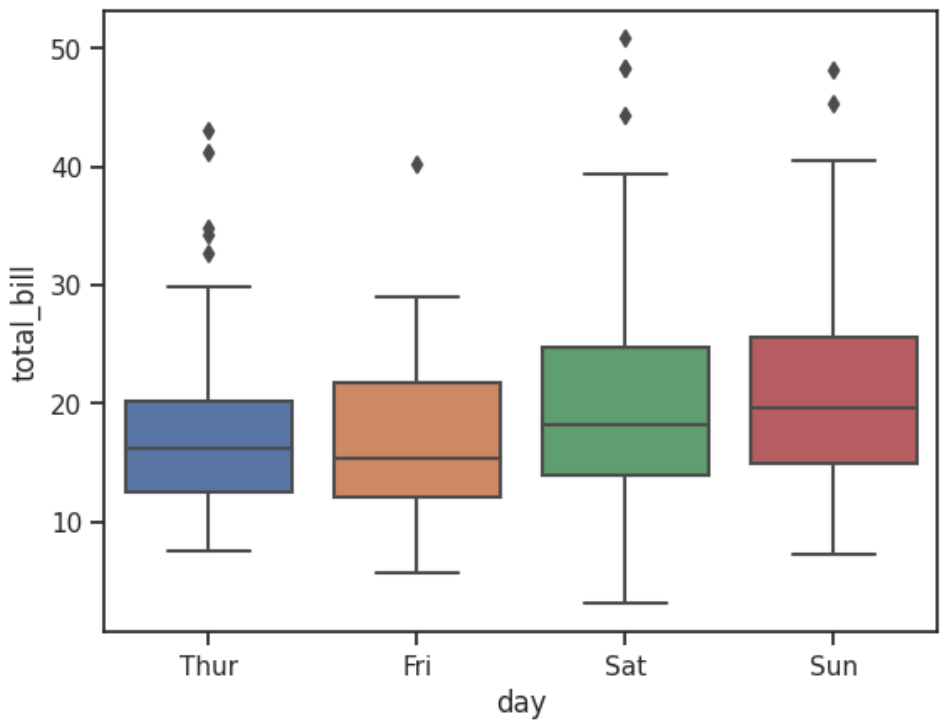

```
fig, axes = plt.subplots(2, 4, figsize=(16, 8))  # Create a figure with two rows and four columns

# Scatterplot colored by "day"
sns.scatterplot(x="day", y="total_bill", data=tips, hue="day", ax=axes[0, 0])  # Scatterplot of total_bill by day
axes[0, 0].set_title("Scatterplot by day")  # Set subplot title

# Scatterplot without jitter effect
sns.stripplot(x="day", y="total_bill", data=tips, hue="day", jitter=False, ax=axes[0, 1])  # Scatterplot of total_bill by day (without jitter)
axes[0, 1].set_title("Scatterplot without jitter")  # Set subplot title

# Swarm plot
sns.swarmplot(x="day", y="total_bill", data=tips, hue="day", ax=axes[0, 2])  # Swarm plot of total_bill by day
axes[0, 2].set_title("Swarm plot")  # Set subplot title

# Swarm plot with colors based on "sex"
sns.swarmplot(x="day", y="total_bill", data=tips, hue="sex", ax=axes[0, 3])  # Swarm plot of total_bill by day (colored by sex)
axes[0, 3].set_title("Swarm plot with sex color")  # Set subplot title

# Scatterplot with "smoker" category and ordered hue
#sns.catplot(x="smoker", y="tip", data=tips, hue="smoker", order=['No', 'Yes'], ax=axes[1, 0])  # Scatterplot of tip by smoker status (ordered hue)
#axes[1, 0].set_title("Scatterplot with ordered hue")  # Set subplot title

# Scatterplot with "smoker" category and reversed hue order
#sns.catplot(x="smoker", y="tip", data=tips, hue="smoker", order=['Yes', 'No'], ax=axes[1, 1])  # Scatterplot of tip by smoker status (reversed hue order)
#axes[1, 1].set_title("Scatterplot with reversed hue order")  # Set subplot title

# Scatterplot with "sex" category and swapped axes
sns.scatterplot(y="day", x="total_bill", data=tips, hue="sex", ax=axes[1, 2])  # Scatterplot of total_bill by day (swapped axes)
axes[1, 2].set_title("Scatterplot with swapped axes")  # Set subplot title

# Scatterplot with filtered data and colors based on "size"
sns.scatterplot(x="size", y="total_bill", data=tips.query("size != 3"), hue="size", ax=axes[1, 3])  # Scatterplot of total_bill by size (excluding size 3)
axes[1, 3].set_title("Scatterplot with filtered data")  # Set subplot title

plt.tight_layout()  # Adjust spacing between subplots
plt.show()  # Display the figure
```



Distributions of observation within categorical variable

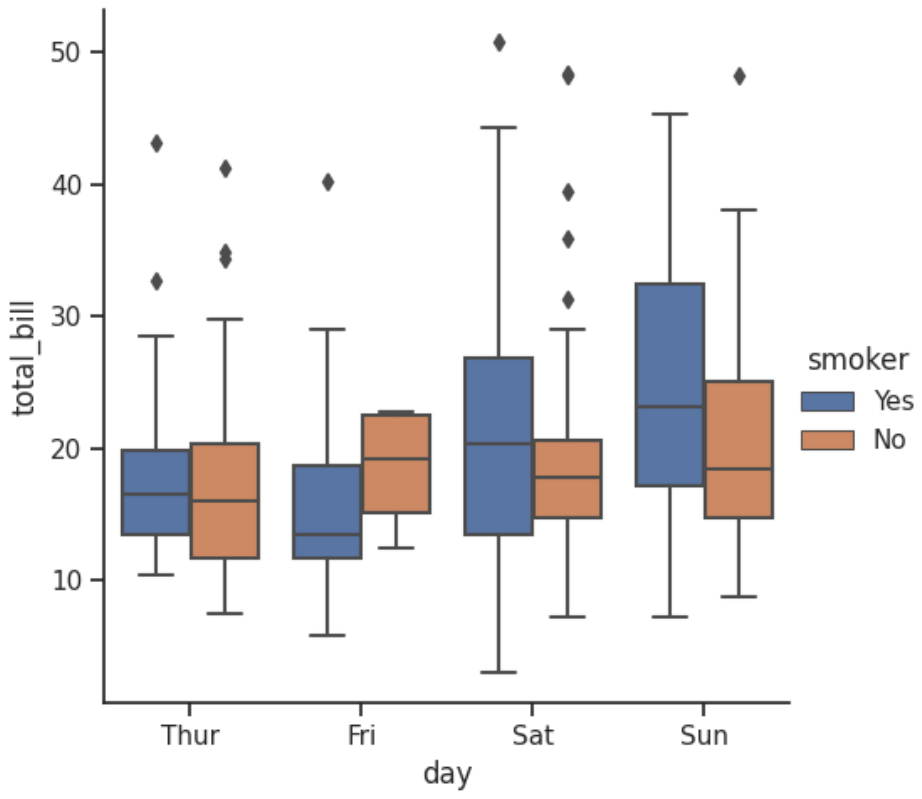
```
# Boxplot
sns.boxplot(x="day", y="total_bill", data=tips)  # Boxplot of total_bill by day
```

```
<Axes: xlabel='day', ylabel='total_bill'>
```



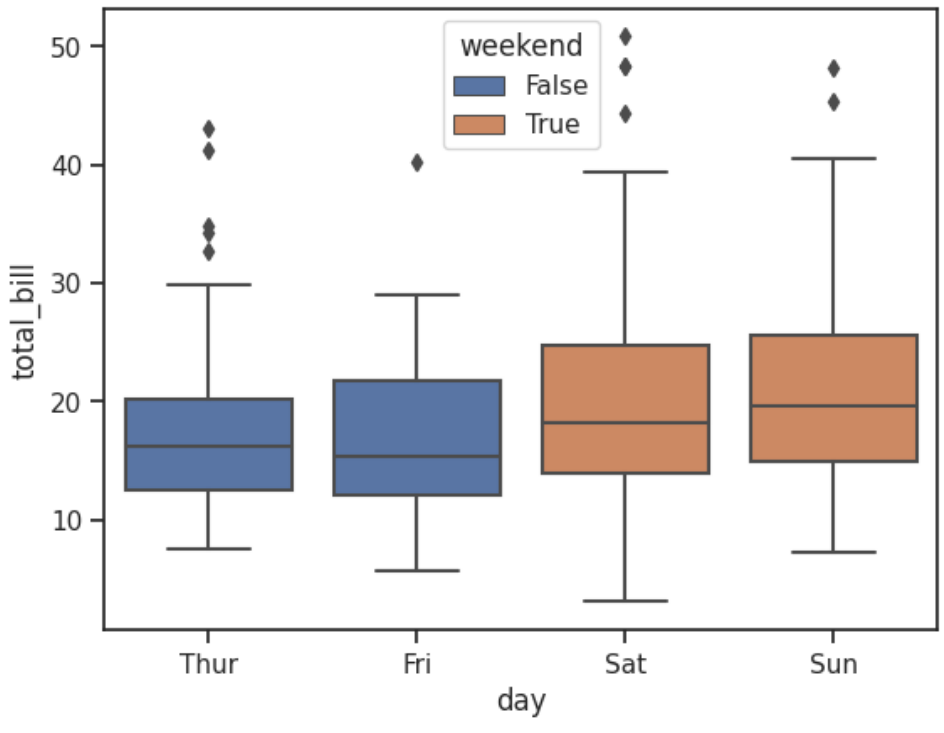

```
# Boxplot with "smoker" category
sns.catplot(x="day", y="total_bill", data=tips, hue="smoker", kind="box")  # Boxplot of total_bill by day (colored by smoker status)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f41799e59c0>
```



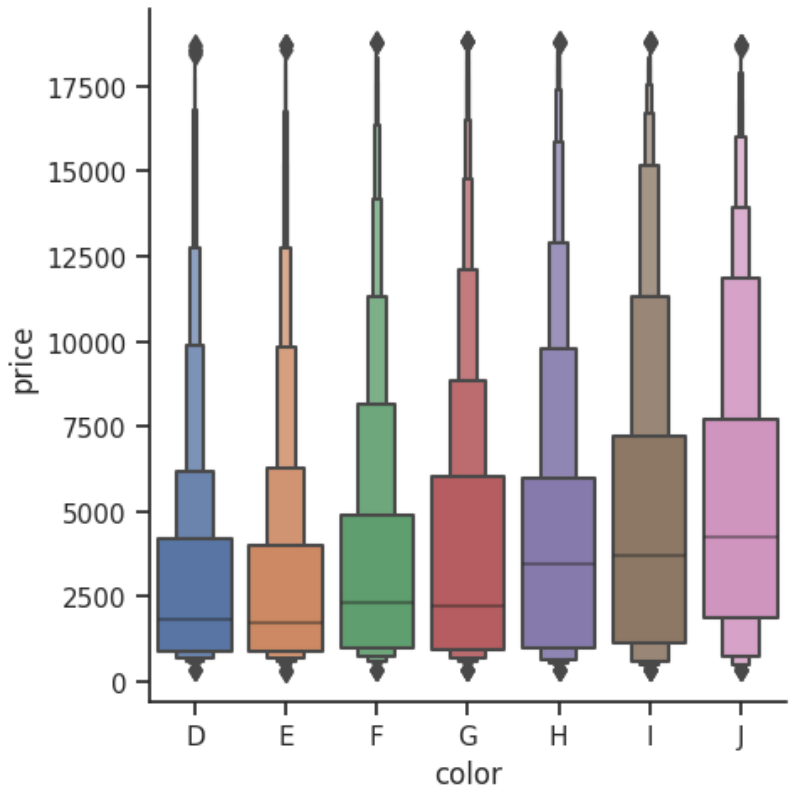

```
# Boxplot with "weekend" category without separating boxes
tips["weekend"] = tips["day"].isin(["Sat", "Sun"])  # Create a new column indicating if it's a weekend
sns.boxplot(x="day", y="total_bill", data=tips, hue="weekend", dodge=False)  # Boxplot of total_bill by day (colored by weekend)
```

```
<Axes: xlabel='day', ylabel='total_bill'>
```



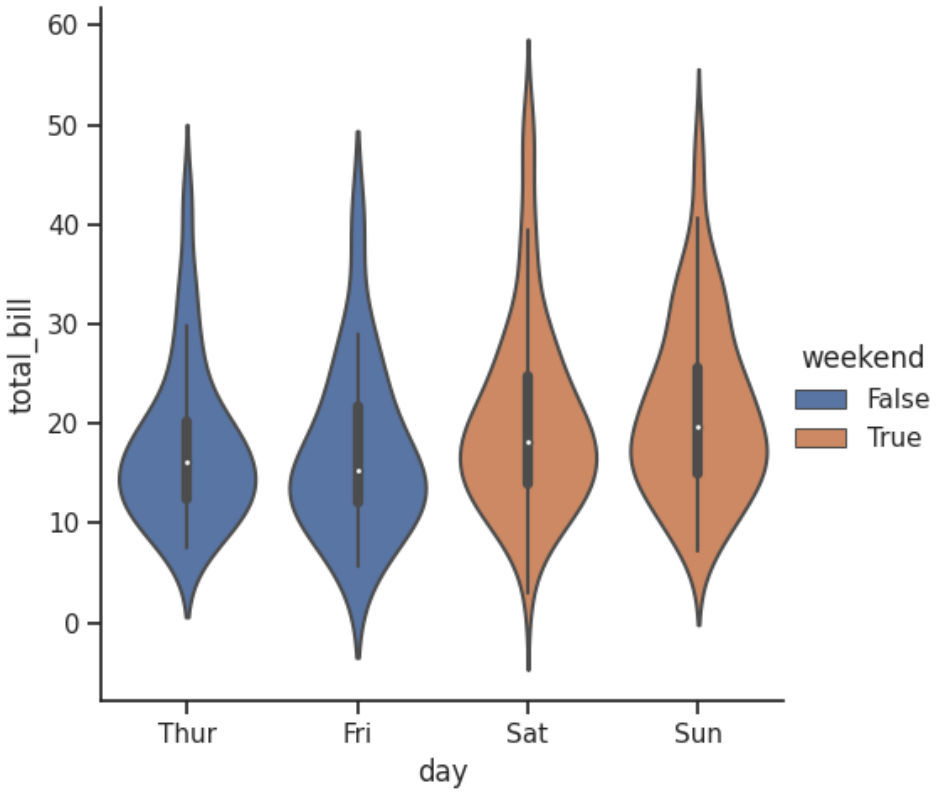

```
# Boxen plot with "color" category
diamonds = sns.load_dataset("diamonds")  # Load the diamonds dataset from seaborn
sns.catplot(x="color", y="price", kind="boxen", data=diamonds.sort_values("color"))  # Boxen plot of price by color
```

```
<seaborn.axisgrid.FacetGrid at 0x7f417bce5630>
```



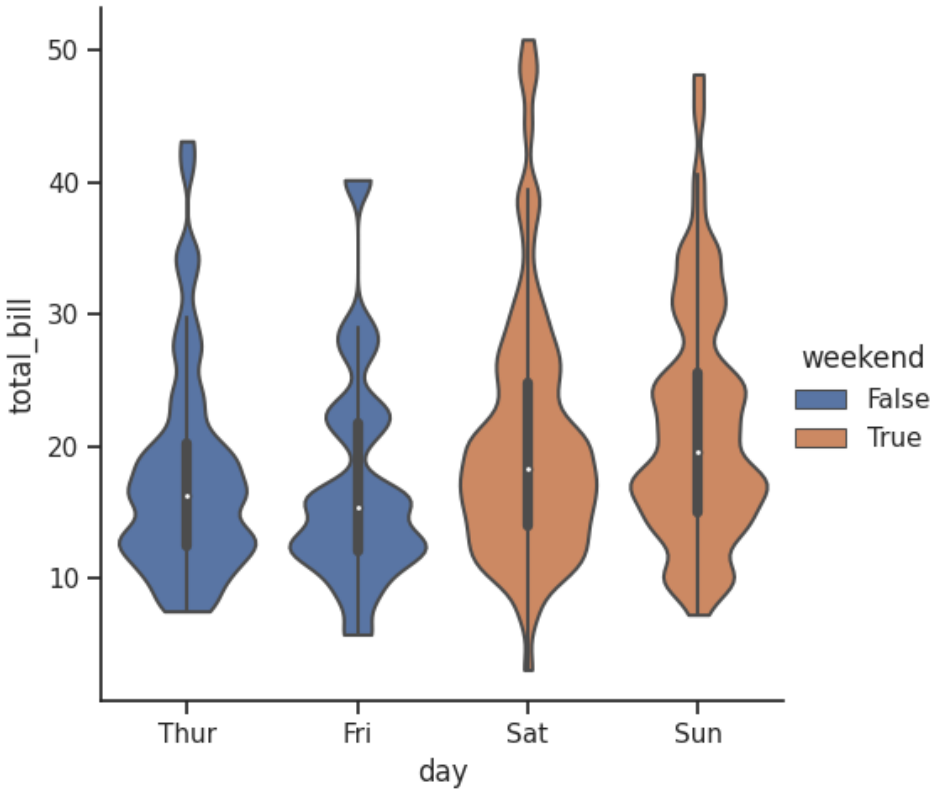

```
# Violin plot with "weekend" category
sns.catplot(x="day", y="total_bill", data=tips, hue="weekend", dodge=False, kind="violin")  # Violin plot of total_bill by day (colored by weekend)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f417937ca30>
```



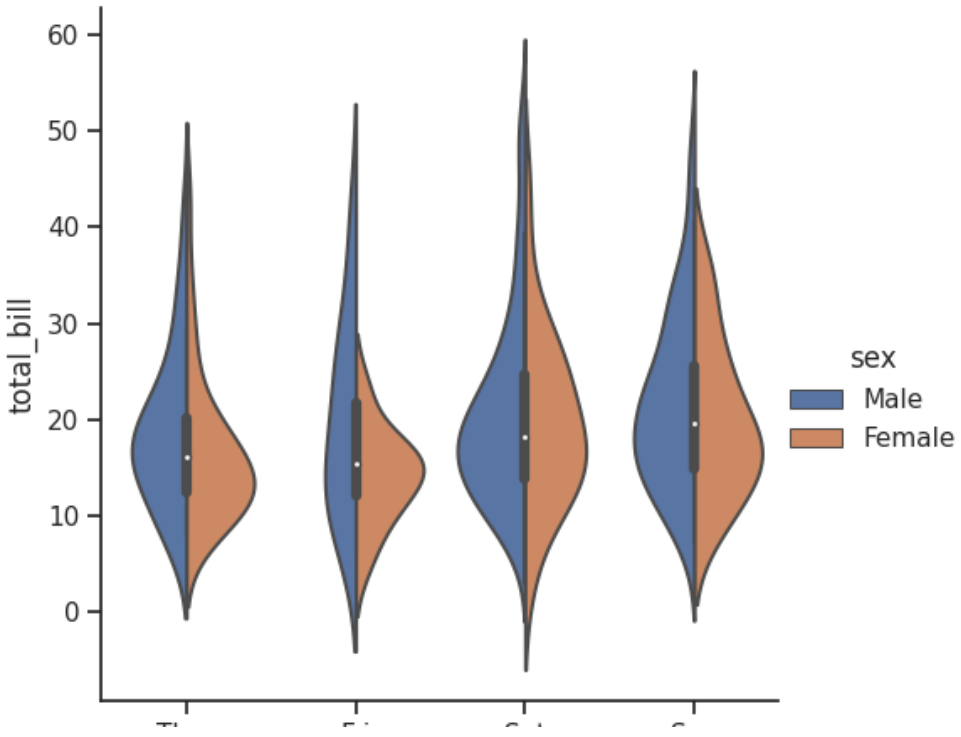

```
# Violin plot with customized bandwidth and cutoff
sns.catplot(x="day", y="total_bill", data=tips, hue="weekend", dodge=False, kind="violin", bw=.15, cut=0)  # Violin plot of total_bill by day (colored by weeke
```

```
<seaborn.axisgrid.FacetGrid at 0x7f417befff10>
```



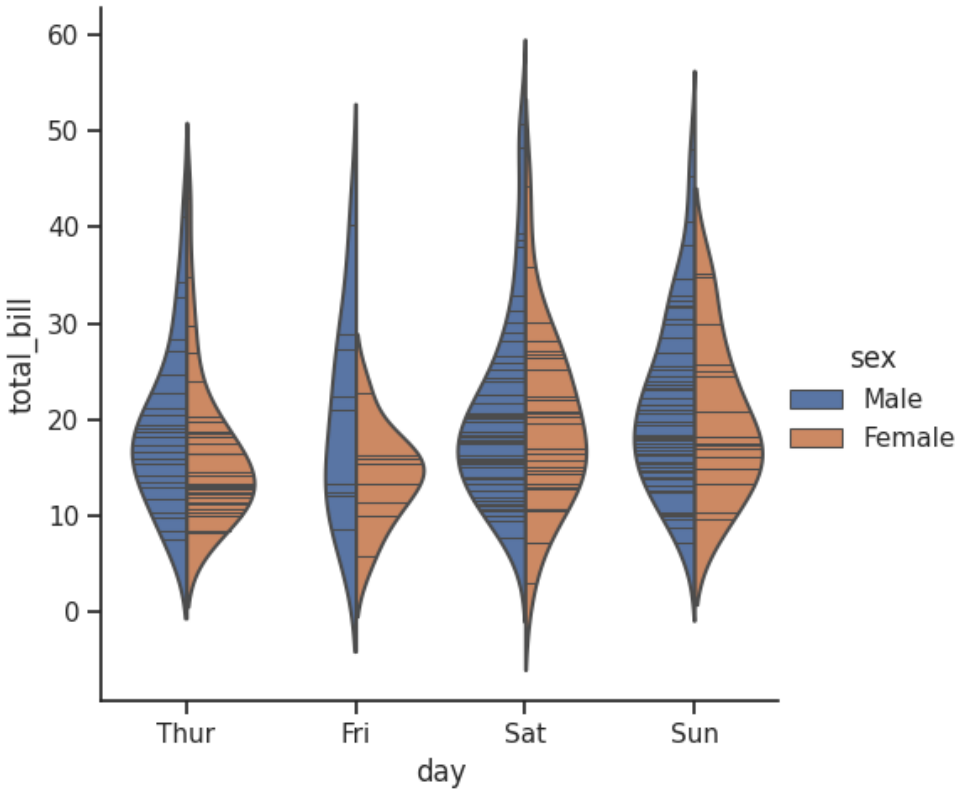

```
# Violin plot with split violins based on "sex"
sns.catplot(x="day", y="total_bill", data=tips, hue="sex", split=True, kind="violin")  # Violin plot of total_bill by day (split by sex)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f417c006050>
```

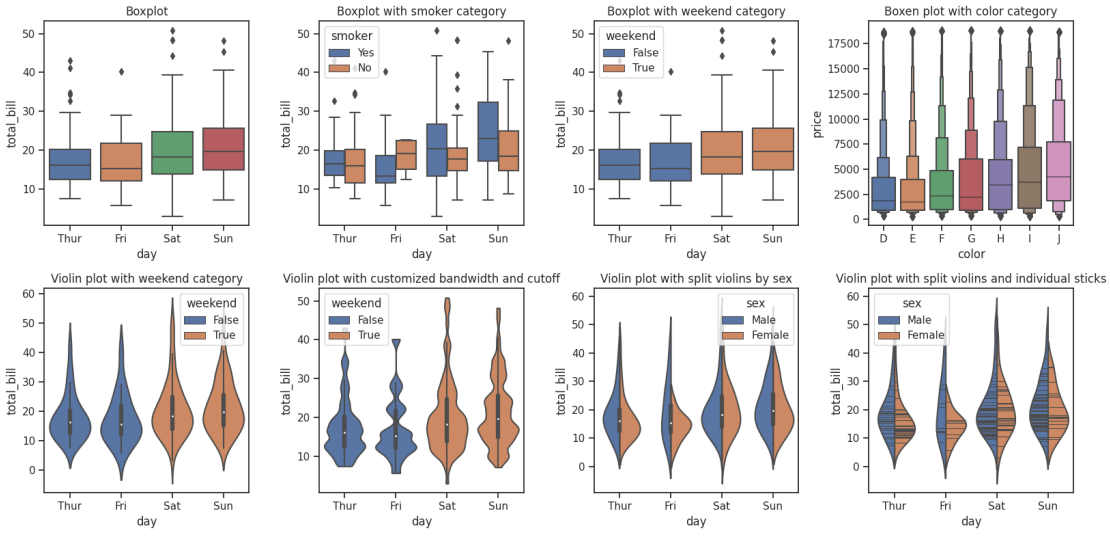



```
# Violin plot with split violins, individual sticks, and "sex" category
sns.catplot(x="day", y="total_bill", data=tips, hue="sex", inner="stick", split=True, kind="violin")  # Violin plot of total_bill by day (split by sex) with in
```

```
<seaborn.axisgrid.FacetGrid at 0x7f417c051e70>
```




```
fig, axes = plt.subplots(2, 4, figsize=(16, 8))

# Boxplot
sns.boxplot(x="day", y="total_bill", data=tips, ax=axes[0, 0])
axes[0, 0].set_title("Boxplot")

# Boxplot with "smoker" category
sns.boxplot(x="day", y="total_bill", data=tips, hue="smoker", ax=axes[0, 1])
axes[0, 1].set_title("Boxplot with smoker category")

# Boxplot with "weekend" category without separating boxes
tips["weekend"] = tips["day"].isin(["Sat", "Sun"])
sns.boxplot(x="day", y="total_bill", data=tips, hue="weekend", dodge=False, ax=axes[0, 2])
axes[0, 2].set_title("Boxplot with weekend category")

# Boxen plot with "color" category
sns.boxenplot(x="color", y="price", data=diamonds.sort_values("color"), ax=axes[0, 3])
axes[0, 3].set_title("Boxen plot with color category")

# Violin plot with "weekend" category
sns.violinplot(x="day", y="total_bill", data=tips, hue="weekend", dodge=False, ax=axes[1, 0])
axes[1, 0].set_title("Violin plot with weekend category")

# Violin plot with customized bandwidth and cutoff
sns.violinplot(x="day", y="total_bill", data=tips, hue="weekend", dodge=False, bw=.15, cut=0, ax=axes[1, 1])
axes[1, 1].set_title("Violin plot with customized bandwidth and cutoff")

# Violin plot with split violins based on "sex"
sns.violinplot(x="day", y="total_bill", data=tips, hue="sex", split=True, ax=axes[1, 2])
axes[1, 2].set_title("Violin plot with split violins by sex")

# Violin plot with split violins, individual sticks, and "sex" category
sns.violinplot(x="day", y="total_bill", data=tips, hue="sex", split=True, inner="stick", ax=axes[1, 3])
axes[1, 3].set_title("Violin plot with split violins and individual sticks")

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


Boxplot
Boxplot with smoker category
smoker
Yes
No
Boxplot with weekend category
weekend
False
True
Boxen plot with color category
price
color
Violin plot with weekend category
Violin plot with customized bandwidth and cutoff
Violin plot with split violins by sex
sex
Male
Female
Violin plot with split violins and individual sticks
total_bill
day
Thur
Fri
Sat
Sun