seaborn

October 19, 2024

Seaborn is a powerful Python data visualization library built on top of Matplotlib. It simplifies complex visualizations and enhances them with attractive, informative statistical graphics. Here's a brief overview:

0.0.1 Key Features of Seaborn:

- **Built on Matplotlib**: It extends Matplotlib's functionality with more refined, high-level interfaces for drawing attractive and informative statistical graphics.
- **Easier Syntax**: Seaborn makes complex plots easier to create with fewer lines of code compared to Matplotlib.
- **In-built Themes**: It comes with built-in themes (darkgrid, whitegrid, etc.), making visualizations aesthetically pleasing.
- **DataFrames Integration**: Seaborn works well with pandas DataFrames, allowing easy plotting of data directly from the DataFrame.
- **Statistical Visualizations**: Seaborn simplifies visualizations for statistical relationships, making plots like regression lines, box plots, and violin plots easy to generate.
- **Automatic Plot Aesthetics**: It manages plot sizes and aesthetic elements like color palettes, legends, and more.

0.0.2 Quick Comparison: Seaborn vs. Matplotlib:

- **Ease of Use**: Seaborn has a simpler syntax and is more intuitive for complex visualizations, whereas Matplotlib may require more configuration and code.
- **Plot Customization**: While Seaborn provides automatic styling, Matplotlib offers more customization control over the finer details of the plot.
- **Statistical Plotting**: Seaborn is designed specifically for statistical visualizations, with built-in support for tasks like plotting regression lines, while Matplotlib requires manual work to achieve similar effects.
- **Integration with Pandas**: Seaborn integrates better with Pandas, allowing easy plotting directly from DataFrames, whereas Matplotlib often requires additional steps.

[1]:
```
# Importing Major Libraries

import numpy as np
import pandas as pd
import seaborn as sns # seaborn library
import matplotlib.pyplot as plt
import plotly.express as px # Plotly
```

```python
# importing additional libraries
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```python
# importing datasets

sns.get_dataset_names()
```

[2]:
```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'dowjones',
 'exercise',
 'flights',
 'fmri',
 'geyser',
 'glue',
 'healthexp',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'seaice',
 'taxis',
 'tips',
 'titanic']
```

[3]:
```python
iris = sns.load_dataset('iris') # seaborn datasets
mpg = sns.load_dataset('mpg') # seaborn datasets
tips = sns.load_dataset('tips') # seaborn datasets
titanic = sns.load_dataset('titanic') # seaborn datasets
```

[4]:
```python
gap = px.data.gapminder() # plotly dataset
```

[5]:
```python
iris.sample(5)
```

[5]:

	sepal_length	sepal_width	petal_length	petal_width	species
132	6.4	2.8	5.6	2.2	virginica
41	4.5	2.3	1.3	0.3	setosa
116	6.5	3.0	5.5	1.8	virginica
111	6.4	2.7	5.3	1.9	virginica
25	5.0	3.0	1.6	0.2	setosa

0.0.3 Iris Dataset

- **Description**: Contains 150 iris flower samples with 4 features each, classified into 3 species.
- **Source**: Introduced by Ronald A. Fisher in 1936.
- **Use**: Classification and clustering tasks.

0.0.4 Features:

1. Sepal Length (cm)
2. Sepal Width (cm)
3. Petal Length (cm)
4. Petal Width (cm)
5. Species (Setosa, Versicolor, Virginica)

0.0.5 Size:

- **Records**: 150
- **Features**: 4 numerical, 1 categorical

0.0.6 Class Distribution:

- Setosa: 50
- Versicolor: 50
- Virginica: 50

0.0.7 Use Cases:

- Classification, clustering, EDA, feature engineering.

[6]:
```
mpg.head()
```

[6]:
```
    mpg  cylinders  displacement  horsepower  weight  acceleration  \
0  18.0          8         307.0       130.0    3504          12.0
1  15.0          8         350.0       165.0    3693          11.5
2  18.0          8         318.0       150.0    3436          11.0
3  16.0          8         304.0       150.0    3433          12.0
4  17.0          8         302.0       140.0    3449          10.5

   model_year origin                       name
0          70    usa  chevrolet chevelle malibu
1          70    usa          buick skylark 320
2          70    usa         plymouth satellite
3          70    usa              amc rebel sst
4          70    usa                ford torino
```

0.0.8 MPG Dataset

- **Description**: Contains fuel consumption data (miles per gallon) for cars from 1970 to 1982, along with various car attributes.
- **Source**: Collected by the U.S. Environmental Protection Agency (EPA).
- **Use**: Regression and exploratory data analysis.

0.0.9 Features:

1. **MPG**: Miles per gallon (numeric, target)
2. **Cylinders**: Number of engine cylinders (numeric)
3. **Displacement**: Engine displacement (cubic inches) (numeric)
4. **Horsepower**: Engine horsepower (numeric, some missing values)
5. **Weight**: Vehicle weight (pounds) (numeric)
6. **Acceleration**: Time to accelerate from 0 to 60 mph (numeric)
7. **Model Year**: Year of the car (numeric)
8. **Origin**: Country of origin (categorical: USA, Europe, Japan)
9. **Car Name**: Name of the car (string)

0.0.10 Size:

- **Records**: 398
- **Features**: 8 (7 numeric, 1 categorical)

0.0.11 Missing Values:

- **Horsepower**: Contains missing values.

0.0.12 Use Cases:

- Regression, data preprocessing, feature engineering, and predictive modeling.

0.0.13 Challenges:

- Missing values in horsepower.
- Mixed data types (numeric, categorical, string).

[7]:
```
tips.head()
```

[7]:

	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
```

0.0.14 Tips Dataset

- **Description**: Contains information about tips given by customers in a restaurant, including various factors influencing the tip amount.
- **Source**: Collected by a restaurant and commonly used for data visualization and regression tasks.
- **Use**: Regression, exploratory data analysis, and statistics.

0.0.15 Features:

1. **Total Bill**: Total bill amount (numeric)
2. **Tip**: Tip amount given (numeric, target)
3. **Sex**: Gender of the person paying (categorical: Male, Female)
4. **Smoker**: Whether the person is a smoker (categorical: Yes, No)
5. **Day**: Day of the week (categorical: Thurs, Fri, Sat, Sun)
6. **Time**: Time of the meal (categorical: Lunch, Dinner)
7. **Size**: Number of people in the party (numeric)

0.0.16 Size:

- **Records**: 244
- **Features**: 7 (4 categorical, 3 numeric)

0.0.17 Missing Values:

- No missing values.

0.0.18 Use Cases:

- Regression (predicting tips), data visualization, statistical analysis.

0.0.19 Challenges:

- Small dataset, useful for simple models and visualization.

[8]:
```
titanic.head()
```

[8]:
```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

0.0.20 Titanic Dataset

- **Description**: Contains information about the passengers aboard the RMS Titanic, including details about their survival status and various personal attributes.
- **Source**: Collected from the Titanic disaster (1912), commonly used for data science and machine learning tasks.
- **Use**: Classification, exploratory data analysis, and machine learning.

0.0.21 Features:

1. **PassengerId**: Unique ID for each passenger (numeric)
2. **Survived**: Survival status (0 = No, 1 = Yes) (binary, target)
3. **Pclass**: Passenger class (1st, 2nd, or 3rd) (categorical)
4. **Name**: Name of the passenger (string)
5. **Sex**: Gender of the passenger (categorical: Male, Female)
6. **Age**: Age of the passenger (numeric, some missing values)
7. **SibSp**: Number of siblings/spouses aboard (numeric)
8. **Parch**: Number of parents/children aboard (numeric)
9. **Ticket**: Ticket number (string)
10. **Fare**: Ticket fare (numeric)
11. **Cabin**: Cabin number (string, some missing values)
12. **Embarked**: Port of embarkation (categorical: C = Cherbourg, Q = Queenstown, S = Southampton)

0.0.22 Size:

- **Records**: 891 (in the training set)
- **Features**: 12 (8 numeric, 4 categorical)

0.0.23 Missing Values:

- **Age**: Some missing values.
- **Cabin**: Many missing values.
- **Embarked**: Few missing values.

0.0.24 Use Cases:

- Classification (predicting survival), data preprocessing, exploratory data analysis, and feature engineering.

0.0.25 Challenges:

- Missing values in Age and Cabin.
- Mixed data types (numeric, categorical, string).

[9]:
```
gap.sample(5)
```

[9]:
```
                       country continent  year  lifeExp      pop    gdpPercap  \
1252               Puerto Rico  Americas  1972   72.160  2847132  9123.041742
258   Central African Republic    Africa  1982   48.295  2476971   956.752991
199               Burkina Faso    Africa  1987   49.557  7586551   912.063142
1286                    Rwanda    Africa  1962   43.000  3051242   597.473073
1696                  Zimbabwe    Africa  1972   55.635  5861135   799.362176

     iso_alpha  iso_num
1252       PRI      630
258        CAF      140
199        BFA      854
1286       RWA      646
1696       ZWE      716
```

0.0.26 Gapminder Dataset

- **Description**: Contains global data on various socioeconomic indicators, including life expectancy, income, and population across different countries over time.
- **Source**: Compiled by the Gapminder Foundation, an organization that promotes sustainable global development.
- **Use**: Data visualization, exploratory data analysis, and socioeconomic research.

0.0.27 Features:

1. **Country**: Name of the country (categorical)
2. **Year**: Year of observation (numeric)
3. **Continent**: Continent of the country (categorical)

4. **Life Expectancy**: Average life expectancy (numeric)
5. **GDP per Capita**: Gross Domestic Product per capita (numeric)
6. **Population**: Total population (numeric)
7. **Income Group**: Classification of countries into income groups (categorical: Low, Lower-Middle, Upper-Middle, High)

0.0.28 Size:

- **Records**: Approximately 1700 (varies by specific version)
- **Features**: 7 (4 numeric, 3 categorical)

0.0.29 Missing Values:

- Some countries and years may have missing values for GDP or life expectancy.

0.0.30 Use Cases:

- Data visualization (e.g., scatter plots, time series), statistical analysis, and trend analysis in socioeconomic research.

0.0.31 Challenges:

- Handling missing data and ensuring accurate interpretation of trends over time and across different regions.

0.0.32 1. Axes-Level Functions:

- These functions create **one plot at a time** on a specific part of the figure (called an "axes").
- If you want multiple plots, you have to **manually set up the layout** (like the size, number of plots, etc.).
- They give you more **control** over the details of the plot, which is useful for customizing things like titles, labels, or axes.
- **Examples**: `sns.scatterplot()`, `sns.barplot()`, `sns.lineplot()`.
- Use axes-level functions when you are working on **single plots** or want to fully control how the plot looks.

0.0.33 2. Figure-Level Functions:

- These functions automatically handle the creation of **multiple plots** and the overall layout (the whole "figure").
- You don't need to worry about setting up the figure or arranging plots, as Seaborn does it for you.
- They are great for creating **complex visualizations** or when you want multiple plots in one figure (e.g., subplots, grids).

- **Examples**: `sns.catplot()`, `sns.relplot()`, `sns.pairplot()`.
- Use figure-level functions when you need to **create multiple plots at once** or want Seaborn to manage the layout automatically.

In summary: - **Axes-level**: Best for **single, simple plots** with more control. - **Figure-level**: Best for **complex layouts** or when you want Seaborn to handle multiple plots automatically.

0.0.34 Relational Plots in Seaborn

Relational plots are used to visualize relationships between two variables. Two common types of relational plots in Seaborn are **scatter plots** and **line plots**.

1. **Scatter Plot**:
    - **Purpose**: Displays individual data points on a two-dimensional graph, showing the relationship between two numerical variables. Each point represents an observation, and its position is determined by the values of the two variables.
    - **Use Cases**:
        - To identify trends, clusters, or outliers in the data.
        - To visualize how one variable affects another, helping in understanding correlations.
    - **Features**:
        - Can include additional dimensions through point size, color, and style to represent categorical variables.
        - Useful for both small and large datasets.
2. **Line Plot**:
    - **Purpose**: Connects individual data points with lines to show the trend over time or ordered categories. It's especially useful for visualizing continuous data.
    - **Use Cases**:
        - To display trends over a period (like sales over months or temperature changes over days).
        - To compare different groups or categories across the same x-axis values.
    - **Features**:
        - Can show multiple lines on the same plot for different groups, making it easy to compare their trends.
        - Allows for additional formatting, like markers at data points to enhance clarity.

0.0.35 Summary:

- **Scatter plots** visualize the relationship between two numerical variables, helping to identify patterns and correlations. **Line plots** show trends over time or ordered categories, making it easy to compare data points across a continuous scale. Both types of plots are essential for exploratory data analysis and understanding relationships in datasets.

[10]:
```
# sns.lineplot // Axes level
# sns.scatterplot // Axes level
# sns.relplot // Figure level
```

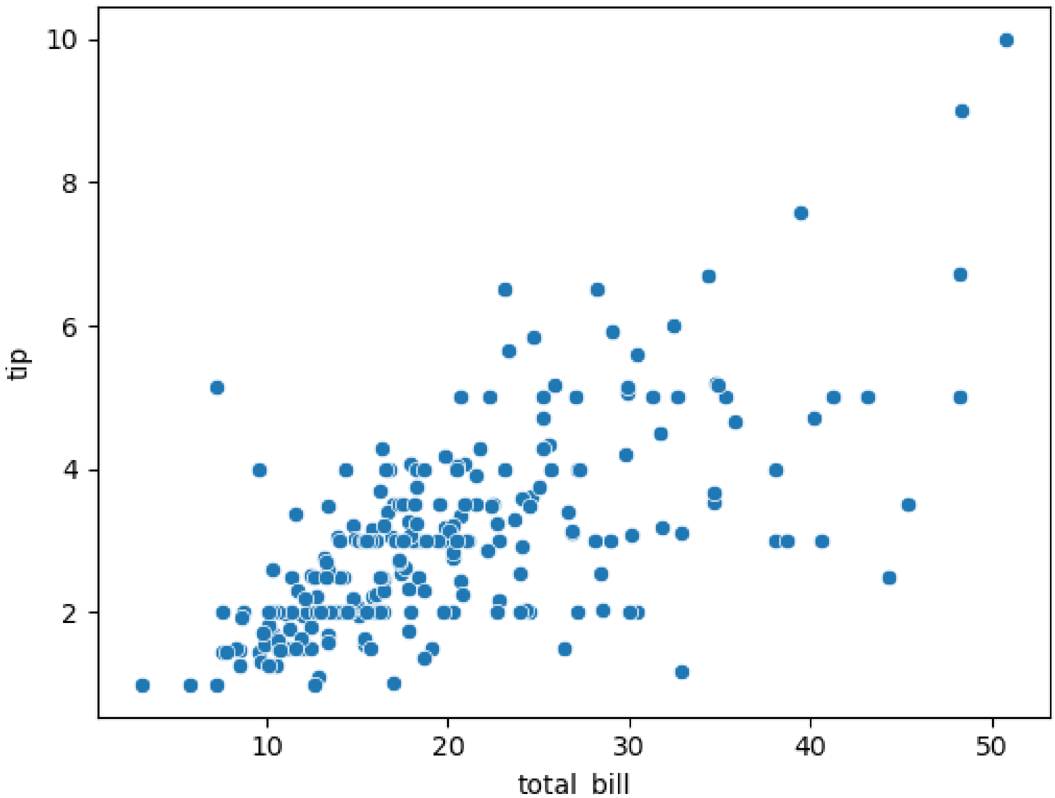
[11]:
```
# scatterplots

tips.sample(5)
```

[11]:

	total_bill	tip	sex	smoker	day	time	size
197	43.11	5.00	Female	Yes	Thur	Lunch	4
31	18.35	2.50	Male	No	Sat	Dinner	4
84	15.98	2.03	Male	No	Thur	Lunch	2
170	50.81	10.00	Male	Yes	Sat	Dinner	3
102	44.30	2.50	Female	Yes	Sat	Dinner	3

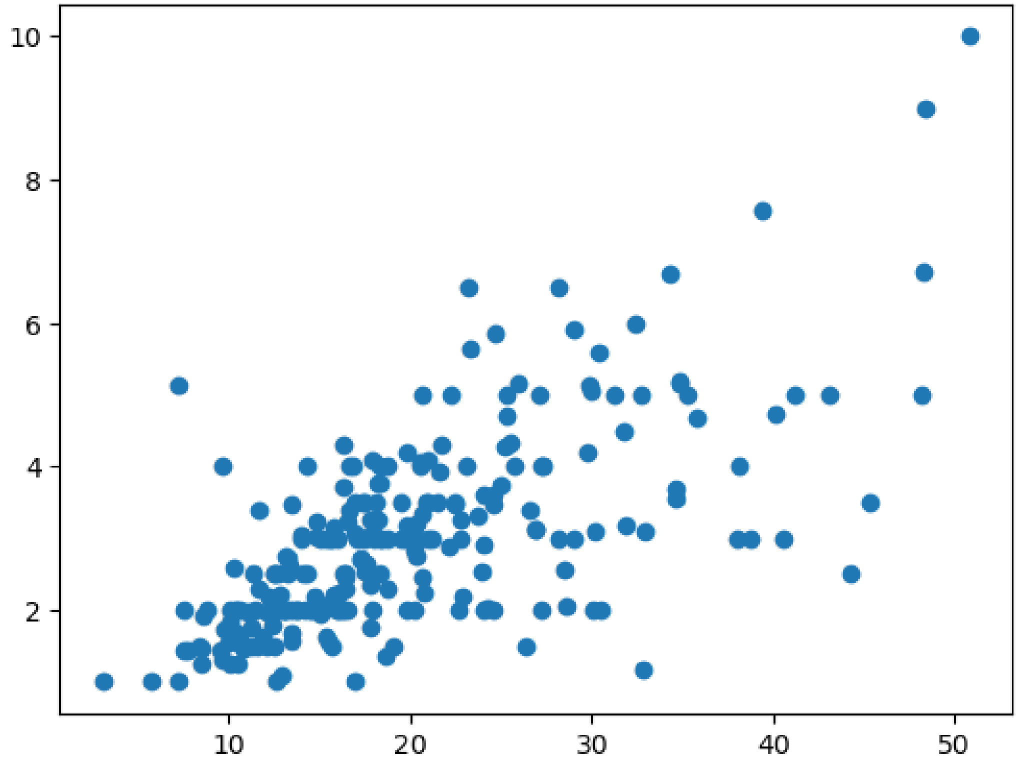
[12]:
```
# difference between matplotlib graphs and Seaborn graphs
# Seaborn Graph
sns.scatterplot(data= tips, x= 'total_bill',y= 'tip')
```

[12]: <Axes: xlabel='total_bill', ylabel='tip'>

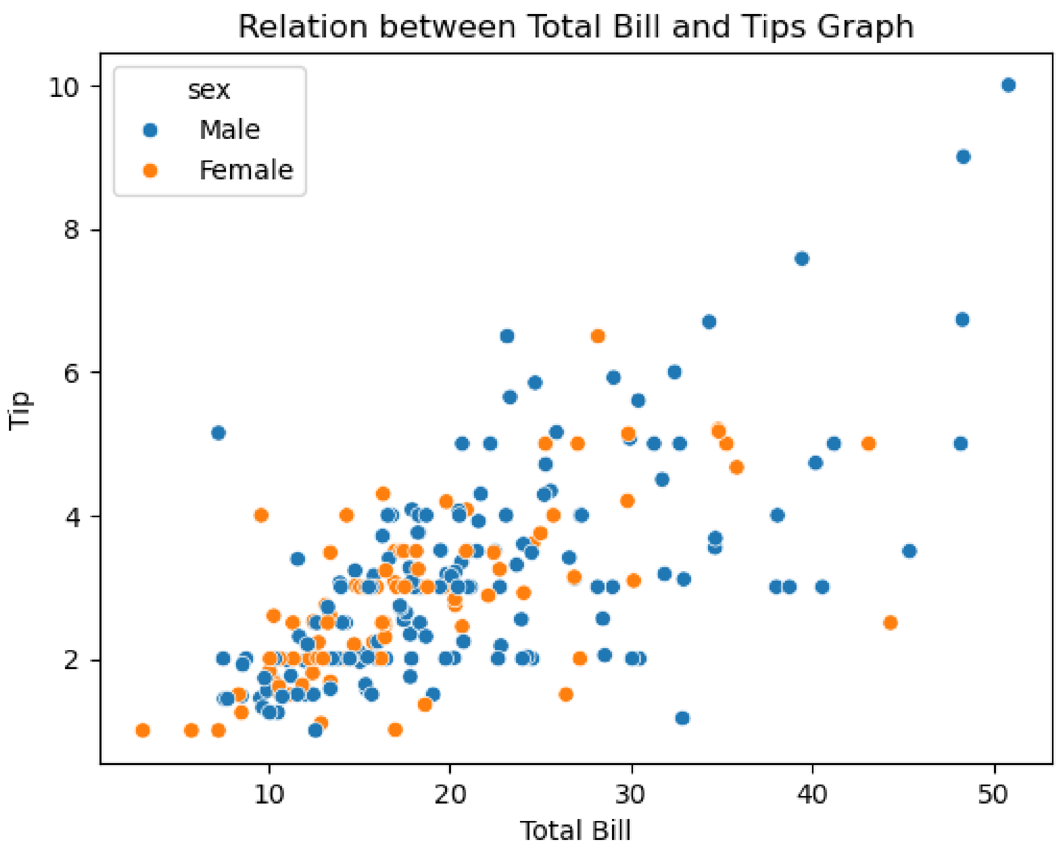


[13]:
```
# Matplotlib graph
plt.scatter(tips.total_bill,tips.tip)
```

[13]: <matplotlib.collections.PathCollection at 0x2821791be10>



[14]:
```
# axes level - Scatterplot

sns.scatterplot(data= tips, x= 'total_bill',y= 'tip',hue='sex') #␣
 ↪style='time',size='size'
plt.title('Relation between Total Bill and Tips Graph')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```



0.0.36 Hue Parameter in Seaborn

- **Definition**: `hue` adds color differentiation based on a categorical variable in visualizations.
- **Purpose**: Enhances data visualization by grouping data points, making patterns easier to identify.
- **Usage**: Applicable in various plots like:
  - `sns.scatterplot()`
  - `sns.lineplot()`
  - `sns.barplot()`
  - `sns.boxplot()`
  - `sns.histplot()`
- **Examples**:

```
sns.scatterplot(data=tips, x='total_bill', y='tip', hue='day')
sns.lineplot(data=tips, x='total_bill', y='tip', hue='time')
sns.boxplot(data=tips, x='day', y='total_bill', hue='sex')
```

- **Customization**:
  - **Palette**: Customize colors using `palette`.

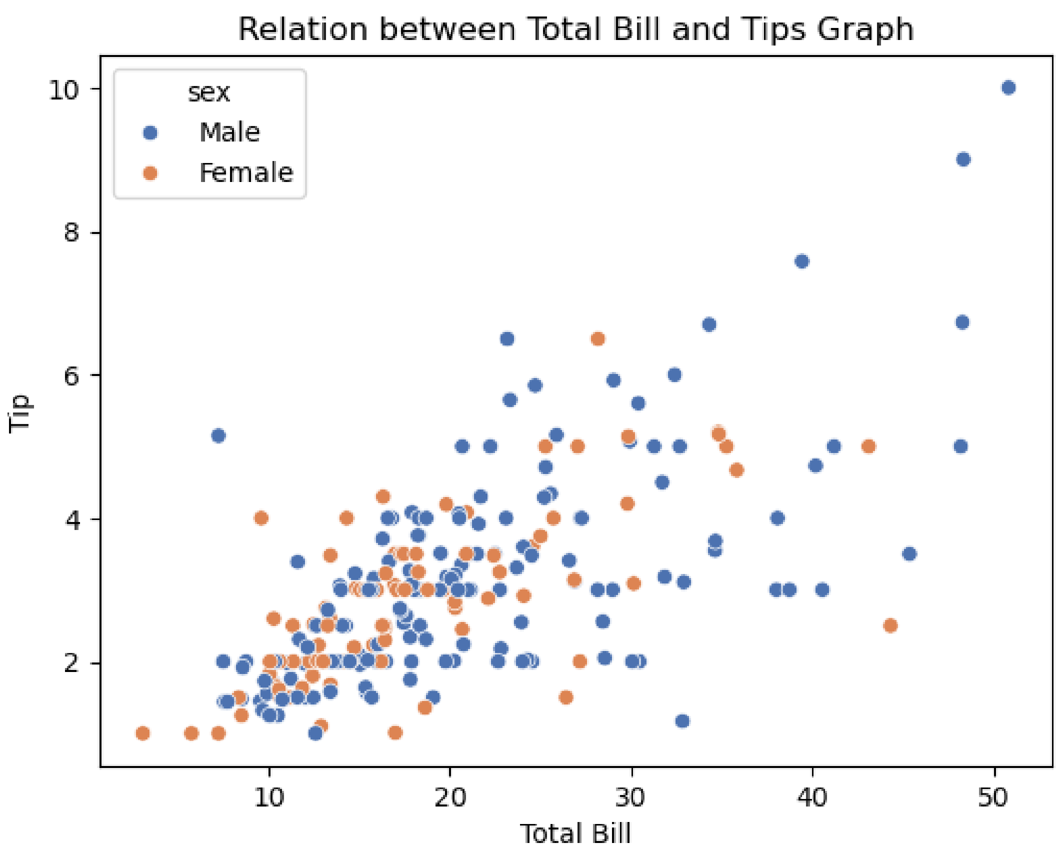
```
sns.scatterplot(data=tips, x='total_bill', y='tip', hue='day', palette='Set2')
```

  - **Legend**: Control display with `legend=False`.

- **Considerations**:
  - Limit unique categories to avoid clutter.
  - Use color-blind friendly palettes.

- **Conclusion**: The `hue` parameter is essential for enhancing visualizations by grouping data with color coding.

[15]:
```
sns.scatterplot(data= tips, x= 'total_bill',y= 'tip',hue='sex',palette='deep')␣
↪# style='time',size='size'
plt.title('Relation between Total Bill and Tips Graph')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```



Seaborn offers a variety of palettes for different types of data. Here are the main categories and examples of available palettes:

0.0.37 1. Qualitative Palettes

- Used for categorical data where the order does not matter.
- **Examples**:
  - `deep`
  - `muted`
  - `pastel`
  - `dark`
  - `colorblind`
  - `Set1`
  - `Set2`
  - `Set3`
  - `Paired`
  - `Accent`

0.0.38 2. Sequential Palettes

- Used for ordered data where values range from low to high.
- **Examples**:
  - `Blues`
  - `Greens`
  - `Purples`
  - `Oranges`
  - `Reds`
  - `Greys`
  - `BuGn`
  - `YlGn`
  - `OrRd`

0.0.39 3. Diverging Palettes

- Used for data with a critical midpoint (e.g., differences).
- **Examples**:
  - `coolwarm`
  - `RdBu`
  - `Spectral`
  - `BrBG`
  - `PiYG`
  - `PuOr`
  - `RdYlBu`
  - `RdYlGn`

0.0.40 4. Custom Palettes

- You can create custom palettes using `sns.color_palette()` or by defining a list of colors.

```
custom_palette = sns.color_palette(["#ff0000", "#00ff00", "#0000ff"])
```

0.0.41 5. Color Brewer Palettes

- Seaborn integrates Color Brewer palettes for better color schemes:
  - **Qualitative**: `Set1`, `Set2`, `Set3`
  - **Sequential**: `Blues`, `Greens`, `Purples`
  - **Diverging**: `RdBu`, `Spectral`, `BrBG`

0.0.42 Usage Example:

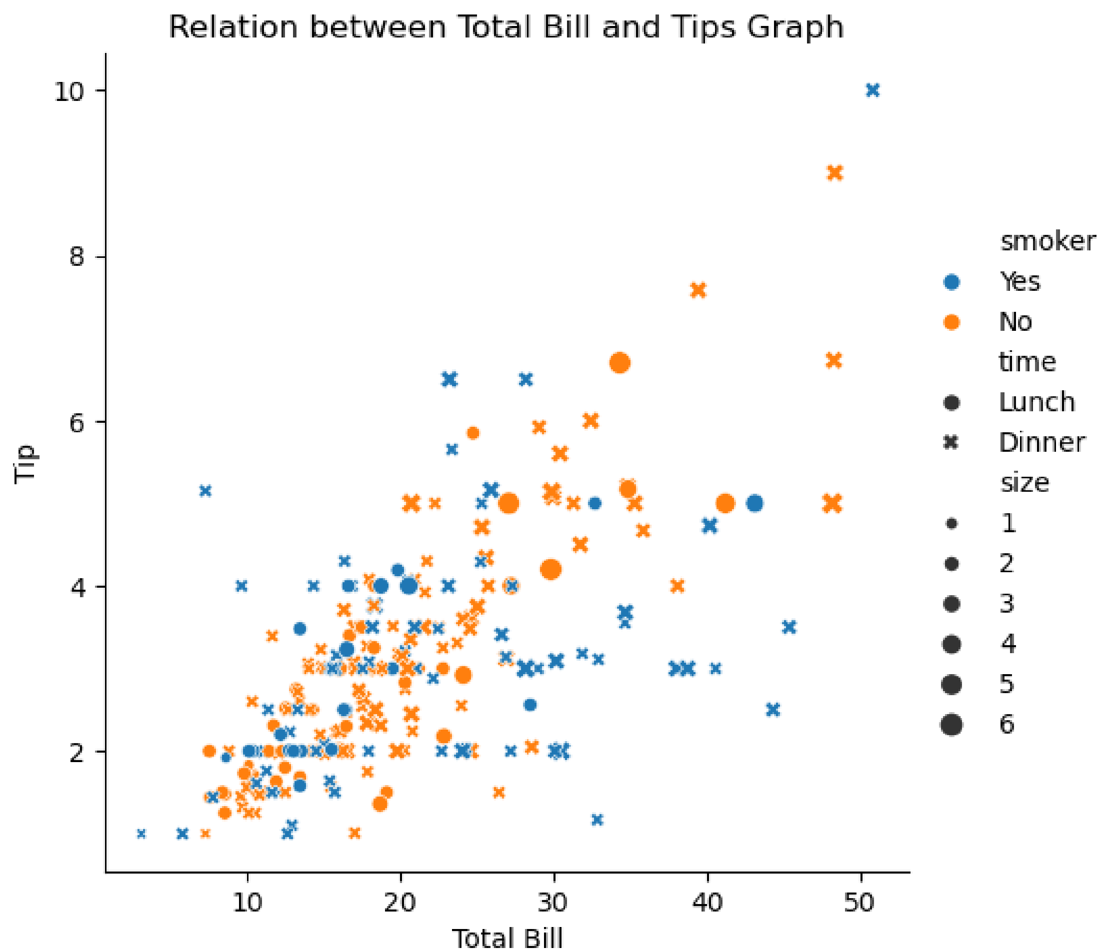
To use a specific palette in your plots, you can specify the `palette` parameter:

```
sns.scatterplot(data=tips, x='total_bill', y='tip', hue='day', palette='Set2')
```

0.0.43 Conclusion

Seaborn provides a wide range of palettes to cater to different visualization needs, enhancing the ability to interpret data through effective color coding.

[16]:
```
# figure level- Scatterplot

sns.relplot(data= tips, x= 'total_bill',y=␣
 ↪'tip',hue='smoker',style='time',size='size')
plt.title('Relation between Total Bill and Tips Graph')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```



1. **Style Parameter**
   - **Purpose**: The `style` parameter is used to differentiate the markers in the scatter plot based on a categorical variable. It changes the marker type (shape) based on the values of the specified column.
   - **Example**: In your case, `style='time'` indicates that the markers will have different shapes based on the values in the `time` column from the `tips` dataset.
     - **Common Styles**:
       * `'o'`: Circle
       * `'s'`: Square
       * `'^'`: Triangle
       * Other custom marker shapes can also be used.

2. **Size Parameter**
   - **Purpose**: The `size` parameter adjusts the size of the markers in the scatter plot based on a numerical variable. This provides a visual representation of another dimension of data.

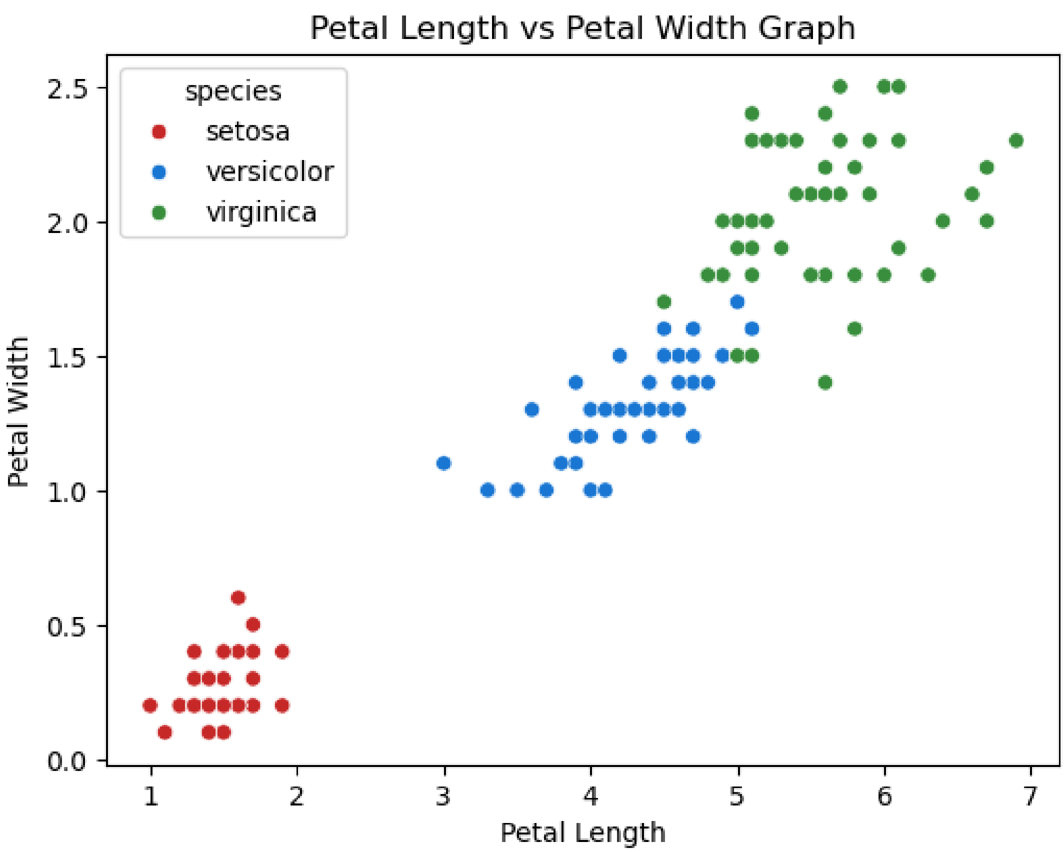
- **Example**: In your case, `size='size'` means the size of each marker will vary according to the values in the `size` column of the `tips` dataset.
  - **Size Mapping**: The size of markers will scale based on the numerical values, allowing for a better representation of the data distribution.

[17]:
```
iris.sample(5)
```

[17]:

	sepal_length	sepal_width	petal_length	petal_width	species
4	5.0	3.6	1.4	0.2	setosa
18	5.7	3.8	1.7	0.3	setosa
79	5.7	2.6	3.5	1.0	versicolor
142	5.8	2.7	5.1	1.9	virginica
70	5.9	3.2	4.8	1.8	versicolor

[18]:
```
sns.scatterplot(data=iris,x='petal_length',y='petal_width',hue='species',␣
↪palette=["#C62828", "#1976D2", "#388E3C"])
plt.title('Petal Length vs Petal Width Graph')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
```



[19]:
```
gap.head()
```

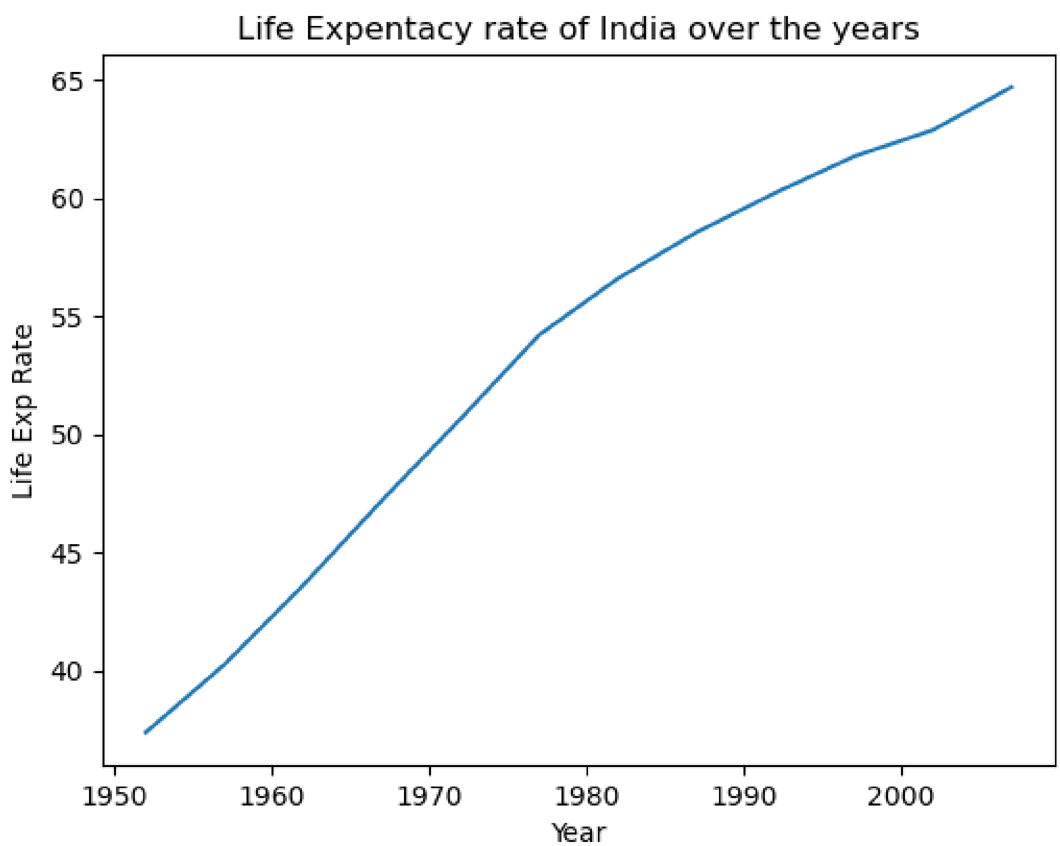
[19]:
```
       country continent  year  lifeExp       pop   gdpPercap iso_alpha  \
0  Afghanistan      Asia  1952   28.801   8425333  779.445314       AFG
1  Afghanistan      Asia  1957   30.332   9240934  820.853030       AFG
2  Afghanistan      Asia  1962   31.997  10267083  853.100710       AFG
3  Afghanistan      Asia  1967   34.020  11537966  836.197138       AFG
4  Afghanistan      Asia  1972   36.088  13079460  739.981106       AFG

   iso_num
0        4
1        4
2        4
3        4
4        4
```

[20]:
```
# time series analysis of India
# year vs Population

india = gap[gap.country=='India']
india
```

[20]:
```
    country continent  year  lifeExp         pop    gdpPercap iso_alpha  \
696   India      Asia  1952   37.373   372000000   546.565749       IND
697   India      Asia  1957   40.249   409000000   590.061996       IND
698   India      Asia  1962   43.605   454000000   658.347151       IND
699   India      Asia  1967   47.193   506000000   700.770611       IND
700   India      Asia  1972   50.651   567000000   724.032527       IND
701   India      Asia  1977   54.208   634000000   813.337323       IND
702   India      Asia  1982   56.596   708000000   855.723538       IND
703   India      Asia  1987   58.553   788000000   976.512676       IND
704   India      Asia  1992   60.223   872000000  1164.406809       IND
705   India      Asia  1997   61.765   959000000  1458.817442       IND
706   India      Asia  2002   62.879  1034172547  1746.769454       IND
707   India      Asia  2007   64.698  1110396331  2452.210407       IND

     iso_num
696      356
697      356
698      356
699      356
700      356
701      356
702      356
703      356
704      356
705      356
```

```
706       356
707       356
```

[21]:
```
sns.lineplot(data=india,x='year',y='lifeExp')
plt.title('Life Expentacy rate of India over the years')
plt.xlabel('Year')
plt.ylabel('Life Exp Rate')
plt.show()
```



[22]:
```
# india's neighbours
# India, Pakistan, Afghanistan, Bangladesh, China

neighbours = gap[gap['country'].isin(['India', 'Pakistan', 'Afghanistan',␣
 ↪'Bangladesh', 'China' ])]
```

The `isin()` method in pandas is used to filter DataFrame rows based on whether the values in a specified column are present in a given list or array. It returns a boolean Series indicating whether each element in the Series is contained in the specified list.

0.0.44 How isin() Works

- **Syntax**: `DataFrame['column_name'].isin(list_of_values)`
- **Parameters**:
  - `list_of_values`: A list, set, or array-like object containing the values to check against.

0.0.45 Example Explanation

In your example:

```
neighbours = gap[gap['country'].isin(['India', 'Pakistan', 'Afghanistan', 'Bangladesh', 'China
```

1. **DataFrame**: `gap` is a pandas DataFrame containing country data.
2. **Column**: `gap['country']` selects the `country` column from the DataFrame.
3. **Filtering**:
   - `isin(['India', 'Pakistan', 'Afghanistan', 'Bangladesh', 'China'])` checks each value in the `country` column to see if it is one of the specified countries.
   - This generates a boolean Series where each entry is `True` if the corresponding country is in the list and `False` otherwise.
4. **Result**:
   - `gap[...]` uses this boolean Series to filter the original DataFrame. Only the rows where `country` matches one of the countries in the list will be included in the new DataFrame `neighbours`.

0.0.46 Practical Use

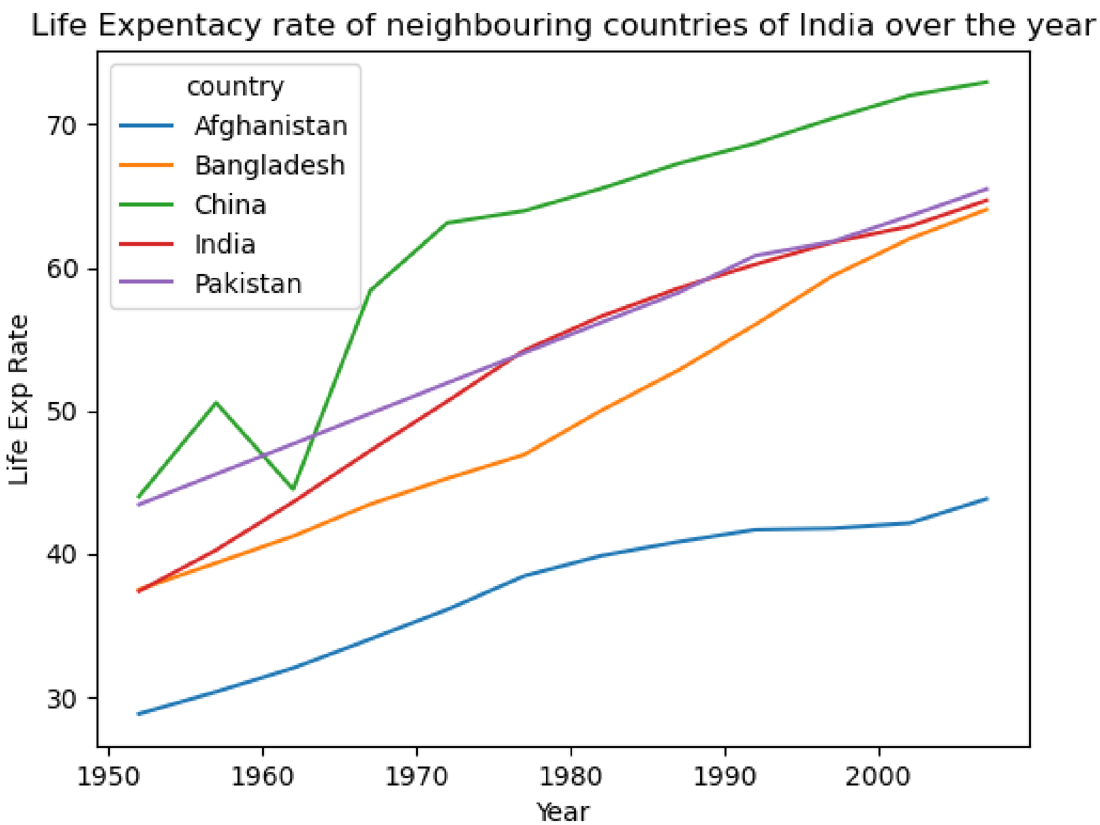
- **Filtering Rows**: `isin()` is useful for selecting multiple categories in a column without having to use multiple conditions with `|` (or).
- **Performance**: It is typically faster and more concise than using multiple conditions.

[23]:
```
neighbours.sample(5)
```

[23]:
```
          country continent  year  lifeExp        pop    gdpPercap iso_alpha  \
9     Afghanistan      Asia  1997   41.763   22227415   635.341351       AFG
1173     Pakistan      Asia  1997   61.818  135564834  2049.350521       PAK
107    Bangladesh      Asia  2007   64.062  150448339  1391.253792       BGD
1171     Pakistan      Asia  1987   58.245  105186881  1704.686583       PAK
698         India      Asia  1962   43.605  454000000   658.347151       IND

      iso_num
9           4
1173      586
107        50
1171      586
698       356
```

[24]:
```
# Axes Level function
sns.lineplot(data= neighbours,x='year',y='lifeExp',hue='country')
plt.title('Life Expentacy rate of neighbouring countries of India over the␣
  ↪year')
```

```
plt.xlabel('Year')
plt.ylabel('Life Exp Rate')
plt.show()
```



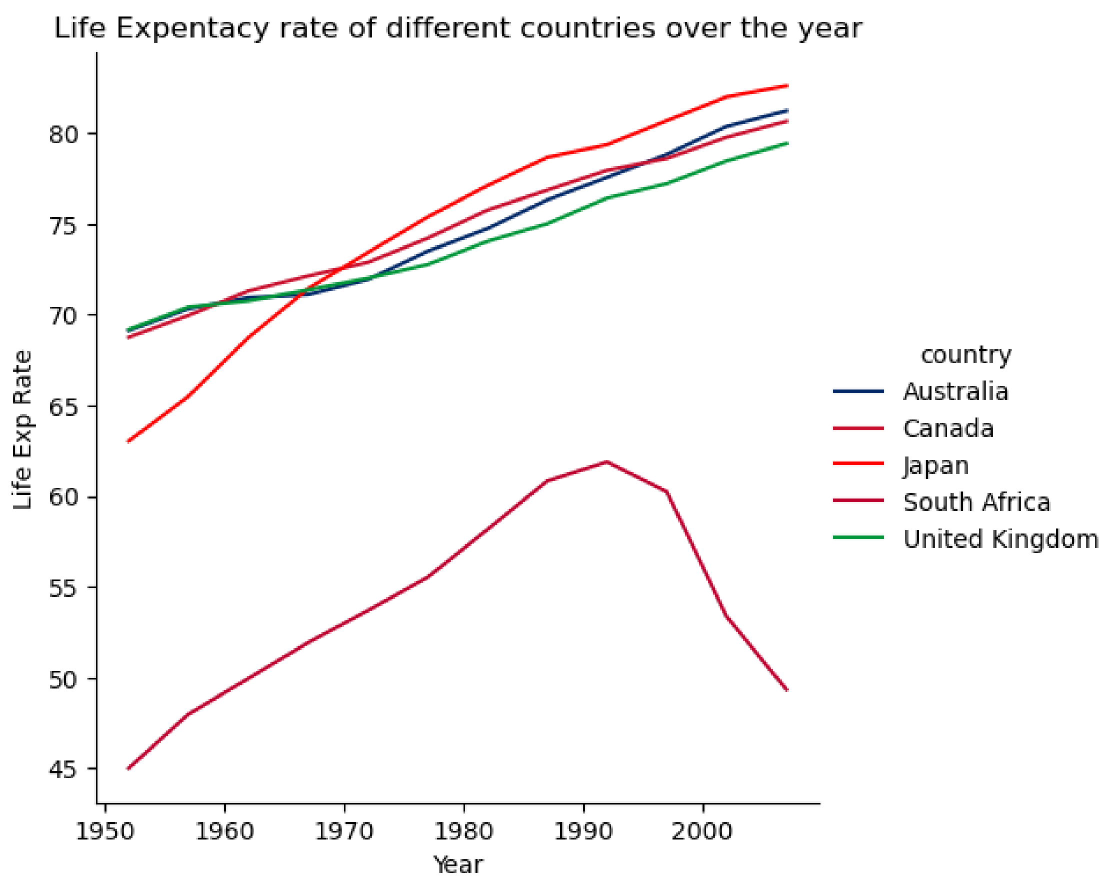
[25]:
```
continent = gap[gap['country'].isin(['Australia', 'South Africa', 'Canada',␣
↪'Japan','United Kingdom'])]
```

[26]:
```
continent.sample(5)
```

[26]:
```
           country continent  year  lifeExp        pop     gdpPercap  \
1411  South Africa    Africa  1987   60.834   35933379   7825.823398
70       Australia   Oceania  2002   80.370   19546792  30687.754730
798          Japan      Asia  1982   77.110  118454974  19384.105710
63       Australia   Oceania  1967   71.100   11872264  14526.124650
797          Japan      Asia  1977   75.380  113872473  16610.377010

     iso_alpha  iso_num
1411       ZAF      710
70         AUS       36
798        JPN      392
```

```
63          AUS       36
797         JPN      392
```

[27]:
```
# Figure level Function, Line plot
sns.
↪relplot(data=continent,x='year',y='lifeExp',hue='country',kind='line',palette=[
    "#002766",  # Australia - Dark Blue
    "#C8102E",  # Australia - Red
    "#FF0000",  # Canada - Red
    "#BC002D",  # Japan - Red
    "#009639",  # South Africa - Green
    "#FDD100",  # South Africa - Yellow
    "#00247D",  # United Kingdom - Blue
    "#CF142B",  # United Kingdom - Red
]) # style='continent'
plt.title('Life Expentacy rate of different countries over the year')
plt.xlabel('Year')
plt.ylabel('Life Exp Rate')
plt.show()
```



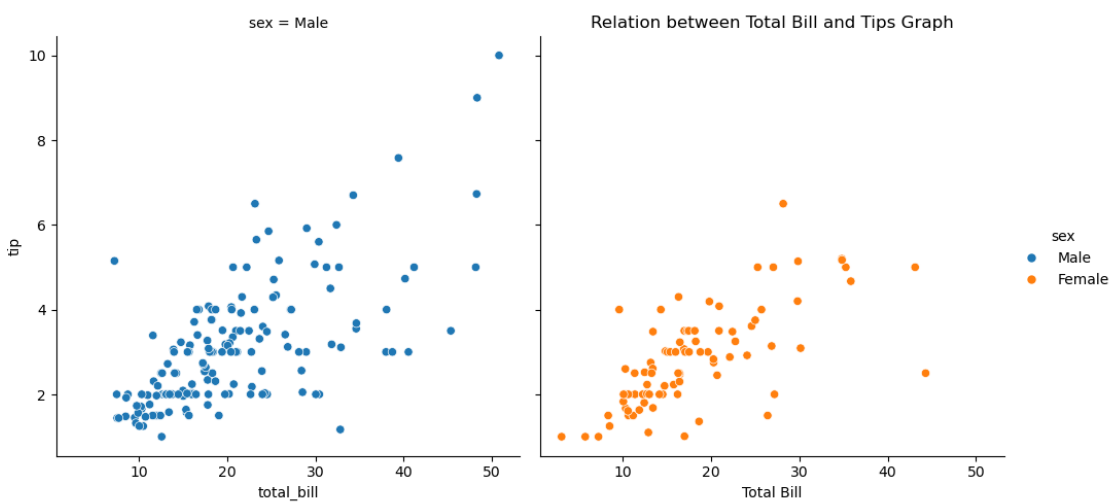
0.0.47 Facet Plots in Seaborn's `relplot`

Seaborn's **relplot** function can create facet plots by splitting the data into multiple subplots based on the values of categorical variables. You can specify which variables to use for the rows and columns of the grid layout, allowing for a comprehensive view of relationships in your dataset.

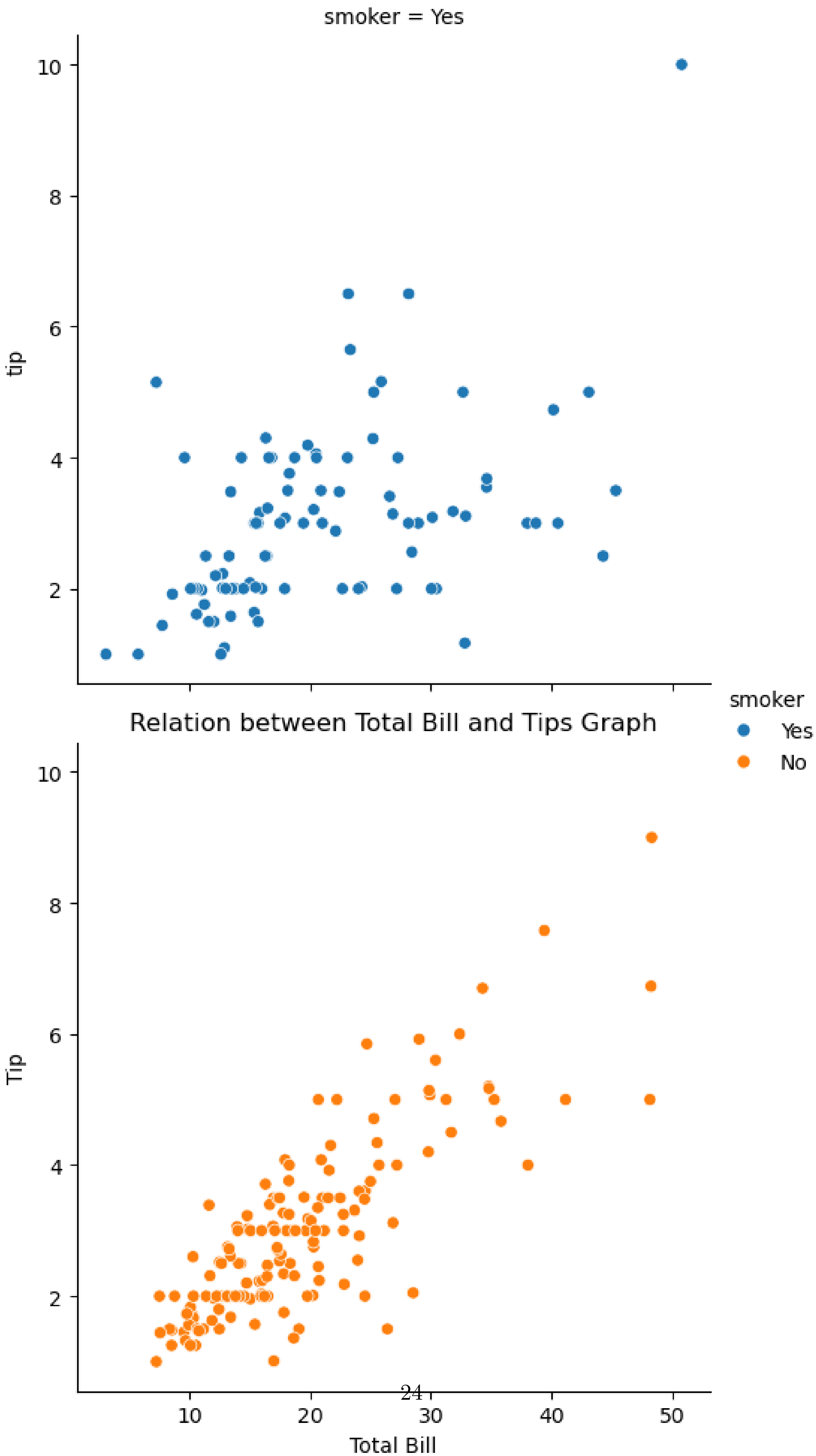
Key Parameters for Faceting:

- **row**: This parameter allows you to create multiple rows of plots, where each row represents a unique value of the specified categorical variable.
- **col**: This parameter allows you to create multiple columns of plots, where each column represents a unique value of another categorical variable.

[28]:
```
# col
sns.relplot(data=tips,x='total_bill',y='tip',col='sex',hue='sex')
plt.title('Relation between Total Bill and Tips Graph')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```

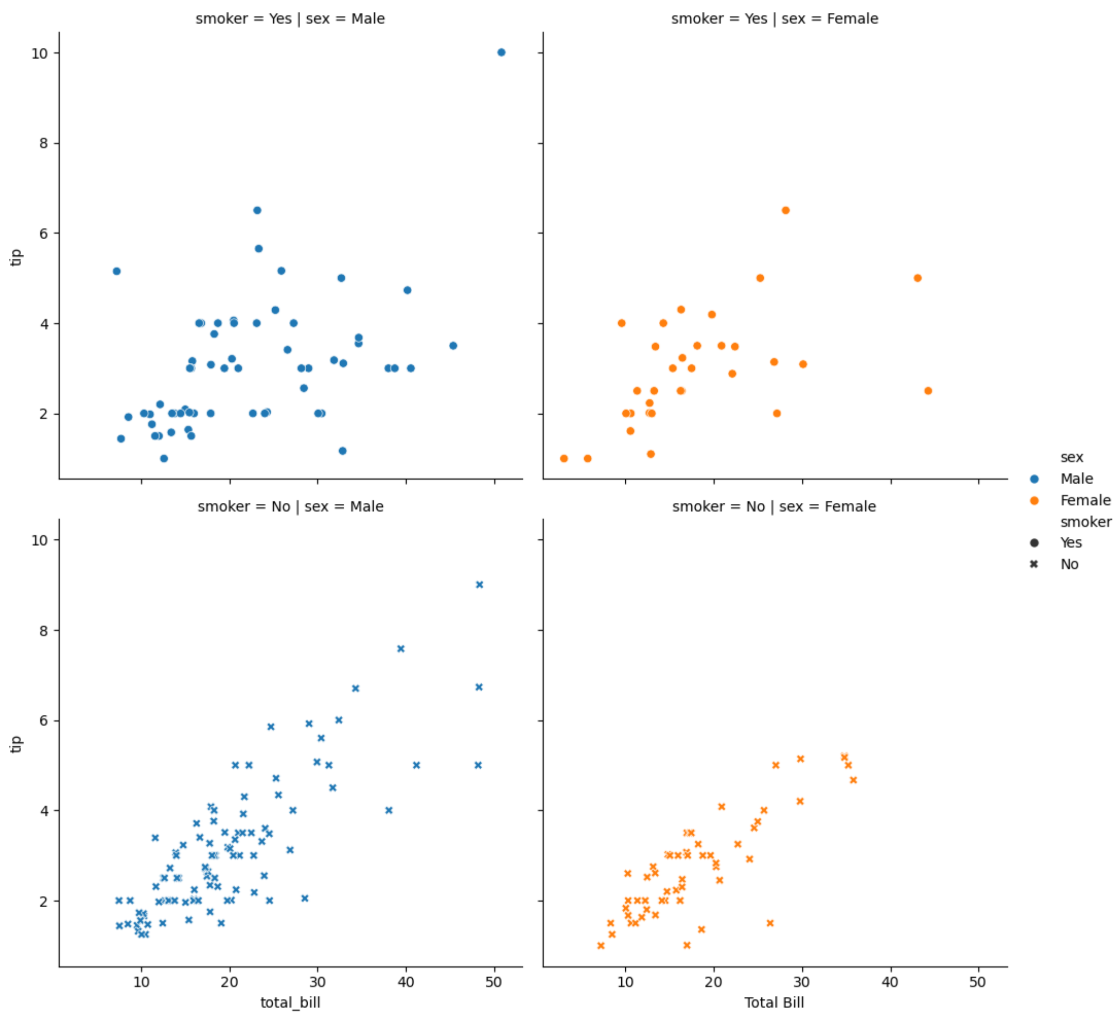


[29]:
```
# row
sns.relplot(data=tips,x='total_bill',y='tip',row='smoker',hue='smoker')
plt.title('Relation between Total Bill and Tips Graph')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```



smoker = Yes
tip
10
8
6
4
2
smoker
Yes
No
Relation between Total Bill and Tips Graph
Tip
10
8
6
4
2
10
20
30
40
50
Total Bill

[30]:
```
# row x col
sns.
 ↪relplot(data=tips,x='total_bill',y='tip',row='smoker',col='sex',hue='sex',style='smoker')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```



0.0.48 Distribution Plots in Seaborn:

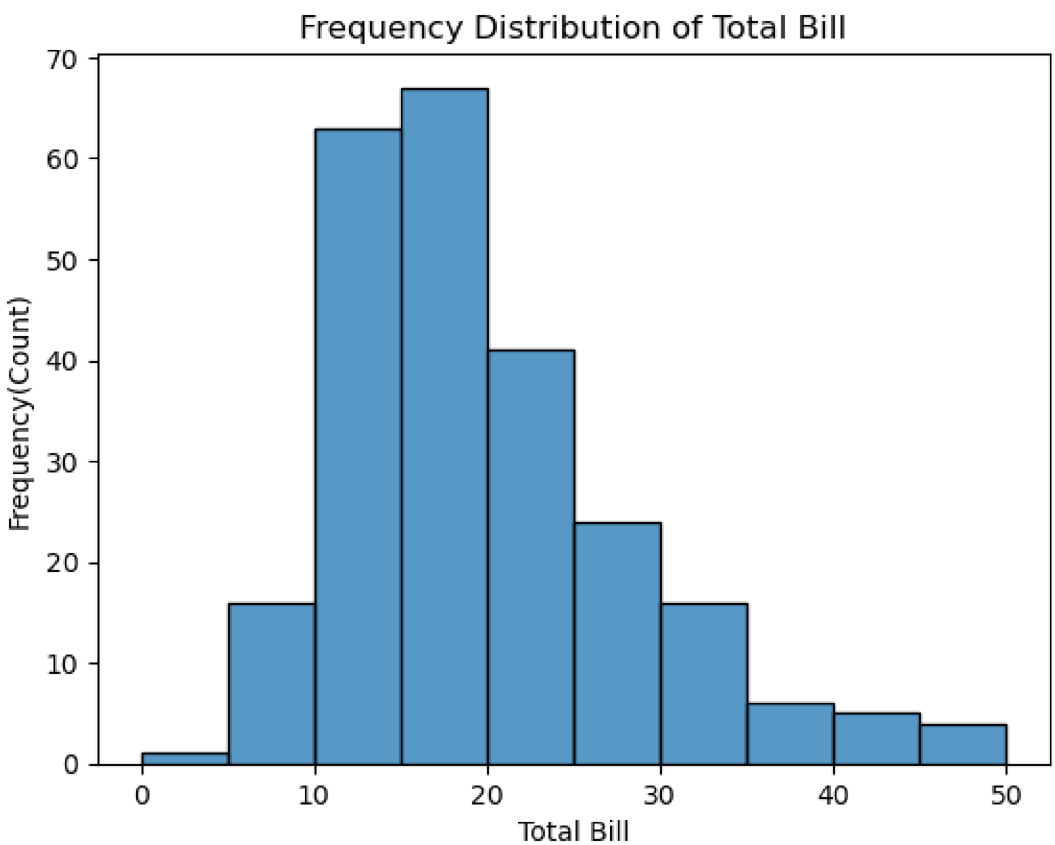
1. **Histogram**:
   - **Purpose**: Displays the distribution of data by grouping values into bins (bars) and counting how many data points fall into each bin. It provides a visual representation of how data is spread across different ranges.
2. **KDE Plot**:

- **Purpose**: Creates a smooth curve that represents the distribution of data. This smoothed version of a histogram gives a clearer visual idea of the data's density and helps identify patterns in the distribution.

3. **Rug Plot**:
   - **Purpose**: Adds small vertical ticks on the x-axis to indicate the exact locations of individual data points. It helps visualize the concentration of data points and can be useful when combined with other plots for added context.

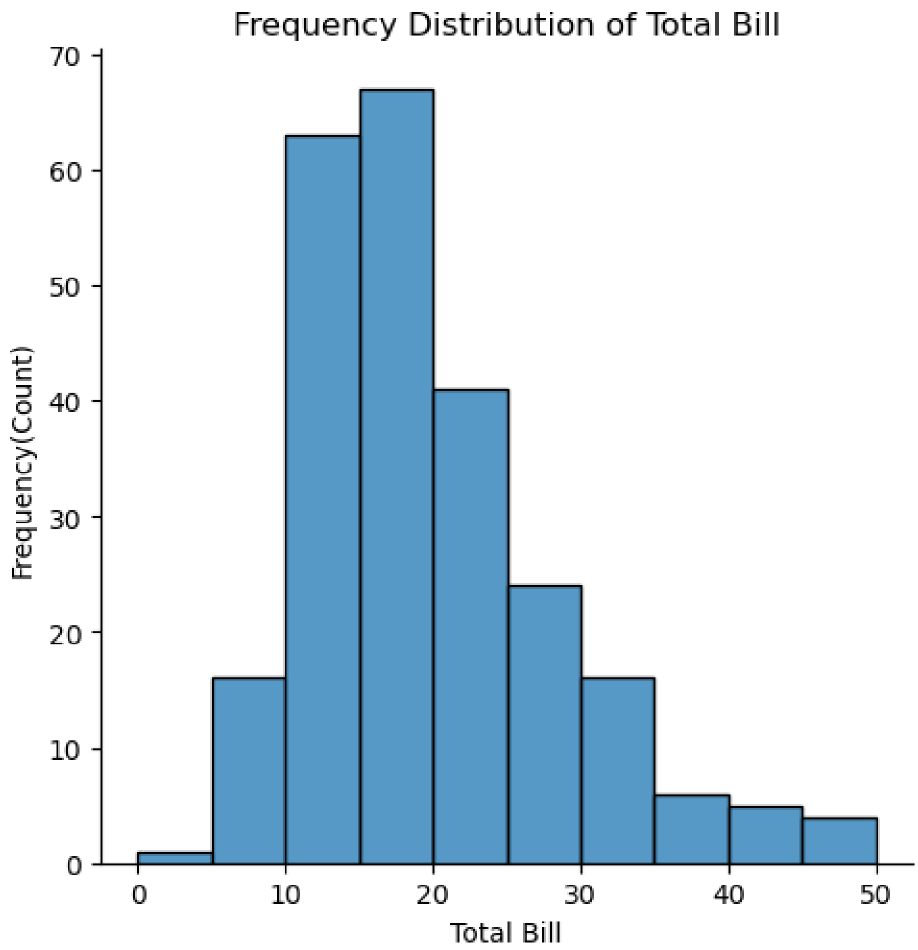
[31]:
```
# sns.histplot // axes level
# sns.kdeplot // axes level
# sns.rugplot // axes level
# sns.displot // figure level
# kind="kde", "rug"
```

[32]:
```
# sns.histplot // axes level
sns.histplot(data=tips,x='total_bill',bins=range(0,51,5))
plt.title('Frequency Distribution of Total Bill')
plt.xlabel('Total Bill')
plt.ylabel('Frequency(Count)')
plt.show()
```



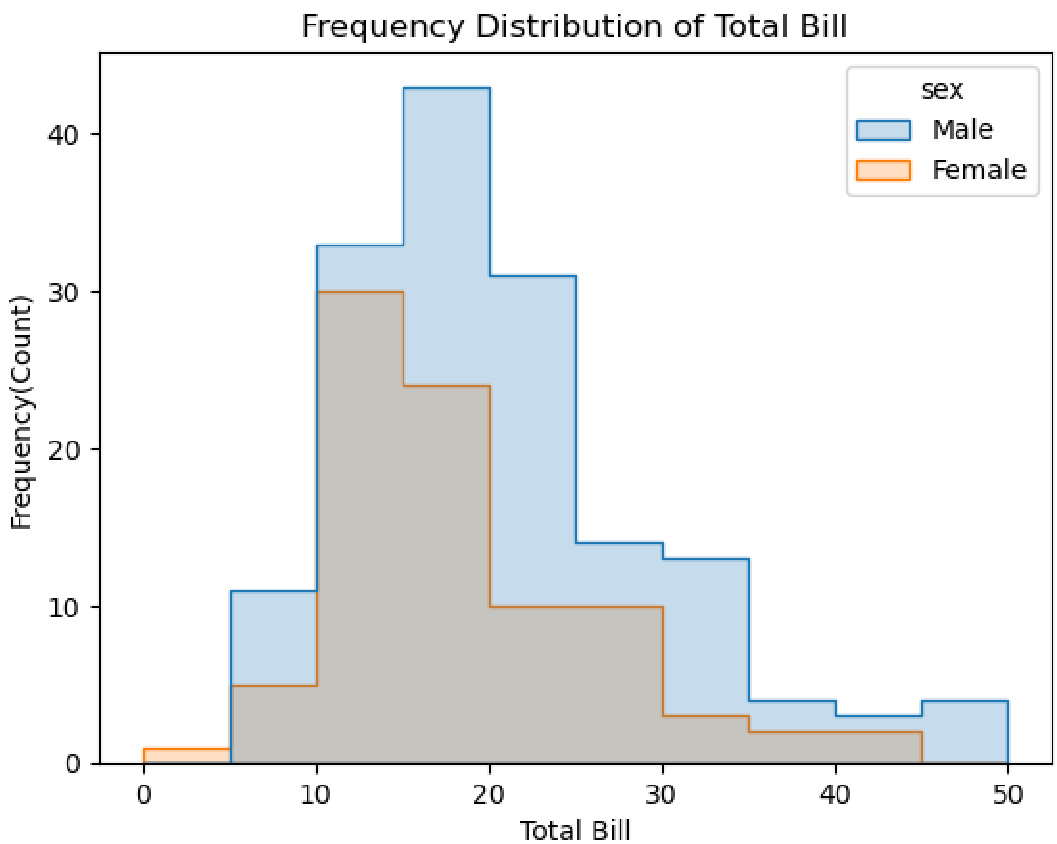
[33]:

```
# sns.displot // figure level

sns.displot(data=tips,x='total_bill',bins=range(0,51,5),kind='hist')
plt.title('Frequency Distribution of Total Bill')
plt.xlabel('Total Bill')
plt.ylabel('Frequency(Count)')
plt.show()
```



[34]:

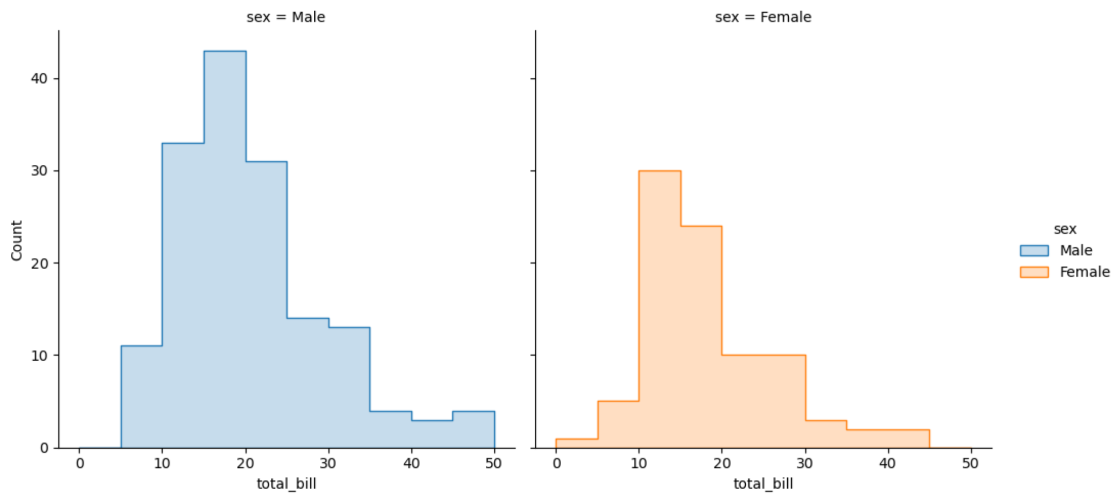
```
sns.
 ↪histplot(data=tips,x='total_bill',bins=range(0,51,5),hue='sex',element='step')
plt.title('Frequency Distribution of Total Bill')
plt.xlabel('Total Bill')
plt.ylabel('Frequency(Count)')
plt.show()

# bar(by default), step, poly
```



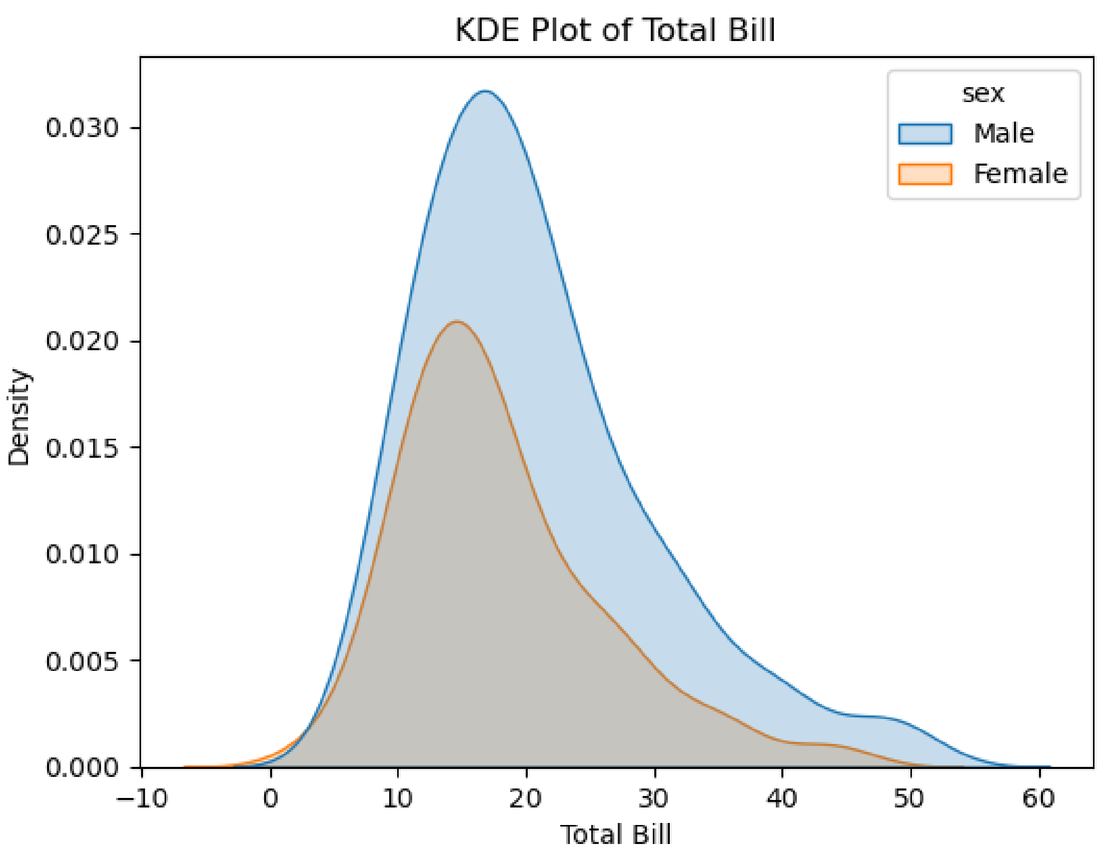
0.0.49 Element Parameter Options

1. `bars`:
   - **Description**: Standard rectangular bars representing frequency counts.
   - **Use Case**: Best for visualizing the distribution of data.
2. `step`:
   - **Description**: Displays a stepped line representing the frequency counts without filling the area under the curve.
   - **Use Case**: Useful for emphasizing the boundaries between bins.
3. `poly`:
   - **Description**: Similar to `step`, but the area under the curve is filled, forming a polygon shape.
   - **Use Case**: Good for visualizing distributions while maintaining a smooth look.

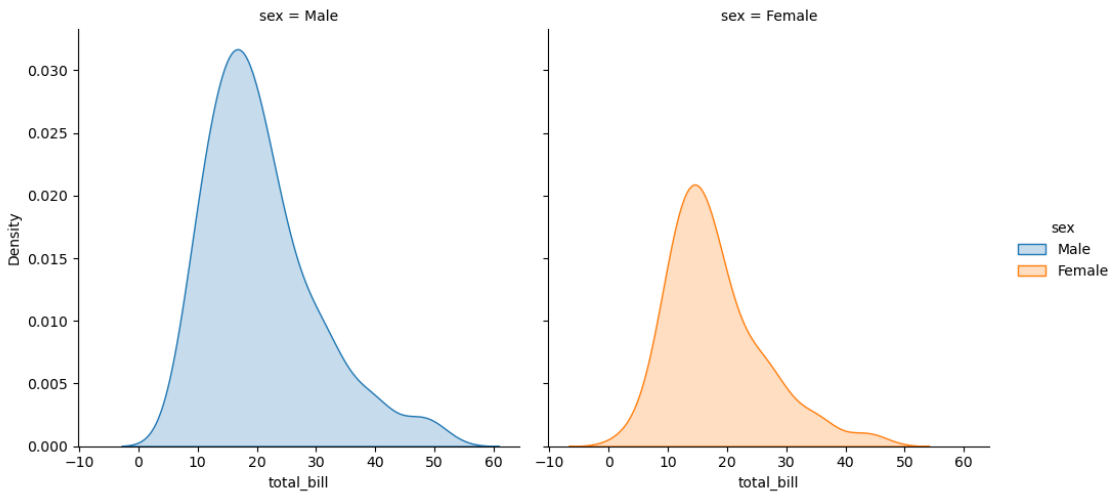
[35]:
```
# facet plot

sns.
 ↪displot(data=tips,x='total_bill',bins=range(0,51,5),kind='hist',col='sex',element='step',hu
plt.ylabel('Frequency(Count)')
plt.show()
```



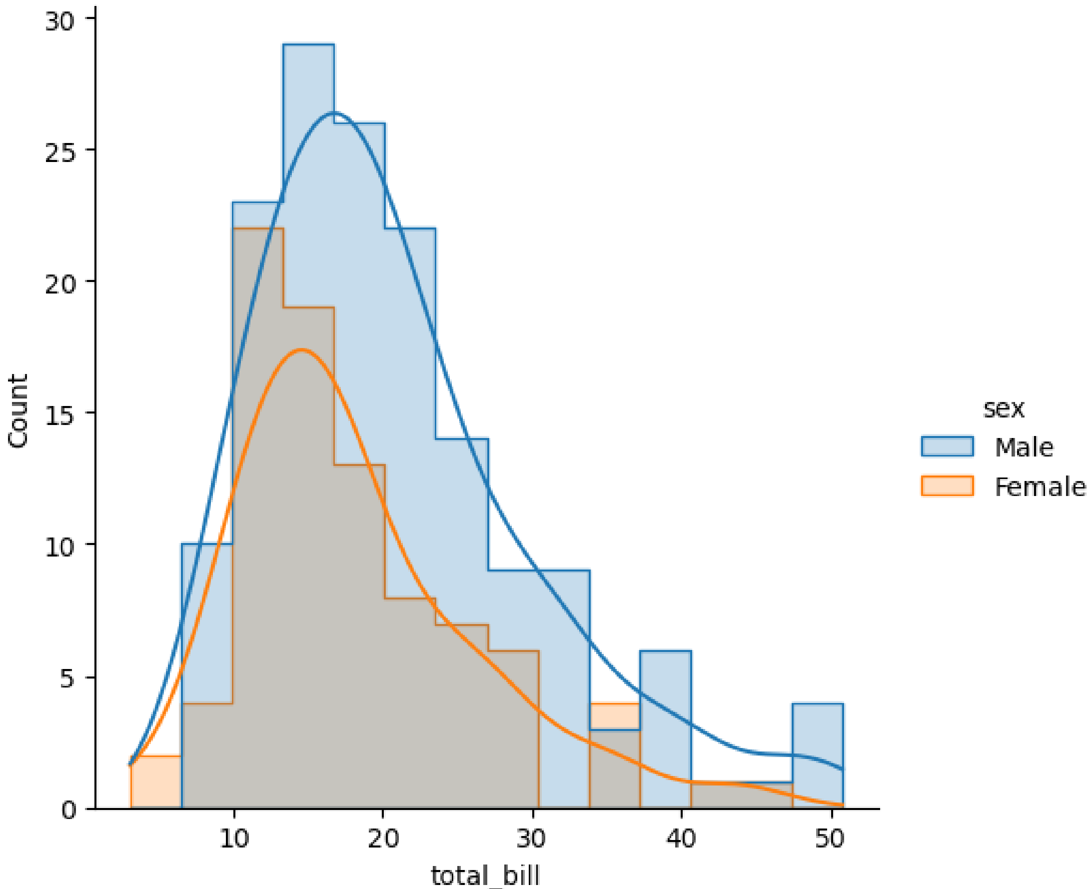
[36]:
```
sns.kdeplot(data=tips,x='total_bill',fill=True,hue='sex')
plt.title('KDE Plot of Total Bill')
plt.xlabel('Total Bill')
plt.show()
```



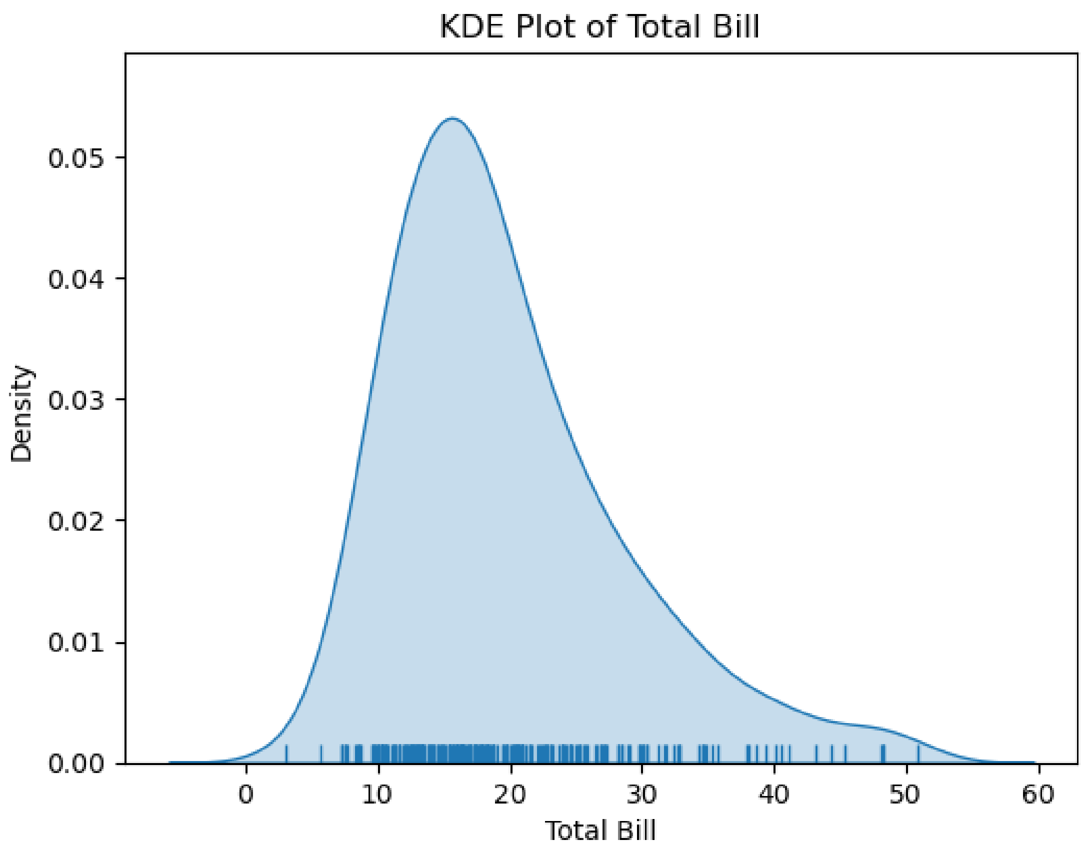
[37]:
```
# facetplot
sns.displot(data=tips,x='total_bill',fill=True,hue='sex',col='sex',kind='kde')
plt.show()
```



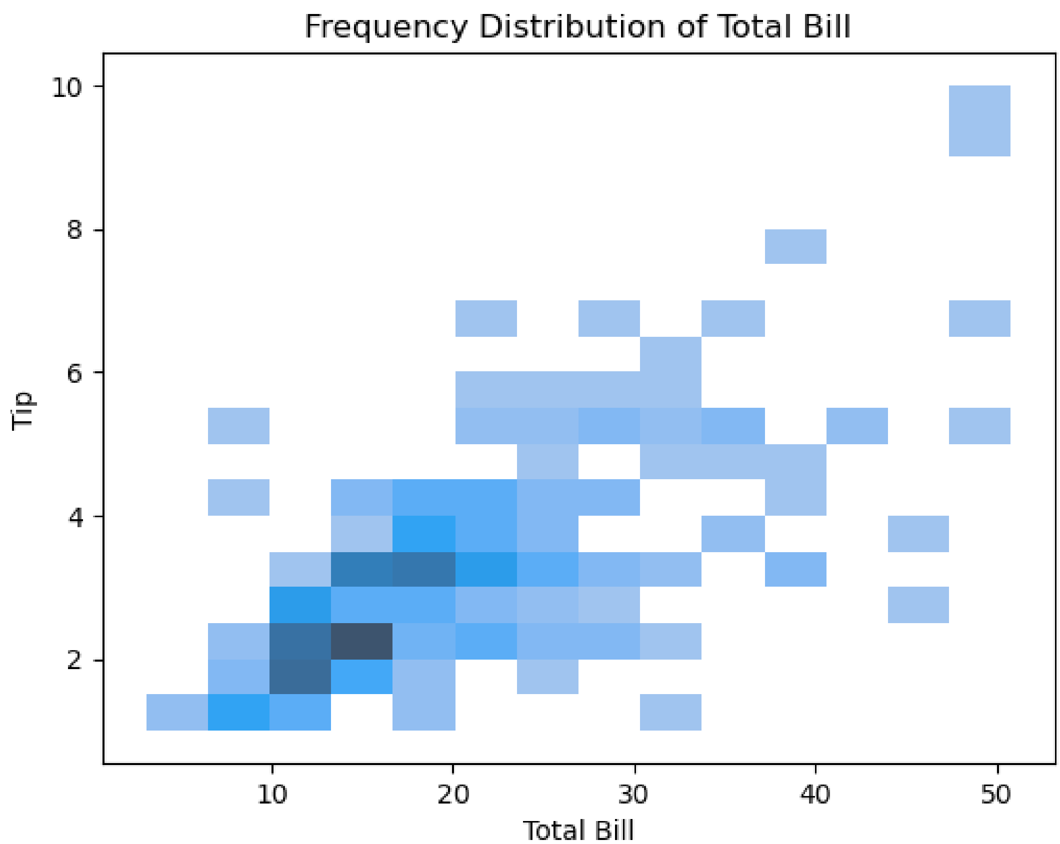
[38]:
```
sns.
↪displot(data=tips,x='total_bill',fill=True,hue='sex',kind='hist',kde=True,element='step')
plt.show()
```



[39]:
```
sns.kdeplot(data=tips,x='total_bill',fill=True)
sns.rugplot(data=tips,x='total_bill')
plt.title('KDE Plot of Total Bill')
plt.xlabel('Total Bill')
plt.show()
```



[40]:
```
# advance plots
# Histogram
sns.histplot(data=tips,x='total_bill',y='tip')
plt.title('Frequency Distribution of Total Bill')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```



0.0.50 What the Graph Shows

- **Type**: This is a **2D histogram**, which shows how often different combinations of **total bills** and **tips** occur.

0.0.51 Axes

- **X-axis (Total Bill)**: This axis shows the total amount customers paid for their meals, ranging from about 10 to 50.
- **Y-axis (Tip)**: This axis shows how much customers tipped, ranging from 0 to 10.

0.0.52 Color Intensity

- **Color**: The colors indicate how many customers fall into each category of total bill and tip:
  - **Darker Colors**: Areas where many customers had that specific combination of total bill and tip.
  - **Lighter Colors**: Areas where fewer customers had that combination.

0.0.53 Key Observations

1. **Trends**:

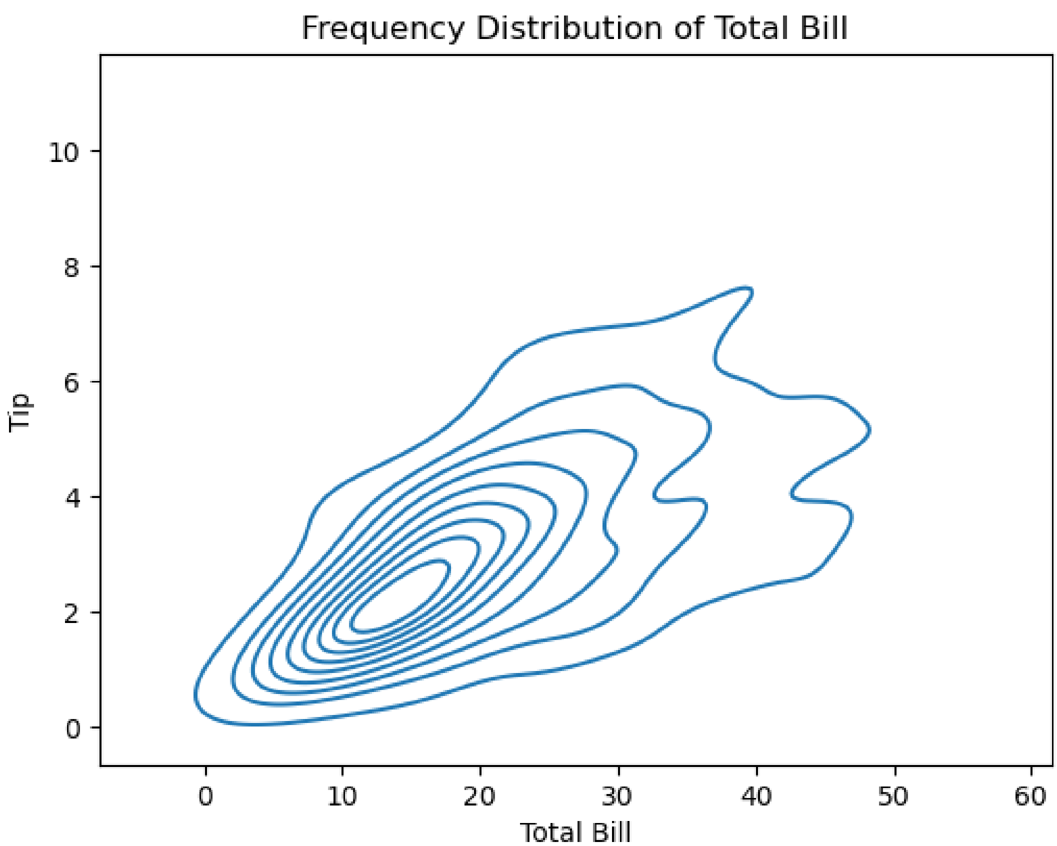
- As the total bill increases, tips also tend to increase. This suggests that people tip more when their bill is higher.

2. **Common Combinations**:
   - There are many customers who paid between 10-20 for the bill and left tips around 2-4.
   - For higher total bills (above 30), tips tend to be higher (around 4-8).

0.0.54 Summary

Overall, this histogram helps visualize the relationship between total bills and tips. It shows that higher total bills are usually associated with higher tips, helping restaurant owners understand customer behavior regarding tipping.

[41]:
```
# advance plots
# kdeplot
sns.kdeplot(data=tips,x='total_bill',y='tip')
plt.title('Frequency Distribution of Total Bill')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```



0.0.55 What the Graph Shows

- **Type**: This is a **KDE (Kernel Density Estimate) plot**. It helps visualize the distribution of two variables, in this case, **total bill** and **tip**.

0.0.56 Axes

- **X-axis (Total Bill)**: This shows the amount customers paid for their meals, ranging from 0 to 60.
- **Y-axis (Tip)**: This shows the amount customers tipped, ranging from 0 to 10.

0.0.57 Lines

- **Contour Lines**: The lines on the graph represent areas where a lot of customers have similar combinations of total bills and tips.
  - **Close Lines**: Areas where there are many customers with similar bills and tips.
  - **Wider Spaces**: Areas with fewer customers.

0.0.58 Key Observations

1. **Trends**:
   - There are noticeable clusters where most customers are found, indicating that higher total bills generally lead to higher tips.
   - The lines suggest that as the total bill increases, the tip also tends to increase, which is a common pattern.
2. **Dense Areas**:
   - The denser parts (where lines are closer together) show where most tips are concentrated for given bill amounts. For example, higher tips are more common for bills in the range of 20-30.

0.0.59 Summary

In simple terms, this KDE plot shows the relationship between the total bill and the tip. It indicates that people usually tip more when they have a higher bill, helping to understand how much customers tend to tip based on their meal costs. It visually represents where most customers fall in terms of their bill and tip amounts.

0.0.60 Pair Plot in Seaborn

A **pair plot** is a powerful visualization tool in Seaborn that allows you to explore relationships between multiple numerical variables in a dataset. It creates a matrix of scatter plots and histograms, helping to visualize how each pair of variables relates to each other.

Key Features:

- **Matrix of Plots**: Each cell in the pair plot matrix shows a scatter plot for a pair of variables, while the diagonal displays the distribution (usually a histogram or KDE) of each variable.
- **Multivariate Analysis**: Useful for visualizing the relationships among multiple variables simultaneously, making it easy to identify correlations, trends, and potential outliers.

- **Categorical Hue**: You can color the points based on a categorical variable, allowing you to see how different groups or classes are distributed across the variable pairs.
- **Customizability**: You can customize aspects of the plot, such as the kind of plots displayed on the diagonal (e.g., histograms, KDE), the color palette, and more.

Use Cases:

- **Exploratory Data Analysis (EDA)**: A pair plot is commonly used during the EDA phase to understand the relationships and distributions of variables in a dataset.
- **Identifying Relationships**: Helps in spotting linear or non-linear relationships, clusters, and patterns among different features.
- **Detecting Multicollinearity**: By examining the scatter plots, you can identify pairs of variables that are highly correlated, which may be relevant for regression analysis.

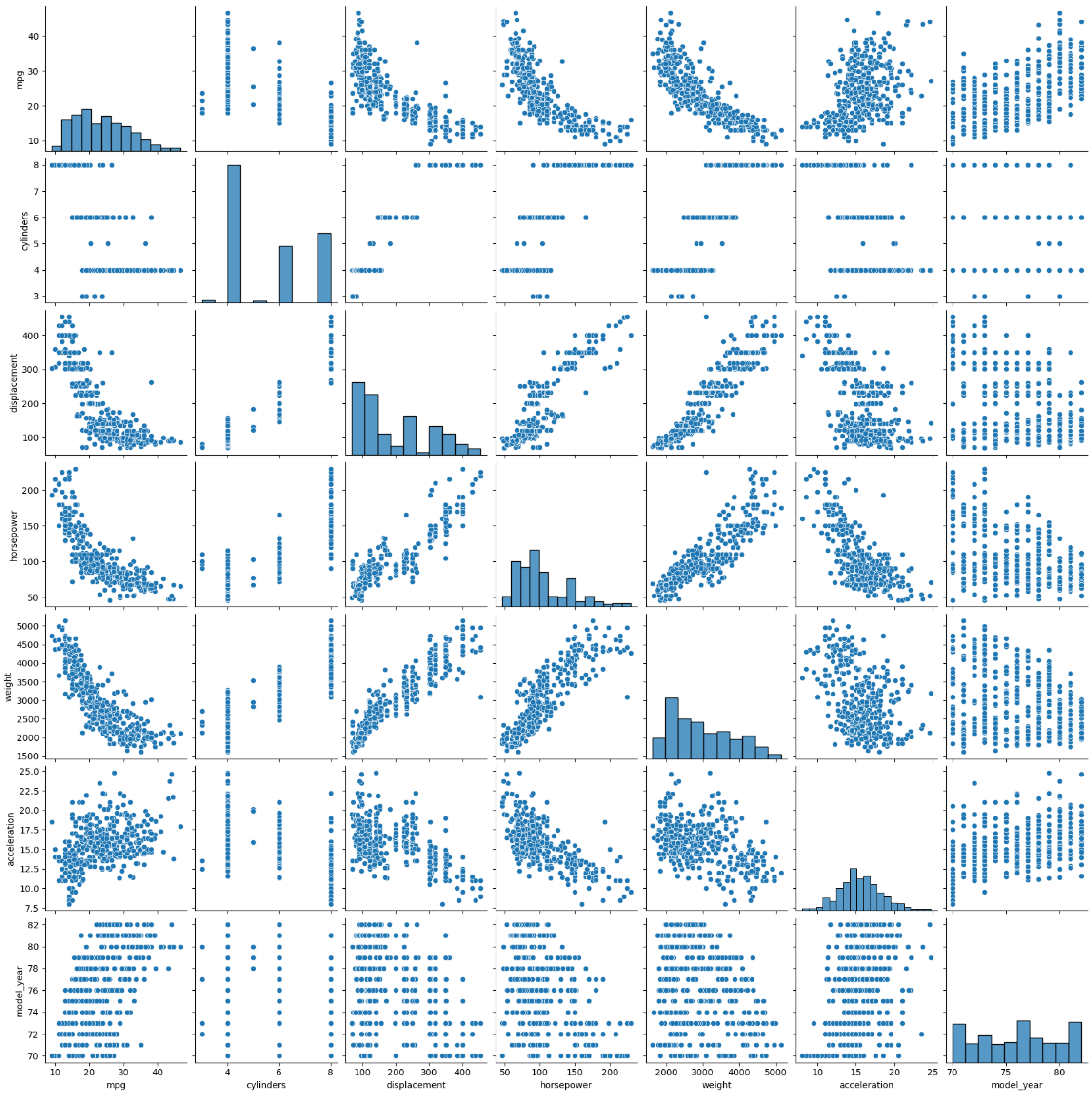
0.0.61 Summary:

A pair plot is an effective way to visualize and analyze the relationships between multiple numerical variables in a dataset. It provides a comprehensive overview, making it easier to identify patterns, correlations, and groupings within the data.

[42]:
```
# sns.paiplot // Figure level
```

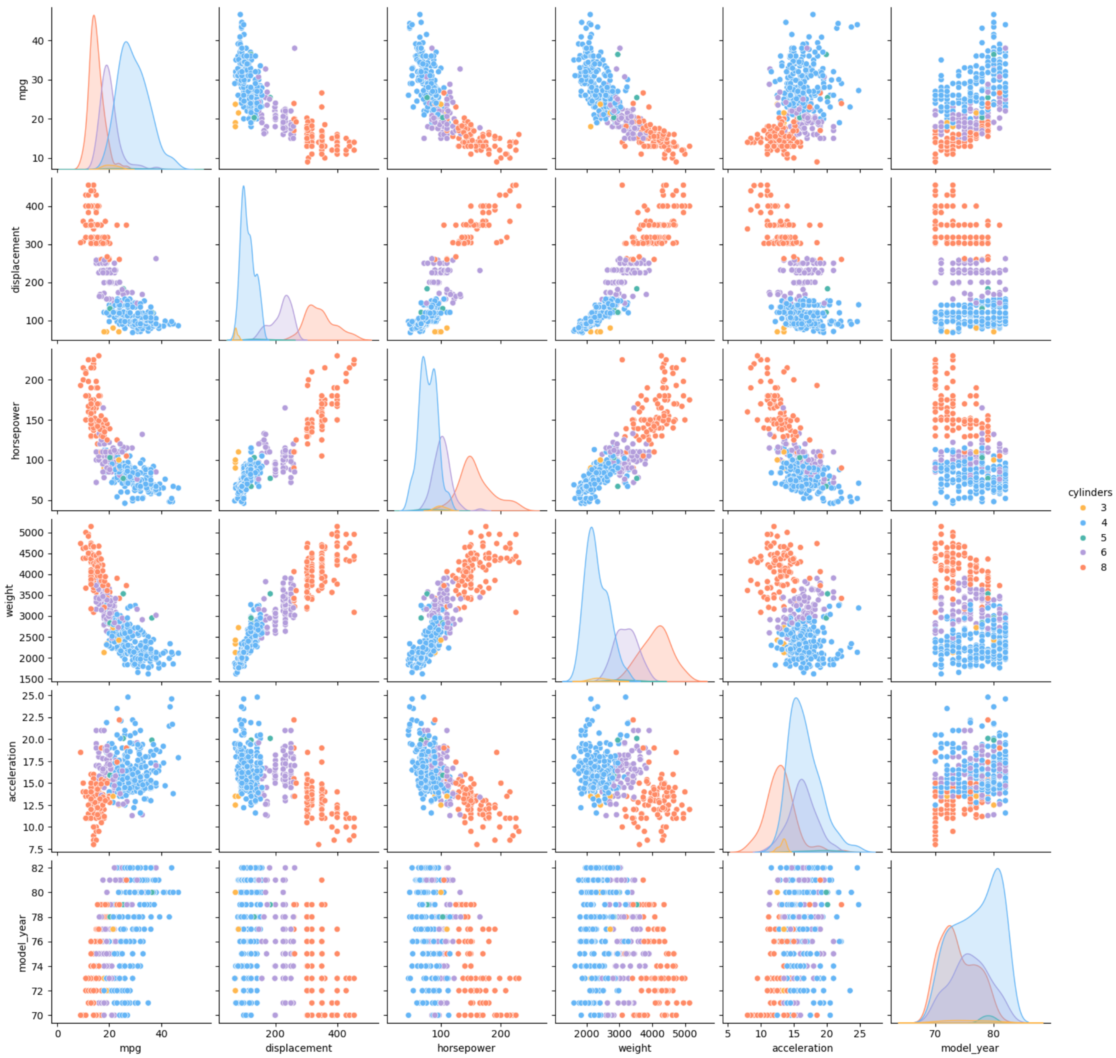
[43]:
```
plt.figure(figsize=(14,8))
sns.pairplot(mpg)
plt.show()
```

```
<Figure size 1400x800 with 0 Axes>
```



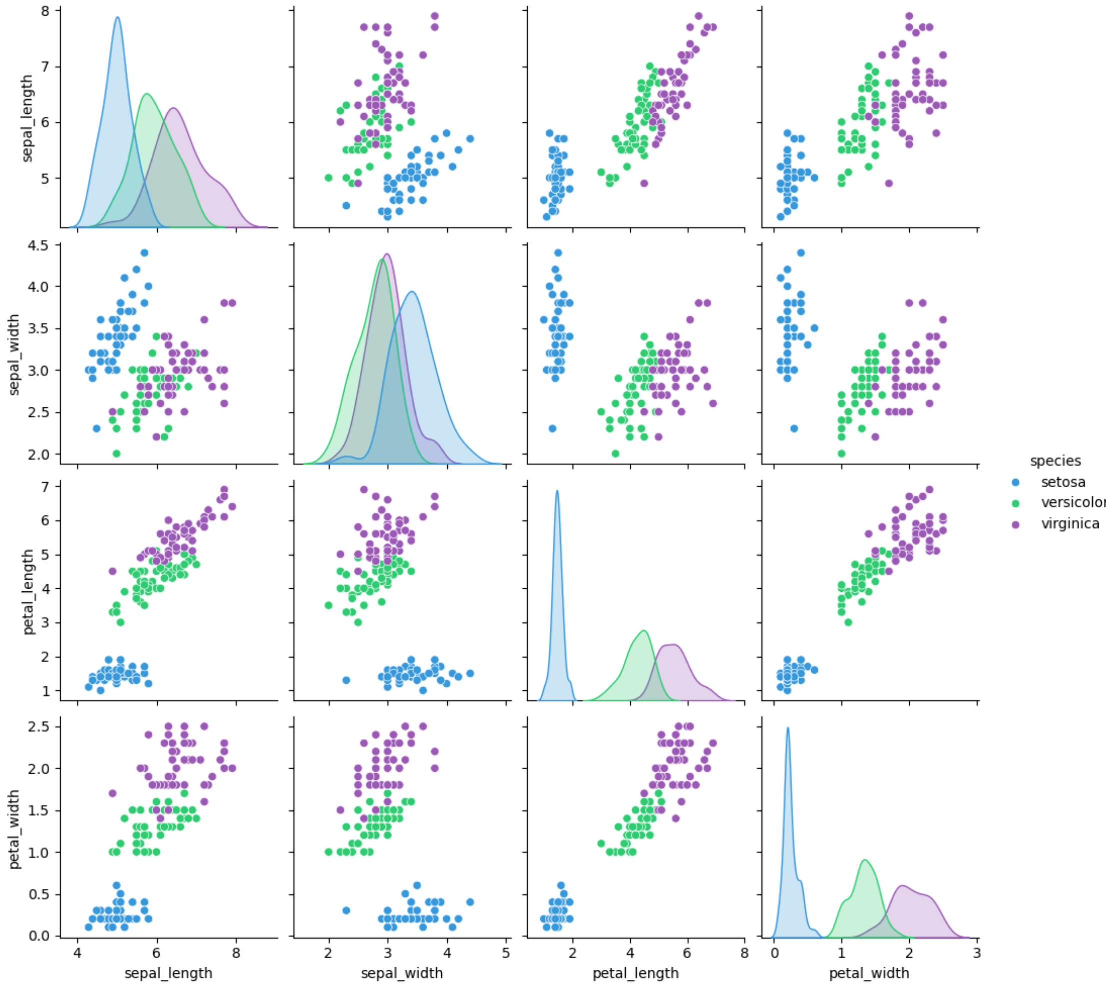
[44]:
```
plt.figure(figsize=(14,8))
sns.pairplot(mpg,hue='cylinders',palette=["#FFB74D", "#64B5F6", "#4DB6AC",␣
↪"#B39DDB", "#FF8A65"])
plt.show()
```

```
<Figure size 1400x800 with 0 Axes>
```

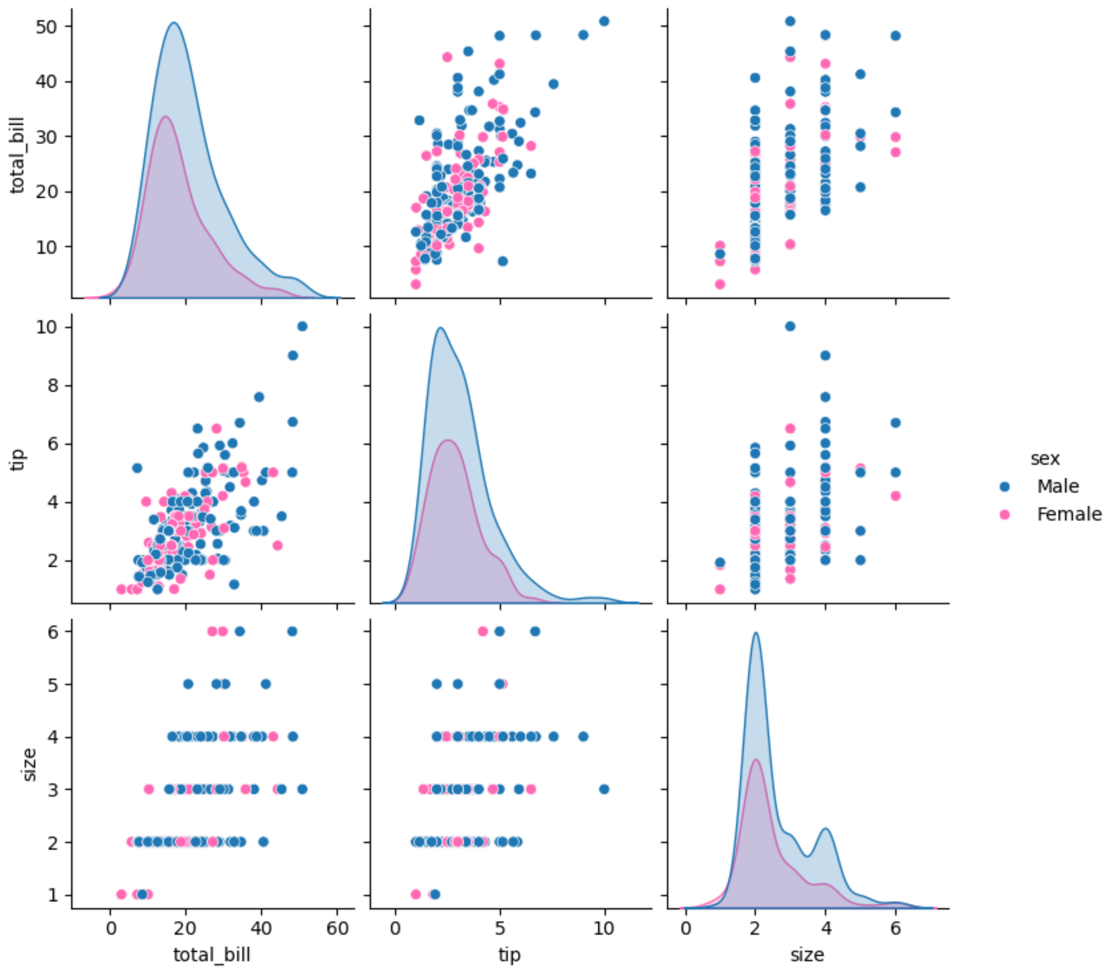


[45]:
```
plt.figure(figsize=(12,6))
sns.pairplot(iris,hue='species',palette=["#3498DB", "#2ECC71", "#9B59B6"])
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



[46]:
```
sns.pairplot(tips,hue='sex',palette=["#1f77b4", "#ff69b4"])
plt.show()
```



0.0.62 Categorical Plots in Seaborn

Categorical plots are used to visualize the distribution and relationships of categorical data. Here are some common types of categorical plots in Seaborn:

1. Bar Plot

- **Purpose**: Displays the mean (or another statistic) of a numerical variable for different categories. Each bar represents a category, and the height of the bar shows the value of the statistic.
- **Use Cases**: Useful for comparing the average values across different groups (e.g., average sales by product category).

2. Count Plot

- **Purpose**: Shows the count of observations in each category. It's a special case of the bar plot where the height of each bar represents the number of occurrences of each category.

- **Use Cases**: Useful for visualizing the frequency distribution of categorical variables (e.g., the number of students in each grade).

3. Box Plot

- **Purpose**: Displays the distribution of a numerical variable for different categories using a summary of statistics (minimum, first quartile, median, third quartile, and maximum). It shows the spread and identifies potential outliers.
- **Use Cases**: Useful for comparing distributions across categories (e.g., comparing test scores across different classes).

4. Swarm Plot

- **Purpose**: Displays all individual data points for a categorical variable, avoiding overlap. It creates a scatter-like effect for categorical data, allowing you to see the distribution of points within each category.
- **Use Cases**: Useful for visualizing the spread of individual observations and identifying clusters or outliers within categories.

5. Strip Plot

- **Purpose**: Similar to a swarm plot, the strip plot shows all individual data points for a categorical variable, but points may overlap. It provides a straightforward way to visualize the distribution of data points along a categorical axis.
- **Use Cases**: Useful for observing the distribution of individual observations within categories, especially when the number of points is relatively small.

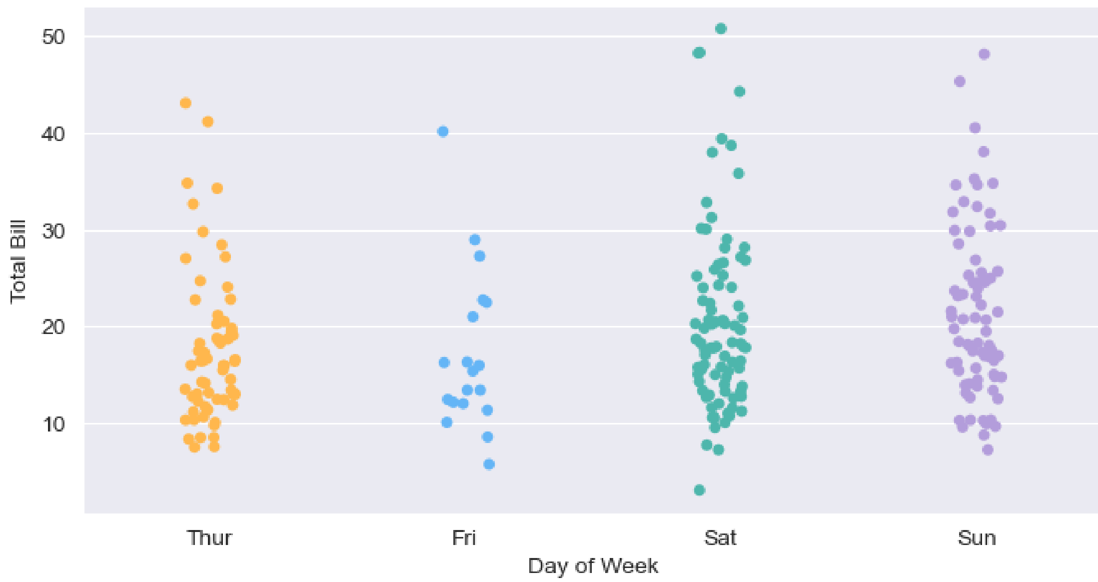
0.0.63 Summary:

- **Bar plots** and **count plots** are great for comparing summary statistics and frequencies across categories.
- **Box plots** provide insights into the distribution of numerical data, while **swarm plots** and **strip plots** give a detailed view of individual data points within categories.
- Together, these plots offer a comprehensive understanding of categorical data distributions and relationships.

[47]:
```
# sns.swarmplot // axes level
# sns.stripplot // axes level
# sns.boxplot // axes level
# sns.barplot // axes level
# sns.countplot // axes level
# sns.catplot // figure level
# kind= "strip", "swarm", "box", "bar", "count"
```

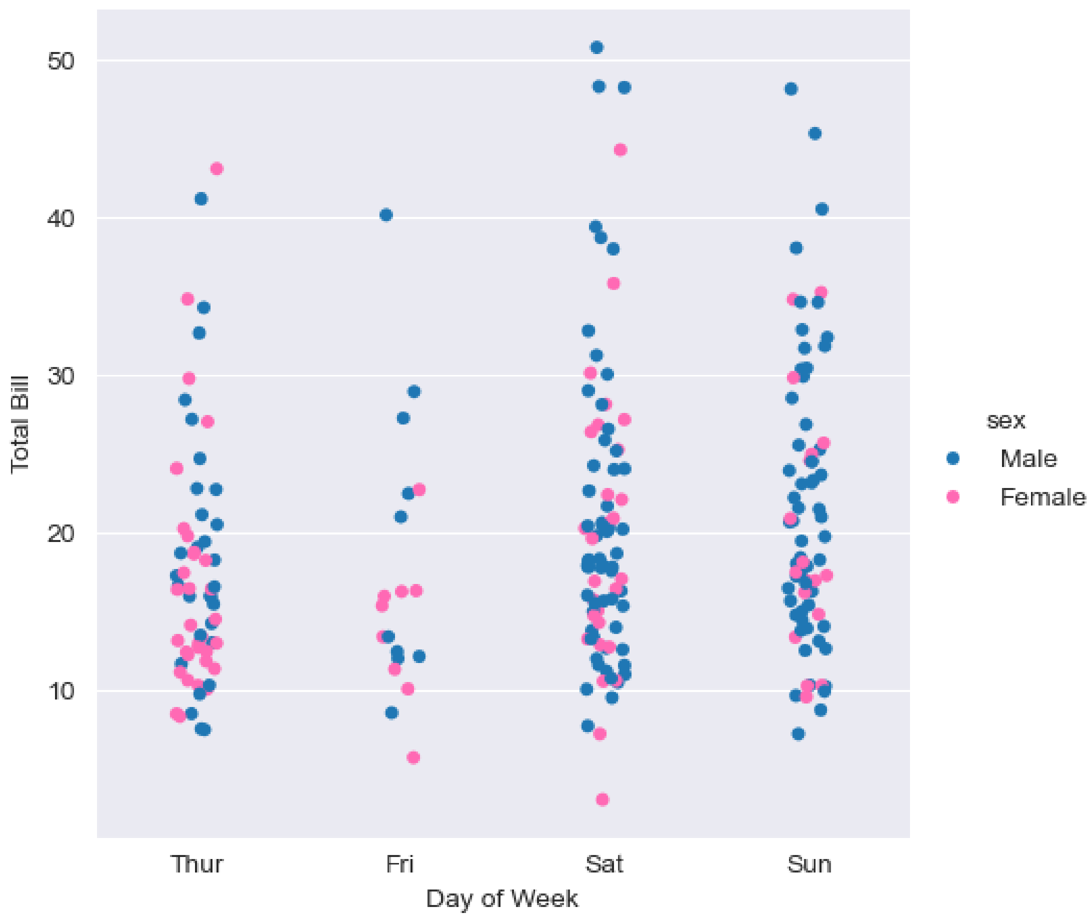
[173]:
```
# Strip Plot
plt.figure(figsize=(8,4))
sns.
 ↪stripplot(data=tips,x='day',y='total_bill',jitter=True,hue='day',palette=["#FFB74D",␣
 ↪"#64B5F6", "#4DB6AC","#B39DDB"])
```

```
plt.xlabel('Day of Week')
plt.ylabel('Total Bill')
plt.show()
```



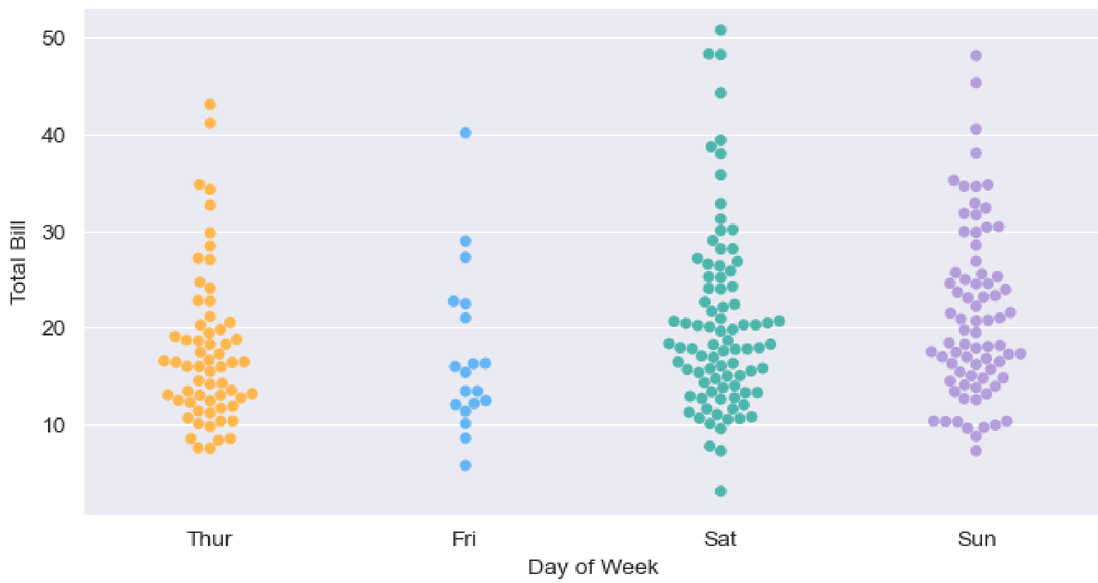
Jitter parameter is used in categorical plots, such as stripplot and swarmplot, to add a small amount of random noise (or "jitter") to the position of data points along the categorical axis.

[172]:
```
# figure level
# By default, strip plot
sns.
 ↪catplot(data=tips,x='day',y='total_bill',jitter=True,hue='sex',palette=["#1f77b4",␣
 ↪"#ff69b4"])
plt.xlabel('Day of Week')
plt.ylabel('Total Bill')
plt.show()
```

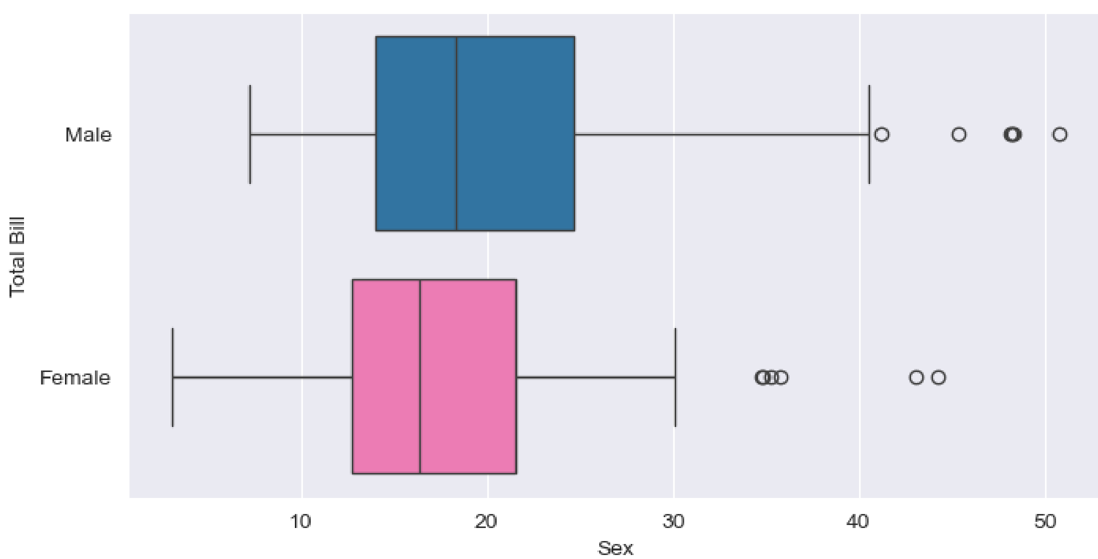


[170]:

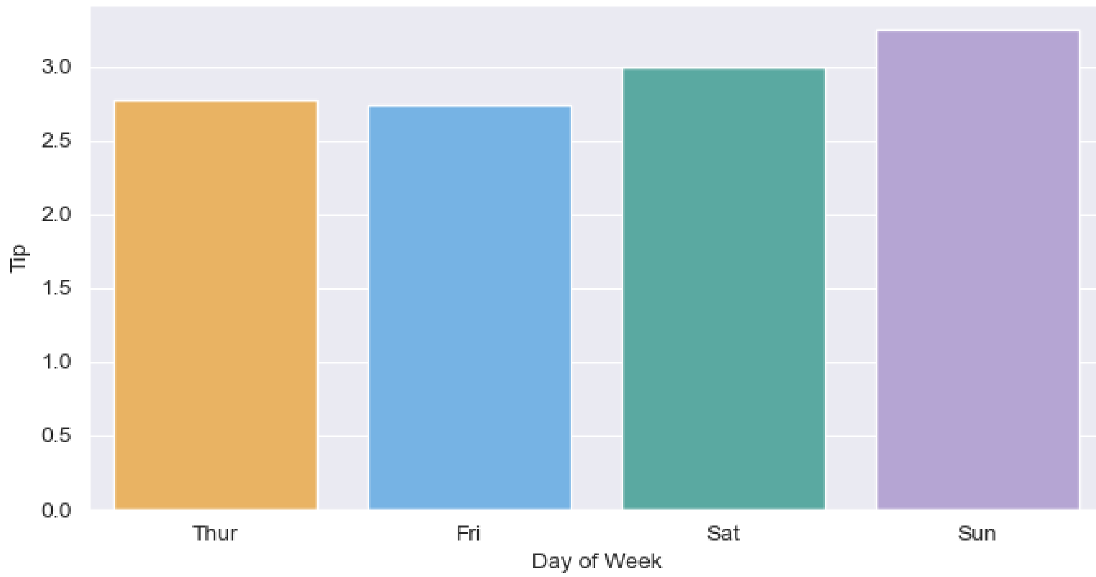
```
# Swarmplot
plt.figure(figsize=(8,4))
sns.swarmplot(data=tips,x='day',y='total_bill',hue='day',palette=["#FFB74D",␣
  ↪"#64B5F6", "#4DB6AC","#B39DDB"])
plt.xlabel('Day of Week')
plt.ylabel('Total Bill')
plt.show()
```



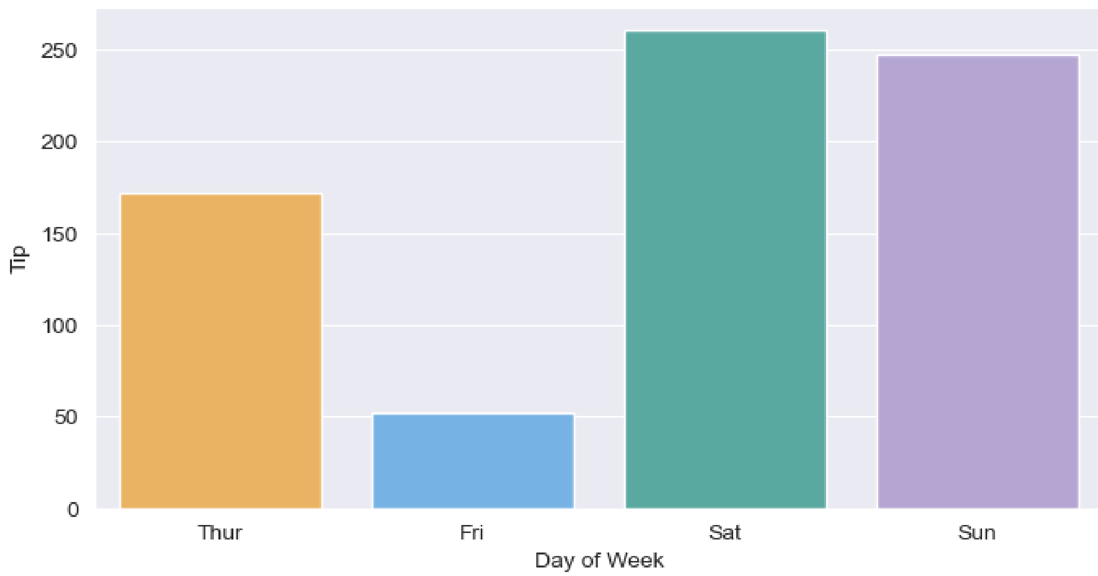
[171]:
```
# Box Plot
plt.figure(figsize=(8,4))
sns.boxplot(data=tips,y='sex',x='total_bill',hue='sex',palette=["#1f77b4",␣
↪"#ff69b4"])
plt.ylabel('Total Bill')
plt.xlabel('Sex')
plt.show()
```



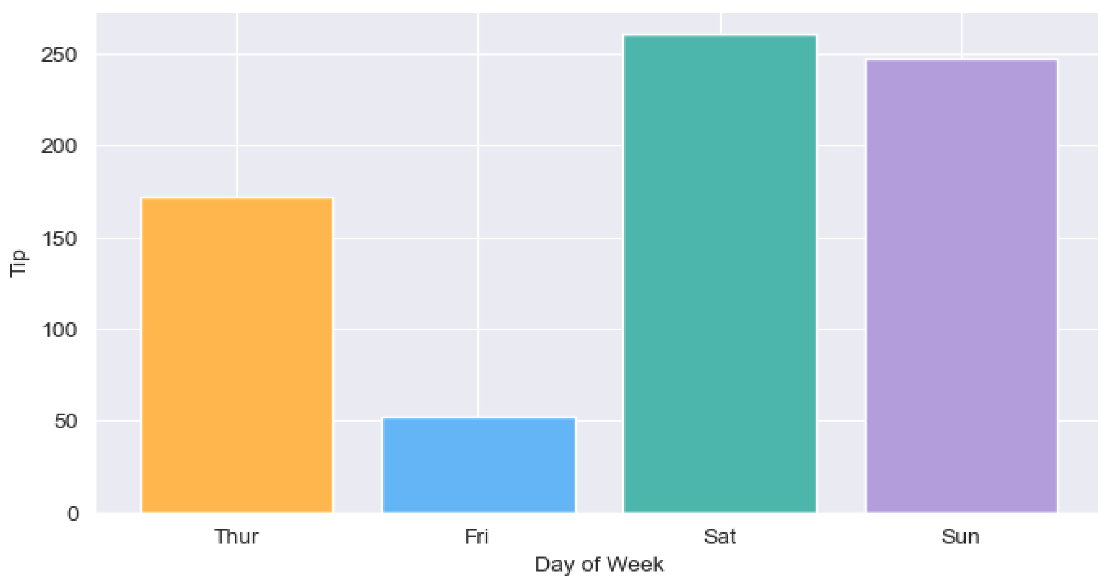
[168]:
```
# Barplot
# By default, average
plt.figure(figsize=(8,4))
sns.barplot(data=tips,x='day',y='tip',hue='day',ci=False,palette=["#FFB74D",␣
↪"#64B5F6", "#4DB6AC","#B39DDB"])
plt.xlabel('Day of Week')
plt.ylabel('Tip')
plt.show()
```



[167]:
```
# Sum
plt.figure(figsize=(8,4))
sns.barplot(data=tips,x='day',y='tip',hue='day',ci=False,palette=["#FFB74D",␣
↪"#64B5F6", "#4DB6AC","#B39DDB"],estimator=sum)
plt.xlabel('Day of Week')
plt.ylabel('Tip')
plt.show()
```

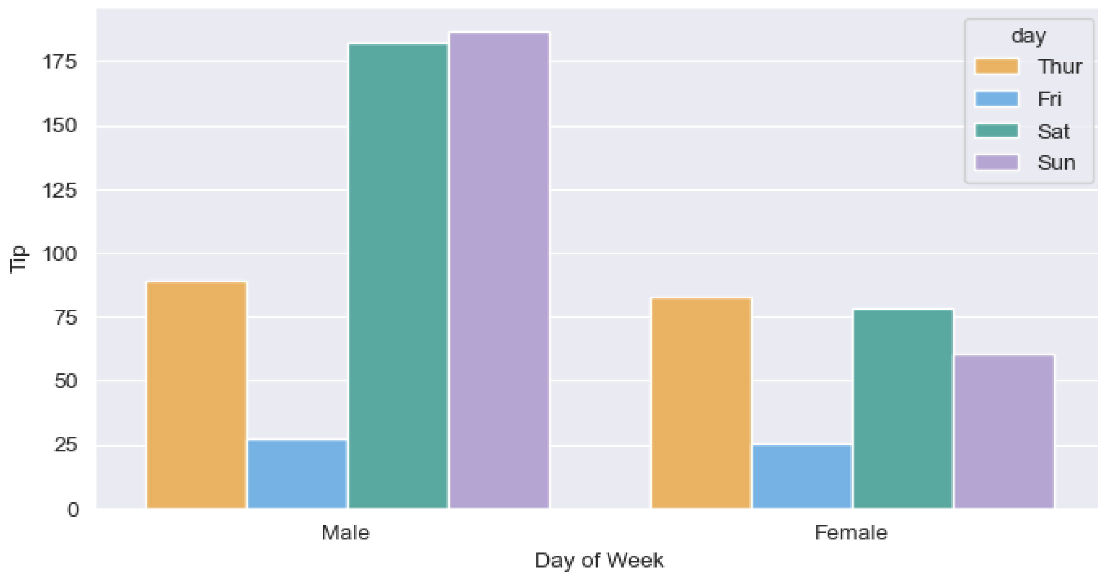


[166]:
```
# by using matplotlib
# comparision
plt.figure(figsize=(8,4))
temp_tip = tips.groupby('day')['tip'].sum().reset_index()
plt.bar(temp_tip.day,temp_tip.tip,color=["#FFB74D", "#64B5F6",␣
 ↪"#4DB6AC","#B39DDB"])
plt.xlabel('Day of Week')
plt.ylabel('Tip')
plt.show()
```



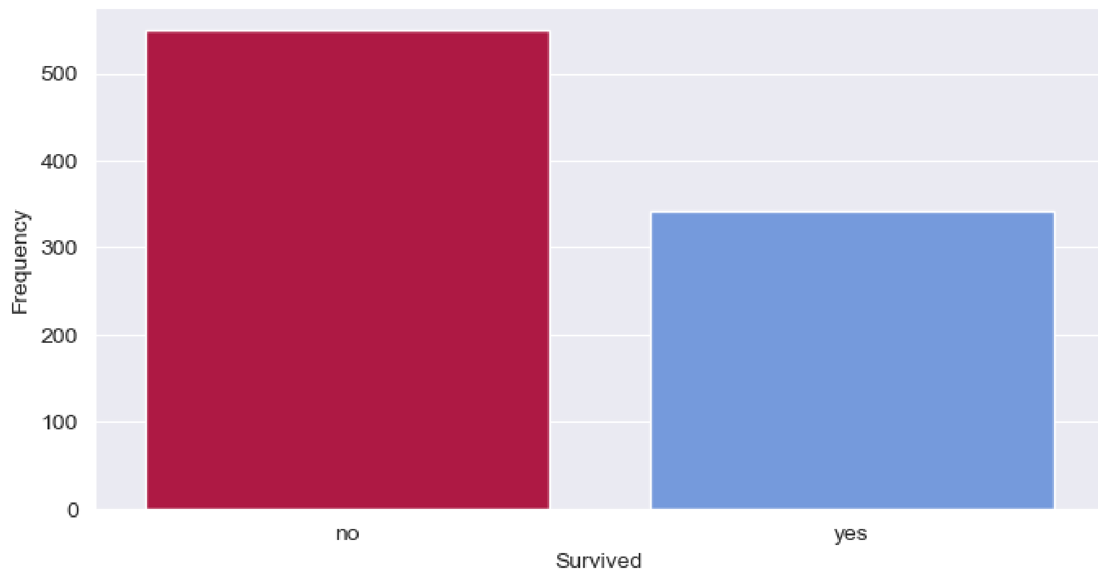
[164]:

```
# clustered column chart

plt.figure(figsize=(8,4))
sns.barplot(data=tips,x='sex',y='tip',hue='day',ci=False,palette=["#FFB74D",␣
↪"#64B5F6", "#4DB6AC","#B39DDB"],estimator=sum)
plt.xlabel('Day of Week')
plt.ylabel('Tip')
plt.show()
```



[165]:

```
plt.figure(figsize=(8,4))
sns.countplot(data=titanic,x='alive',hue='alive',palette=['#C70039','#6495ED'])
plt.xlabel('Survived')
plt.ylabel('Frequency')
plt.show()
```



0.0.64 Matrix Plots in Seaborn

Matrix plots are used to visualize data in a two-dimensional grid format, where individual values are represented by colors. They are particularly useful for visualizing relationships between variables in a dataset, especially when dealing with large amounts of data.

1. Heatmap

- **Purpose**: A heatmap displays data values in a matrix format using color gradients. Each cell in the matrix represents a value, and the color intensity corresponds to the magnitude of that value.
- **Use Cases**:
  - Visualizing correlation matrices to understand relationships between multiple variables.
  - Representing the frequency or intensity of events across two categorical dimensions (e.g., website traffic by day of the week and hour).
  - Comparing values across categories in a clear and concise manner.
- **Example**: In a heatmap representing a correlation matrix, darker colors might indicate strong correlations, while lighter colors indicate weak correlations.

2. Cluster Plot (or Cluster Map)

- **Purpose**: A cluster plot, often created using the `clustermap` function, combines the features of a heatmap with hierarchical clustering. It not only displays data values with colors but also organizes rows and/or columns based on similar patterns using clustering algorithms.
- **Use Cases**:
  - Identifying groups or clusters within the data, helping to reveal hidden patterns.

  - Useful in exploratory data analysis to see how different observations or variables relate to each other.
  - Often used in genomics, marketing segmentation, and customer profiling.
- **Example**: A clustermap might display gene expression data, where rows represent different genes and columns represent different samples. The clustering will group similar gene expression patterns together, making it easier to identify related genes.

0.0.65 Summary

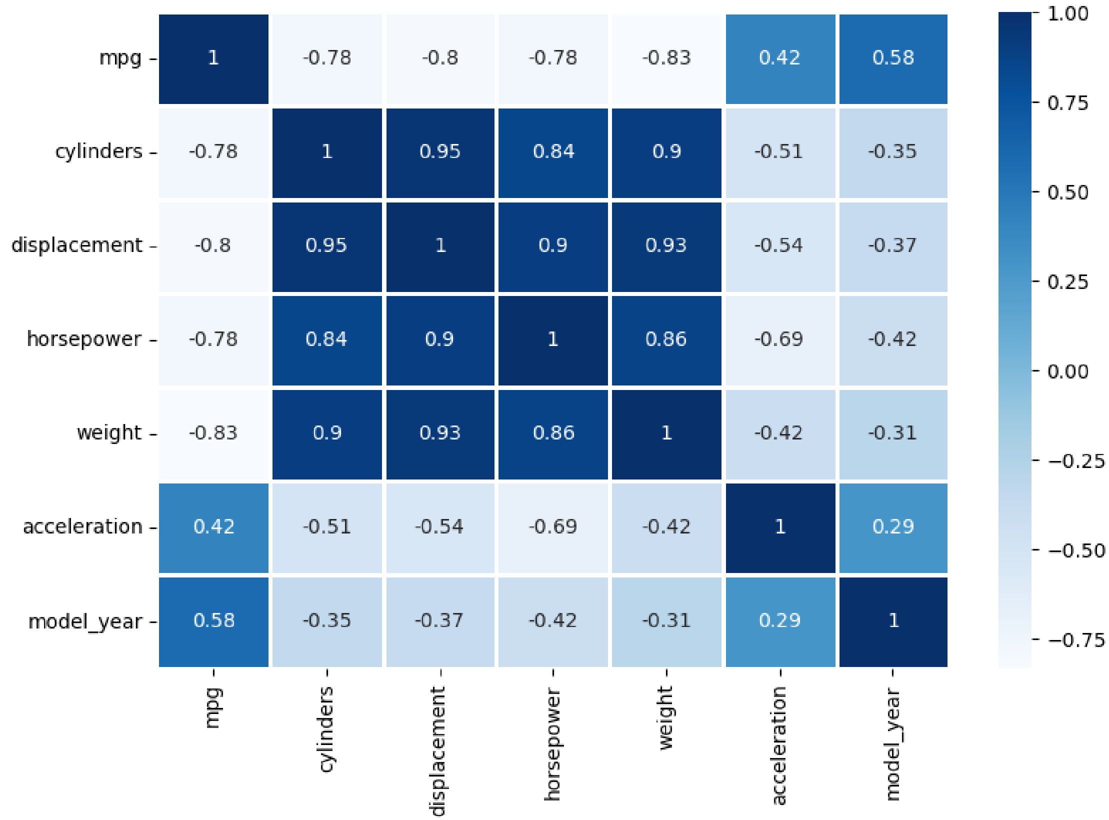
- **Heatmaps** provide a straightforward way to visualize values in a matrix format, where color indicates magnitude.
- **Cluster plots (clustermaps)** enhance heatmaps by adding hierarchical clustering, allowing for the visualization of patterns and relationships among data points.
- Both types of matrix plots are powerful tools for understanding complex datasets and uncovering relationships that may not be immediately apparent.

[48]:
```
# sns.heatmap // axes level
# sns.clustermap // axes level
```

[49]:
```
mpg.corr(numeric_only=True)
```

[49]:
```
                   mpg  cylinders  displacement  horsepower    weight  \
mpg           1.000000  -0.775396     -0.804203   -0.778427 -0.831741
cylinders    -0.775396   1.000000      0.950721    0.842983  0.896017
displacement -0.804203   0.950721      1.000000    0.897257  0.932824
horsepower   -0.778427   0.842983      0.897257    1.000000  0.864538
weight       -0.831741   0.896017      0.932824    0.864538  1.000000
acceleration  0.420289  -0.505419     -0.543684   -0.689196 -0.417457
model_year    0.579267  -0.348746     -0.370164   -0.416361 -0.306564

              acceleration  model_year
mpg               0.420289    0.579267
cylinders        -0.505419   -0.348746
displacement     -0.543684   -0.370164
horsepower       -0.689196   -0.416361
weight           -0.417457   -0.306564
acceleration      1.000000    0.288137
model_year        0.288137    1.000000
```

[50]:
```
plt.figure(figsize=(9,6))
sns.heatmap(mpg.corr(numeric_only=True),annot=True,cmap='Blues',linewidths=1)
plt.show()
```



[51]:
```python
europe = gap[gap.continent=='Europe']
```

[52]:
```python
europe
```

[52]:
```
             country continent  year  lifeExp       pop     gdpPercap  \
12           Albania    Europe  1952   55.230   1282697   1601.056136
13           Albania    Europe  1957   59.280   1476505   1942.284244
14           Albania    Europe  1962   64.820   1728137   2312.888958
15           Albania    Europe  1967   66.220   1984060   2760.196931
16           Albania    Europe  1972   67.690   2263554   3313.422188
…                …        …  …     …      …        …
1603  United Kingdom    Europe  1987   75.007  56981620  21664.787670
1604  United Kingdom    Europe  1992   76.420  57866349  22705.092540
1605  United Kingdom    Europe  1997   77.218  58808266  26074.531360
1606  United Kingdom    Europe  2002   78.471  59912431  29478.999190
1607  United Kingdom    Europe  2007   79.425  60776238  33203.261280

     iso_alpha  iso_num
12         ALB        8
13         ALB        8
14         ALB        8
```

```
15          ALB       8
16          ALB       8
…        …      …
1603        GBR     826
1604        GBR     826
1605        GBR     826
1606        GBR     826
1607        GBR     826

[360 rows x 8 columns]
```

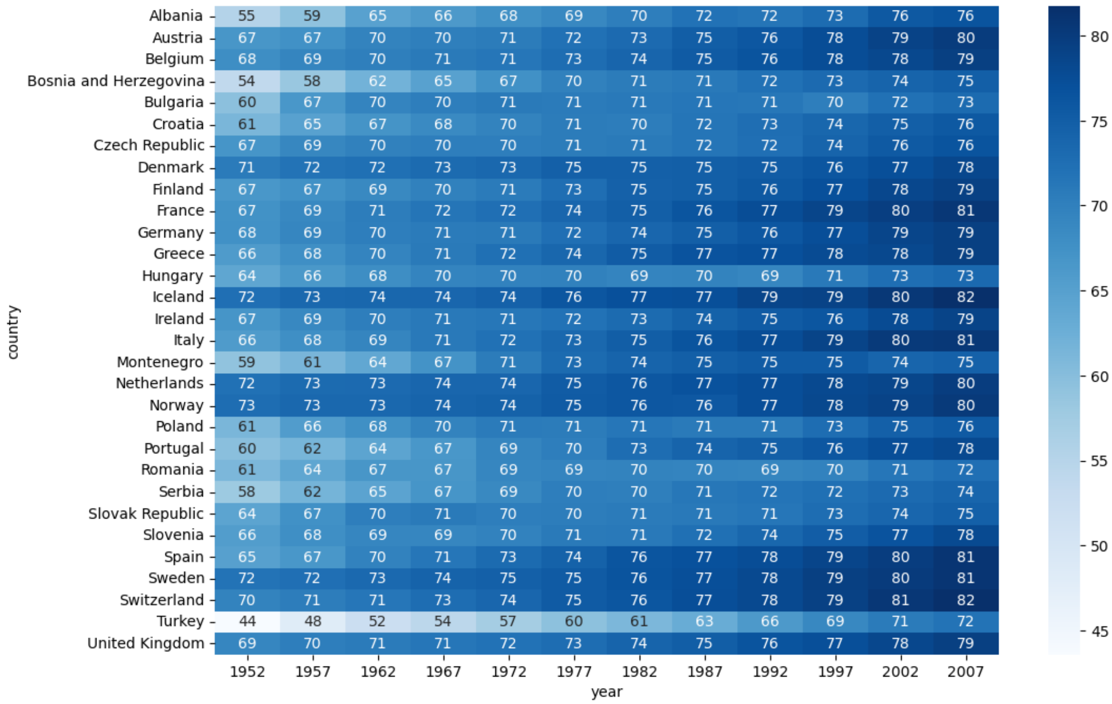
[53]:
```python
# pivots

euro_pivot= europe.pivot(index='country',columns='year',values='lifeExp')
```

[54]:
```python
euro_pivot
```

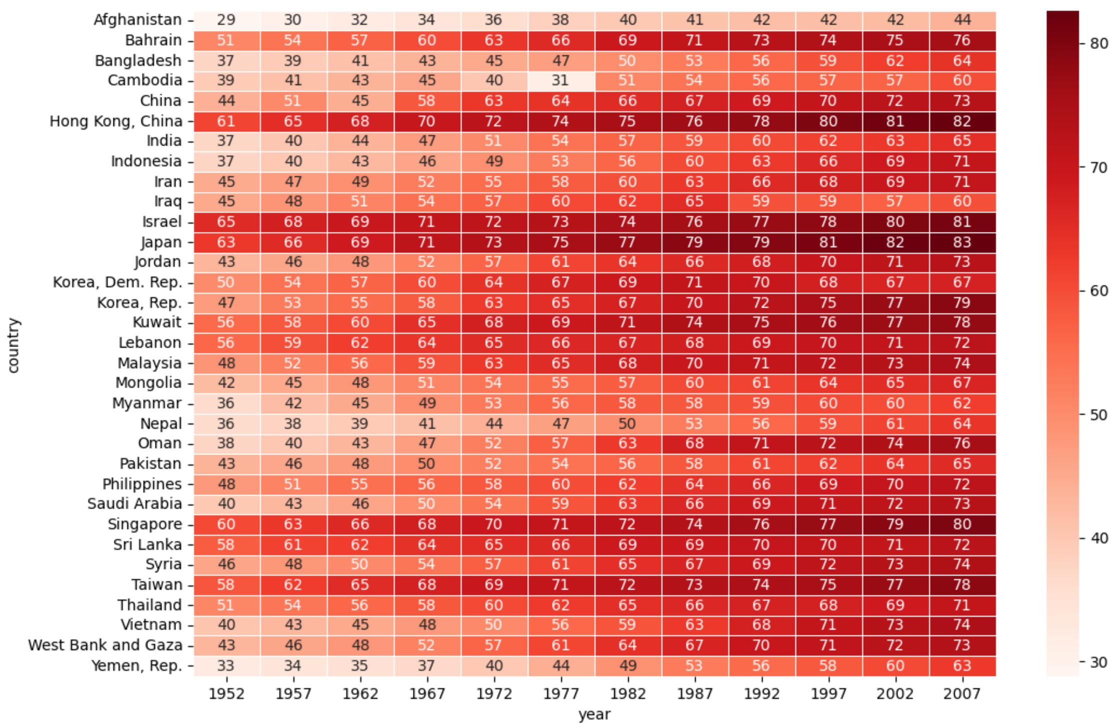
[54]:
```
year                      1952    1957    1962    1967    1972    1977  \
country
Albania                 55.230  59.280  64.820  66.220  67.690  68.930
Austria                 66.800  67.480  69.540  70.140  70.630  72.170
Belgium                 68.000  69.240  70.250  70.940  71.440  72.800
Bosnia and Herzegovina  53.820  58.450  61.930  64.790  67.450  69.860
Bulgaria                59.600  66.610  69.510  70.420  70.900  70.810
Croatia                 61.210  64.770  67.130  68.500  69.610  70.640
Czech Republic          66.870  69.030  69.900  70.380  70.290  70.710
Denmark                 70.780  71.810  72.350  72.960  73.470  74.690
Finland                 66.550  67.490  68.750  69.830  70.870  72.520
France                  67.410  68.930  70.510  71.550  72.380  73.830
Germany                 67.500  69.100  70.300  70.800  71.000  72.500
Greece                  65.860  67.860  69.510  71.000  72.340  73.680
Hungary                 64.030  66.410  67.960  69.500  69.760  69.950
Iceland                 72.490  73.470  73.680  73.730  74.460  76.110
Ireland                 66.910  68.900  70.290  71.080  71.280  72.030
Italy                   65.940  67.810  69.240  71.060  72.190  73.480
Montenegro              59.164  61.448  63.728  67.178  70.636  73.066
Netherlands             72.130  72.990  73.230  73.820  73.750  75.240
Norway                  72.670  73.440  73.470  74.080  74.340  75.370
Poland                  61.310  65.770  67.640  69.610  70.850  70.670
Portugal                59.820  61.510  64.390  66.600  69.260  70.410
Romania                 61.050  64.100  66.800  66.800  69.210  69.460
Serbia                  57.996  61.685  64.531  66.914  68.700  70.300
Slovak Republic         64.360  67.450  70.330  70.980  70.350  70.450
Slovenia                65.570  67.850  69.150  69.180  69.820  70.970
Spain                   64.940  66.660  69.690  71.440  73.060  74.390
Sweden                  71.860  72.490  73.370  74.160  74.720  75.440
Switzerland             69.620  70.560  71.320  72.770  73.780  75.390
```

```
Turkey                  43.585  48.079  52.098  54.336  57.005  59.507
United Kingdom          69.180  70.420  70.760  71.360  72.010  72.760

year                      1982    1987    1992    1997    2002    2007
country
Albania                 70.420  72.000  71.581  72.950  75.651  76.423
Austria                 73.180  74.940  76.040  77.510  78.980  79.829
Belgium                 73.930  75.350  76.460  77.530  78.320  79.441
Bosnia and Herzegovina  70.690  71.140  72.178  73.244  74.090  74.852
Bulgaria                71.080  71.340  71.190  70.320  72.140  73.005
Croatia                 70.460  71.520  72.527  73.680  74.876  75.748
Czech Republic          70.960  71.580  72.400  74.010  75.510  76.486
Denmark                 74.630  74.800  75.330  76.110  77.180  78.332
Finland                 74.550  74.830  75.700  77.130  78.370  79.313
France                  74.890  76.340  77.460  78.640  79.590  80.657
Germany                 73.800  74.847  76.070  77.340  78.670  79.406
Greece                  75.240  76.670  77.030  77.869  78.256  79.483
Hungary                 69.390  69.580  69.170  71.040  72.590  73.338
Iceland                 76.990  77.230  78.770  78.950  80.500  81.757
Ireland                 73.100  74.360  75.467  76.122  77.783  78.885
Italy                   74.980  76.420  77.440  78.820  80.240  80.546
Montenegro              74.101  74.865  75.435  75.445  73.981  74.543
Netherlands             76.050  76.830  77.420  78.030  78.530  79.762
Norway                  75.970  75.890  77.320  78.320  79.050  80.196
Poland                  71.320  70.980  70.990  72.750  74.670  75.563
Portugal                72.770  74.060  74.860  75.970  77.290  78.098
Romania                 69.660  69.530  69.360  69.720  71.322  72.476
Serbia                  70.162  71.218  71.659  72.232  73.213  74.002
Slovak Republic         70.800  71.080  71.380  72.710  73.800  74.663
Slovenia                71.063  72.250  73.640  75.130  76.660  77.926
Spain                   76.300  76.900  77.570  78.770  79.780  80.941
Sweden                  76.420  77.190  78.160  79.390  80.040  80.884
Switzerland             76.210  77.410  78.030  79.370  80.620  81.701
Turkey                  61.036  63.108  66.146  68.835  70.845  71.777
United Kingdom          74.040  75.007  76.420  77.218  78.471  79.425
```

[55]:
```python
plt.figure(figsize=(12,8))
sns.heatmap(euro_pivot,annot=True,cmap='Blues')
plt.show()
```

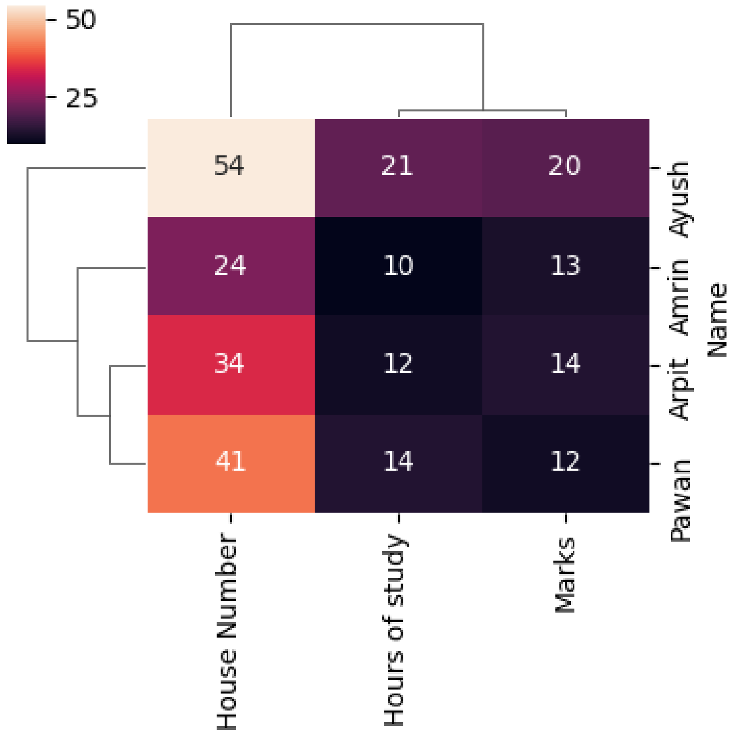


[56]:
```
asia = gap[gap.continent=='Asia']
asia_pivot = asia.pivot(index='country',columns='year',values='lifeExp')
plt.figure(figsize=(12,8))
sns.heatmap(asia_pivot,annot=True,cmap='Reds',linewidths=0.5)
plt.show()
```



[57]:

```
Dict1 = {'Name':['Arpit','Pawan','Amrin','Ayush'],
          'Hours of study':[12,14,10,21],
         'Marks':[14,12,13,20],
         'House Number':[34,41,24,54]}
stud = pd.DataFrame(Dict1)

stud.set_index('Name',inplace=True)

sns.clustermap(stud,annot=True,figsize=(4,4))
plt.show()
```



[64]:
```
stud
```

[64]:
```
        Hours of study  Marks  House Number
Name
Arpit               12     14            34
Pawan               14     12            41
Amrin               10     13            24
Ayush               21     20            54
```

0.0.66 What are Dendrograms?

- **Dendrograms** are tree-like diagrams that show hierarchical relationships between the data points (students in rows and features in columns).
- The dendrograms group similar rows or columns together based on the distance (or similarity) between them. In this case:
  - The rows (students) are clustered based on the similarity of their data (hours of study, marks, house number).
  - The columns (features: Hours of study, Marks, and House Number) are also clustered based on their similarity

[65]:
```
iris.sample(5)
```

[65]:

	sepal_length	sepal_width	petal_length	petal_width	species
100	6.3	3.3	6.0	2.5	virginica
126	6.2	2.8	4.8	1.8	virginica
138	6.0	3.0	4.8	1.8	virginica
94	5.6	2.7	4.2	1.3	versicolor
80	5.5	2.4	3.8	1.1	versicolor

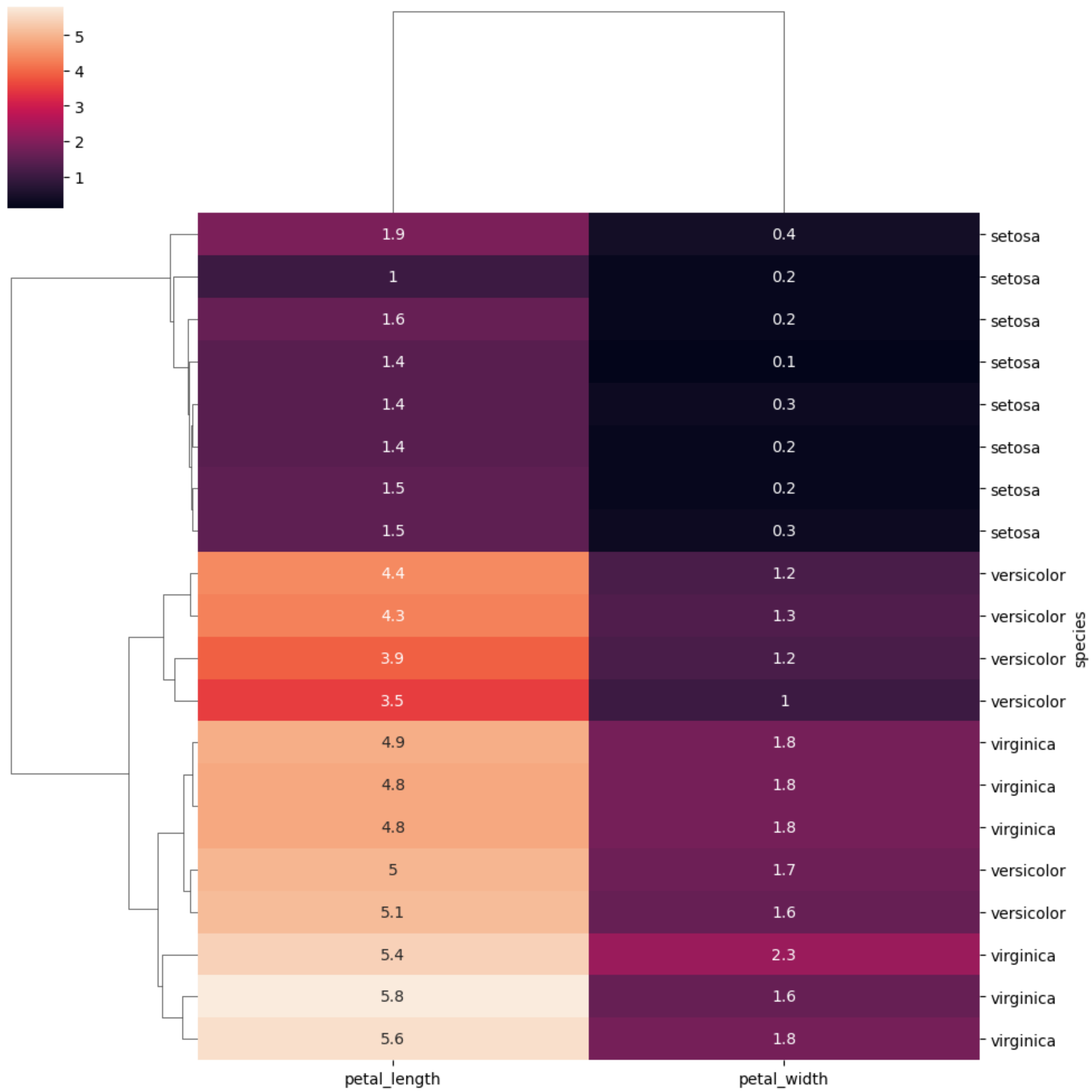
[67]:
```
iris[['petal_length','petal_width']].set_index(iris.species)
```

[67]:

	petal_length	petal_width
species		
setosa	1.4	0.2
setosa	1.4	0.2
setosa	1.3	0.2
setosa	1.5	0.2
setosa	1.4	0.2
…	…	…
virginica	5.2	2.3
virginica	5.0	1.9
virginica	5.2	2.0
virginica	5.4	2.3
virginica	5.1	1.8

[150 rows x 2 columns]

[69]:
```
sns.clustermap(iris[['petal_length','petal_width']].set_index(iris.species).
  ↪sample(20),annot=True)
plt.show()
```



0.0.67 Regression Plot (regplot) in Seaborn

`regplot` is a function in Seaborn that is used to create a scatter plot along with a linear regression line, which helps visualize the relationship between two continuous variables. It combines both scatter plotting and regression analysis, making it a valuable tool for exploratory data analysis.

Key Features:

- **Scatter Plot**: Displays individual data points for the two variables, allowing you to see the distribution of data.
- **Regression Line**: Fits a linear regression model to the data, visually representing the trend or relationship between the variables.

- **Confidence Interval**: By default, `regplot` also includes a shaded area around the regression line that represents the confidence interval of the estimate, usually set at 95%.
- **Customizability**: You can customize various aspects of the plot, such as the color of the points, the type of regression (linear or polynomial), and the inclusion of a scatter plot without the regression line.

0.0.68 Parameters:

- `x`: The name of the variable to be plotted on the x-axis.
- `y`: The name of the variable to be plotted on the y-axis.
- `data`: The dataset containing the variables.
- `order`: Specify the order of the polynomial regression if you want to fit a polynomial regression line (e.g., `order=2` for a quadratic fit).
- `ci`: Confidence interval for the regression estimate. You can set it to `None` if you don't want the confidence interval displayed.

0.0.69 Summary:

`regplot` is an effective tool for visualizing the relationship between two continuous variables through scatter plots and regression lines. It helps in understanding how one variable affects another, identifying trends, and making predictions based on the fitted model. Its ability to display confidence intervals adds further value to the analysis, allowing for better insights into the reliability of the regression estimates.

[163]:
```
plt.figure(figsize=(8,4))
sns.regplot(data=tips,x='total_bill',y='tip',ci=False)
plt.title('Regession Plot Between Total Bill and Tip')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```

[ ]:

0.1 Summary

0.1.1 1. Distribution Plots

- **Histogram**: Shows the frequency distribution of a numerical variable by dividing it into bins.
- **KDE Plot**: Smooths the distribution of a continuous variable, providing a probability density estimate.
- **Rug Plot**: Displays individual data points along the x-axis to visualize distribution.

0.1.2 2. Relational Plots

- **Scatter Plot**: Visualizes the relationship between two continuous variables using individual data points.
- **Line Plot**: Connects data points with a line to show trends over time or ordered categories.

0.1.3 3. Categorical Plots

- **Bar Plot**: Represents the mean or total of a numerical variable for different categories.
- **Count Plot**: Displays the count of observations in each category.
- **Box Plot**: Summarizes the distribution of a numerical variable, highlighting median, quartiles, and outliers.
- **Swarm Plot**: Shows individual data points for categorical variables without overlap.
- **Strip Plot**: Similar to a swarm plot but allows overlapping points.

0.1.4 4. Matrix Plots

- **Heatmap**: Visualizes data values in a matrix format using color gradients.
- **Cluster Plot**: Combines a heatmap with hierarchical clustering to show relationships between data points.

0.1.5 5. Pair Plot

- **Pair Plot**: Displays pairwise relationships in a dataset using a grid of scatter plots.

0.1.6 6. Regression Plot

- **Regplot**: Combines a scatter plot with a linear regression line to visualize the relationship between two continuous variables, including a confidence interval.

These graphs provide essential visual insights into data distributions, relationships, and patterns.

Seaborn provides several built-in themes to style your plots. You can change the aesthetics of your plots using the `set_style()` and `set_context()` functions, and you can control color palettes using `set_palette()`.

0.1.7 1. Themes (Set Using `set_style`)

These themes control the appearance of background, gridlines, and ticks:

- **"darkgrid"** (default): Gridlines on a dark background.
- **"whitegrid"**: Gridlines on a white background.
- **"dark"**: No gridlines, dark background.
- **"white"**: No gridlines, white background.
- **"ticks"**: Adds small ticks to the axes.

Example:

```
import seaborn as sns

# Set theme to "whitegrid"
sns.set_style("whitegrid")
```

0.1.8 2. Context (Set Using `set_context`)

Context themes control the scale of plot elements (e.g., font size, line width) for different use cases. Options are:

- **"paper"** (default): Smallest scale, suitable for small figures.
- **"notebook"**: Medium-sized elements for use in Jupyter notebooks.
- **"talk"**: Larger elements, ideal for presentations.
- **"poster"**: Largest scale, good for large posters and visualizations.

Example:

```
# Set context to "talk"
sns.set_context("talk")
```

0.1.9 3. Color Palettes

Seaborn has various predefined color palettes that you can use or customize, such as:

- **"deep"** (default)
- **"muted"**
- **"pastel"**
- **"bright"**
- **"dark"**
- **"colorblind"**

Example:

```
# Set color palette to "pastel"
sns.set_palette("pastel")
```

You can experiment with these themes and contexts to find what suits your visualizations best!

These can be applied using sns.set_style() for customizing the appearance of your plots.

[162]:
```
sns.set_style('darkgrid')
```

I'm **Anshum Banga**, a Data Scientist and Trainer with expertise in Python, machine learning, and data visualization. I specialize in Matplotlib, Power BI, and Tableau, helping learners develop practical data skills. Connect with me on LinkedIn(www.linkedin.com/in/anshumbanga).