DATA VISUALIZATION WITH SEABORN (PYTHON)

Seaborn is a popular data visualization library in Python that is built on top of Matplotlib. It provides a high-level interface for creating informative and visually appealing statistical graphics. Seaborn is particularly useful for exploratory data analysis and for presenting the relationships between variables in a dataset.

Here are some key features of Seaborn:

1. Statistical Visualization: Seaborn provides a wide range of statistical plots, such as scatter plots, line plots, bar plots, histograms, box plots, violin plots, and heatmaps. These plots often include built-in features for representing underlying statistical relationships, such as confidence intervals, regression lines, or summary statistics.

2. Integration with Pandas: Seaborn seamlessly works with Pandas data structures, such as DataFrames and Series. It allows you to easily visualize patterns and relationships within your data using concise syntax.

3. Aesthetically Pleasing Defaults: Seaborn has built-in themes and color palettes that enhance the visual appeal of plots. The default styles are often more visually appealing than those in Matplotlib, making it easier to create attractive plots with minimal effort.

4. Categorical Data Support: Seaborn has specialized functions for visualizing categorical data. It allows you to create grouped bar plots, count plots, and point plots to showcase relationships between variables based on categorical attributes.

5. Statistical Estimation: Seaborn provides functions for estimating and plotting various statistical models, such as linear regression models and logistic regression models. These functions make it straightforward to visualize the results of statistical analyses.

Overall, Seaborn is a powerful tool for creating visually appealing and informative data visualizations in Python, particularly when working with statistical data.

**DISTRIBUTION PLOTS** Let's discuss some plots that I used here to visualize the distribution of a data set. These plots are:

distplot jointplot pairplot rugplot kdeplot

In [1]:
```
import seaborn as sns
%matplotlib inline
```

**DATA** Seaborn comes with built-in data sets!

In [45]:
```
tips = sns.load_dataset('tips')
flights = sns.load_dataset('flights')
tips = sns.load_dataset('tips')
iris = sns.load_dataset('iris')
```

In [3]:
```
tips.head()
```

Out[3]:

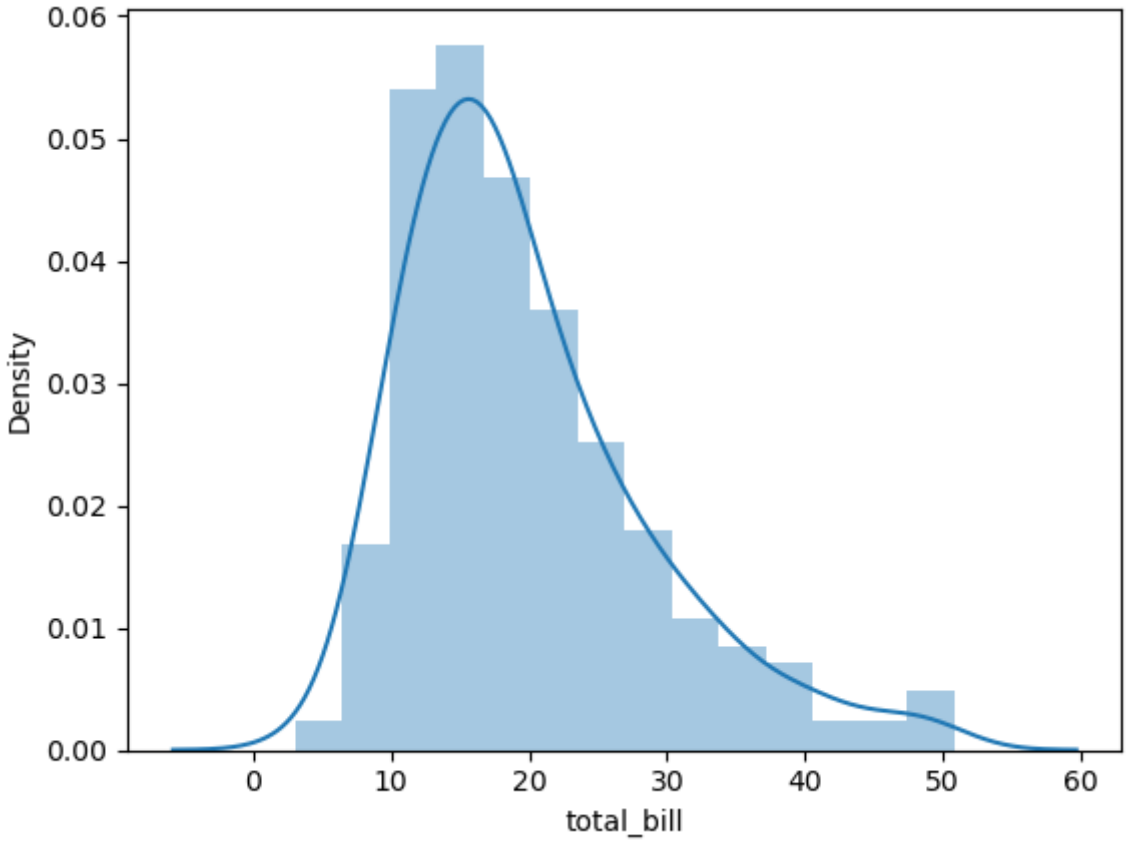
| | total_bill | tip | sex | smoker | day | time | size |
|---|---|---|---|---|---|---|---|
| **0** | 16.99 | 1.01 | Female | No | Sun | Dinner | 2 |
| **1** | 10.34 | 1.66 | Male | No | Sun | Dinner | 3 |
| **2** | 21.01 | 3.50 | Male | No | Sun | Dinner | 3 |
| **3** | 23.68 | 3.31 | Male | No | Sun | Dinner | 2 |
| **4** | 24.59 | 3.61 | Female | No | Sun | Dinner | 4 |

**DISTPLOT** The distplot shows the distribution of a univariate set of observations.

In [4]:
```
sns.distplot(tips['total_bill'])
```

```
/var/folders/15/bdksz9nj30gfrtmt__dlzxdh0000gn/T/ipykernel_12242/4271412032.p
y:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(tips['total_bill'])
```

Out[4]:
```
<Axes: xlabel='total_bill', ylabel='Density'>
```

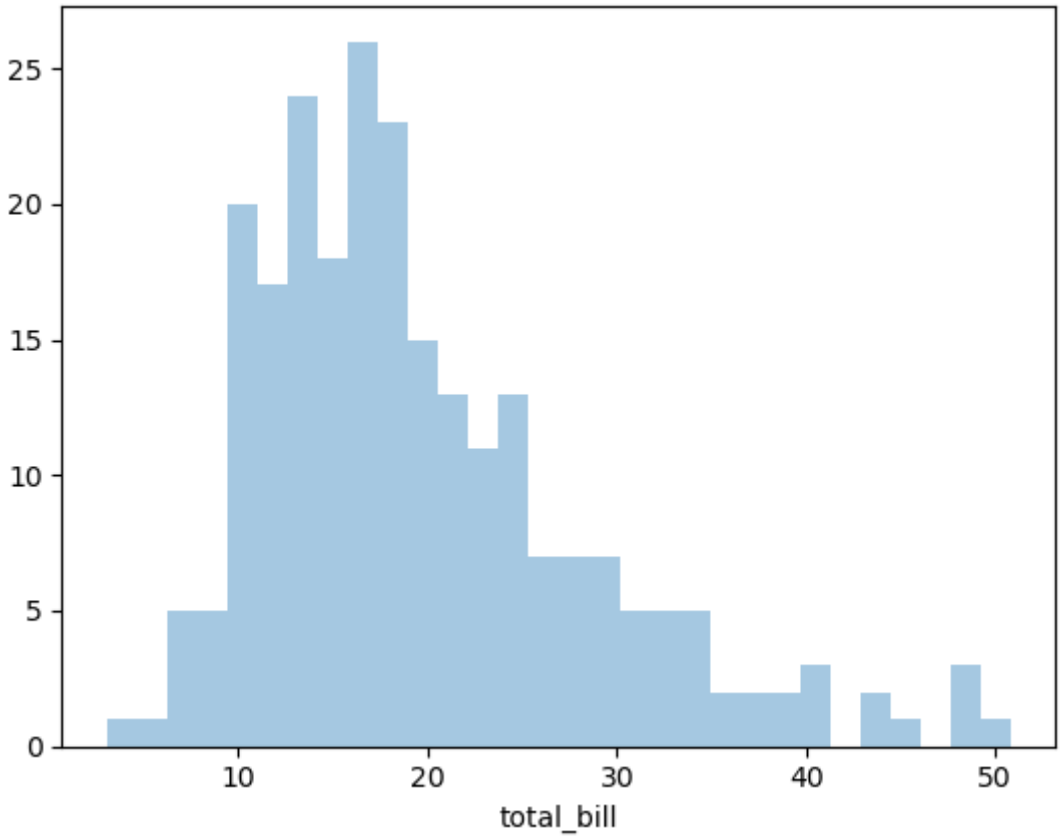


To remove the kde layer and just have the histogram use:

In [5]:
```
sns.distplot(tips['total_bill'],kde=False,bins=30)
```

```
/var/folders/15/bdksz9nj30gfrtmt__dlzxdh0000gn/T/ipykernel_12242/1274391954.p
y:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(tips['total_bill'],kde=False,bins=30)
```

Out[5]:
```
<Axes: xlabel='total_bill'>
```

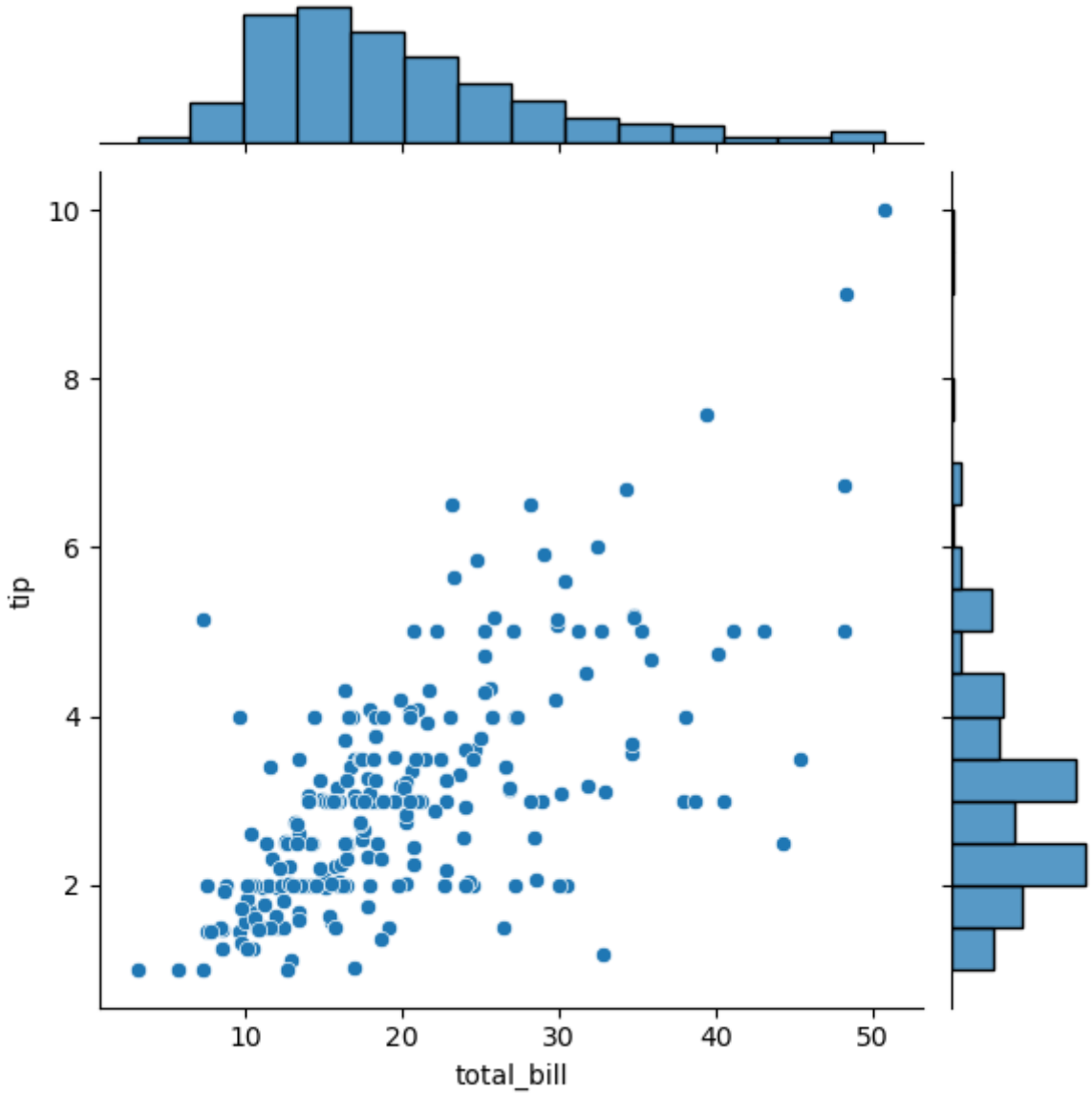



**JOINTPLOT** jointplot() allows me to basically match up two distplots for bivariate data. Different parameters used here are:

"scatter" "reg" "resid" "kde" "hex"

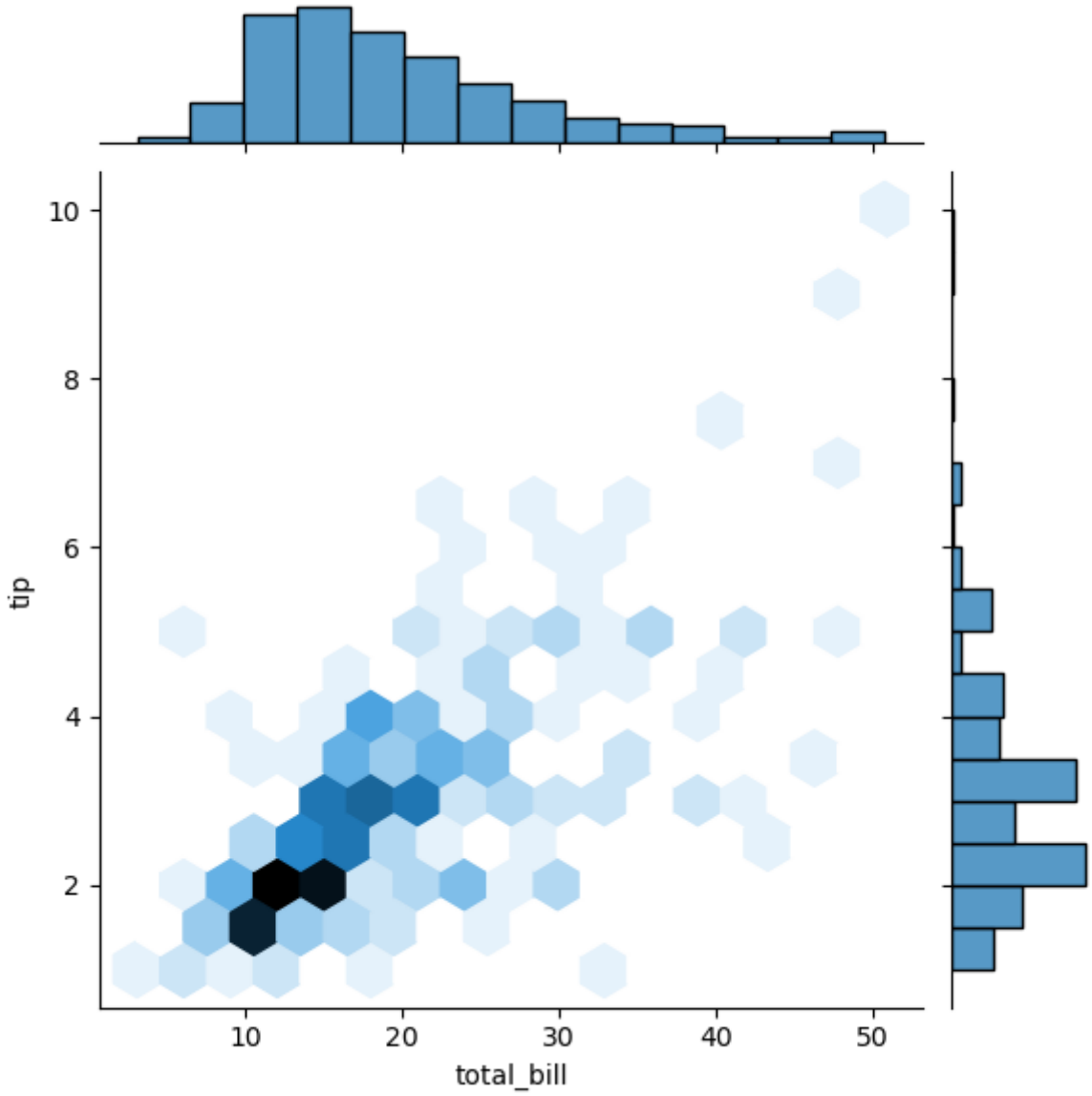
In [6]:
```
sns.jointplot(x='total_bill',y='tip',data=tips,kind='scatter')
```

Out[6]:
```
<seaborn.axisgrid.JointGrid at 0x136863460>
```



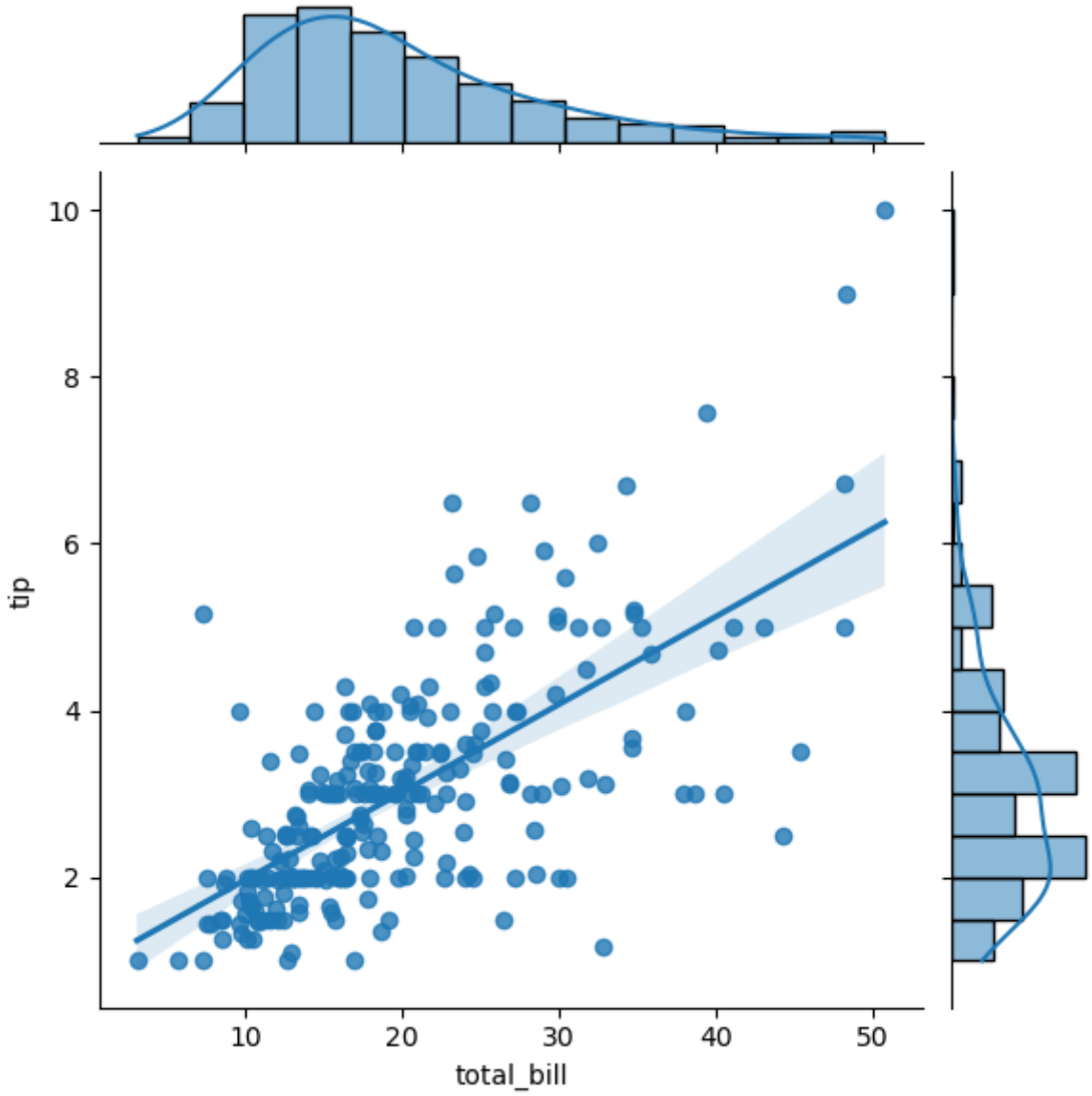

In [7]:
```
sns.jointplot(x='total_bill',y='tip',data=tips,kind='hex')
```

Out[7]:
```
<seaborn.axisgrid.JointGrid at 0x136acdd50>
```




In [8]:
```
sns.jointplot(x='total_bill',y='tip',data=tips,kind='reg')
```

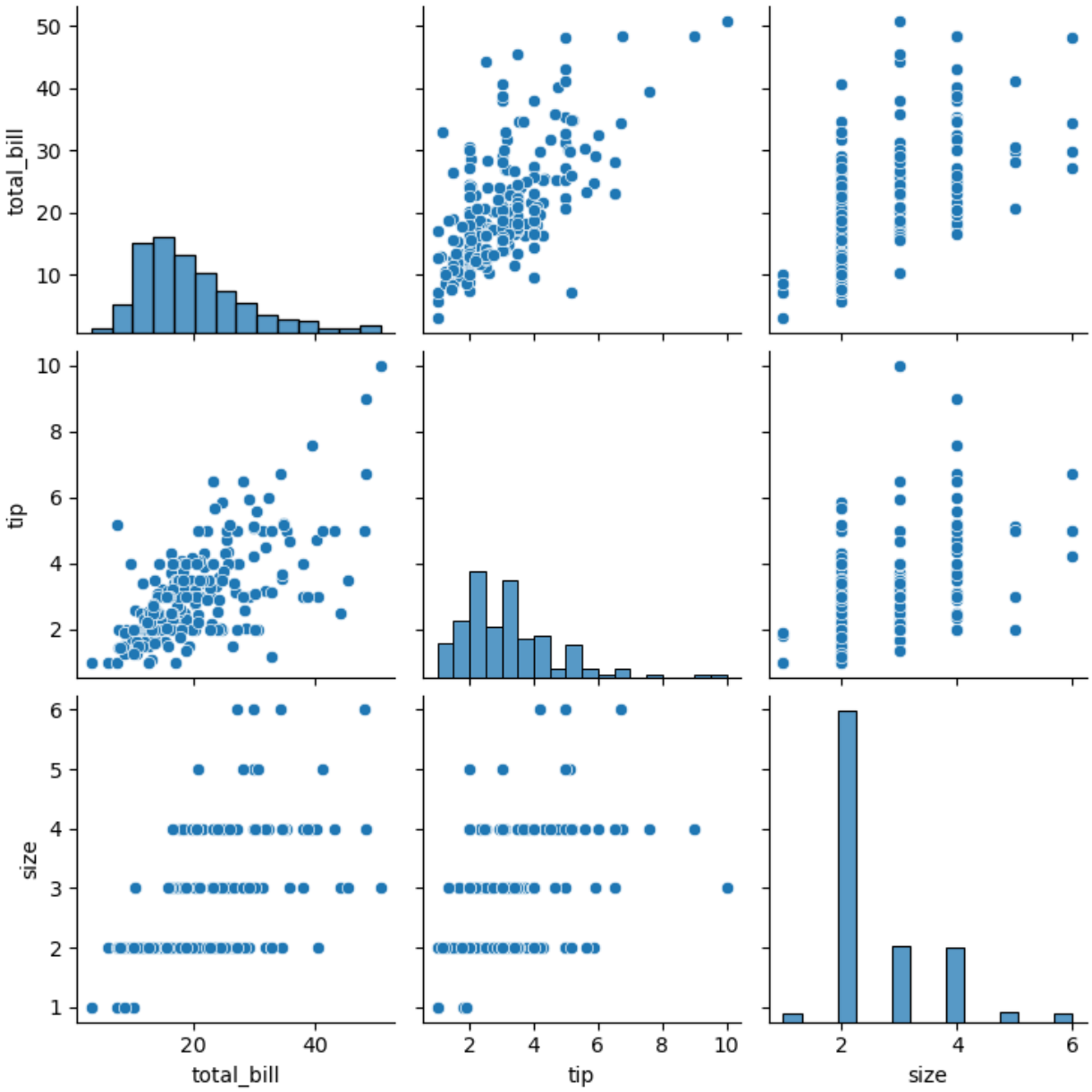
Out[8]:
```
<seaborn.axisgrid.JointGrid at 0x136f3f3a0>
```




**PAIRPLOT** pairplot will plot pairwise relationships across an entire dataframe (for the numerical columns) and supports a color hue argument (for categorical columns).

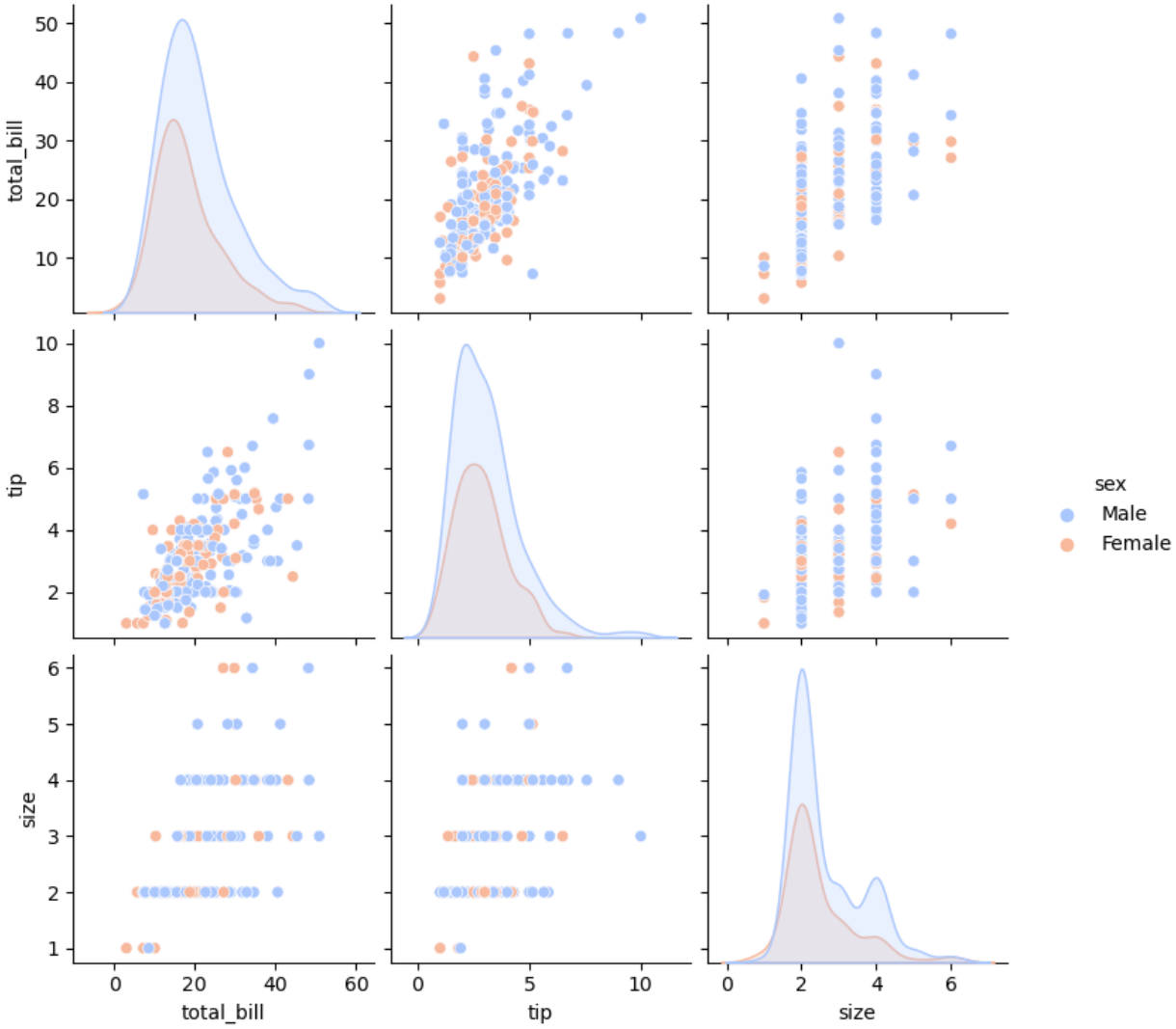
In [9]:
```
sns.pairplot(tips)
```

Out[9]:
```
<seaborn.axisgrid.PairGrid at 0x137109360>
```




In [10]:
```
sns.pairplot(tips,hue='sex',palette='coolwarm')
```

Out[10]:
```
<seaborn.axisgrid.PairGrid at 0x1375b2c20>
```




**CATEGORICAL DATA PLOTS** Now, I will use seaborn to plot categorical data! There are a few main plot types for this:

factorplot boxplot violinplot stripplot swarmplot barplot countplot

In [11]:
```
import seaborn as sns
%matplotlib inline
```

In [12]:
```
tips = sns.load_dataset('tips')
tips.head()
```

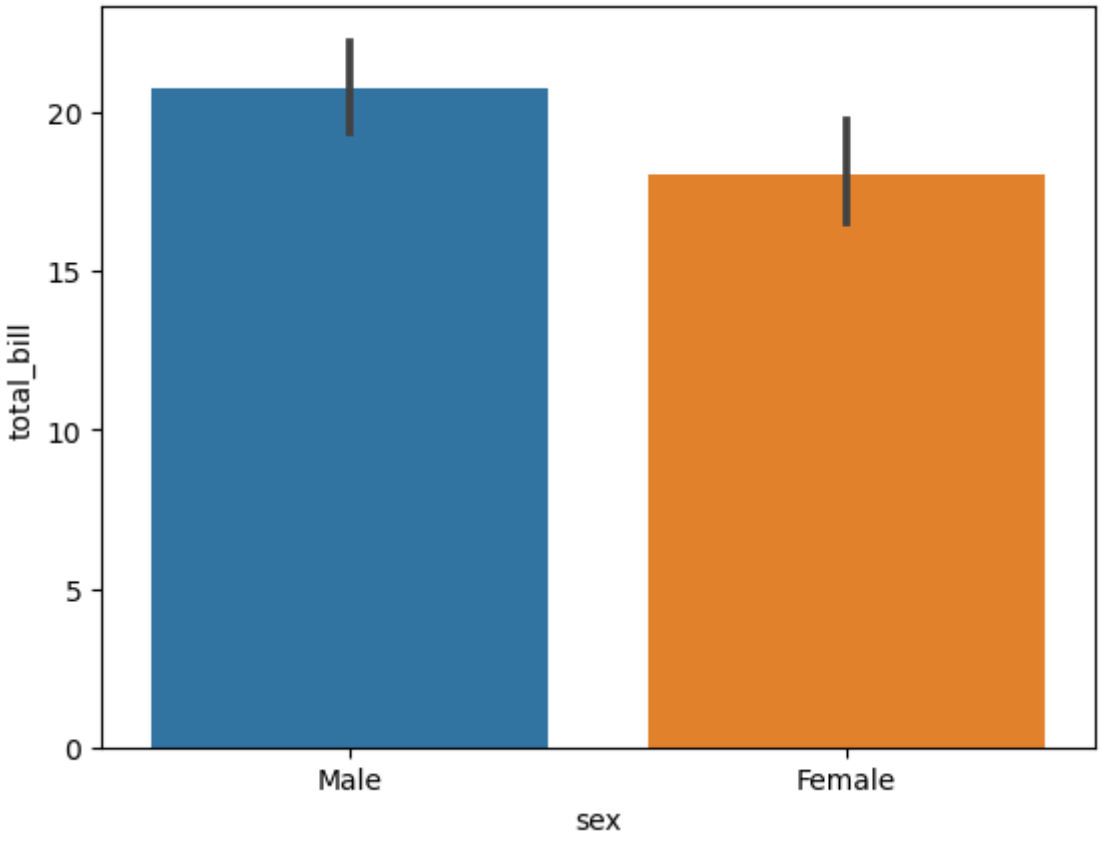
Out[12]:

| | 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 |

**BARPLOT** and **COUNTPLOT** These very similar plots allows to get aggregate data of a categorical feature in my data.

In [13]:
```
sns.barplot(x='sex',y='total_bill',data=tips)
```

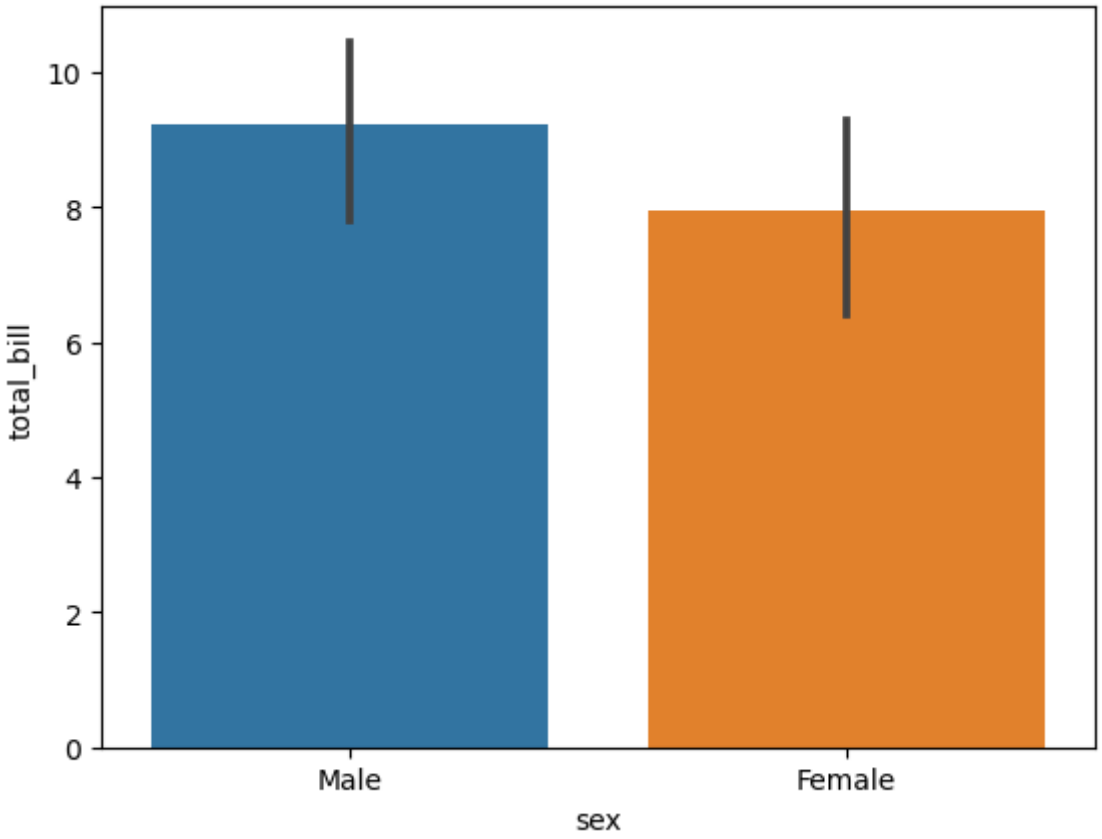
Out[13]:
```
<Axes: xlabel='sex', ylabel='total_bill'>
```




In [14]:
```
import numpy as np
```

In [15]:
```
sns.barplot(x='sex',y='total_bill',data=tips,estimator=np.std)
```

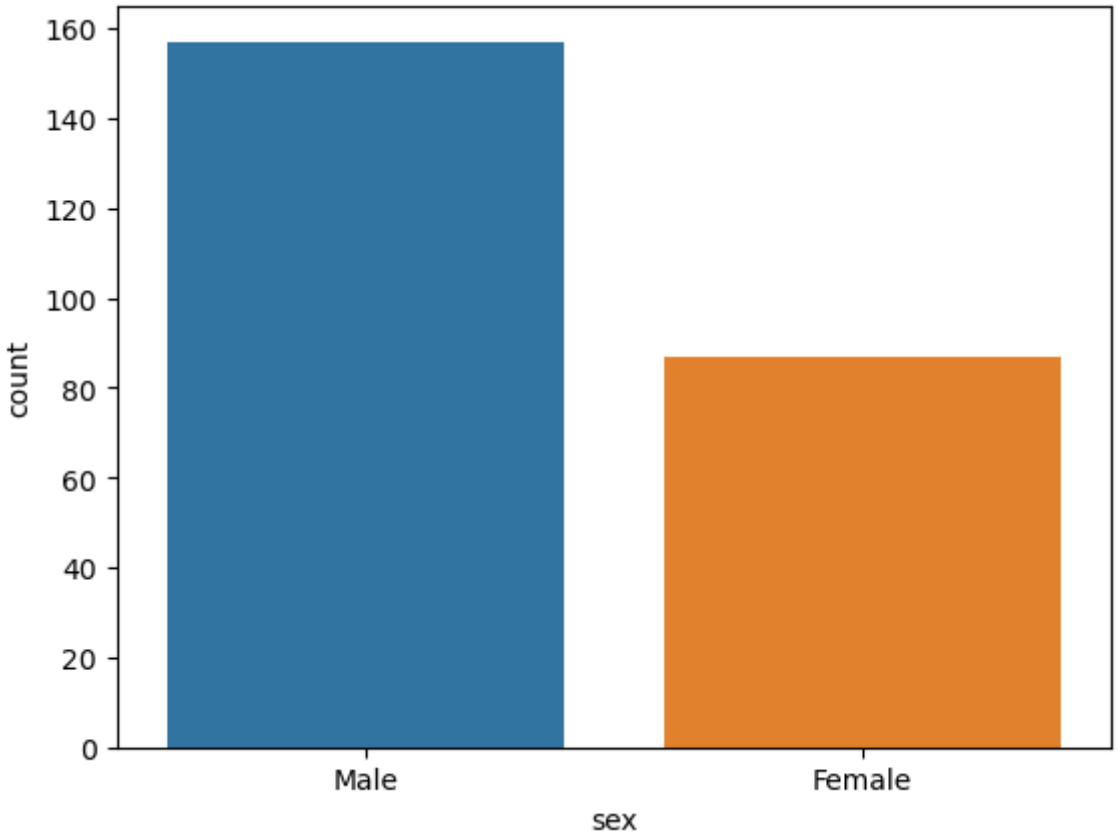
Out[15]:
```
<Axes: xlabel='sex', ylabel='total_bill'>
```




**COUNTPLOT** This is essentially the same as barplot except the estimator is explicitly counting the number of occurrences. Which is why I only pass the x value:

In [16]:
```
sns.countplot(x='sex',data=tips)
```

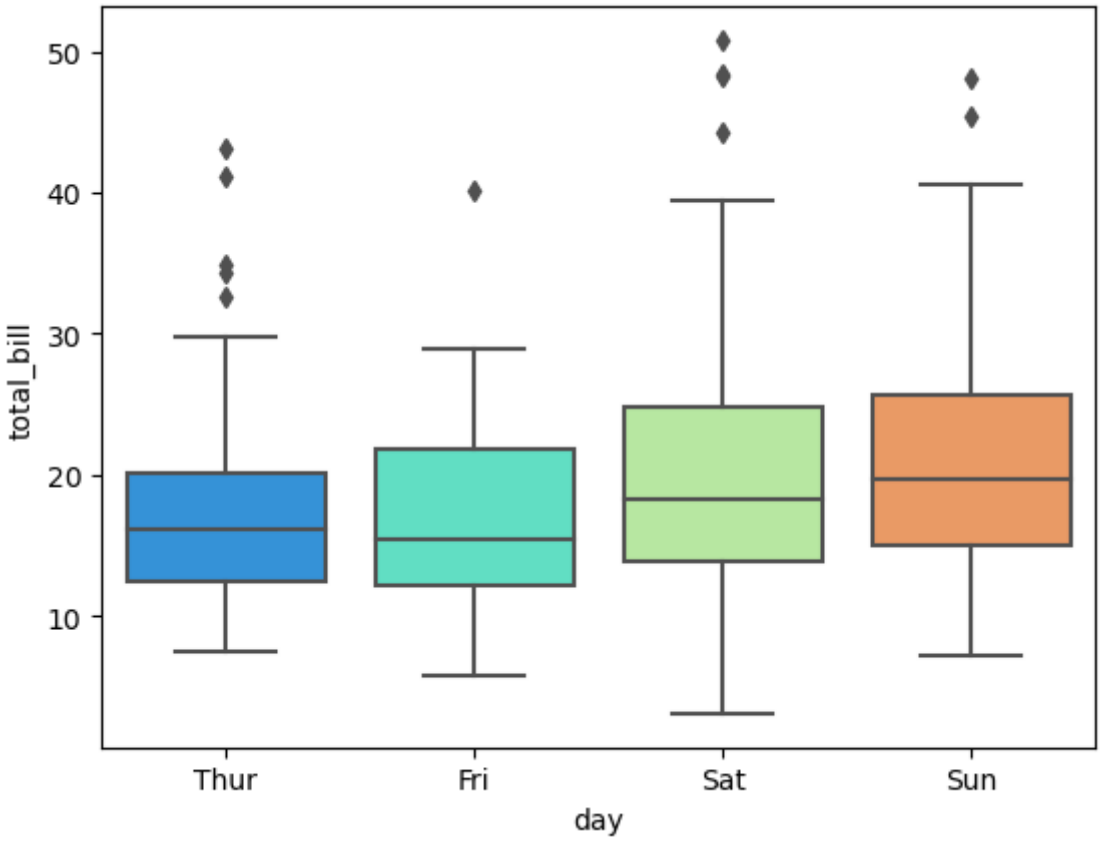
Out[16]:
```
<Axes: xlabel='sex', ylabel='count'>
```




**BOXPLOT** and **VIOLINPLOT** boxplots and violinplots are used to shown the distribution of categorical data. A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be “outliers” using a method that is a function of the inter-quartile range.

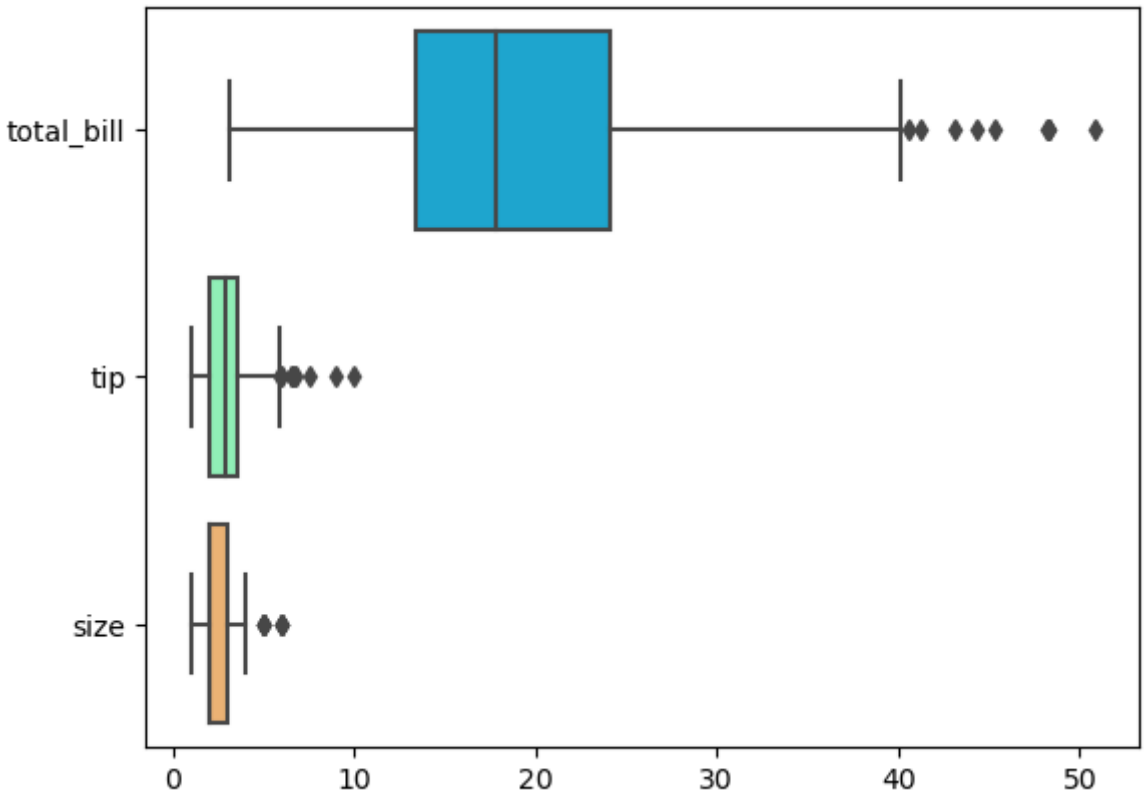
In [17]:
```
sns.boxplot(x="day", y="total_bill", data=tips,palette='rainbow')
```

Out[17]:
```
<Axes: xlabel='day', ylabel='total_bill'>
```



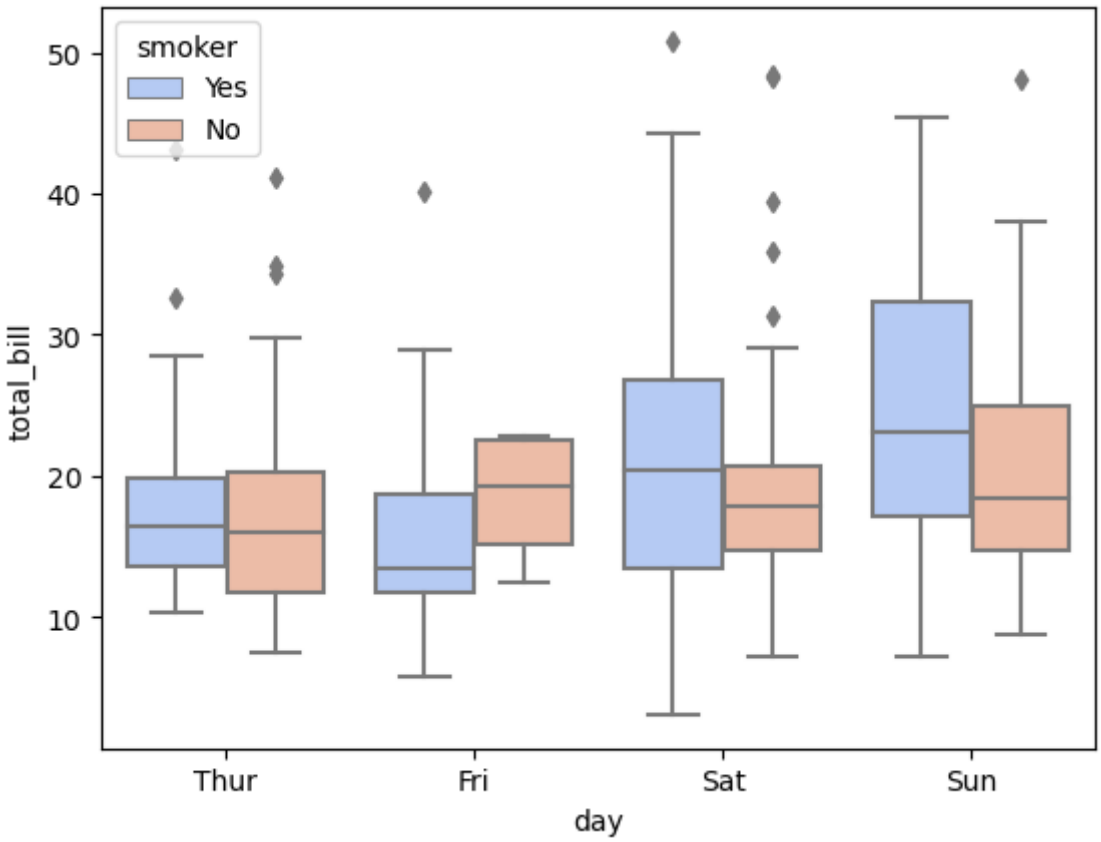

In [18]:
```
# Can do entire dataframe with orient='h'
sns.boxplot(data=tips,palette='rainbow',orient='h')
```

Out[18]:
```
<Axes: >
```




In [19]:
```
sns.boxplot(x="day", y="total_bill", hue="smoker",data=tips, palette="coolwarm'
```

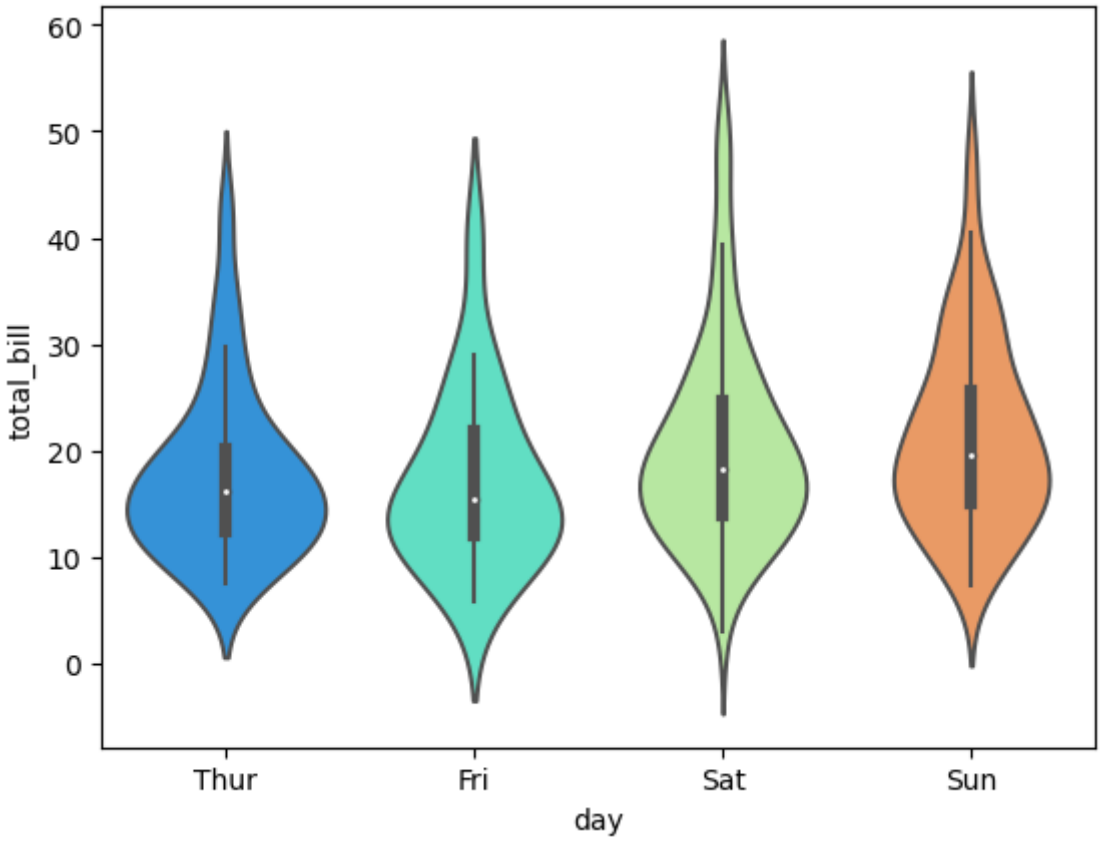
Out[19]: <Axes: xlabel='day', ylabel='total_bill'>




**VIOLINPLOT** A violin plot plays a similar role as a box and whisker plot. It shows the distribution of quantitative data across several levels of one (or more) categorical variables such that those distributions can be compared. Unlike a box plot, in which all of the plot components correspond to actual datapoints, the violin plot features a kernel density estimation of the underlying distribution.

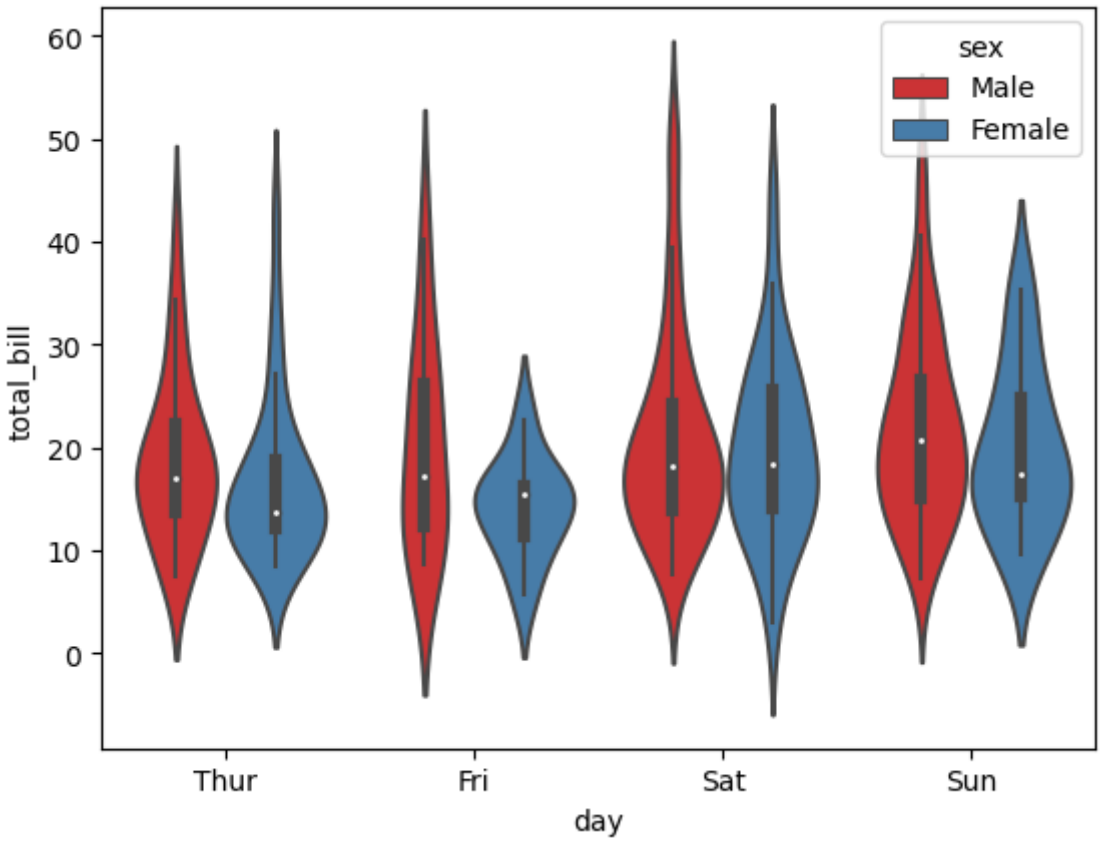
In [20]:
```
sns.violinplot(x="day", y="total_bill", data=tips,palette='rainbow')
```

Out[20]: <Axes: xlabel='day', ylabel='total_bill'>



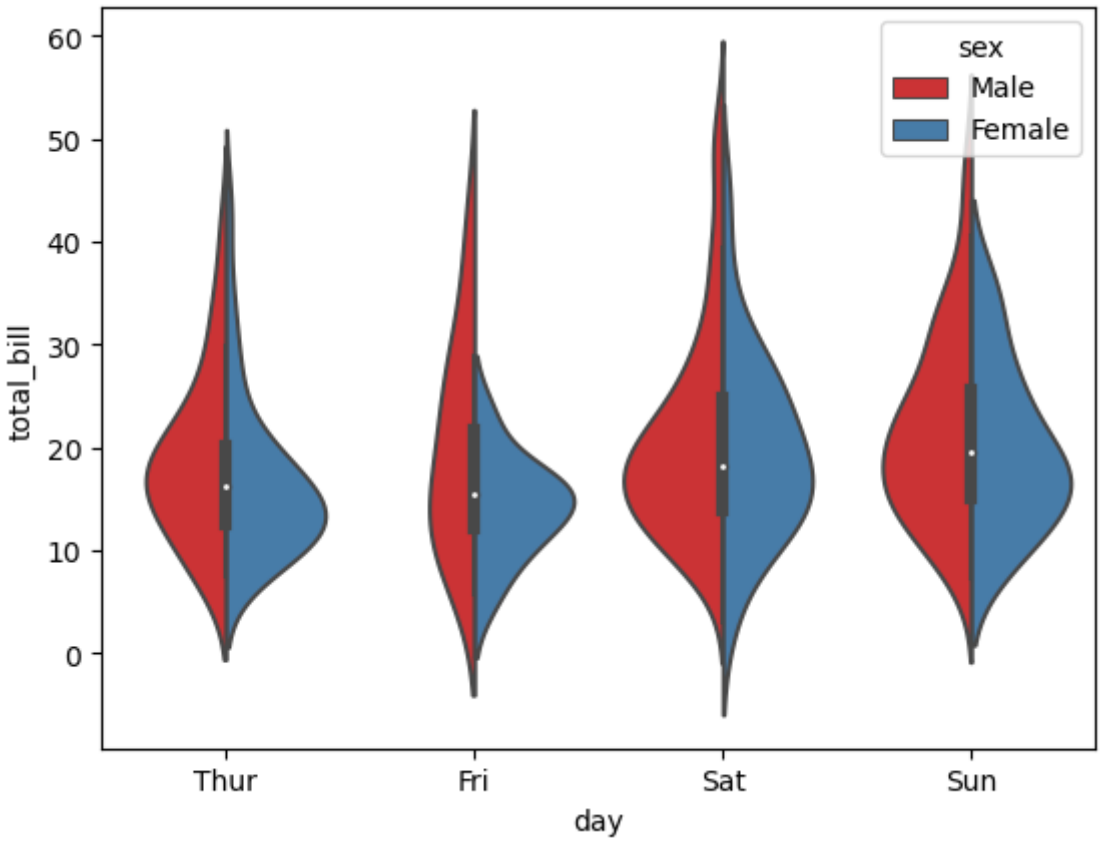

In [21]:
```
sns.violinplot(x="day", y="total_bill", data=tips,hue='sex',palette='Set1')
```

Out[21]:
```
<Axes: xlabel='day', ylabel='total_bill'>
```



In [22]:
```
sns.violinplot(x="day", y="total_bill", data=tips,hue='sex',split=True,palette=
```

Out[22]:
```
<Axes: xlabel='day', ylabel='total_bill'>
```

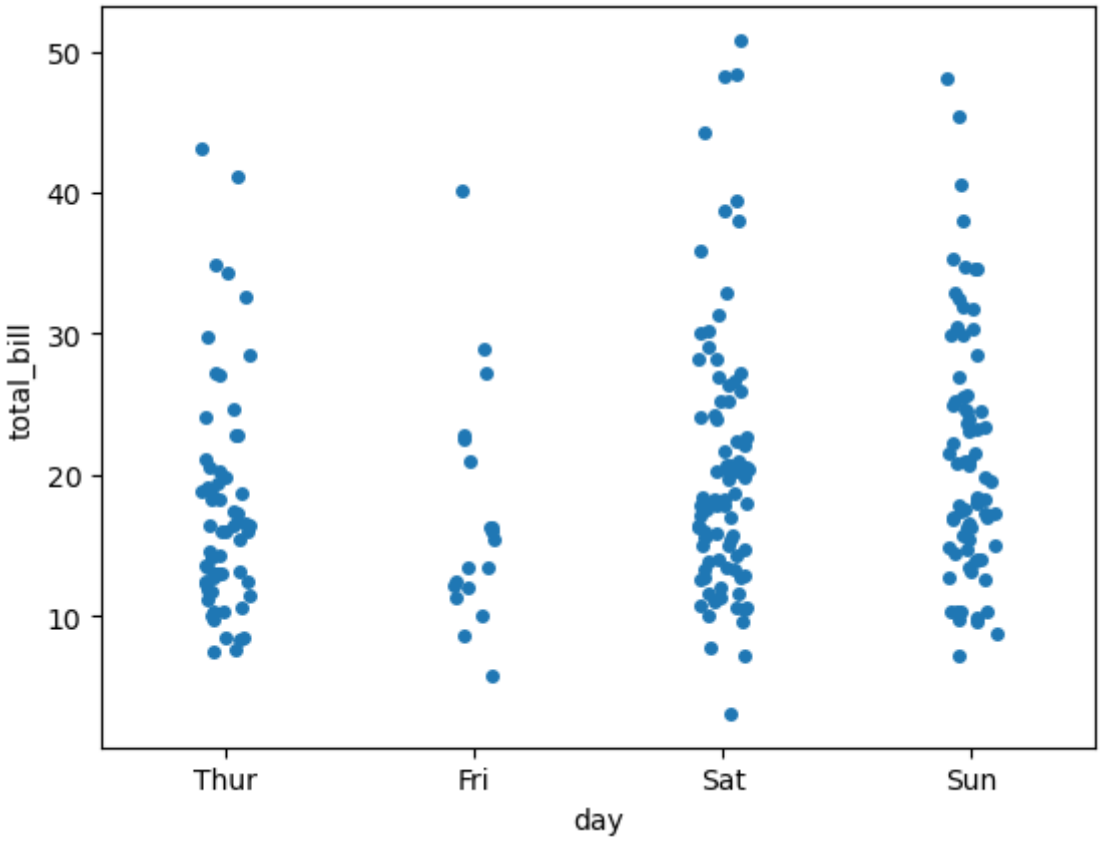



**STRIPPLOT** and **SWARMPLOT** The stripplot will draw a scatterplot where one variable is categorical. A strip plot can be drawn on its own, but it is also a good complement to a box or violin plot in cases where I want to show all observations along with some representation of the underlying distribution.

The swarmplot is similar to stripplot(), but the points are adjusted (only along the categorical axis) so that they don't overlap. This gives a better representation of the distribution of values, although it does not scale as well to large numbers of observations (both in terms of the ability to show all the points and in terms of the computation needed to arrange them).

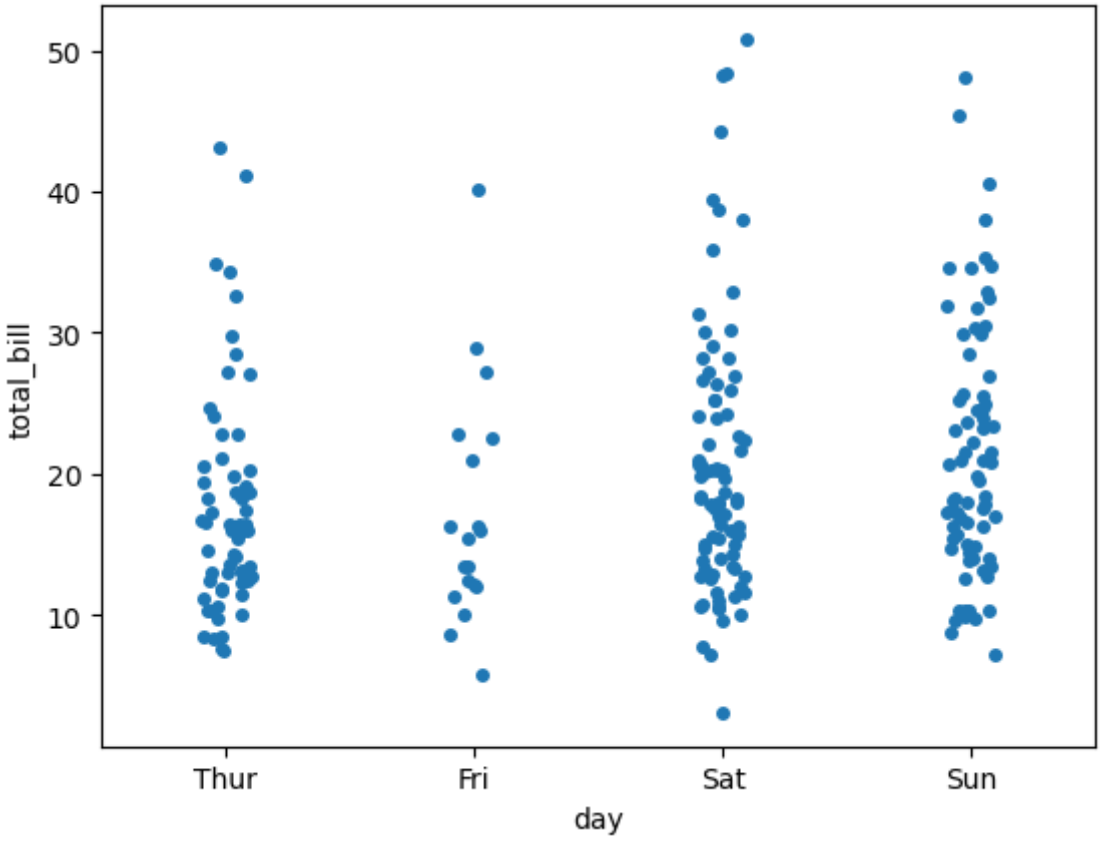
In [23]:
```
sns.stripplot(x="day", y="total_bill", data=tips)
```

Out[23]:
```
<Axes: xlabel='day', ylabel='total_bill'>
```



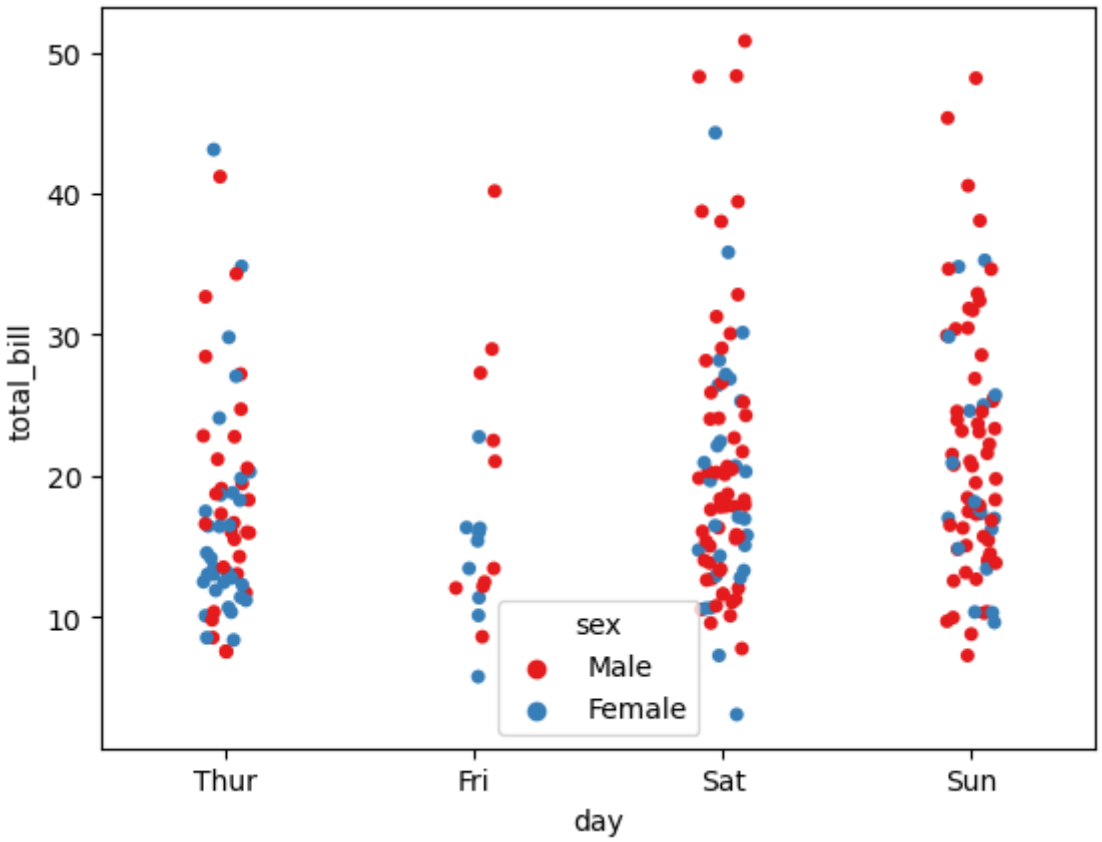

In [24]:
```
sns.stripplot(x="day", y="total_bill", data=tips,jitter=True)
```

Out[24]:
```
<Axes: xlabel='day', ylabel='total_bill'>
```



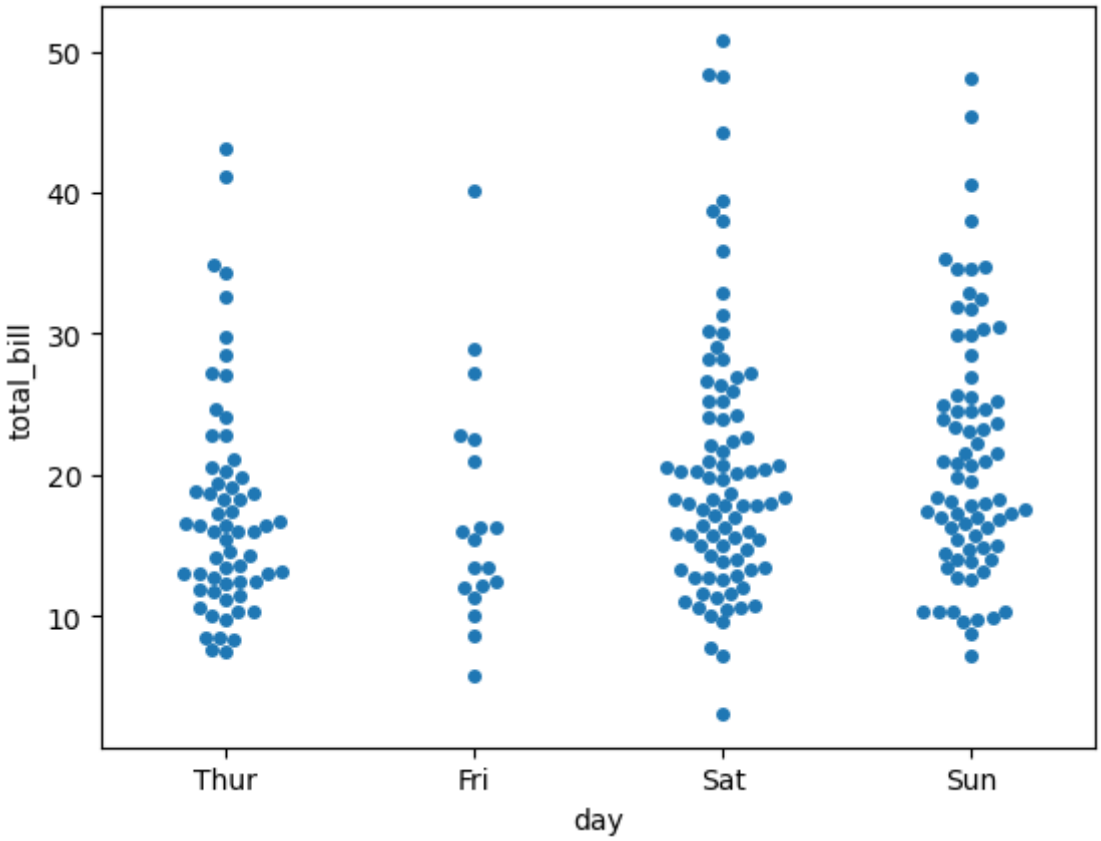
In [25]:
```
sns.stripplot(x="day", y="total_bill", data=tips,jitter=True,hue='sex',palette=
```

Out[25]:
```
<Axes: xlabel='day', ylabel='total_bill'>
```




In [27]:
```
sns.swarmplot(x="day", y="total_bill", data=tips)
```

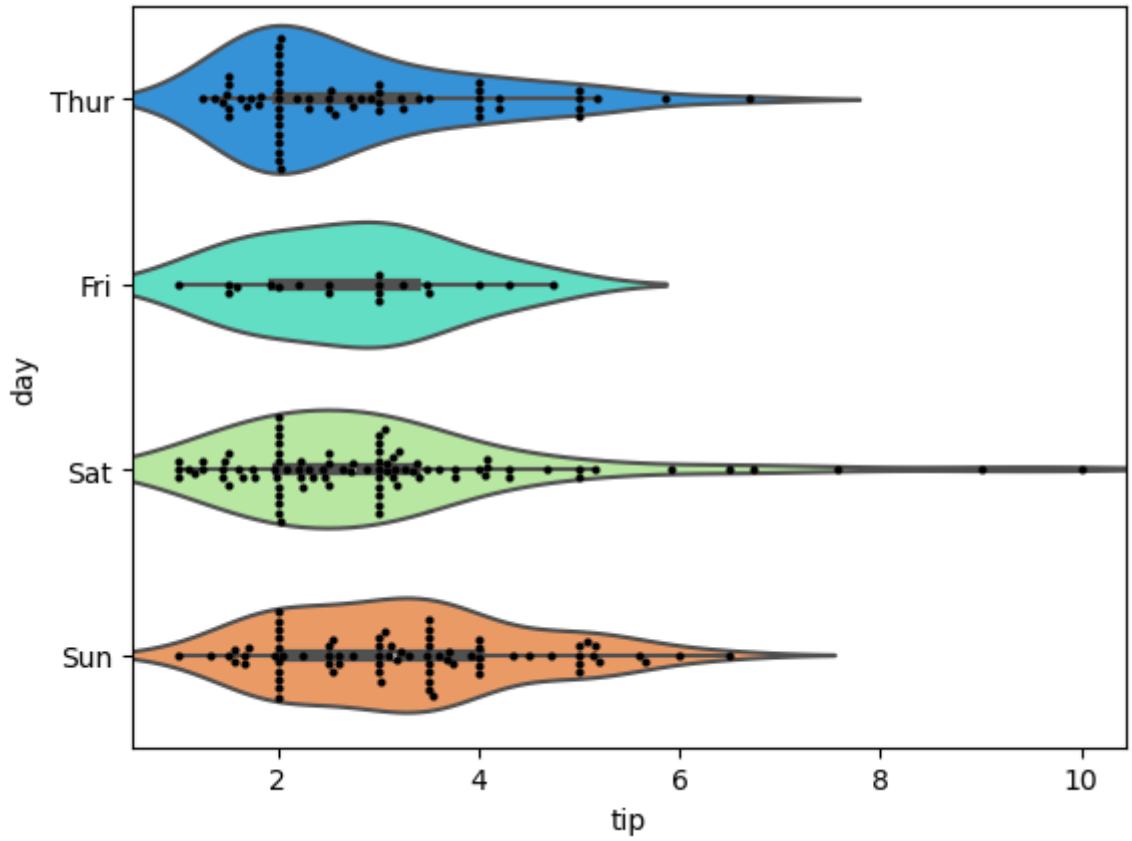
Out[27]:
```
<Axes: xlabel='day', ylabel='total_bill'>
```




**COMBINING CATEGORICAL PLOTS**

In [29]:
```
sns.violinplot(x="tip", y="day", data=tips,palette='rainbow')
sns.swarmplot(x="tip", y="day", data=tips,color='black',size=3)
```

Out[29]:
```
<Axes: xlabel='tip', ylabel='day'>
```




**MATRIX PLOTS** Matrix plots allows to plot data as color-encoded matrices and can also be used to indicate clusters within the data

In [35]:
```
tips.head()
```

Out[35]:

| | 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 |

In [36]:
```
flights.head()
```

Out[36]:

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

**HEATMAP** In order for a heatmap to work properly, the data should already be in a matrix form, the sns.heatmap function basically just colors it in.

In [37]:
```
# Matrix form for correlation data
tips.corr()
```

```
/var/folders/15/bdksz9nj30gfrtmt__dlzxdh0000gn/T/ipykernel_12242/3812684929.p
y:2: FutureWarning: The default value of numeric_only in DataFrame.corr is dep
recated. In a future version, it will default to False. Select only valid colu
mns or specify the value of numeric_only to silence this warning.
  tips.corr()
```

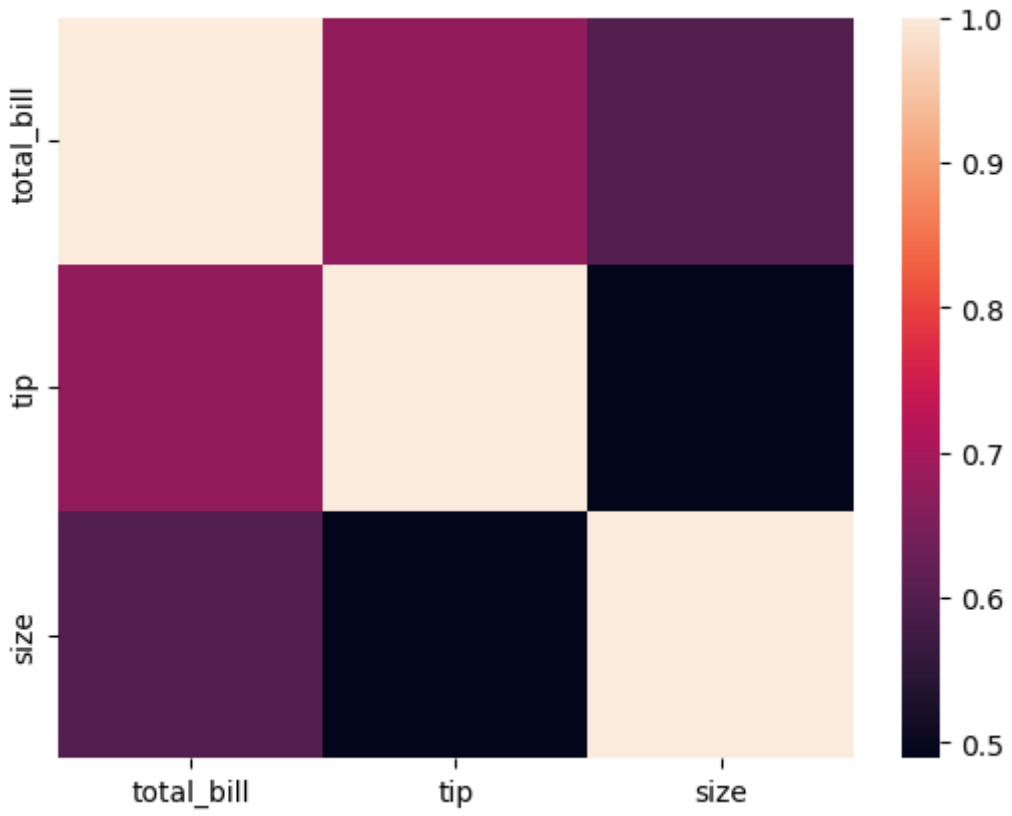
Out[37]:

| | total_bill | tip | size |
|---|---|---|---|
| **total_bill** | 1.000000 | 0.675734 | 0.598315 |
| **tip** | 0.675734 | 1.000000 | 0.489299 |
| **size** | 0.598315 | 0.489299 | 1.000000 |

In [38]:
```
sns.heatmap(tips.corr())
```

```
/var/folders/15/bdksz9nj30gfrtmt__dlzxdh0000gn/T/ipykernel_12242/1579789445.p
y:1: FutureWarning: The default value of numeric_only in DataFrame.corr is dep
recated. In a future version, it will default to False. Select only valid colu
mns or specify the value of numeric_only to silence this warning.
  sns.heatmap(tips.corr())
```

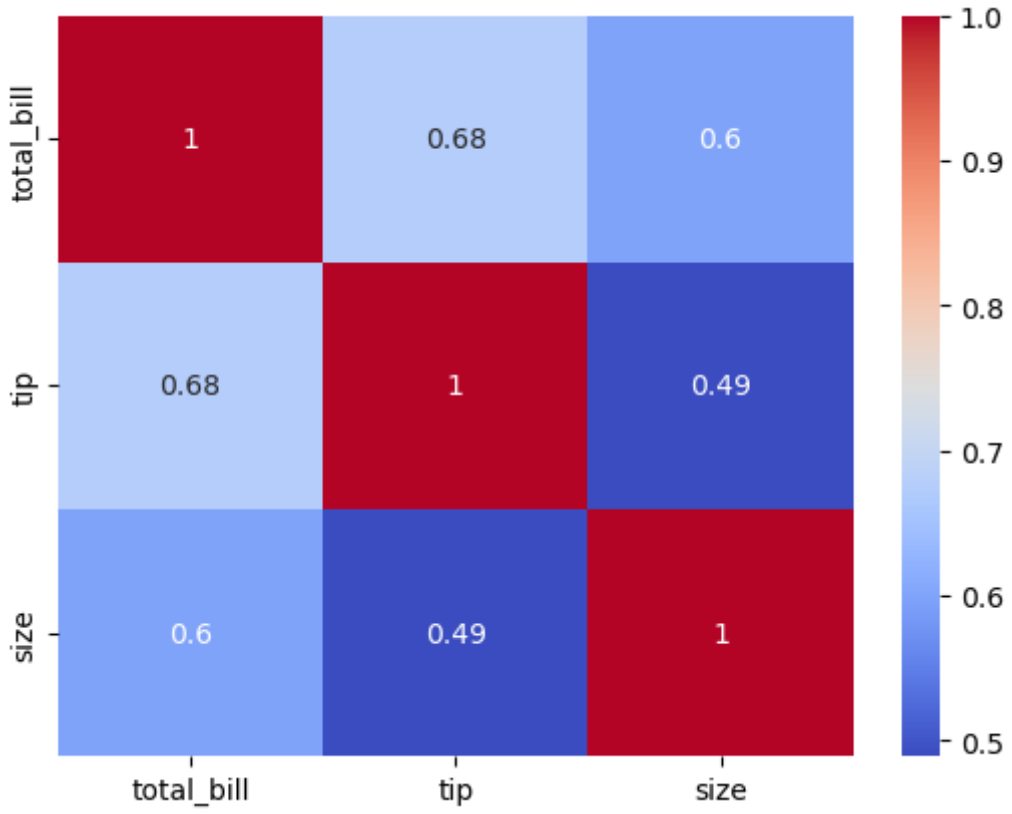
Out[38]:
```
<Axes: >
```




In [39]:
```
sns.heatmap(tips.corr(),cmap='coolwarm',annot=True)
```

```
/var/folders/15/bdksz9nj30gfrtmt__dlzxdh0000gn/T/ipykernel_12242/1408459415.p
y:1: FutureWarning: The default value of numeric_only in DataFrame.corr is dep
recated. In a future version, it will default to False. Select only valid colu
mns or specify the value of numeric_only to silence this warning.
  sns.heatmap(tips.corr(),cmap='coolwarm',annot=True)
```

Out[39]: <Axes: >




Or for the flights data:

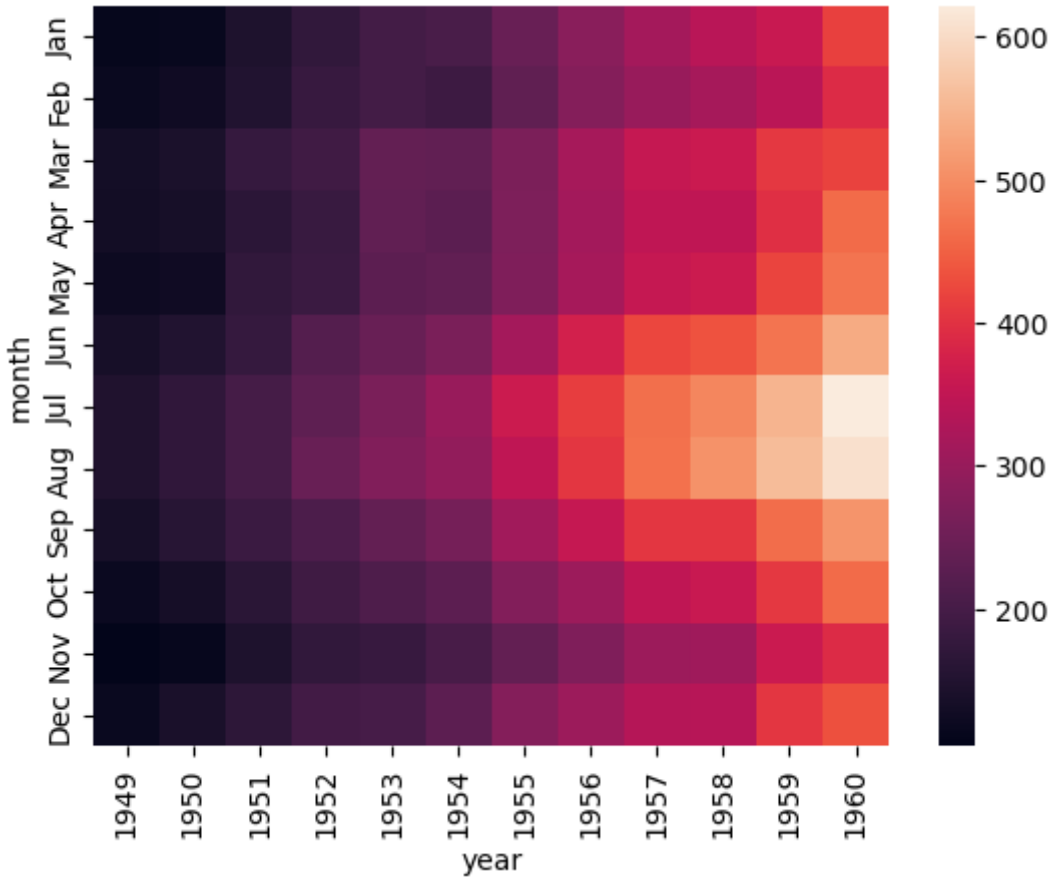
In [40]:
```
flights.pivot_table(values='passengers',index='month',columns='year')
```

Out[40]:

| year | 1949 | 1950 | 1951 | 1952 | 1953 | 1954 | 1955 | 1956 | 1957 | 1958 | 1959 | 1960 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| month | | | | | | | | | | | | |
| **Jan** | 112 | 115 | 145 | 171 | 196 | 204 | 242 | 284 | 315 | 340 | 360 | 417 |
| **Feb** | 118 | 126 | 150 | 180 | 196 | 188 | 233 | 277 | 301 | 318 | 342 | 391 |
| **Mar** | 132 | 141 | 178 | 193 | 236 | 235 | 267 | 317 | 356 | 362 | 406 | 419 |
| **Apr** | 129 | 135 | 163 | 181 | 235 | 227 | 269 | 313 | 348 | 348 | 396 | 461 |
| **May** | 121 | 125 | 172 | 183 | 229 | 234 | 270 | 318 | 355 | 363 | 420 | 472 |
| **Jun** | 135 | 149 | 178 | 218 | 243 | 264 | 315 | 374 | 422 | 435 | 472 | 535 |
| **Jul** | 148 | 170 | 199 | 230 | 264 | 302 | 364 | 413 | 465 | 491 | 548 | 622 |
| **Aug** | 148 | 170 | 199 | 242 | 272 | 293 | 347 | 405 | 467 | 505 | 559 | 606 |
| **Sep** | 136 | 158 | 184 | 209 | 237 | 259 | 312 | 355 | 404 | 404 | 463 | 508 |
| **Oct** | 119 | 133 | 162 | 191 | 211 | 229 | 274 | 306 | 347 | 359 | 407 | 461 |
| **Nov** | 104 | 114 | 146 | 172 | 180 | 203 | 237 | 271 | 305 | 310 | 362 | 390 |
| **Dec** | 118 | 140 | 166 | 194 | 201 | 229 | 278 | 306 | 336 | 337 | 405 | 432 |

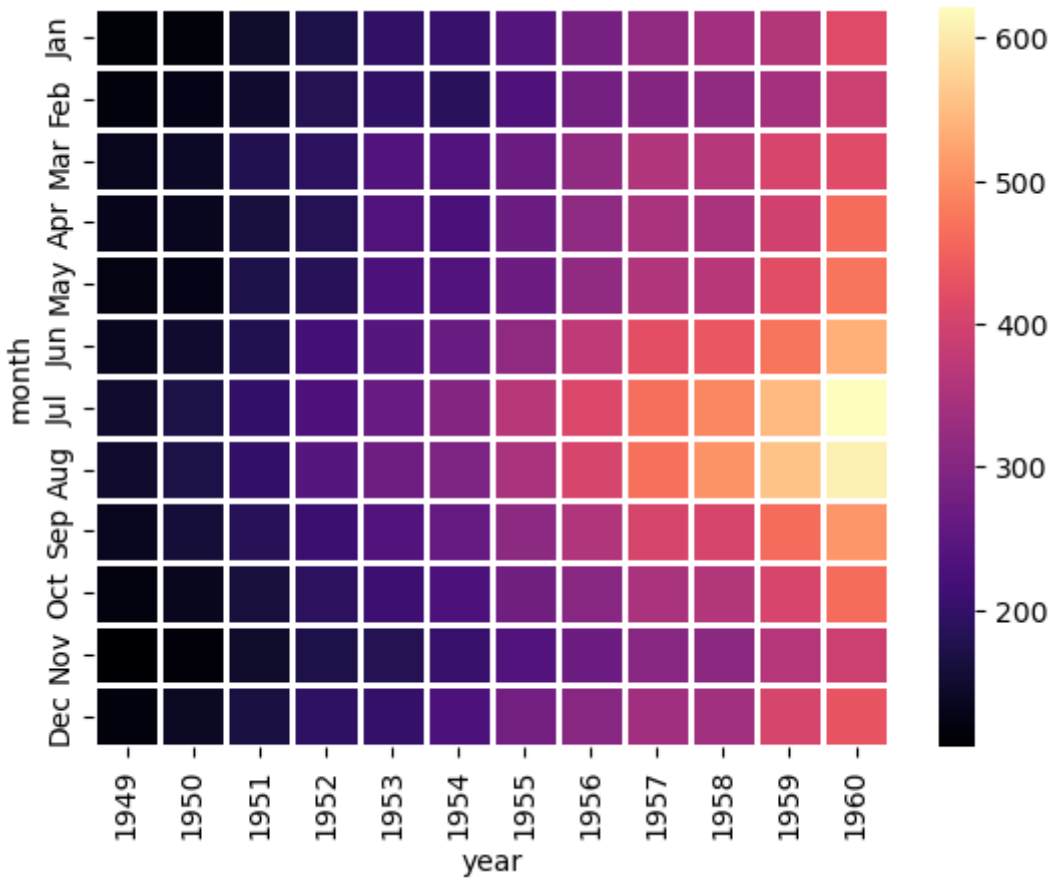
In [41]:
```
pvflights = flights.pivot_table(values='passengers',index='month',columns='year
sns.heatmap(pvflights)
```

Out[41]:
```
<Axes: xlabel='year', ylabel='month'>
```




In [42]:
```
sns.heatmap(pvflights,cmap='magma',linecolor='white',linewidths=1)
```

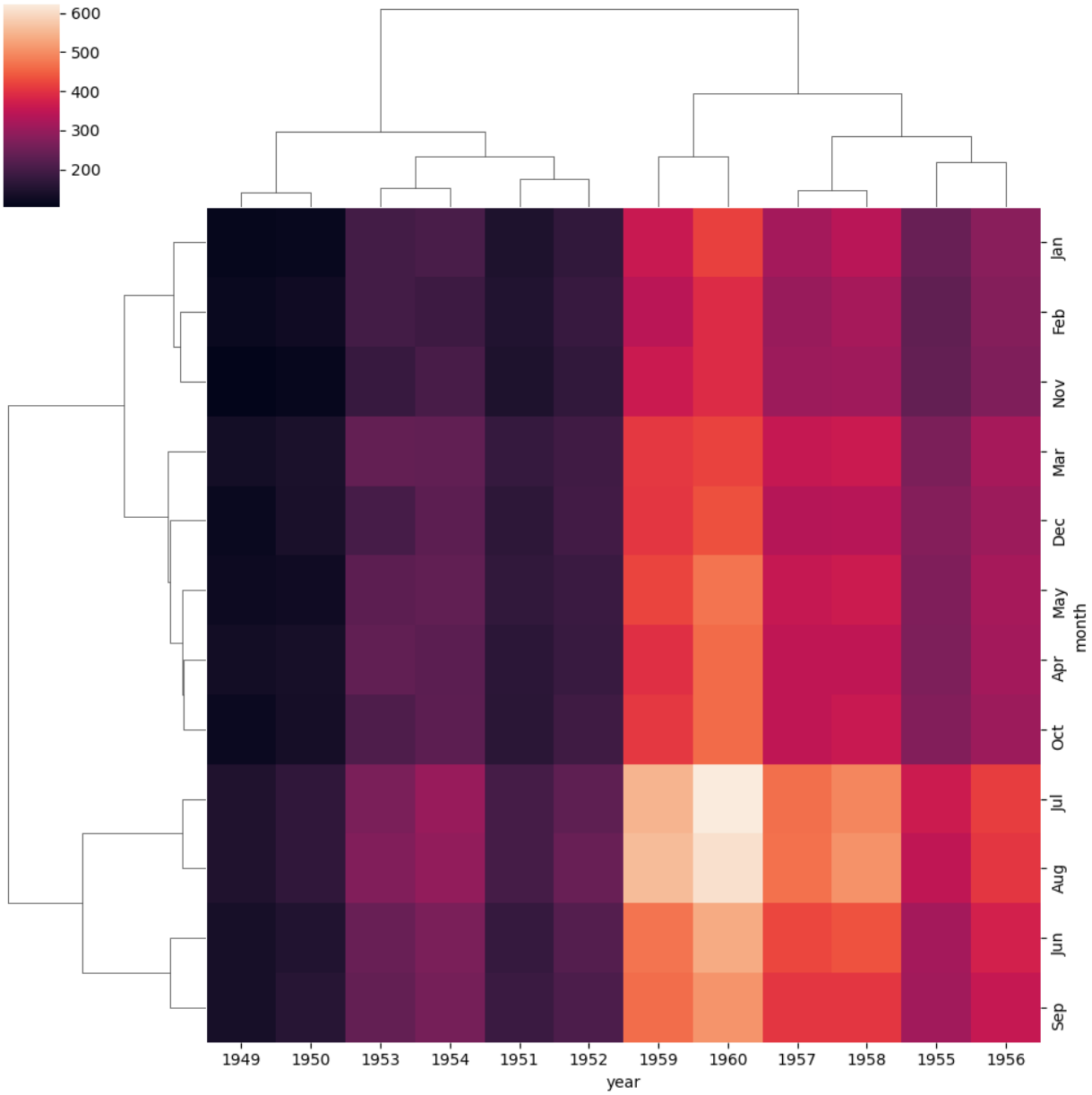
Out[42]: <Axes: xlabel='year', ylabel='month'>




**CLUSTERMAP** The clustermap uses hierarchal clustering to produce a clustered version of the heatmap. For example:

In [43]:
```
sns.clustermap(pvflights)
```

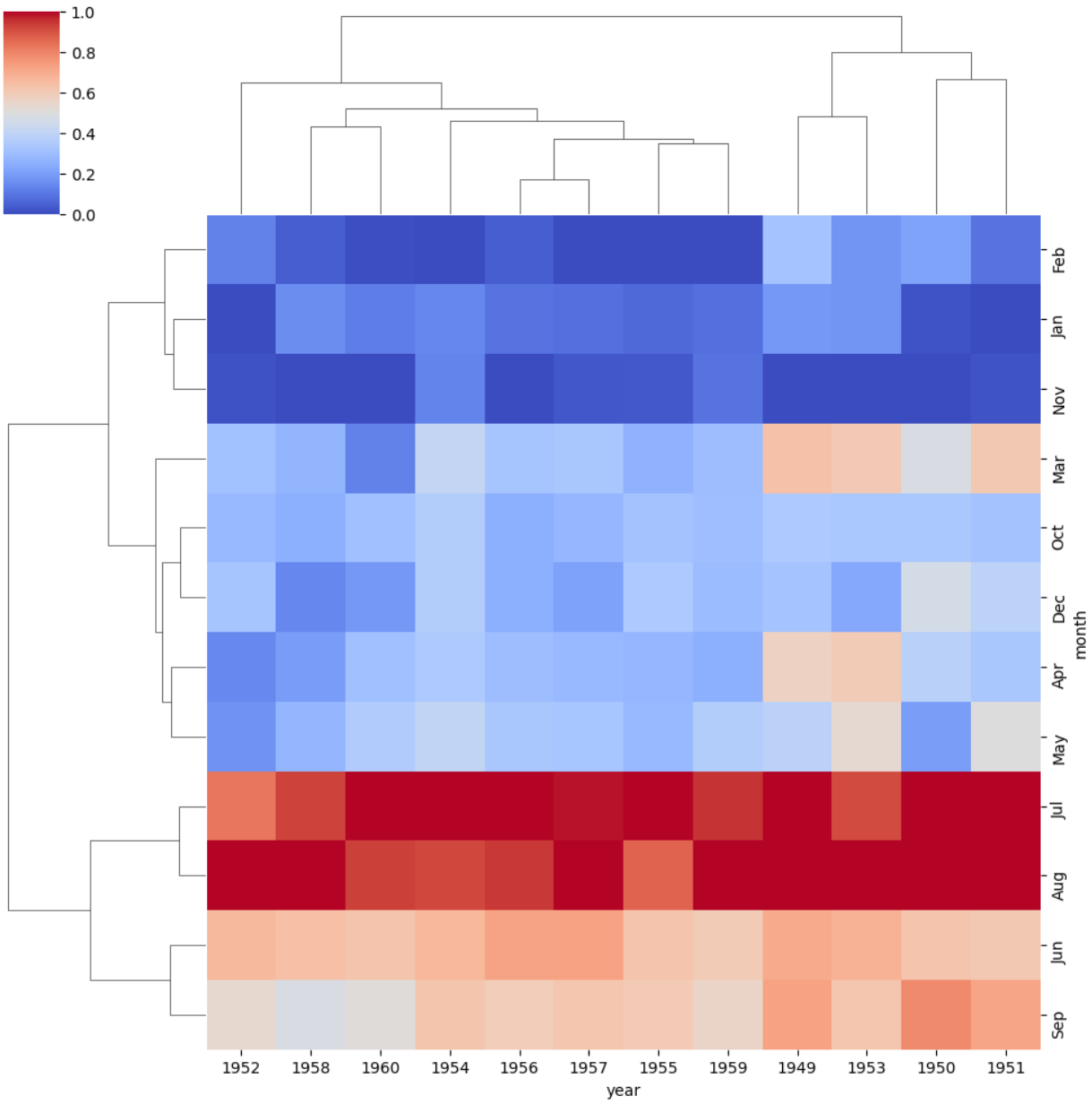
Out[43]: <seaborn.matrix.ClusterGrid at 0x14690a4a0>




Notice now how the years and months are no longer in order, instead they are grouped by similarity in value (passenger count). That means I can begin to infer things from this plot, such as August and July being similar.

In [44]:
```
# More options to get the information a little clearer like normalization
sns.clustermap(pvflights,cmap='coolwarm',standard_scale=1)
```

Out[44]:
```
<seaborn.matrix.ClusterGrid at 0x146aaf340>
```




**GRIDS** Grids are general types of plots that allows to map plot types to rows and columns of a grid, this helps to create similar plots separated by features.

In [47]:
```
iris.head()
```

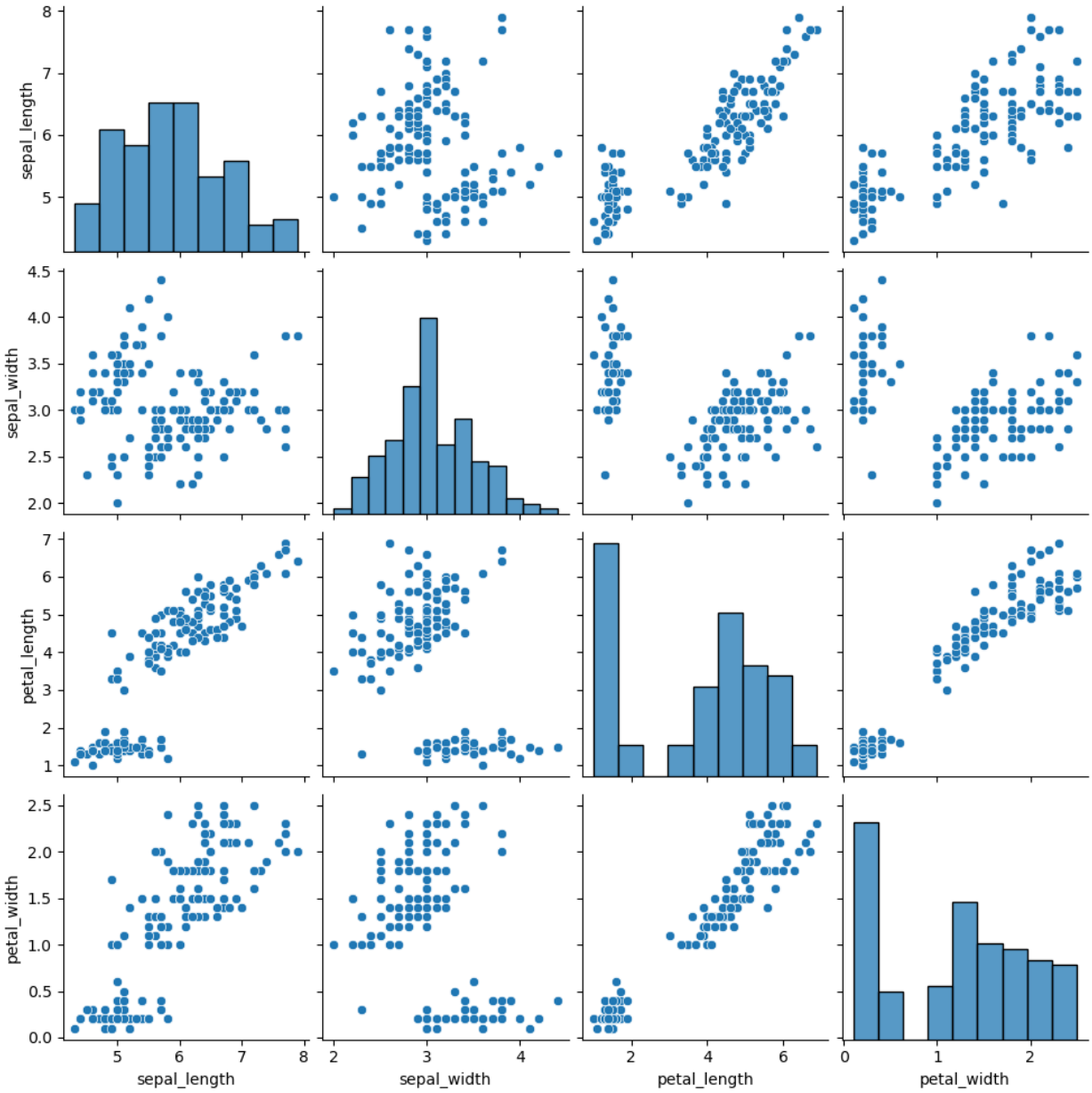
Out[47]:

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | setosa |

**PAIRGRID** Pairgrid is a subplot grid for plotting pairwise relationships in a dataset.

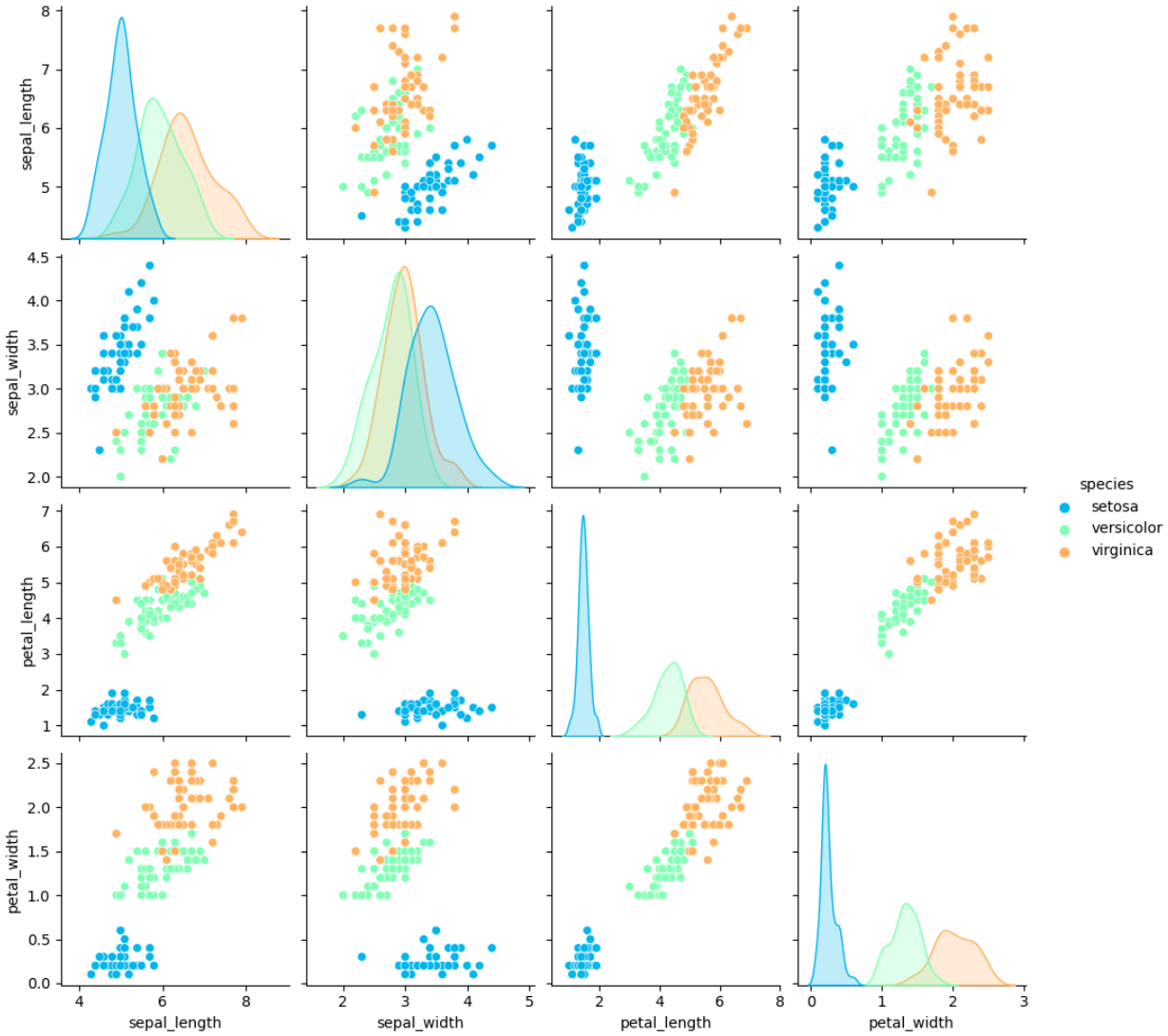
In [52]:
```
sns.pairplot(iris)
```

Out[52]:
```
<seaborn.axisgrid.PairGrid at 0x1463d7c70>
```




In [53]:
```
sns.pairplot(iris,hue='species',palette='rainbow')
```

Out[53]:
```
<seaborn.axisgrid.PairGrid at 0x1515b1960>
```

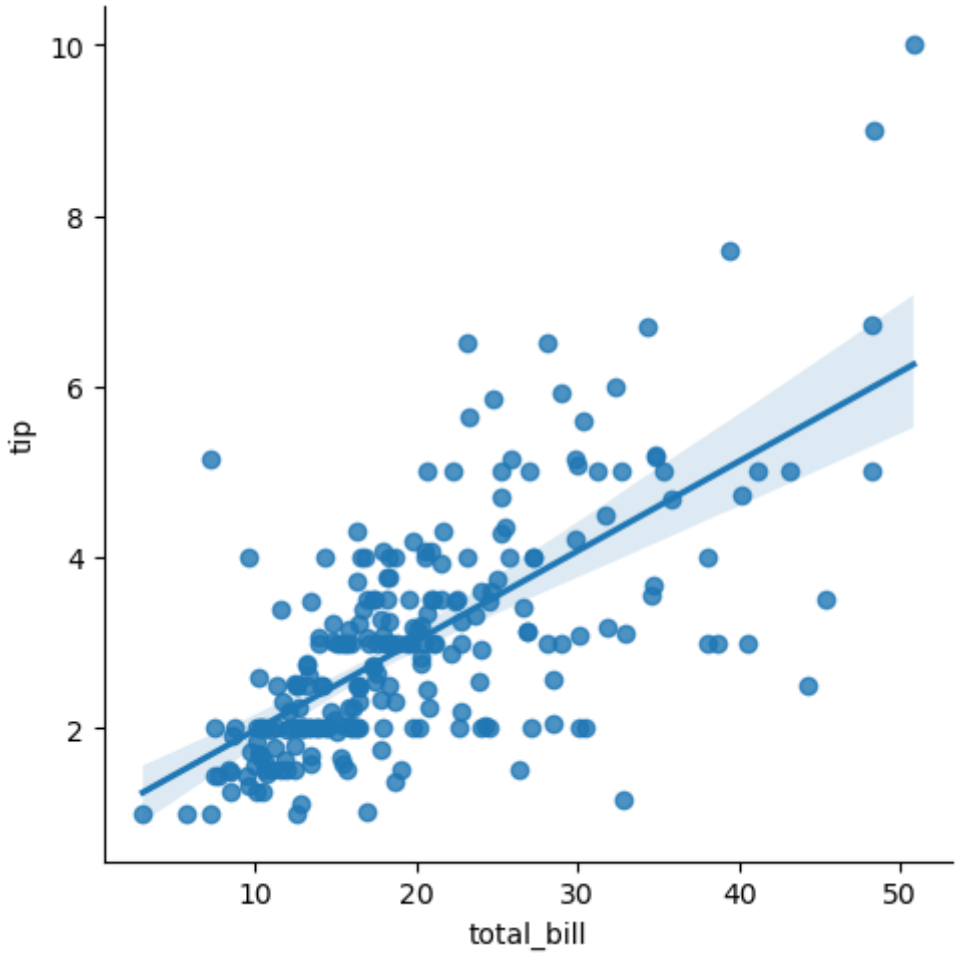



**REGRESSION PLOTS** lmplot allows me to display linear models, but it also conveniently allows me to split up those plots based off of features, as well as coloring the hue based off of features.

Let's explore how this works:

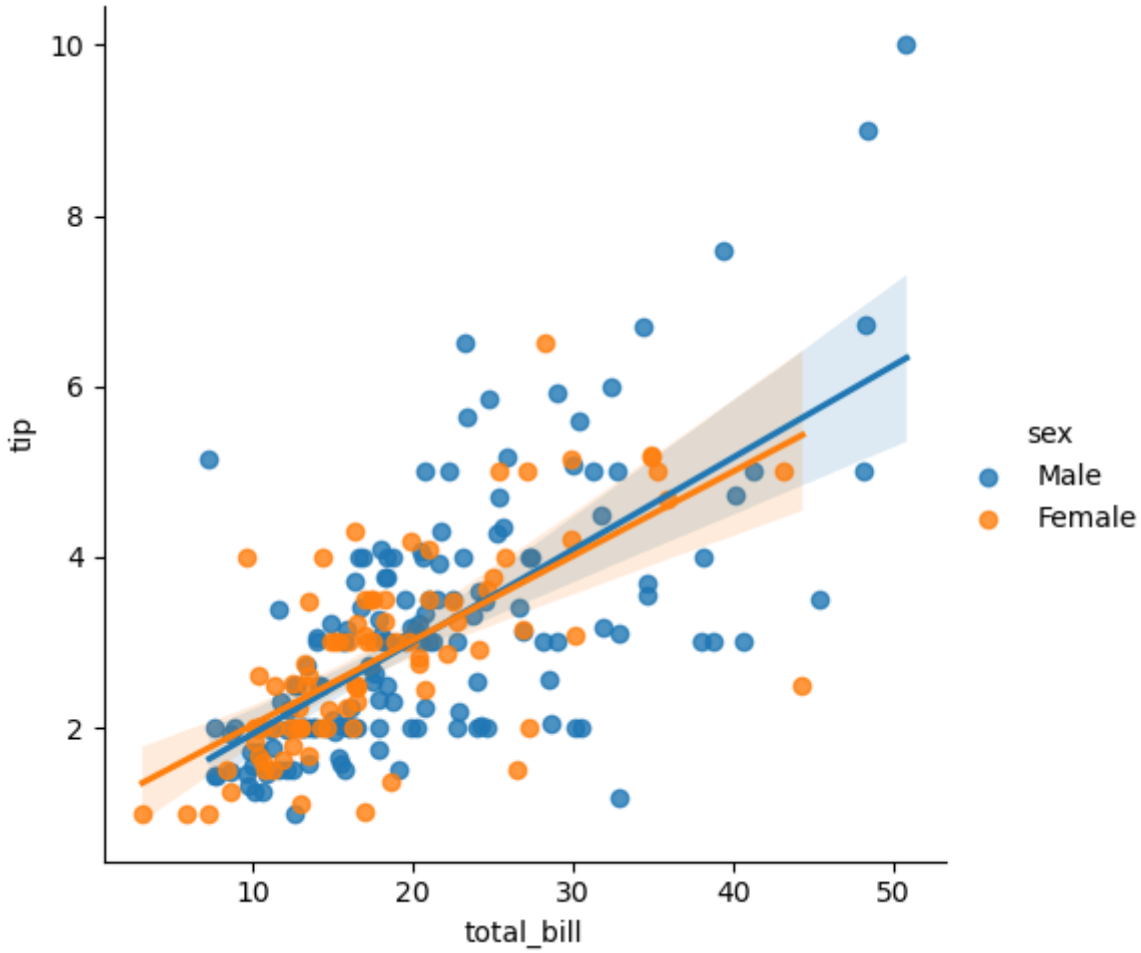
In [58]:
```
sns.lmplot(x='total_bill',y='tip',data=tips)
```

Out[58]:
```
<seaborn.axisgrid.FacetGrid at 0x15215fe20>
```



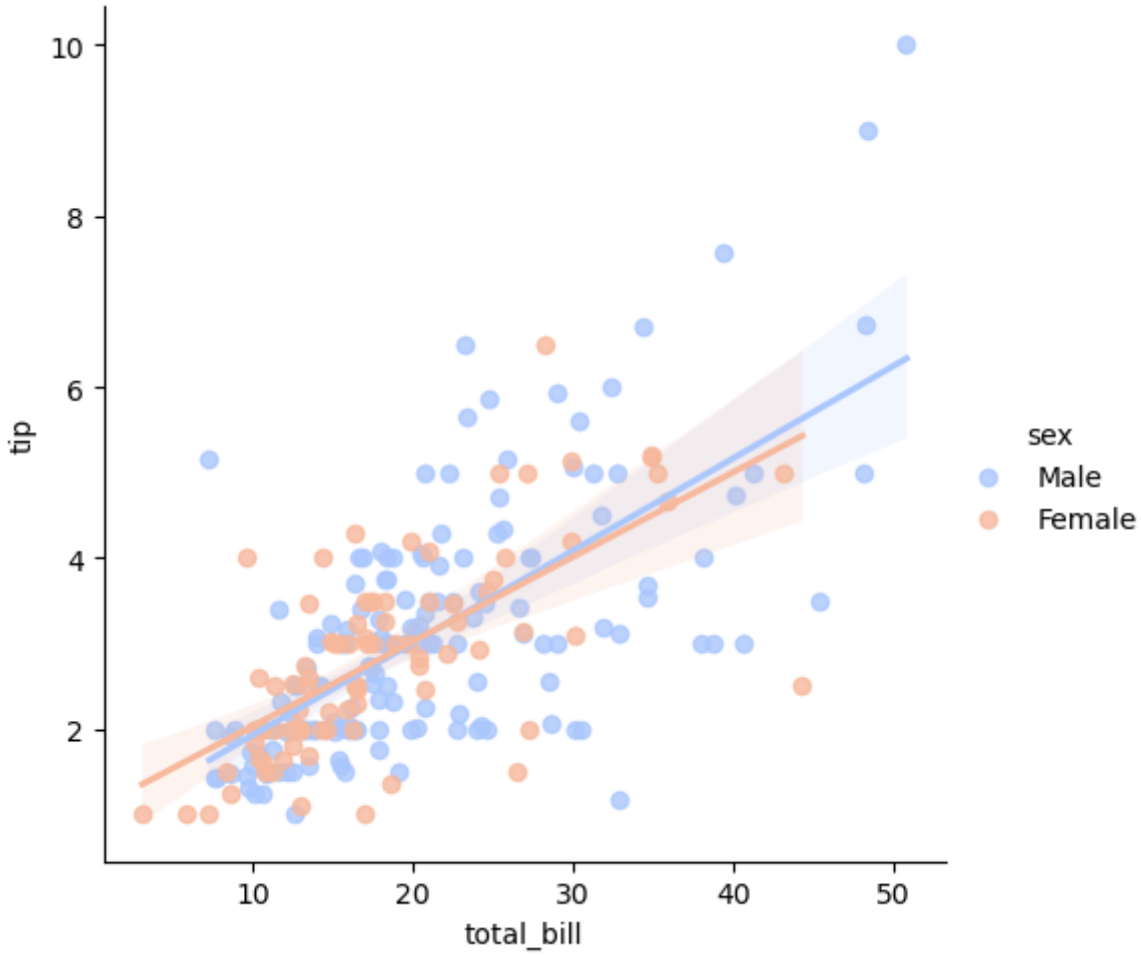

In [59]:
```
sns.lmplot(x='total_bill',y='tip',data=tips,hue='sex')
```

Out[59]:
```
<seaborn.axisgrid.FacetGrid at 0x152446cb0>
```




In [60]:
```
sns.lmplot(x='total_bill',y='tip',data=tips,hue='sex',palette='coolwarm')
```

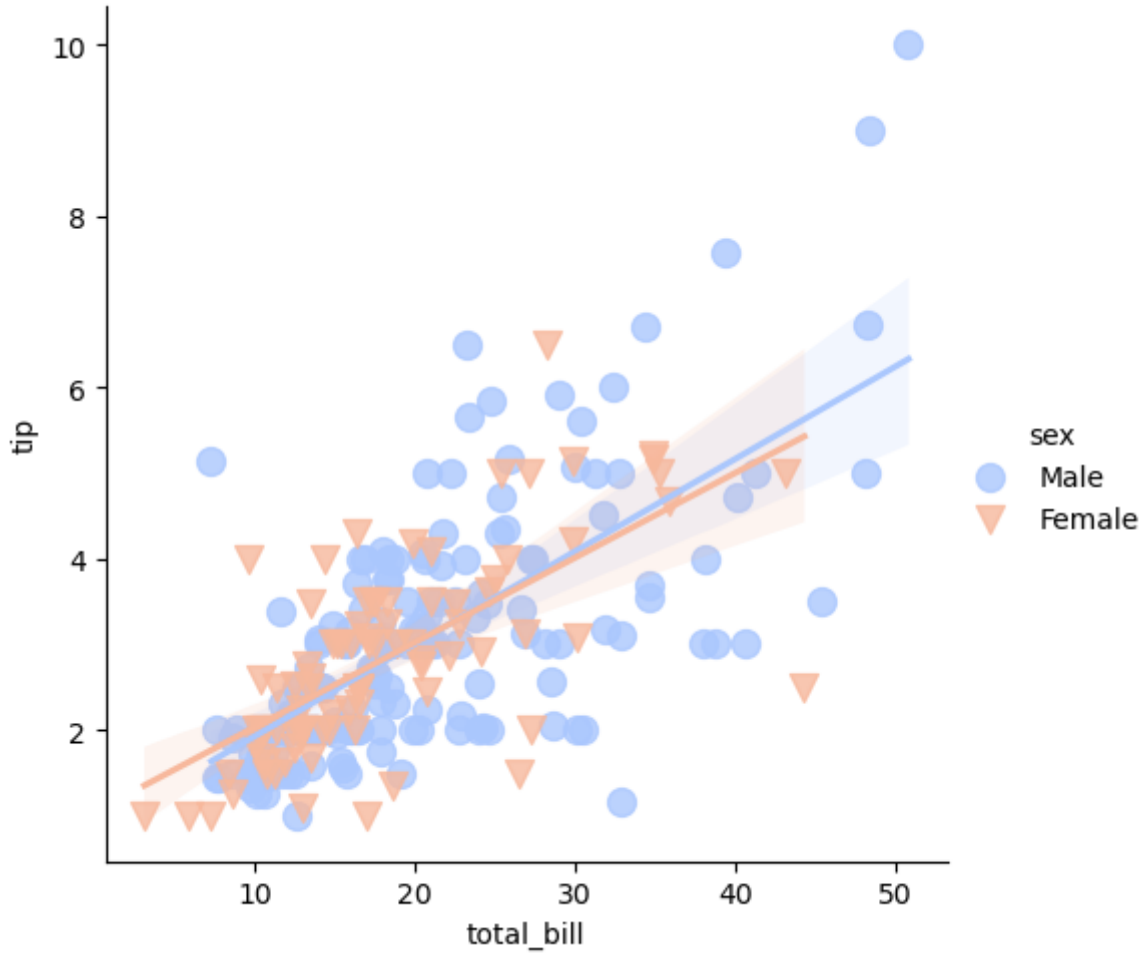
Out[60]:
```
<seaborn.axisgrid.FacetGrid at 0x1524e4c70>
```




**WORKING WITH MARKERS**

In [61]:
```
sns.lmplot(x='total_bill',y='tip',data=tips,hue='sex',palette='coolwarm',
           markers=['o','v'],scatter_kws={'s':100})
```

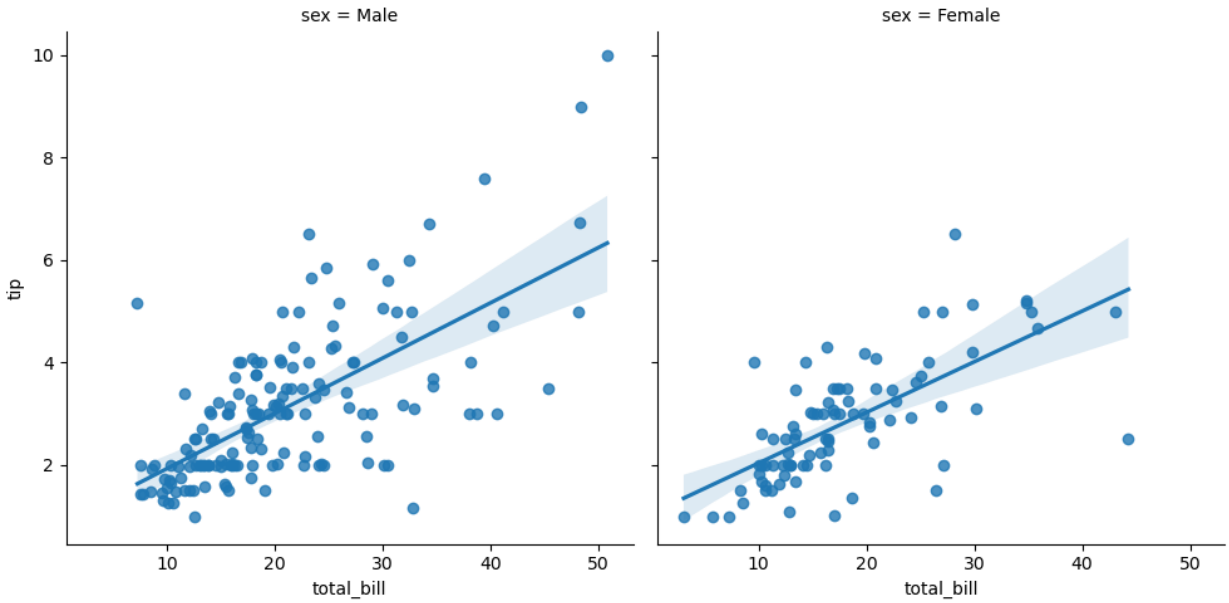
Out[61]:
```
<seaborn.axisgrid.FacetGrid at 0x152589420>
```




**USING A GRID** I can add more variable separation through columns and rows with the use of a grid.

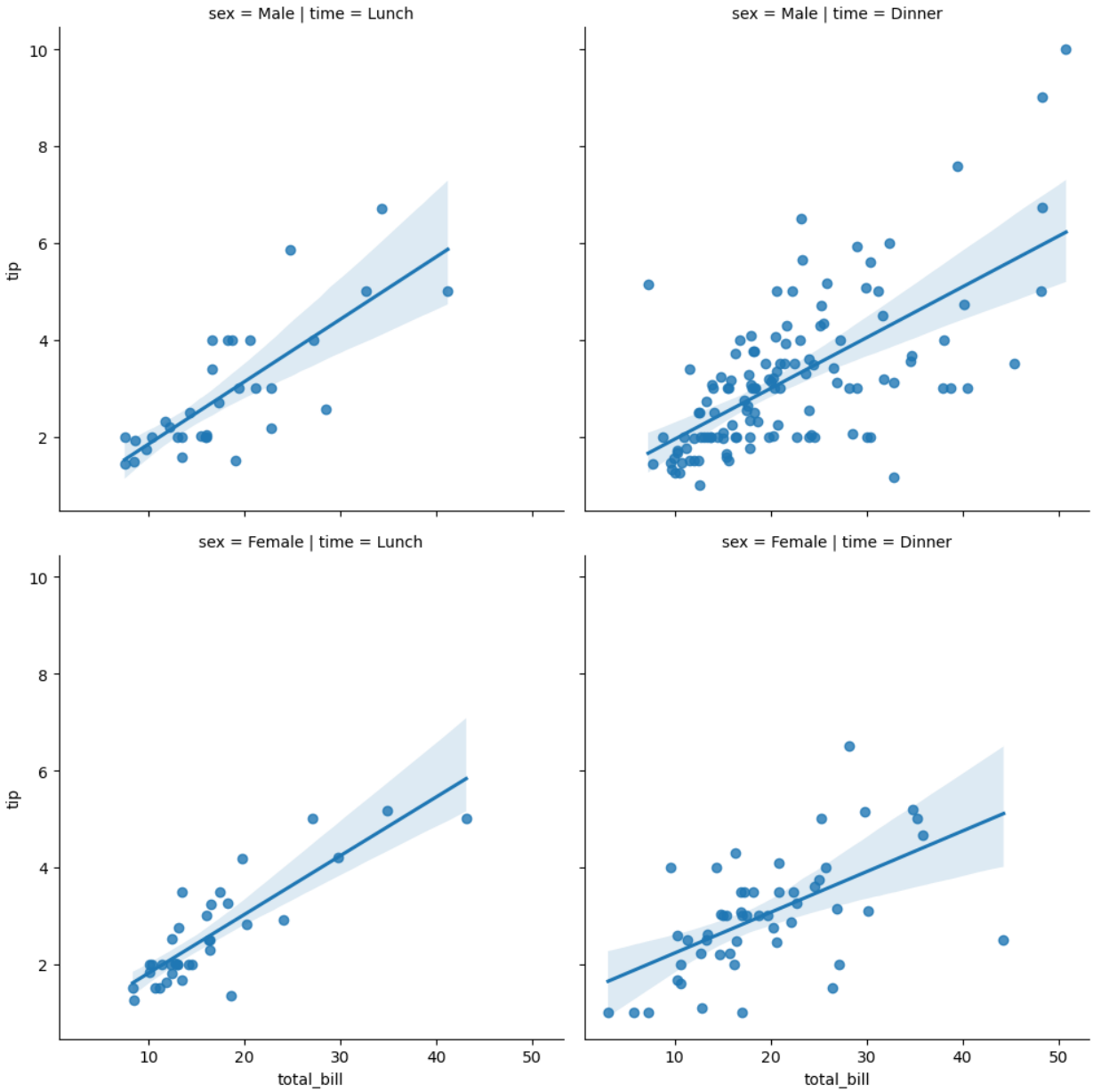
In [62]:
```
sns.lmplot(x='total_bill',y='tip',data=tips,col='sex')
```

Out[62]:
```
<seaborn.axisgrid.FacetGrid at 0x1525fe8c0>
```




In [63]:
```
sns.lmplot(x="total_bill", y="tip", row="sex", col="time",data=tips)
```

Out[63]: <seaborn.axisgrid.FacetGrid at 0x152417c40>




In [64]:
```
sns.lmplot(x='total_bill',y='tip',data=tips,col='day',hue='sex',palette='coolwa
```

Out[64]: <seaborn.axisgrid.FacetGrid at 0x152031450>