Basic Matplotlib

In [1]:

```
import matplotlib.pyplot as plt
```

In [2]:

```
%matplotlib inline
```

In [3]:

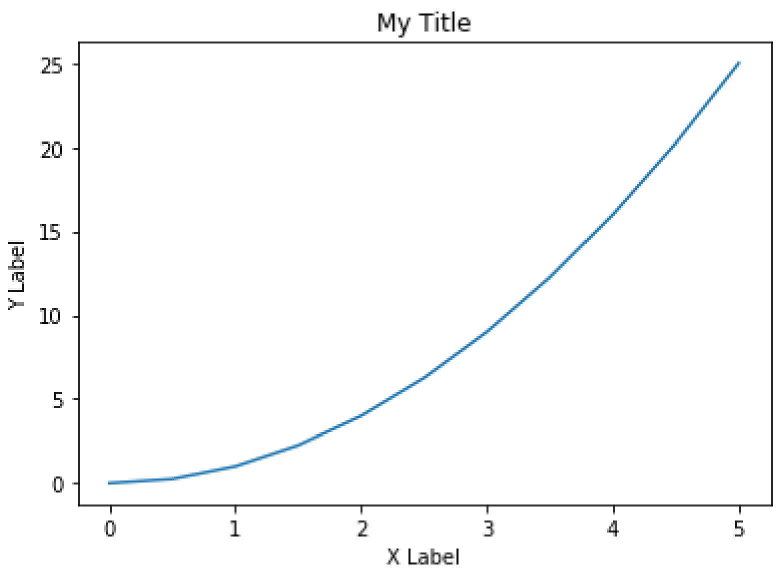
```
import numpy as np

x = np.linspace(0,5,11)
y = x**2
```

In [4]:

```
# Object Oriented Method

fig = plt.figure()

# Set Axes : [left,bottom,width,height]

axes = fig.add_axes([0.1,0.1,0.8,0.8])

# plotting the data

axes.plot(x,y)

# Setting label and title

axes.set_xlabel('X Label')
axes.set_ylabel('Y Label')
axes.set_title('My Title')
```

Out[4]:

```
Text(0.5, 1.0, 'My Title')
```

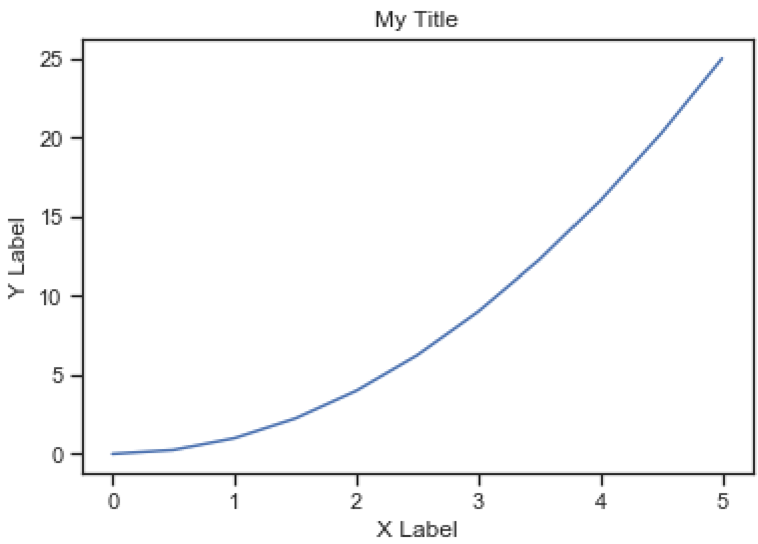


In [273]:

```
# Functional Method

plt.plot(x,y)
plt.xlabel('X Label')
plt.ylabel('Y Label')
plt.title('My Title')
```

Out[273]:

```
Text(0.5, 1.0, 'My Title')
```

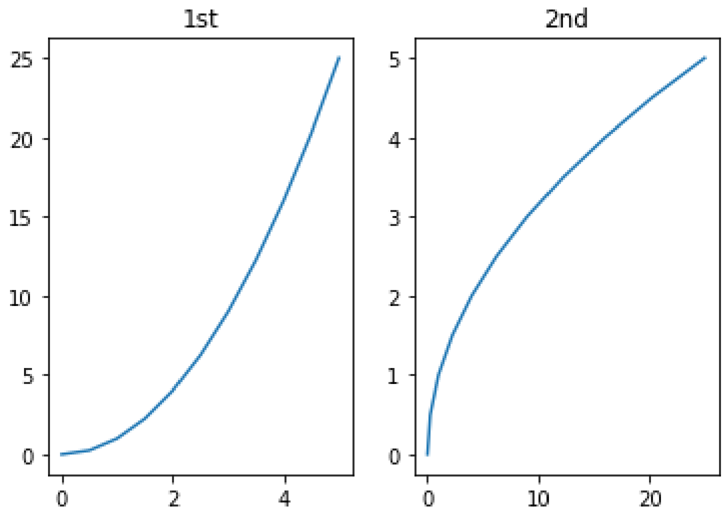


In [5]:

```
# Adding Subplots

fig_subplot,axes1 = plt.subplots(nrows = 1 , ncols=2)

axes1[0].plot(x,y)
axes1[0].set_title('1st')

axes1[1].plot(y,x)
axes1[1].set_title('2nd')
```

Out[5]:

```
Text(0.5, 1.0, '2nd')
```



Special Plot Types

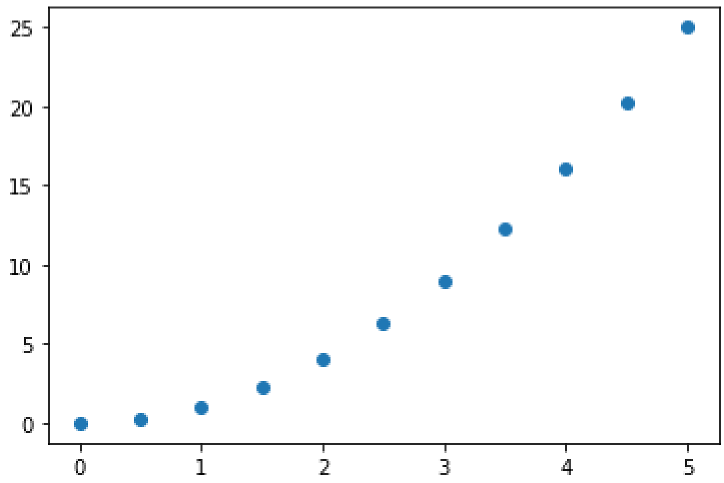
Scatter Plot

In [6]:

```
plt.scatter(x,y)
```

Out[6]:

```
<matplotlib.collections.PathCollection at 0x1cf60efbe08>
```



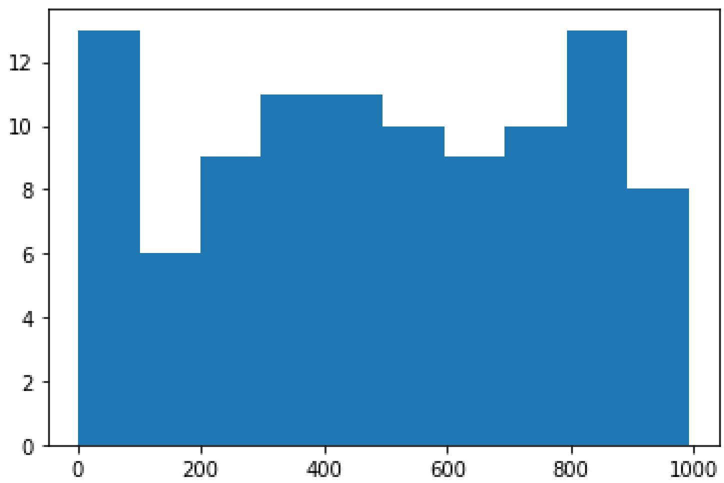
Histogram

In [7]:

```
sample_data = np.random.randint(1,1000,100)
plt.hist(sample_data)
```

Out[7]:

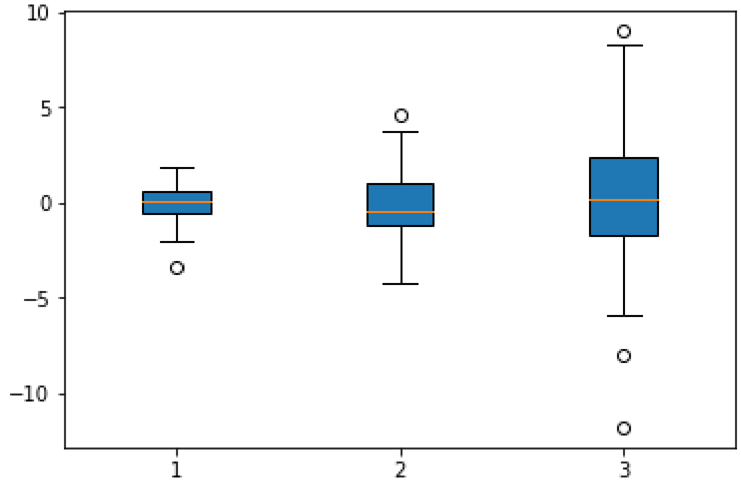
```
(array([13.,  6.,  9., 11., 11., 10.,  9., 10., 13.,  8.]),
 array([  2. , 100.9, 199.8, 298.7, 397.6, 496.5, 595.4, 694.3, 793.2,
        892.1, 991. ]),
 <a list of 10 Patch objects>)
```



Rectangular Box Plot

In [9]:

```
boxplt_data = [ np.random.normal(0,std,100) for std in range(1,4)]

plt.boxplot(boxplt_data,vert=True,patch_artist=True);
```



Seaborn

Distribution Plots

Here are some plots that allow us to visualize the distribution of a data set.

1. distplot
2. jointplot
3. pairplot
4. rugplot

In [10]:

```
import seaborn as sns
%matplotlib inline
```

In [11]:

```
# Seaborn comes with built-in data sets

tips = sns.load_dataset('tips')
tips.head()
```

Out[11]:

	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 [12]:

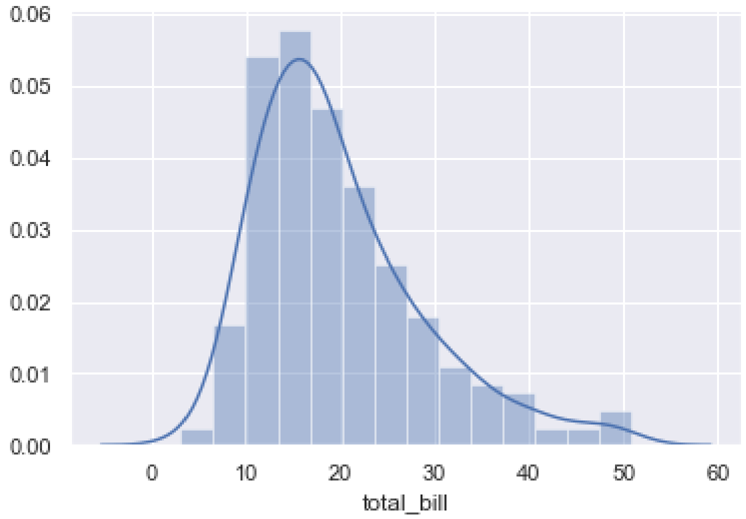
```
sns.set()
```

In [13]:

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

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf63f2e308>
```

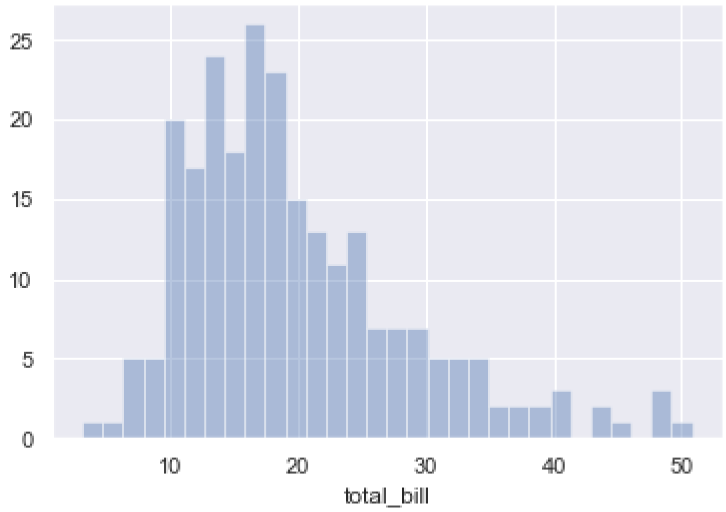


In [14]:

```
# Lets remove the kde layer and specify the no of bins

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

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf63f2ef48>
```



jointplot

jointplot() allows you to basically match up two distplots for bivariate data. With choice of what **kind** parameter to compare with

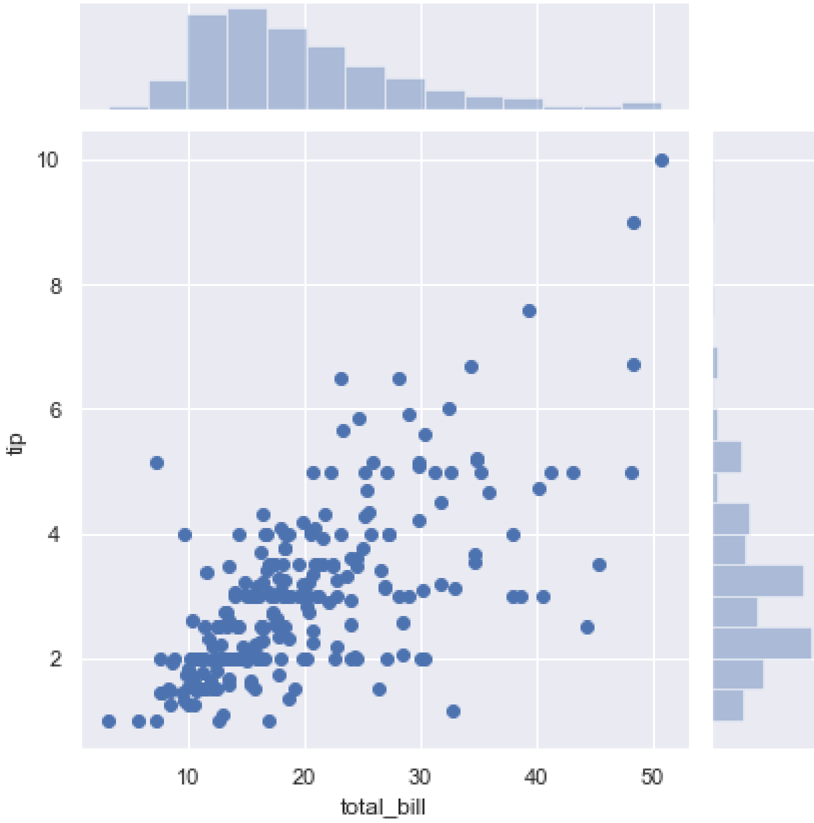
- scatter
- reg
- resid
- kde
- hex

In [15]:

```
sns.jointplot( x='total_bill', y ='tip', data=tips, kind='scatter' )
```

Out[15]:

```
<seaborn.axisgrid.JointGrid at 0x1cf6408b7c8>
```

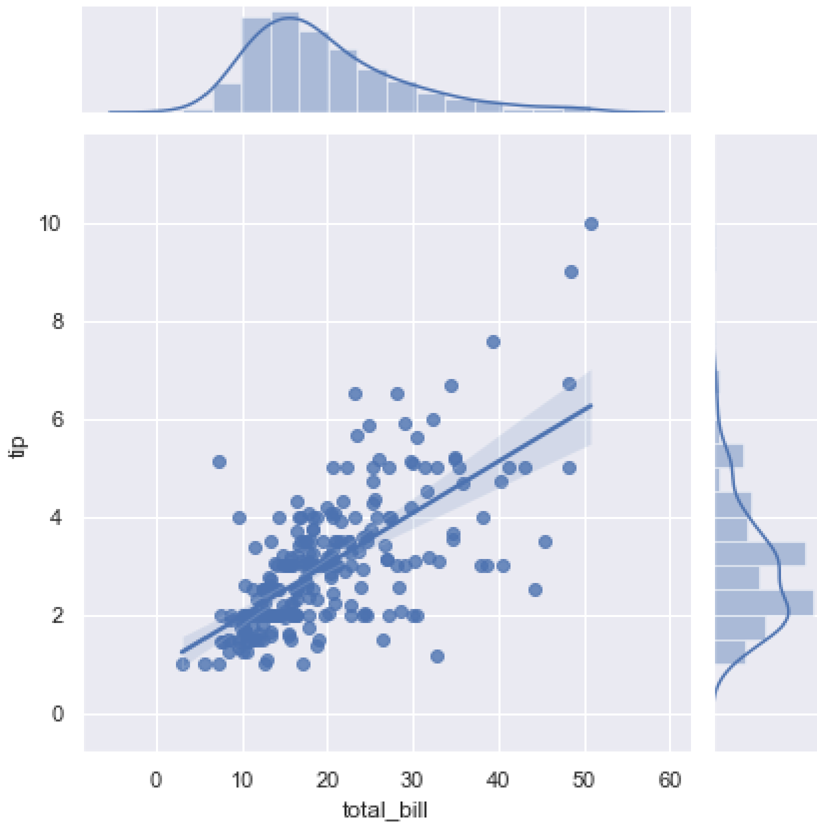


In [16]:

```
sns.jointplot( x='total_bill', y='tip', data=tips, kind='reg')
```

Out[16]:

```
<seaborn.axisgrid.JointGrid at 0x1cf641abd88>
```

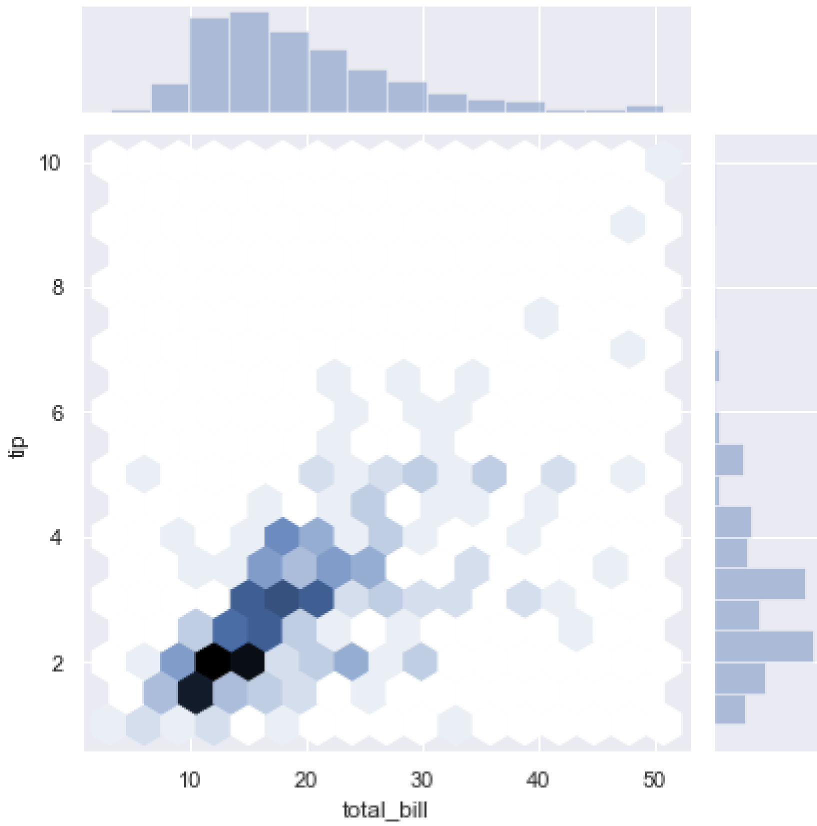


In [17]:

```
sns.jointplot( x='total_bill', y='tip', data=tips, kind='hex')
```

Out[17]:

```
<seaborn.axisgrid.JointGrid at 0x1cf642c3788>
```

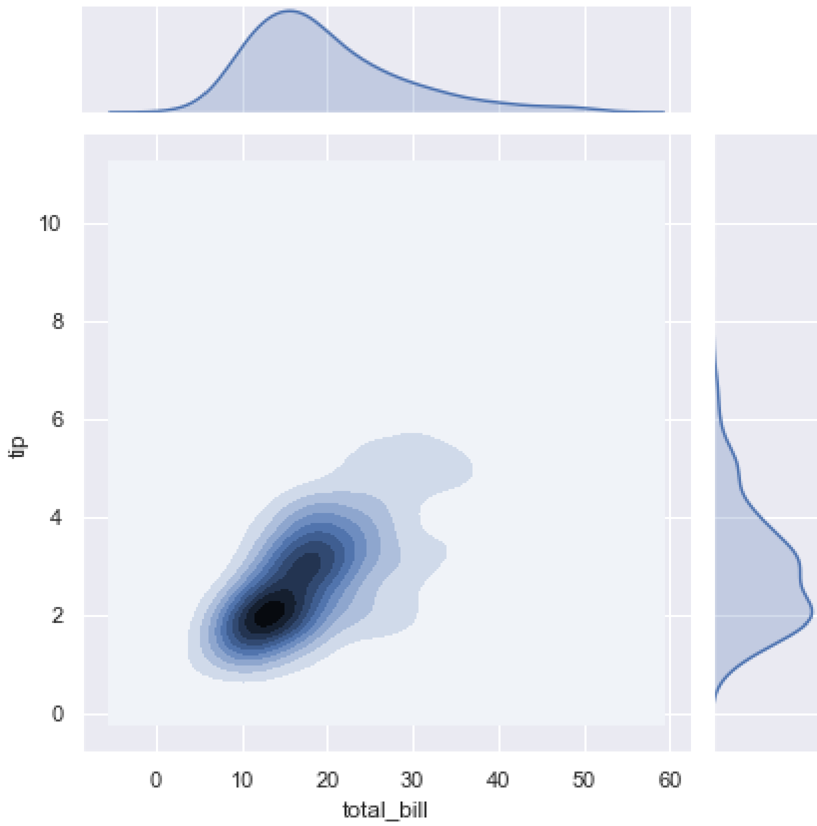


In [18]:

```
sns.jointplot( x='total_bill', y='tip', data=tips, kind='kde')
```

Out[18]:

```
<seaborn.axisgrid.JointGrid at 0x1cf6449dcc8>
```

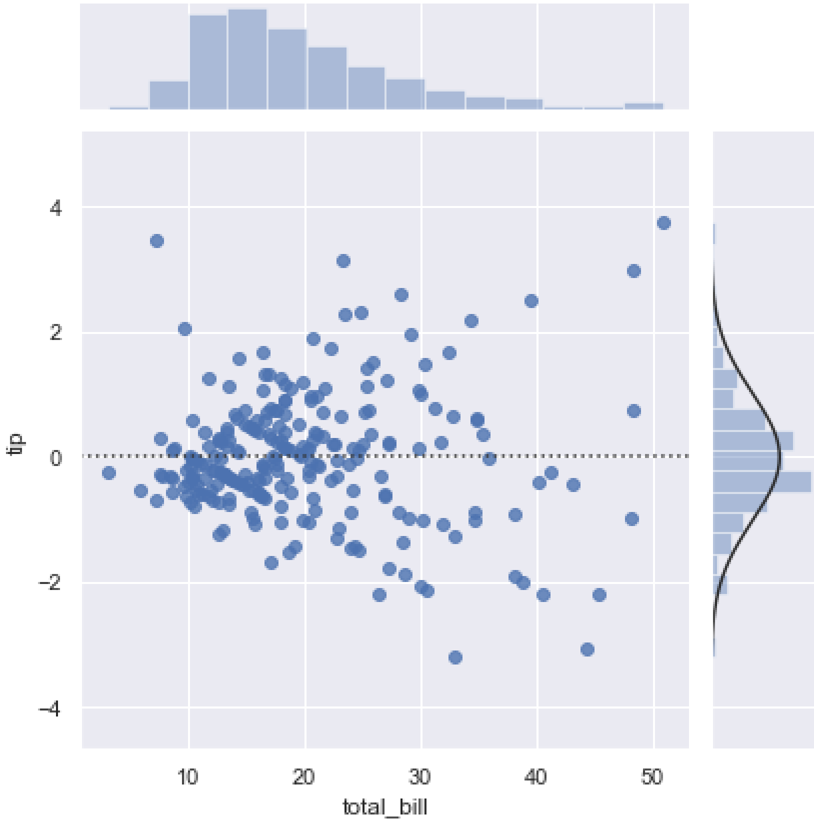


In [19]:

```
sns.jointplot( x='total_bill', y='tip', data=tips, kind='resid')
```

Out[19]:

```
<seaborn.axisgrid.JointGrid at 0x1cf645cf448>
```



pairplot

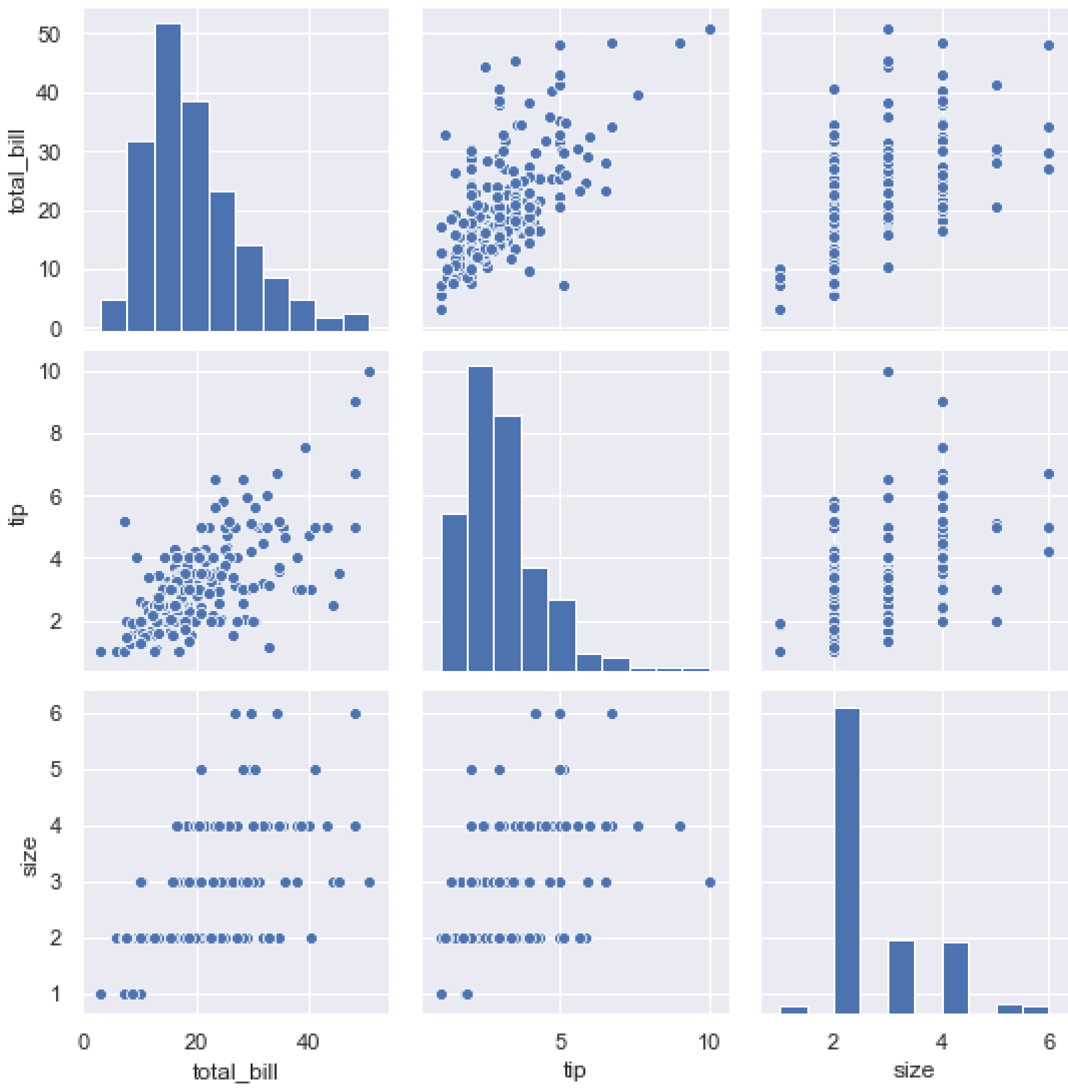
pairplot will plot pairwise relationships across an entire dataframe (for the numerical columns) and supports a color hue argument (for categorical columns).

In [20]:

```
sns.pairplot(tips)
```

Out[20]:

```
<seaborn.axisgrid.PairGrid at 0x1cf645b3288>
```

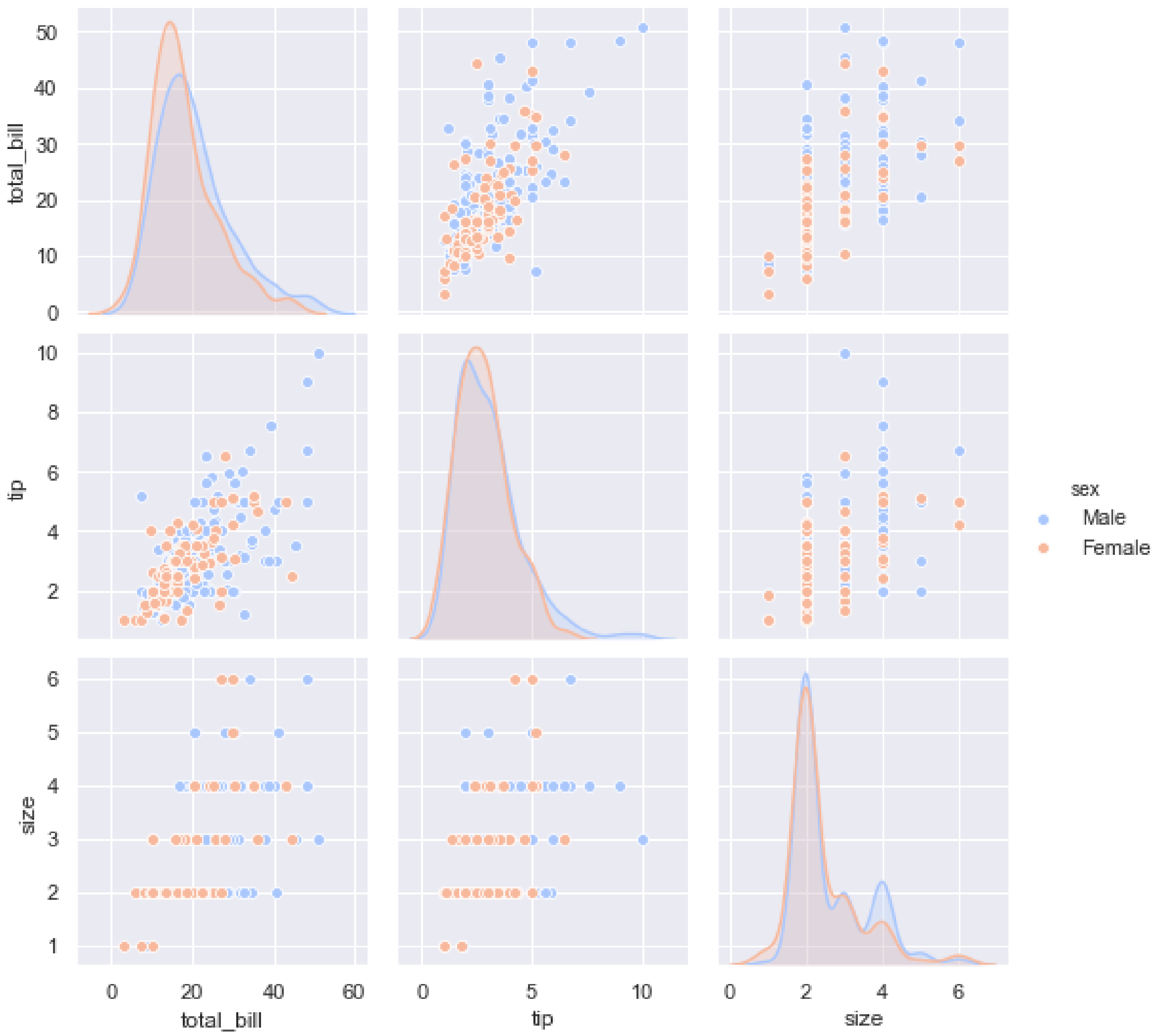


In [21]:

```
sns.pairplot(tips,hue='sex',palette='coolwarm')
```

Out[21]:

```
<seaborn.axisgrid.PairGrid at 0x1cf65c345c8>
```



rugplot

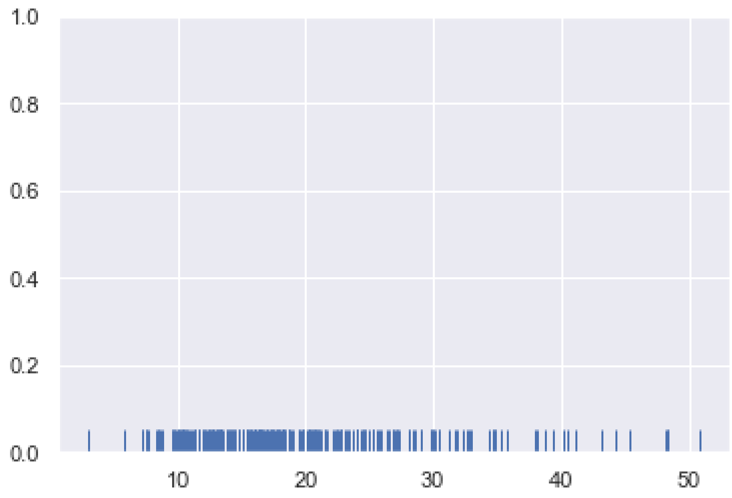
rugplots are actually a very simple concept, they just draw a dash mark for every point on a univariate distribution. They are the building block of a KDE plot:

In [22]:

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

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf660884c8>
```



Categorical Data Plots

There are a few main plot types for this:

1. barplot
2. countplot
3. boxplot
4. catplot

barplot and countplot

These very similar plots allow you to get aggregate data off a categorical feature in your data. **barplot** is a general plot that allows you to aggregate the categorical data based off some function, by default the mean:

In [24]:

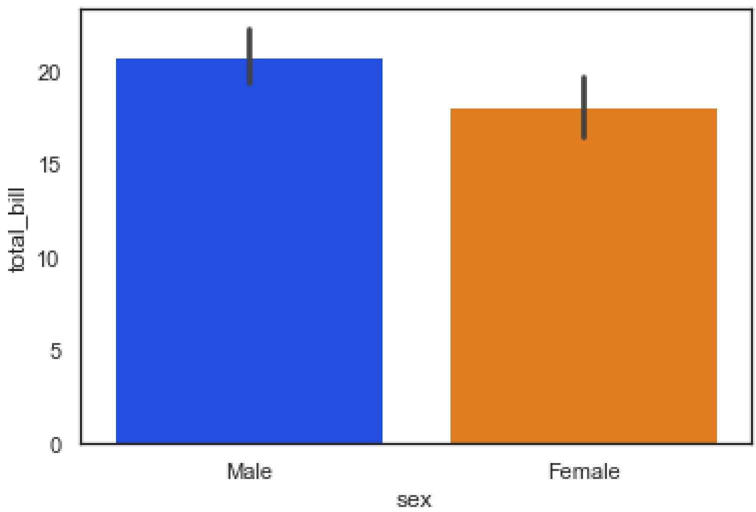
```
sns.set(style='white',palette='bright')
```

In [25]:

```
sns.barplot(x = 'sex', y= 'total_bill', data = tips)
```

Out[25]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf67a02b08>
```

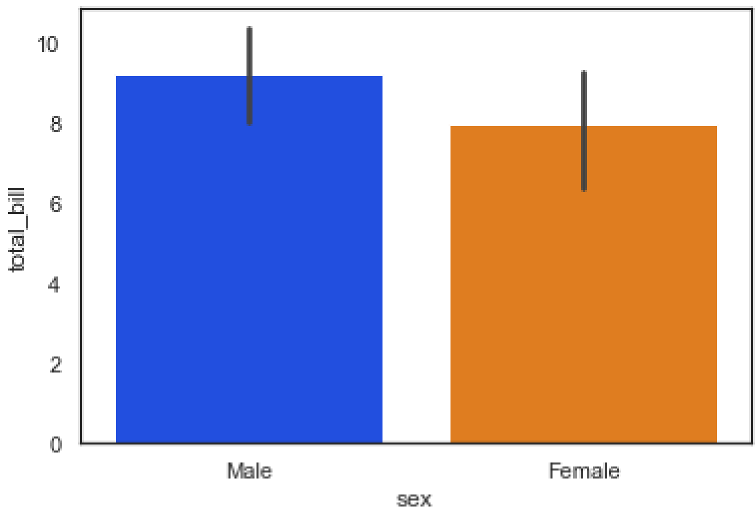


In [27]:

```
# We can change the estimator object that converts a vector to a scalar

sns.barplot(x='sex', y='total_bill', data=tips, estimator=np.std)
```

Out[27]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf67d4e808>
```



countplot

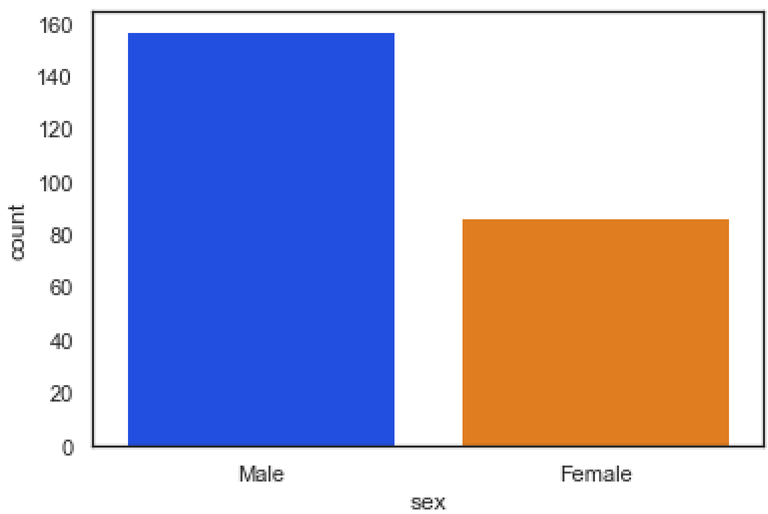
This is essentially the same as barplot except the estimator is explicitly counting the number of occurrences. Which is why we only pass any of the x or y

In [28]:

```
sns.countplot(x='sex', data=tips)
```

Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf67fe7108>
```

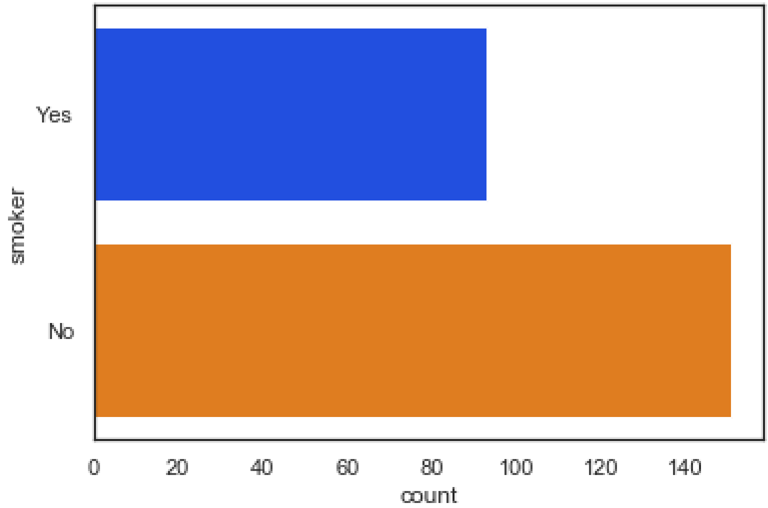


In [29]:

```
sns.countplot(y='smoker', data=tips)
```

Out[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf67bbbd08>
```



boxplot

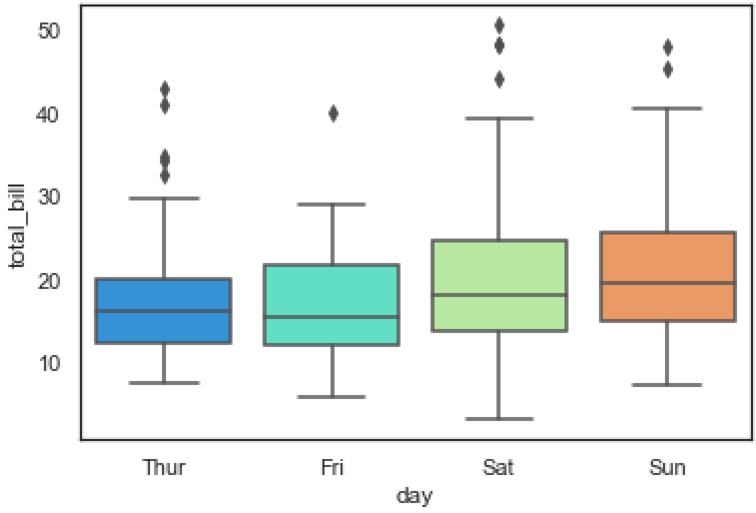
boxplots are used to show the distribution of categorical data. A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. The box shows the quartiles of the dataset while the whiskers 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 [31]:

```
sns.boxplot(x='day', y='total_bill', data=tips, palette='rainbow')
```

Out[31]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf67e30908>
```

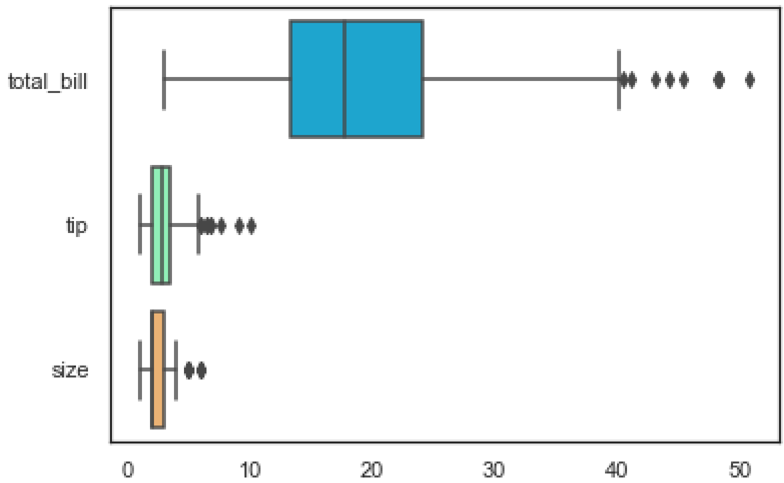


In [35]:

```
sns.boxplot(data=tips, palette='rainbow',orient='h')
```

Out[35]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf69638e88>
```

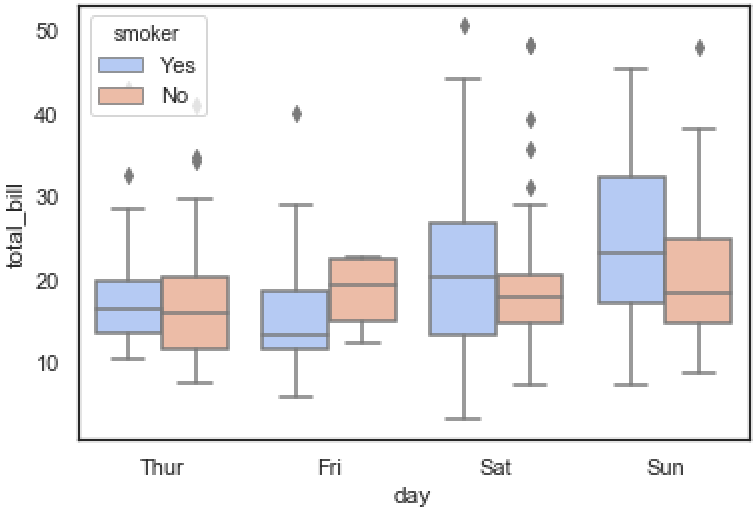


In [42]:

```
sns.boxplot(x='day', y='total_bill', data=tips, palette='coolwarm', hue='smoker')
```

Out[42]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf69843488>
```



catplot

catplot is the most general form of a categorical plot. It can take **kind** as a parameter to adjust the plot type:

In [54]:

```
# let's load the 'Excercise' dataset

exercise = sns.load_dataset('exercise')
exercise.head()
```

Out[54]:

	Unnamed: 0	id	diet	pulse	time	kind
0	0	1	low fat	85	1 min	rest
1	1	1	low fat	85	15 min	rest
2	2	1	low fat	88	30 min	rest
3	3	2	low fat	90	1 min	rest
4	4	2	low fat	92	15 min	rest

In [55]:

```
exercise.set_index('Unnamed: 0',inplace=True)
exercise.rename_axis('index',inplace=True)
exercise.rename(columns={'kind':'kind_exer'}, inplace=True)
exercise.head()
```

Out[55]:

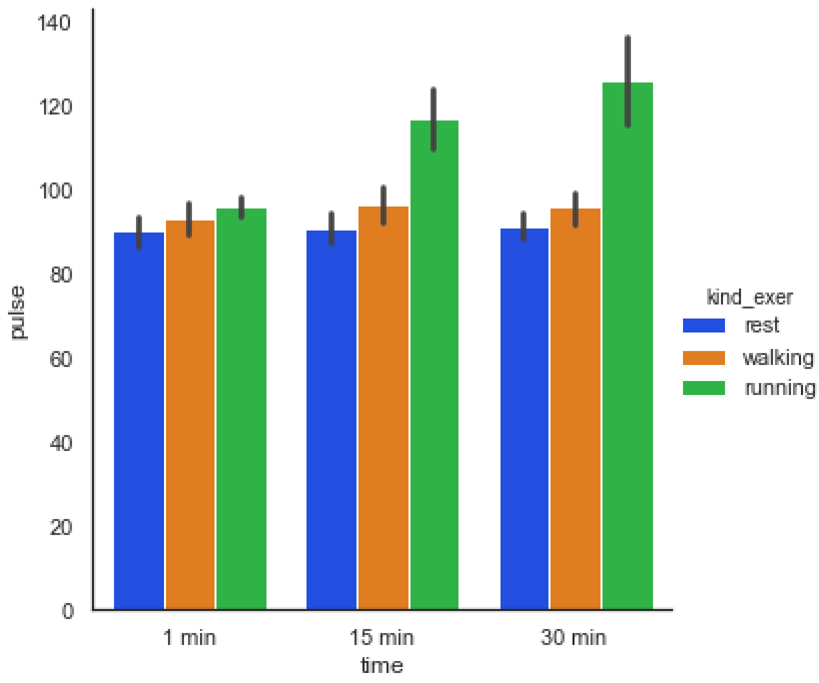
	id	diet	pulse	time	kind_exer
index					
0	1	low fat	85	1 min	rest
1	1	low fat	85	15 min	rest
2	1	low fat	88	30 min	rest
3	2	low fat	90	1 min	rest
4	2	low fat	92	15 min	rest

In [58]:

```
sns.catplot(x='time', y='pulse', data=exercise, kind='bar', hue='kind_exer')
```

Out[58]:

```
<seaborn.axisgrid.FacetGrid at 0x1cf69820688>
```



Grids

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

- PairGrid
- FacetGrid
- JointGrid

In [59]:

```
# let's load iris dataset

iris = sns.load_dataset('iris')
iris.head()
```

Out[59]:

	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

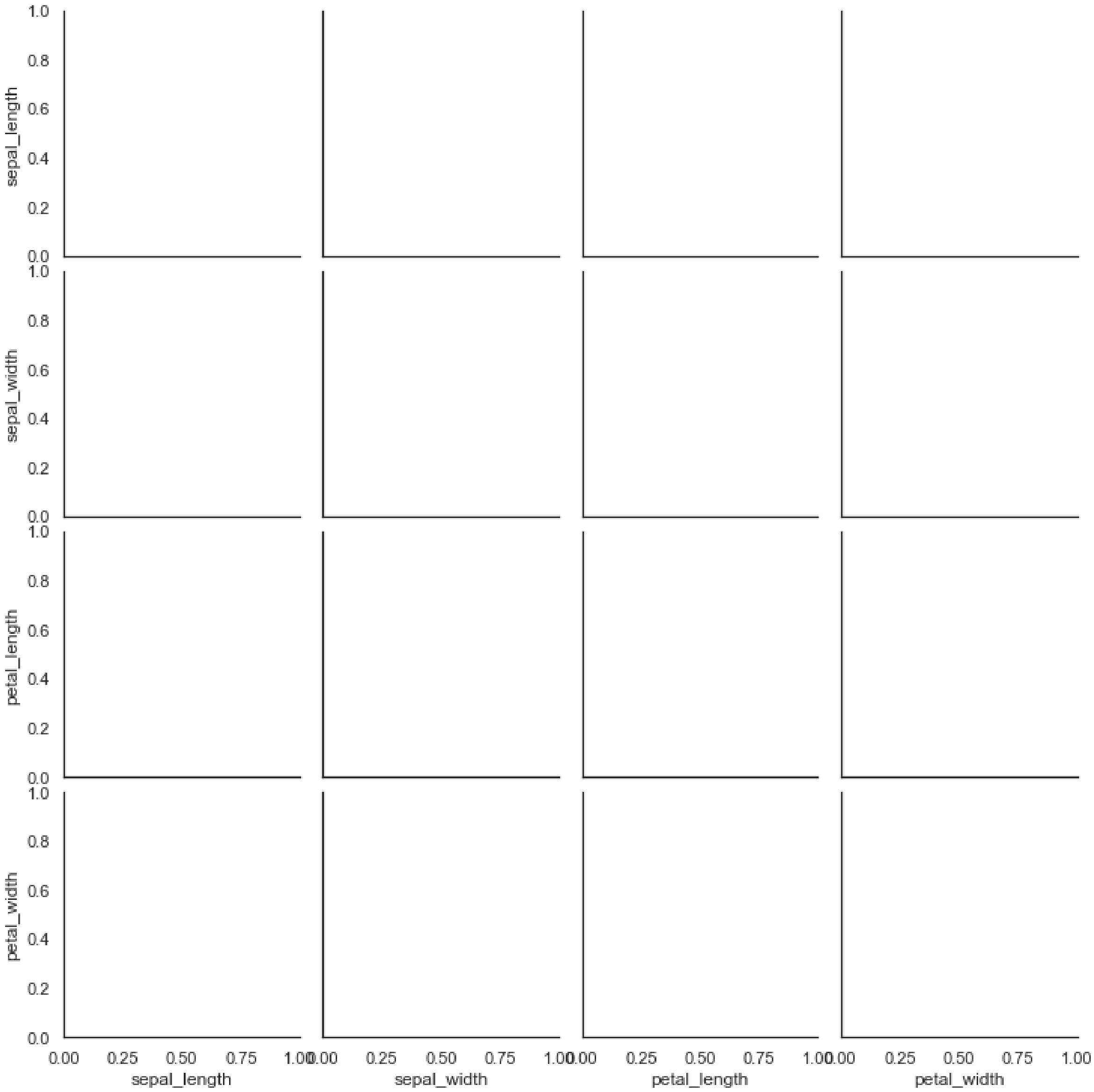
Pairgrid is a subplot grid for plotting pairwise relationships in a dataset.

In [60]:

```
# Empty Grid

sns.PairGrid(iris)
```

Out[60]:

```
<seaborn.axisgrid.PairGrid at 0x1cf6a088248>
```

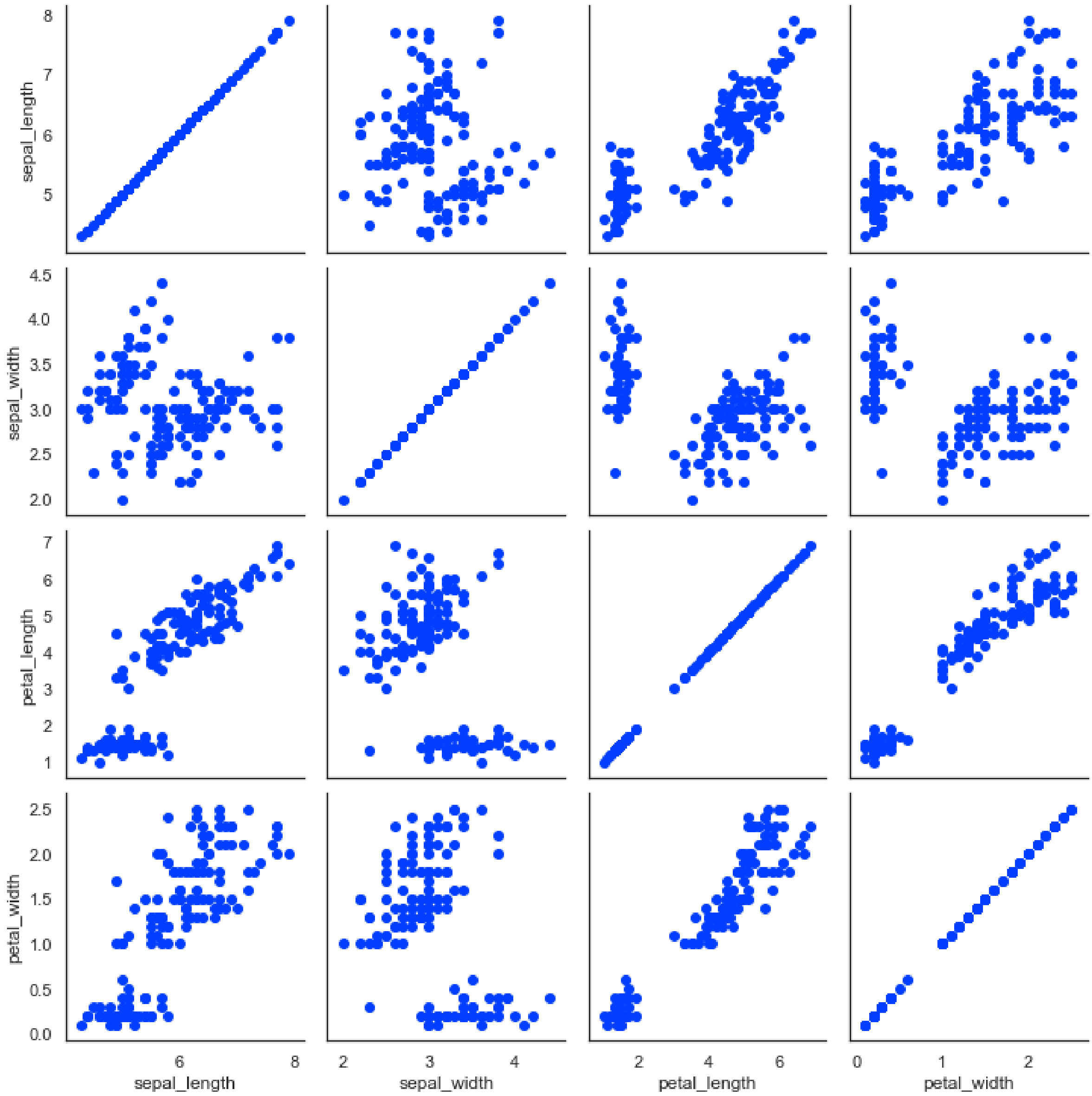


In [62]:

```
# then we can map data to the grid

grid_plot = sns.PairGrid(iris)
grid_plot.map(plt.scatter)
```

Out[62]:

```
<seaborn.axisgrid.PairGrid at 0x1cf6b4c8b48>
```

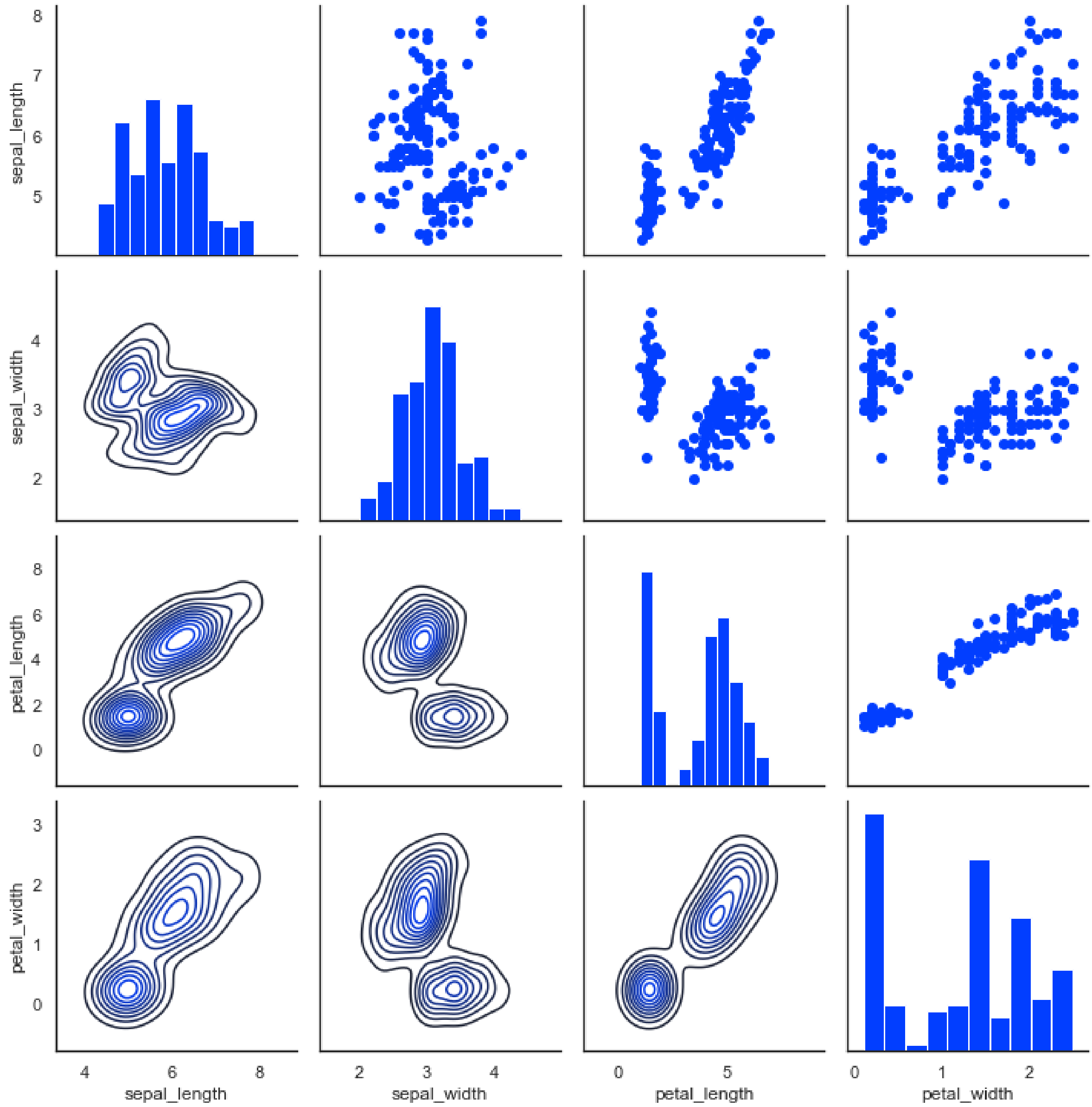


In [68]:

```
# We can map different types of plots to diagonal , lower half and upper half

grid_plot = sns.PairGrid(iris)
grid_plot.map_diag(plt.hist)
grid_plot.map_lower(sns.kdeplot)
grid_plot.map_upper(plt.scatter)
```

Out[68]:

```
<seaborn.axisgrid.PairGrid at 0x1cf6e02a488>
```



pairplot

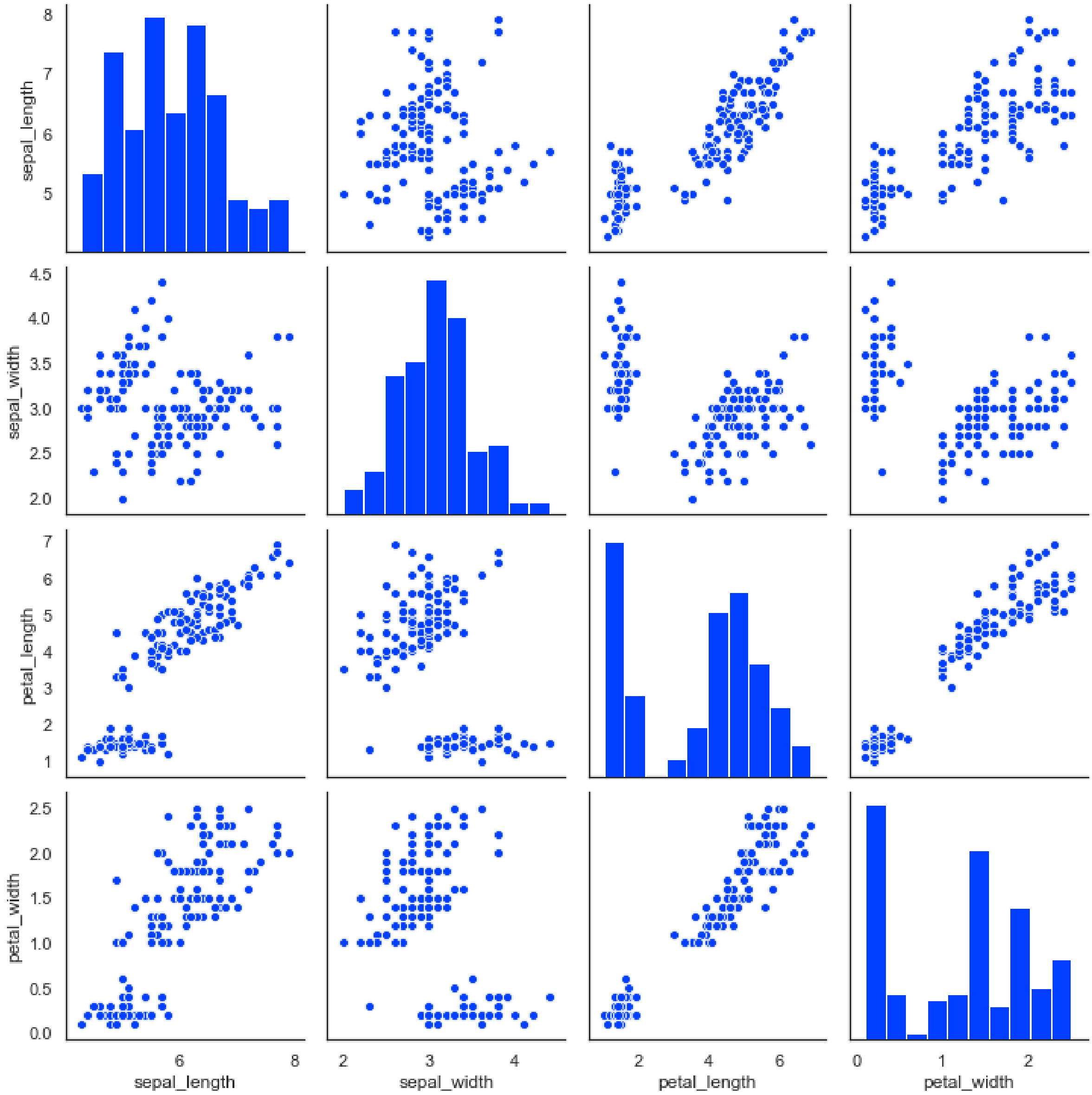
pairplot is a simpler version of PairGrid

In [70]:

```
sns.pairplot(iris)
```

Out[70]:

<seaborn.axisgrid.PairGrid at 0x1cf6e8d9b48>

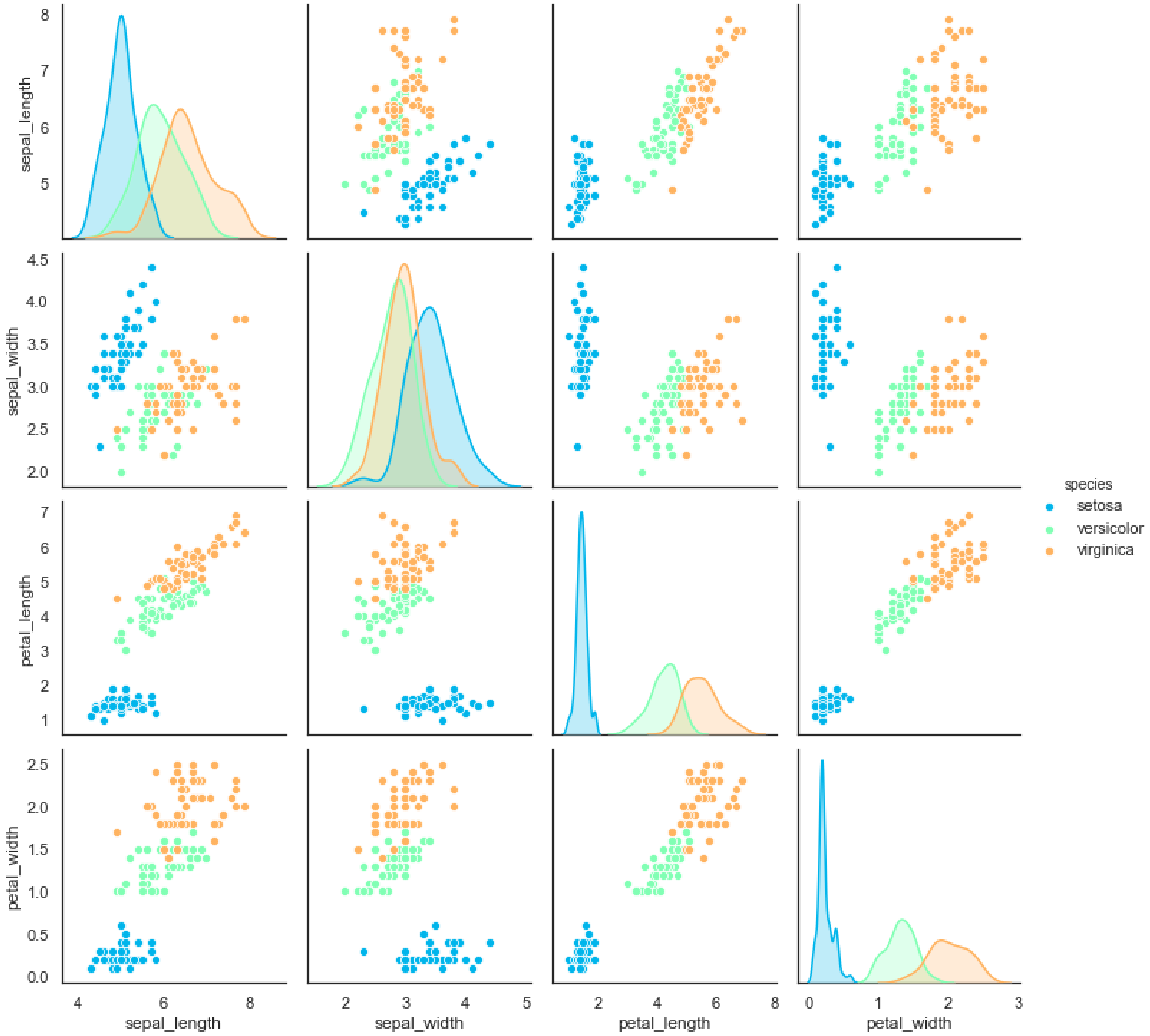


In [71]:

```
sns.pairplot(iris, hue='species', palette='rainbow')
```

Out[71]:

```
<seaborn.axisgrid.PairGrid at 0x1cf705c7088>
```



Facet Grid

FacetGrid is the general way to create grids of plots based of a feature

In [72]:

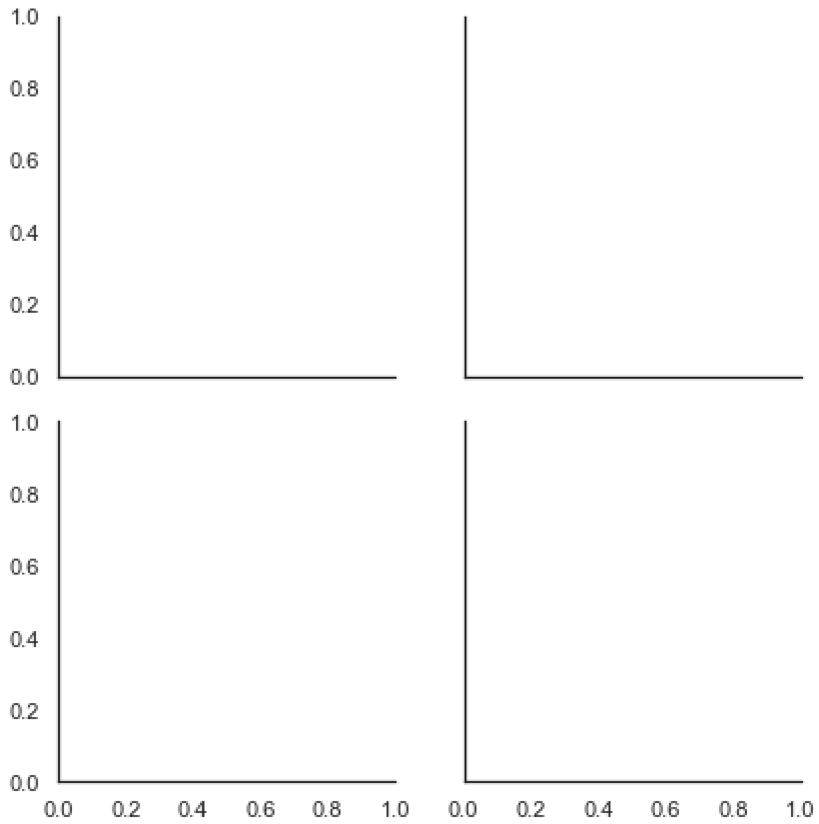
```
tips.head()
```

Out[72]:

	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 [73]:

```
# Plotting the Grid

plot_facet = sns.FacetGrid(data=tips, row='smoker', col='time')
```

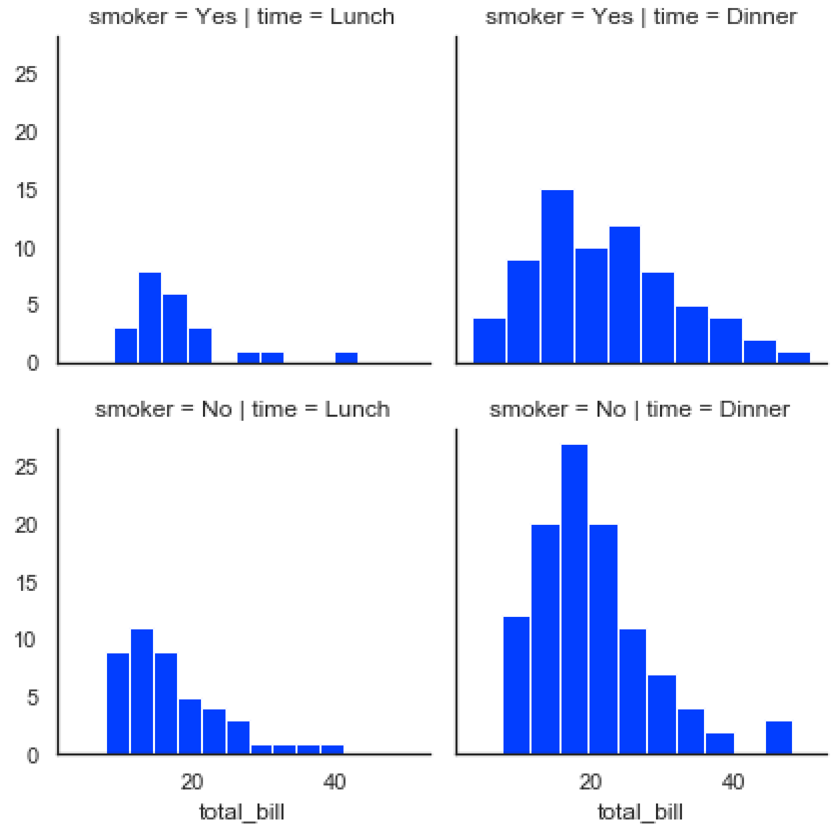


In [75]:

```
# mapping the data to grid

plot_facet = sns.FacetGrid(data=tips, row='smoker', col='time')
plot_facet.map(plt.hist, 'total_bill')
```

Out[75]:

```
<seaborn.axisgrid.FacetGrid at 0x1cf710de4c8>
```

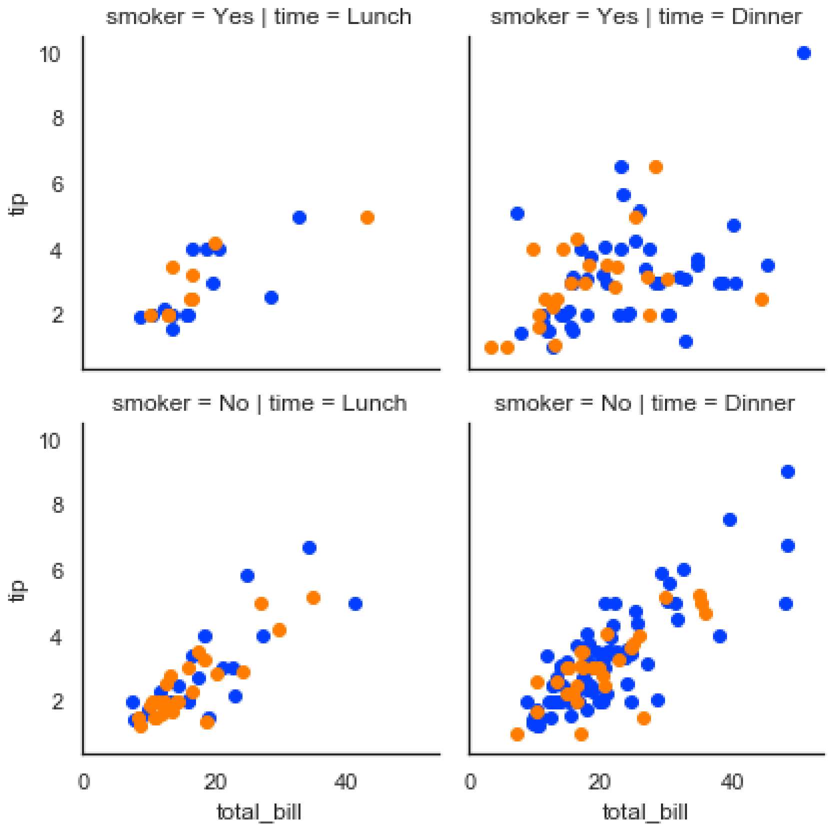


In [78]:

```
plot_facet = sns.FacetGrid(data=tips, row='smoker', col='time',hue='sex')
plot_facet.map(plt.scatter, 'total_bill','tip')

# 2 Arguments are passed for scatter plot
```

Out[78]:

```
<seaborn.axisgrid.FacetGrid at 0x1cf717a9548>
```

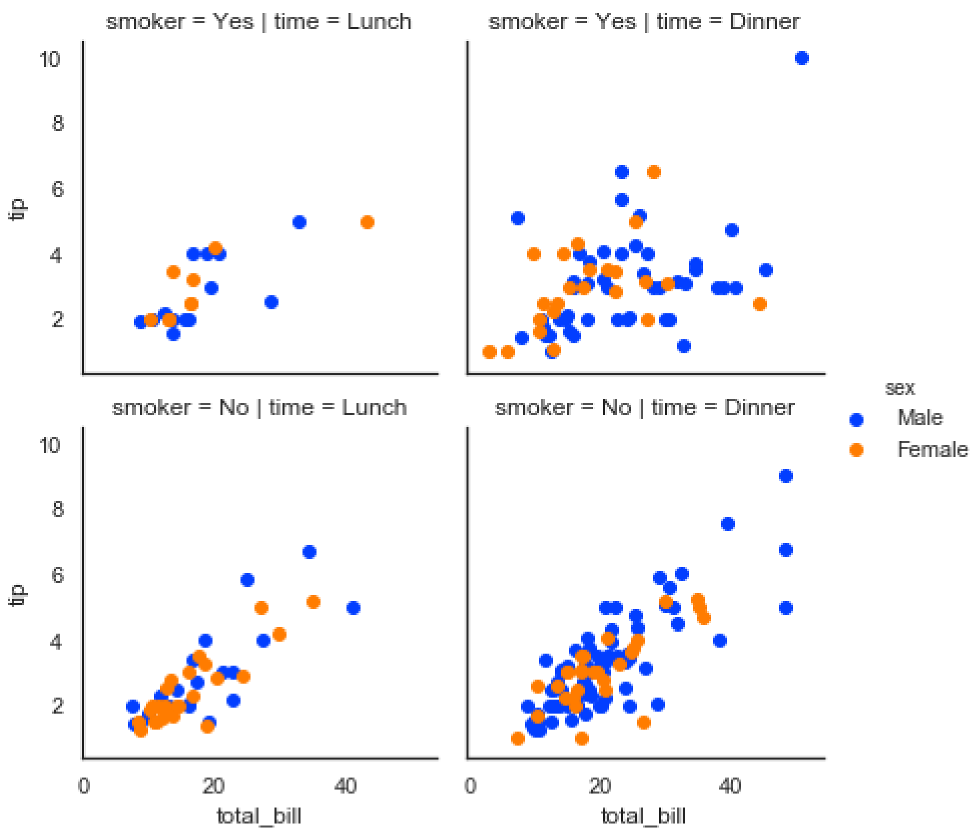


In [79]:

```
# Adding legends

plot_facet = sns.FacetGrid(data=tips, row='smoker', col='time',hue='sex')
plot_facet.map(plt.scatter, 'total_bill','tip').add_legend()
```

Out[79]:

```
<seaborn.axisgrid.FacetGrid at 0x1cf718e0108>
```

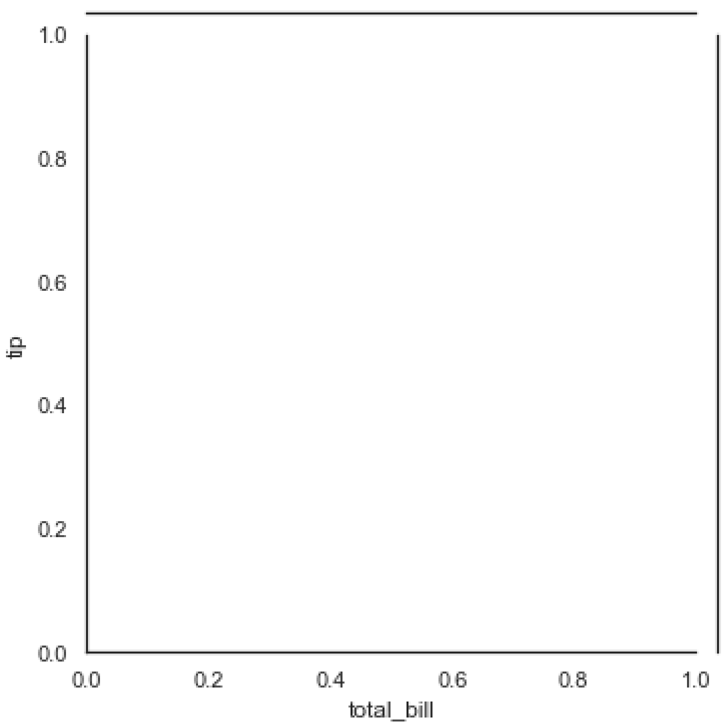


JointGrid

JointGrid is the general version for jointplot() type grids

In [81]:

```
# plotting the Grid

plot_joint = sns.JointGrid(x='total_bill', y='tip', data=tips)
```

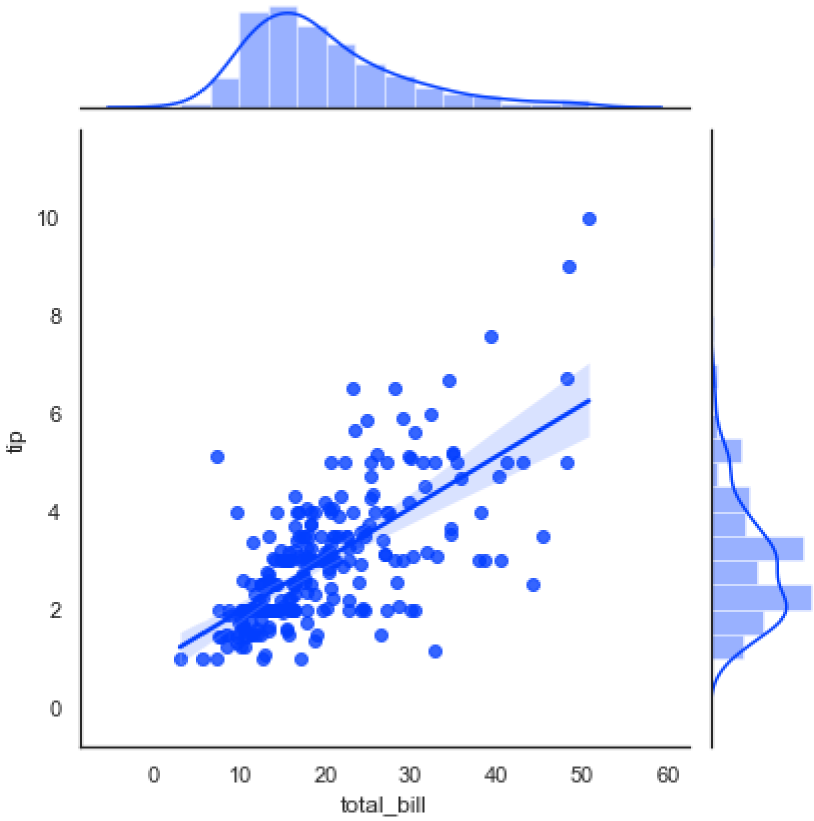


In [84]:

```
# plotting the data

plot_joint = sns.JointGrid(x='total_bill', y='tip', data=tips)
plot_joint.plot(sns.regplot,sns.distplot)
```

Out[84]:

```
<seaborn.axisgrid.JointGrid at 0x1cf73cf5d08>
```



Matrix Plots

Matrix plots allow us to plot data as color-encoded matrices and can also be used to indicate clusters within the data

In [85]:

```
# Loading the fights data

flights = sns.load_dataset('flights')
flights.head()
```

Out[85]:

	year	month	passengers
0	1949	January	112
1	1949	February	118
2	1949	March	132
3	1949	April	129
4	1949	May	121

Heatmap

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

In [88]:

```
tips.head()
```

Out[88]:

	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 [91]:

```
# Matrix form for correlation data

tips_corr = tips.corr()
tips_corr
```

Out[91]:

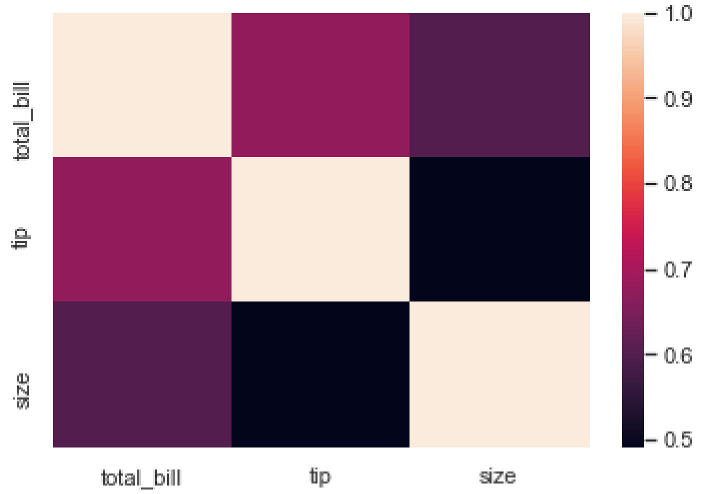
	total_bill	tip	size
total_bill	1.000000	0.675734	0.598315
tip	0.675734	1.000000	0.489299
size	0.598315	0.489299	1.000000

In [92]:

```
sns.heatmap(tips_corr)
```

Out[92]:

<matplotlib.axes._subplots.AxesSubplot at 0x1cf73a068c8>

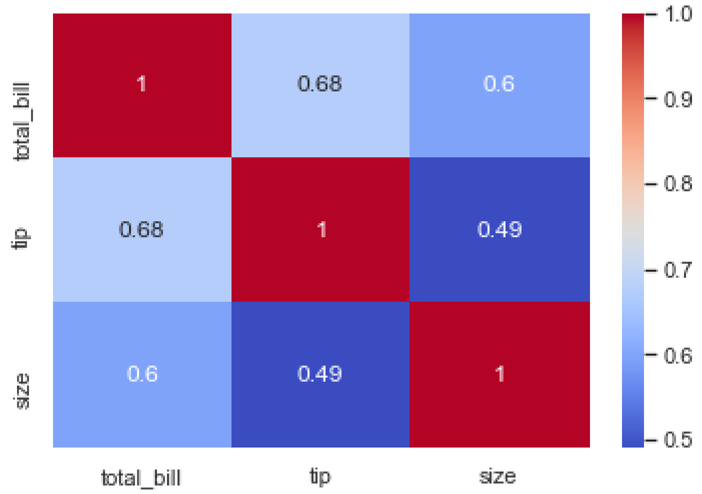


In [97]:

```
sns.heatmap(tips_corr,cmap='coolwarm',annot=True)
```

Out[97]:

<matplotlib.axes._subplots.AxesSubplot at 0x1cf73f28c88>



In [98]:

```
# Let's take pivot of flights data to convert it into Matrix form

flights_pivot = flights.pivot_table(values='passengers',index='month',columns='year')
flights_pivot
```

Out[98]:

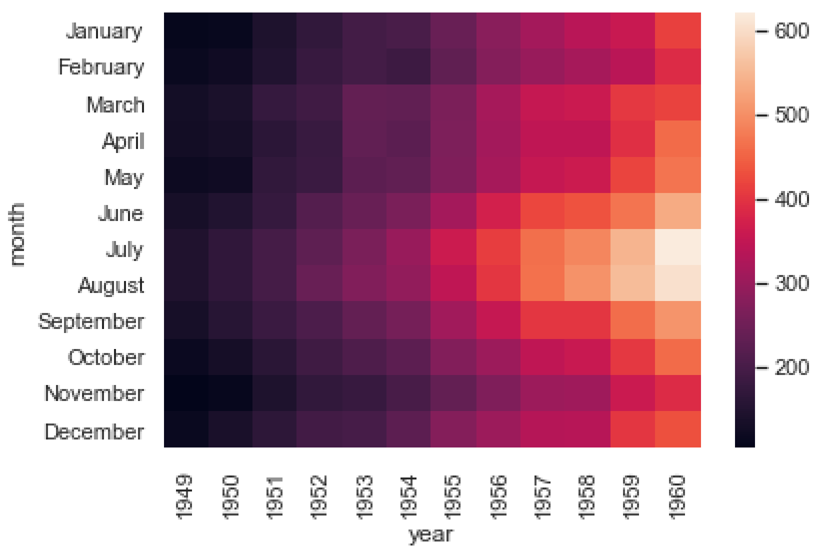
year month	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960
January	112	115	145	171	196	204	242	284	315	340	360	417
February	118	126	150	180	196	188	233	277	301	318	342	391
March	132	141	178	193	236	235	267	317	356	362	406	419
April	129	135	163	181	235	227	269	313	348	348	396	461
May	121	125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

In [99]:

```
sns.heatmap(flights_pivot)
```

Out[99]:

<matplotlib.axes._subplots.AxesSubplot at 0x1cf73ea8b08>

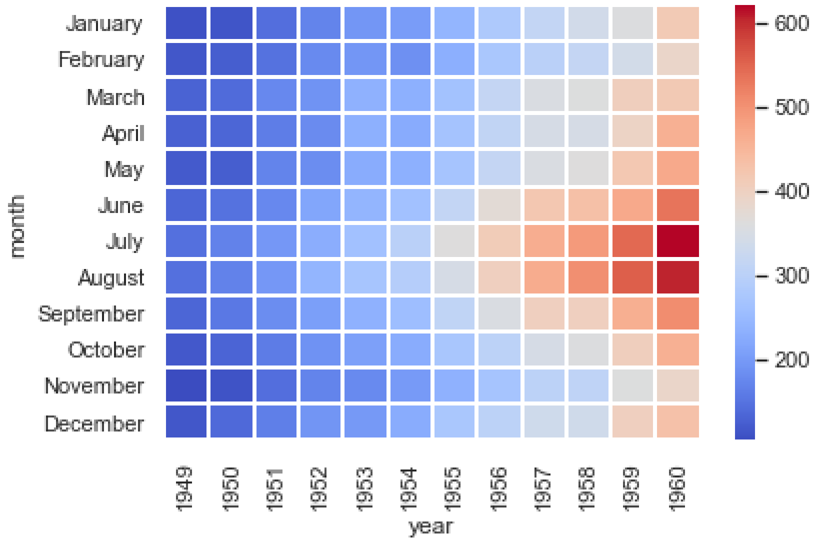


In [103]:

```
sns.heatmap(flights_pivot,cmap='coolwarm',linecolor='white',linewidths=1)
```

Out[103]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf742fed48>
```



clustermap

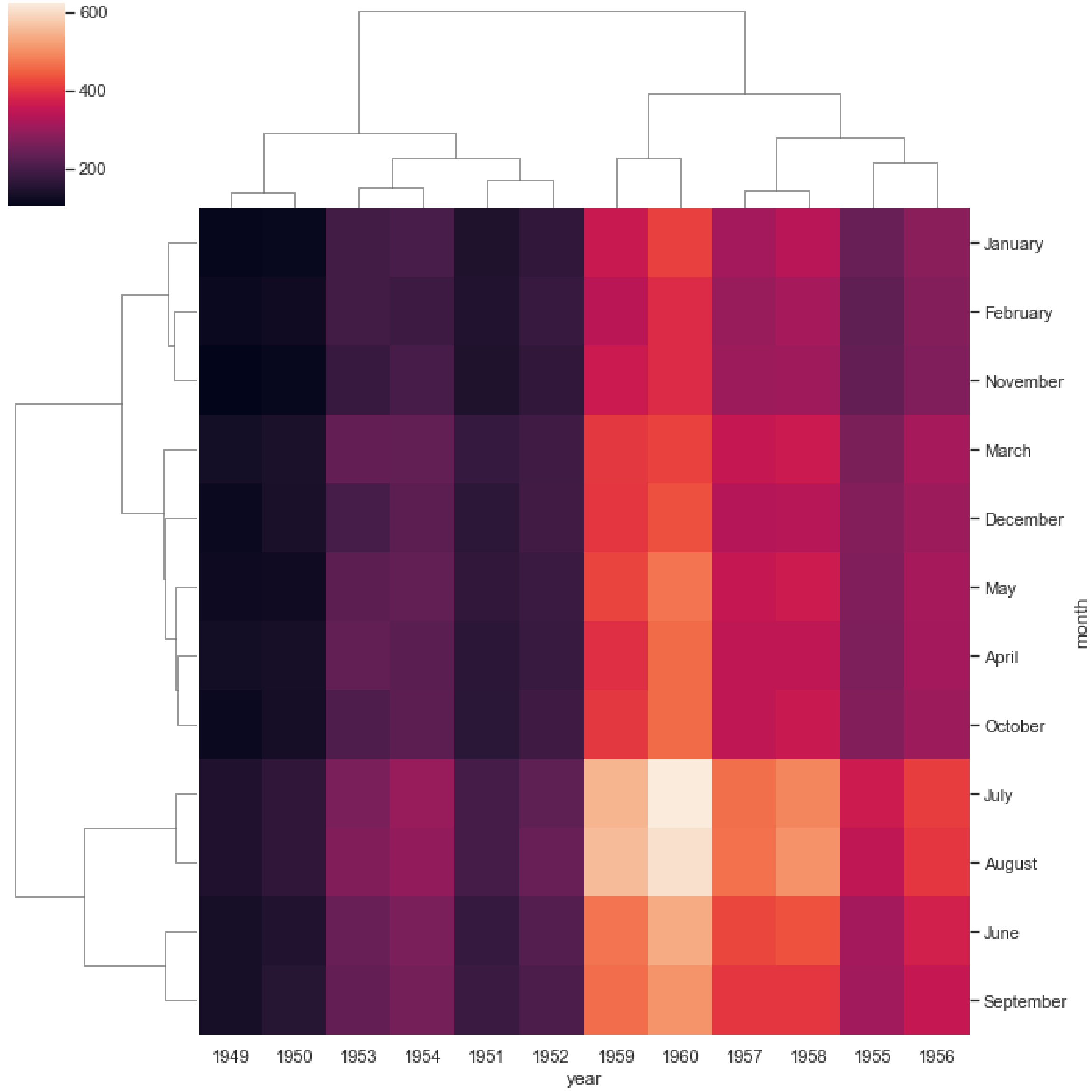
The clustermap uses hierarchal clustering to produce a clustered version of the heatmap.

In [105]:

```
sns.clustermap(flights_pivot)
```

Out[105]:

<seaborn.matrix.ClusterGrid at 0x1cf74740808>



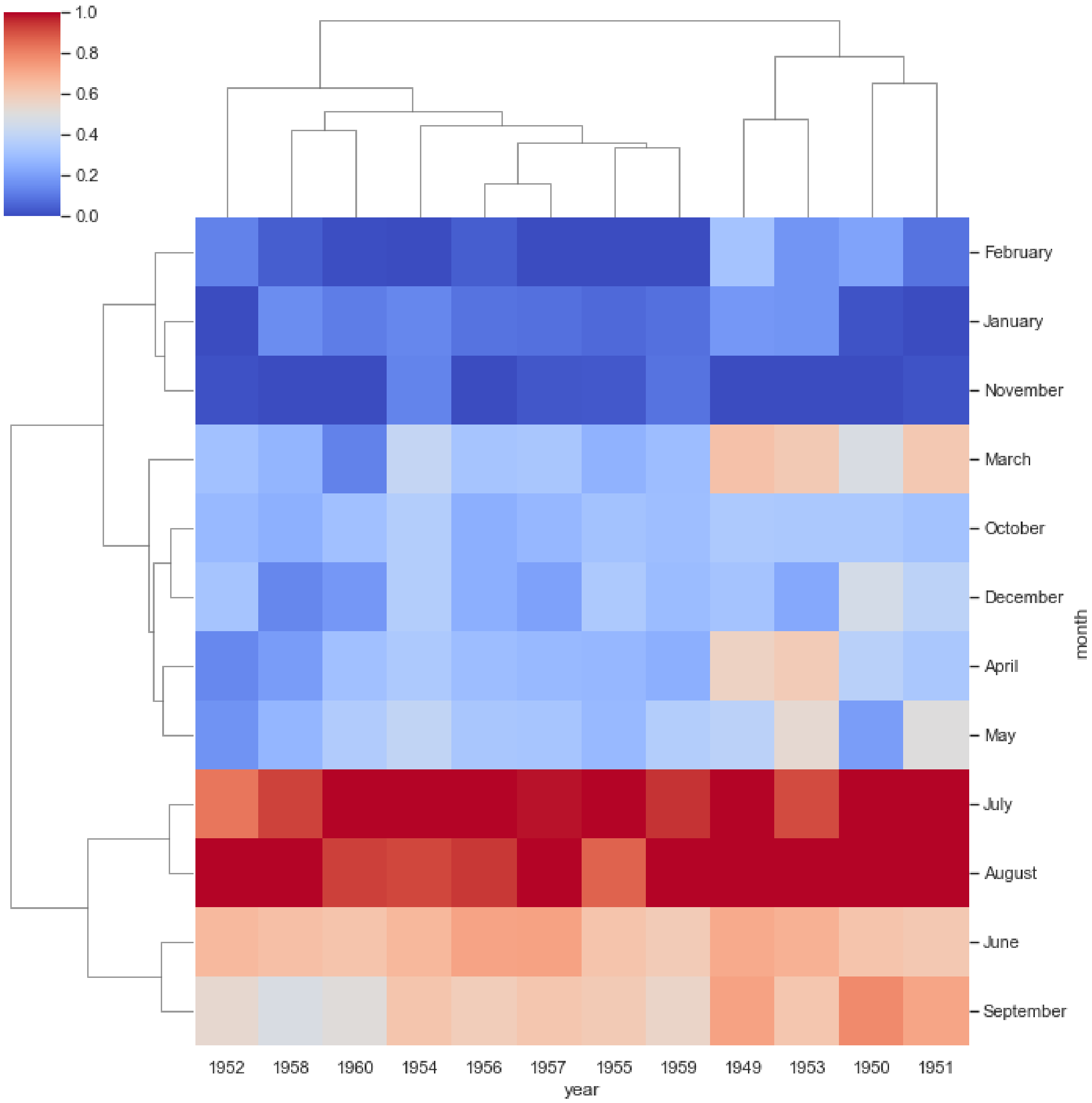
Notice now how the years and months are no longer in order, instead they are grouped by similarity in value (passenger count). That means we can begin to infer things from this plot, such as August and July being similar

In [110]:

```
# More options to get the information a little clearer like normalization

sns.clustermap(flights_pivot,cmap='coolwarm', standard_scale=1)
```

Out[110]:

```
<seaborn.matrix.ClusterGrid at 0x1cf76a6ad08>
```



lmplot()

- lmplot is one of the Regression plot.
- lmplot allows you to display linear models, but it also conveniently allows you to split up those plots based of features, as well as coloring the hue based of features

In [275]:

```
tips.head()
```

Out[275]:

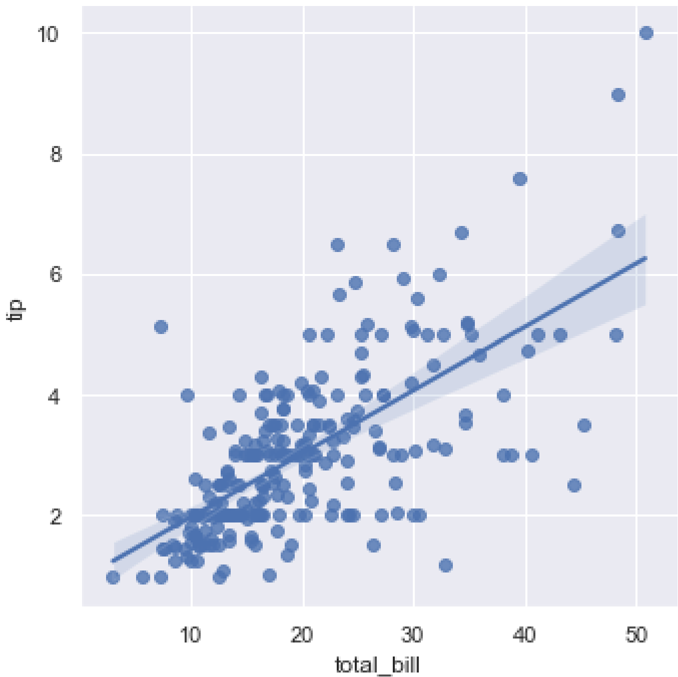
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

In [280]:

```
sns.lmplot(x='total_bill', y ='tip', data=tips)
```

Out[280]:

```
<seaborn.axisgrid.FacetGrid at 0x1cf04aef208>
```

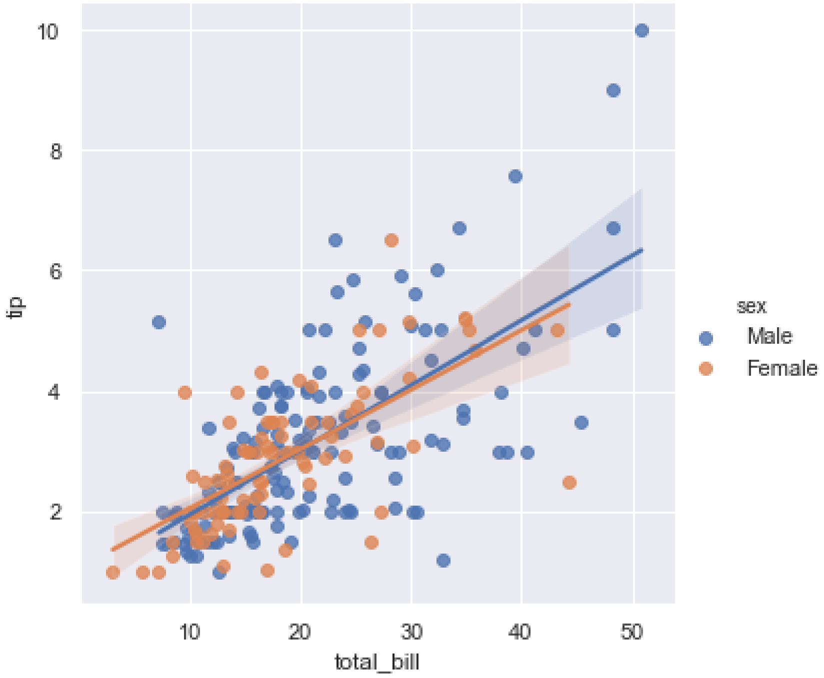


In [281]:

```
sns.lmplot(x='total_bill', y='tip', data=tips, hue='sex')
```

Out[281]:

<seaborn.axisgrid.FacetGrid at 0x1cf04e9b788>

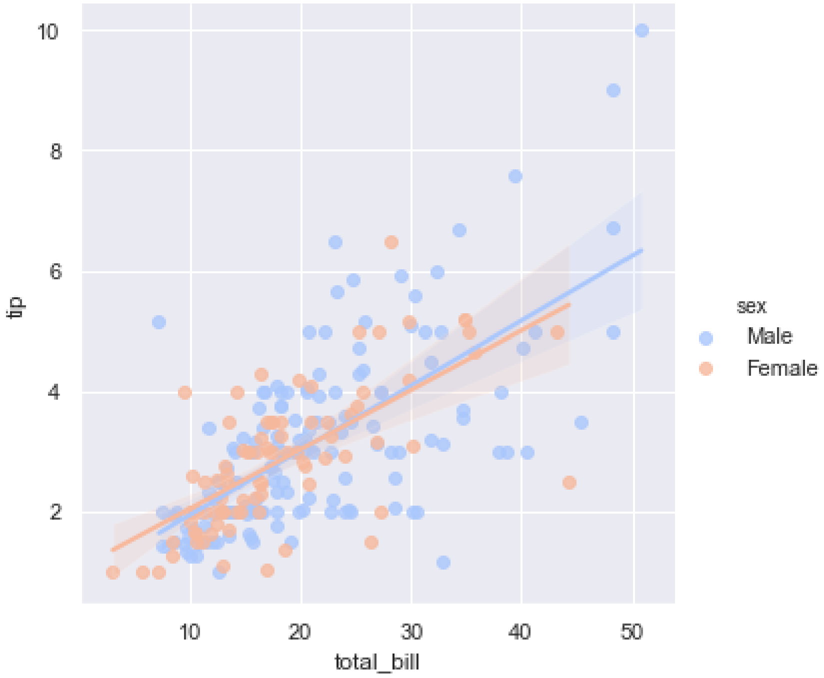


In [282]:

```
sns.lmplot(x='total_bill', y='tip', data=tips, hue='sex',palette='coolwarm')
```

Out[282]:

```
<seaborn.axisgrid.FacetGrid at 0x1cf04b76548>
```



Working with Markers

lmplot kwargs get passed through to regplot which is a more general form of lmplot(). regplot has a scatter_kws parameter that gets passed to plt.scatter. So we have to set the s parameter in that dictionary, which corresponds to the squared markersize.

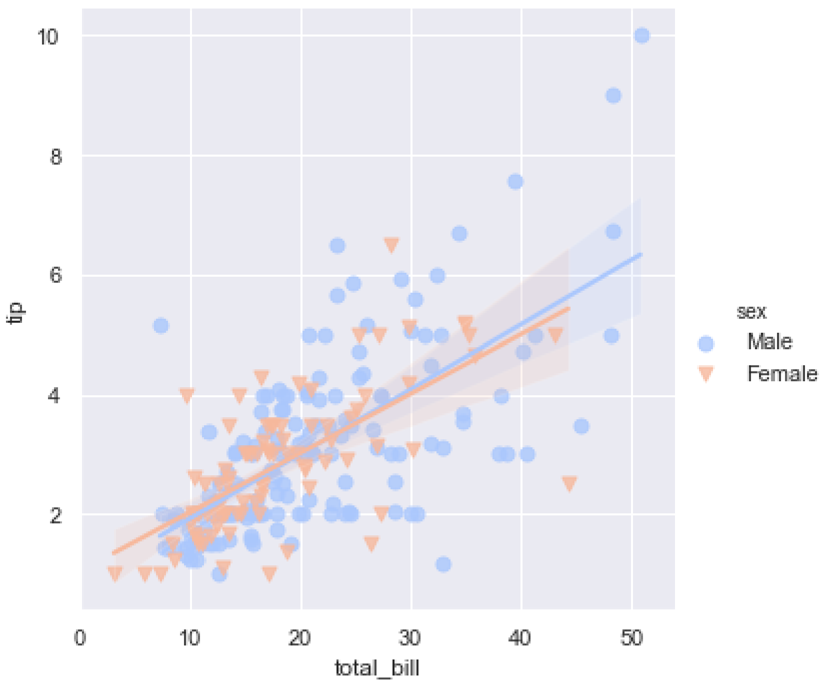
[http://matplotlib.org/api/markers_api.html (http://matplotlib.org/api/markers_api.html)](http://matplotlib.org/api/markers_api.html)

In [289]:

```
sns.lmplot(x='total_bill', y='tip', data=tips, hue='sex',palette='coolwarm',markers=['o','v
```

Out[289]:

```
<seaborn.axisgrid.FacetGrid at 0x1cf06f6cc08>
```



Using a Grid

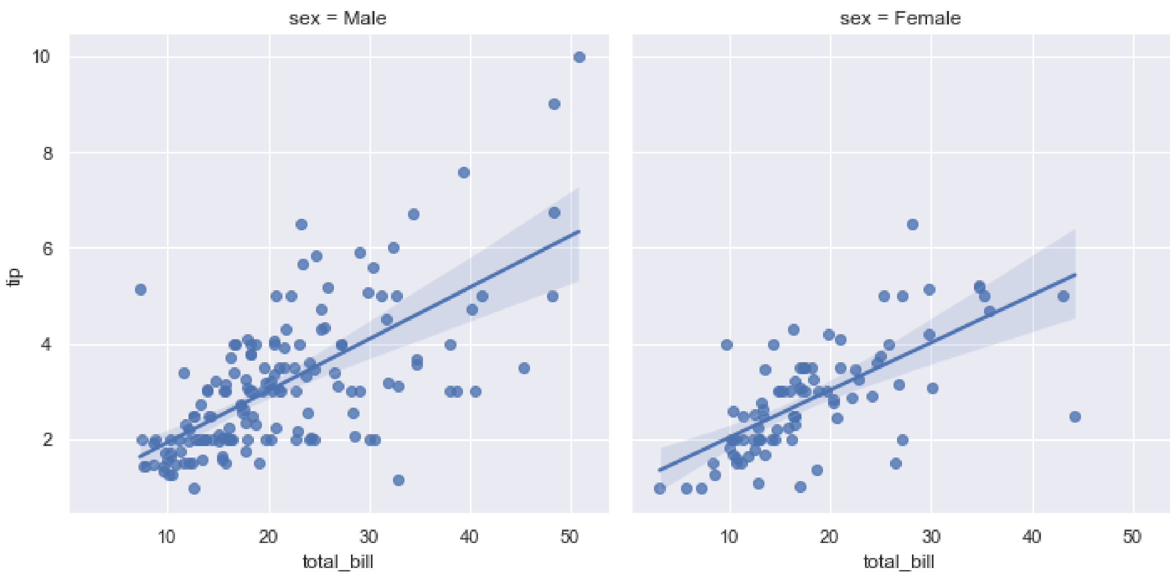
We can add more variable separation through columns and rows with the use of a grid. Just indicate this with the col or row arguments

In [290]:

```
sns.lmplot(x='total_bill', y='tip', data=tips, palette='coolwarm', col='sex')
```

Out[290]:

<seaborn.axisgrid.FacetGrid at 0x1cf07065688>

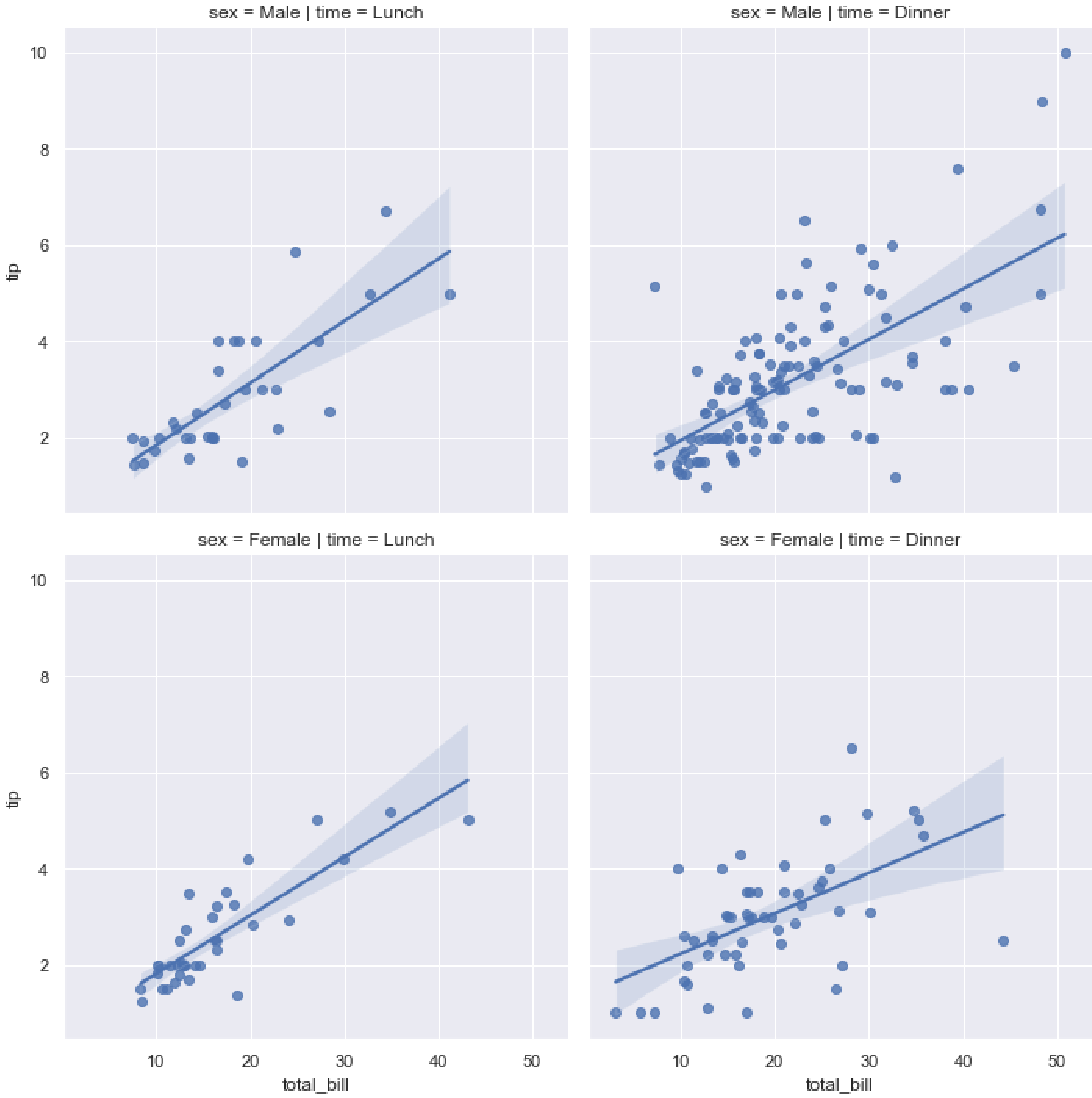


In [295]:

```
sns.lmplot(x='total_bill', y='tip', data=tips, row='sex', col='time', palette='coolwarm')
```

Out[295]:

```
<seaborn.axisgrid.FacetGrid at 0x1cf12f58b48>
```

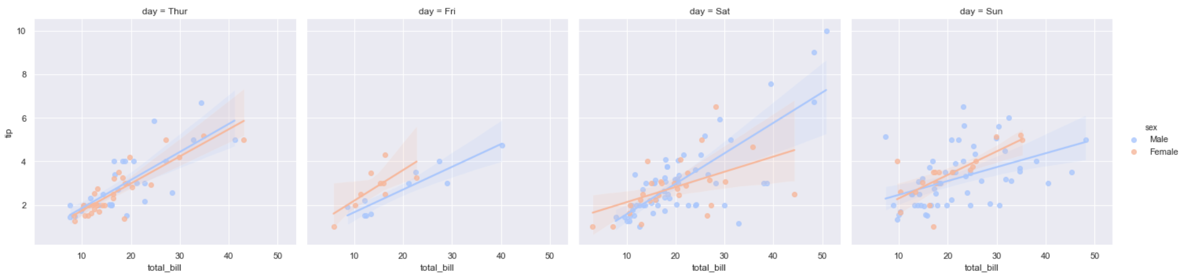


In [292]:

```
sns.lmplot(x='total_bill', y='tip', data=tips, palette='coolwarm', col='day', hue='sex')
```

Out[292]:

```
<seaborn.axisgrid.FacetGrid at 0x1cf11ac4288>
```

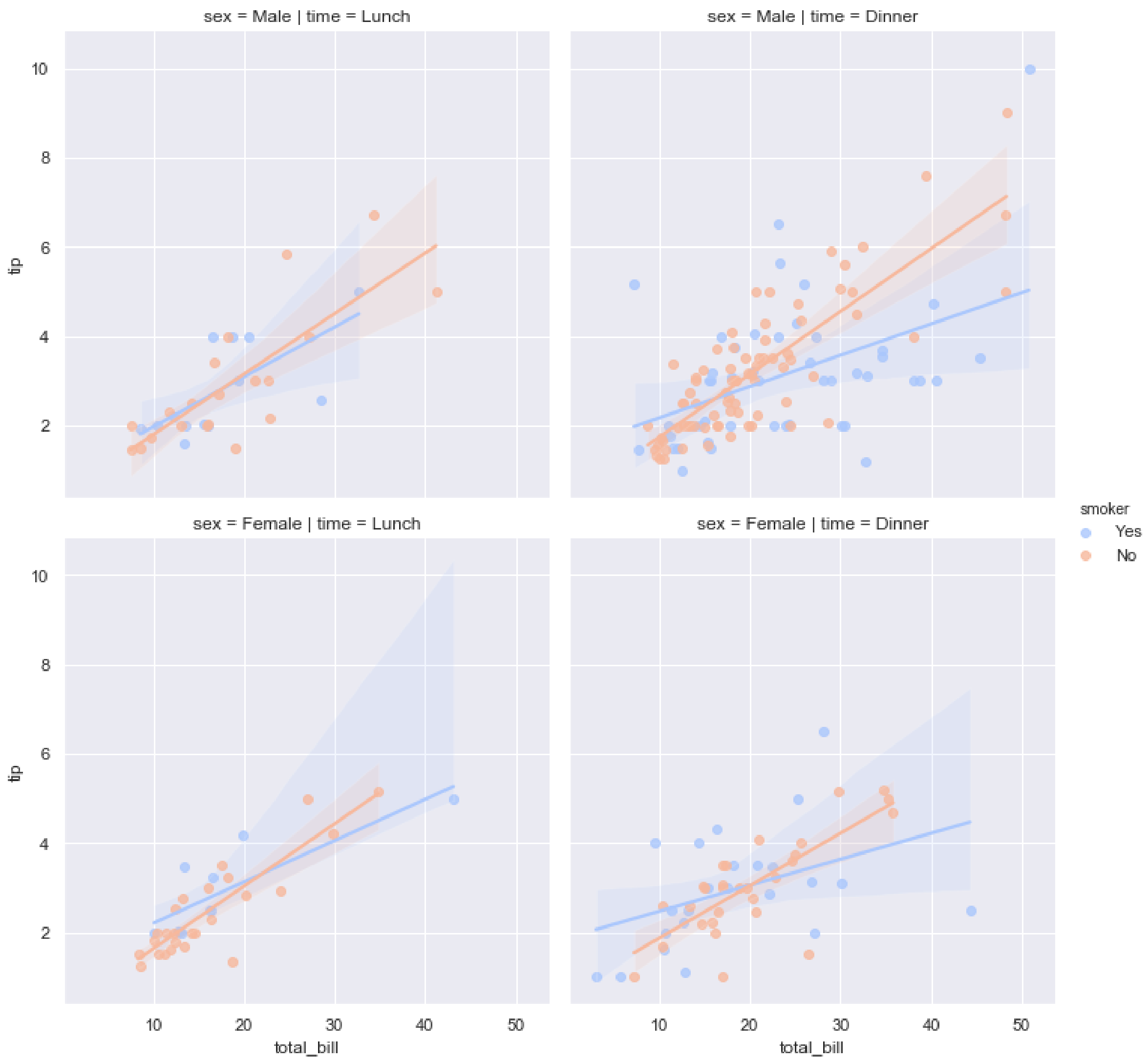


In [296]:

```
sns.lmplot(x='total_bill', y='tip', data=tips, row='sex', col='time', palette='coolwarm',hu
```

Out[296]:

```
<seaborn.axisgrid.FacetGrid at 0x1cf13230108>
```



Style and Color

- Let's see how to control the aesthetics in seaborn

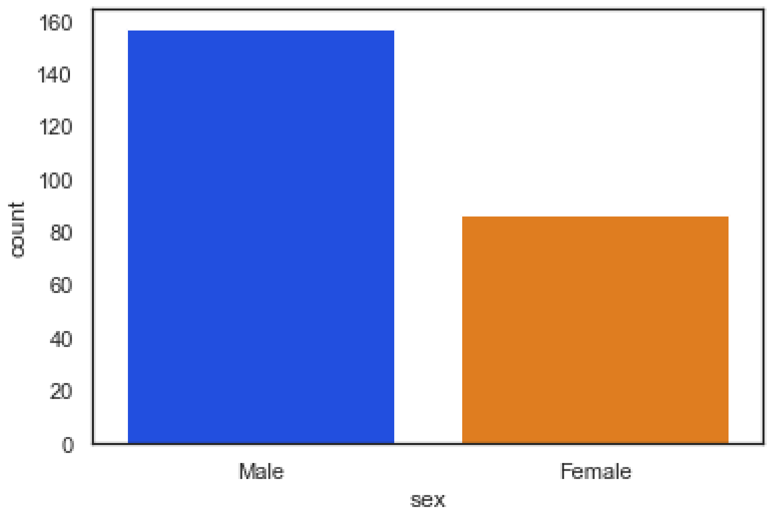
Styles

In [113]:

```
sns.countplot(x='sex', data=tips)
```

Out[113]:

<matplotlib.axes._subplots.AxesSubplot at 0x1cf779f6748>

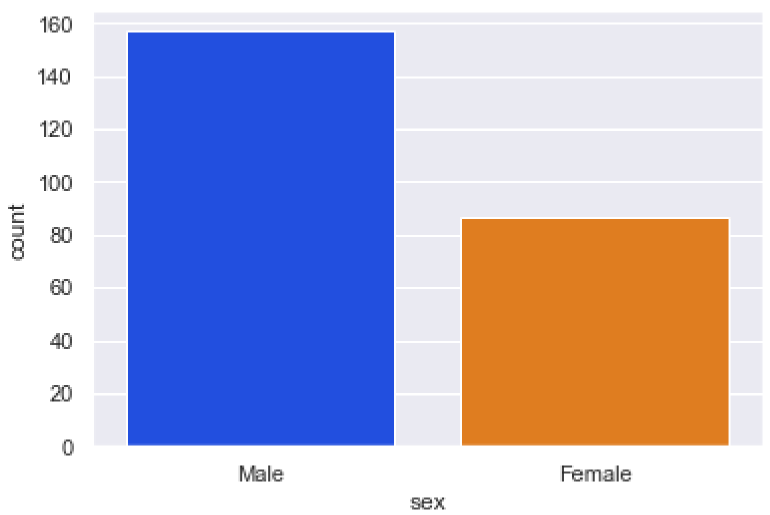


In [115]:

```
sns.set_style('darkgrid')
sns.countplot(x='sex', data=tips)
```

Out[115]:

<matplotlib.axes._subplots.AxesSubplot at 0x1cf77a2cdc8>

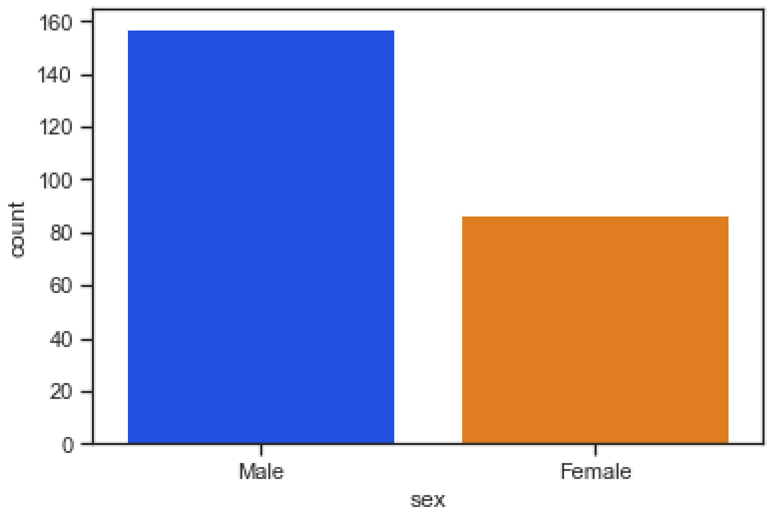


In [116]:

```
sns.set_style('ticks')
sns.countplot(x='sex', data=tips)
```

Out[116]:

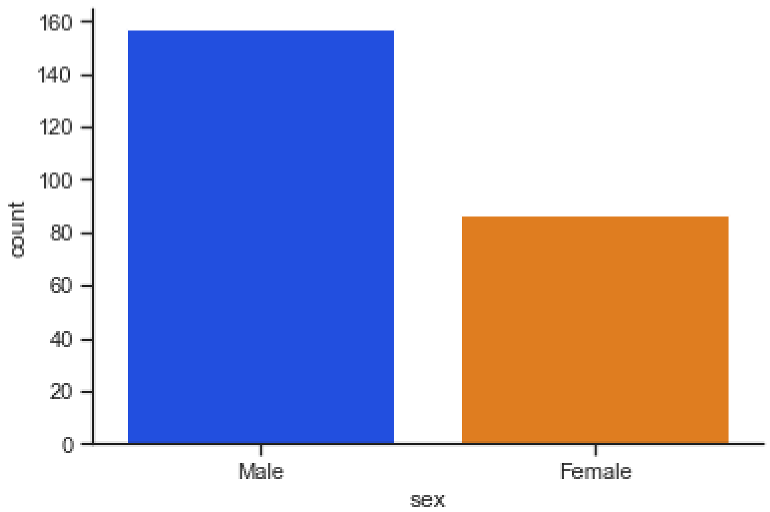
```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf77aec208>
```



Spine Removal

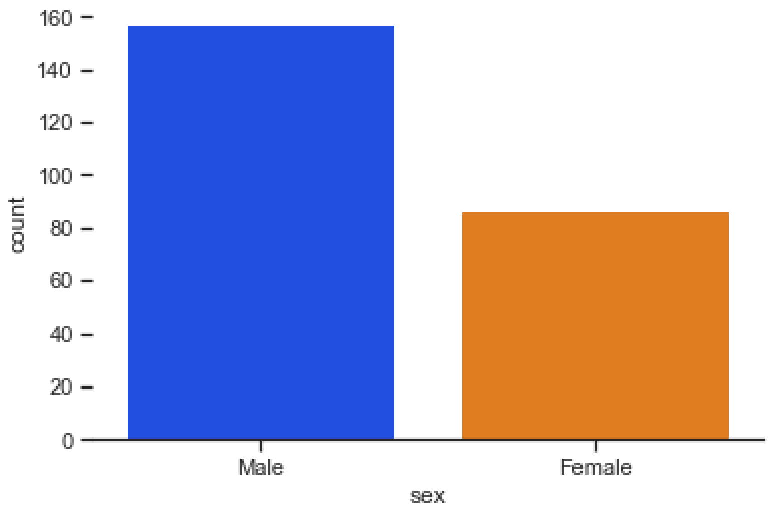
In [117]:

```
sns.countplot(x='sex', data=tips)
sns.despine()
```



In [119]:

```
sns.countplot(x='sex', data=tips)
sns.despine(left=True)
```



Size and Aspect

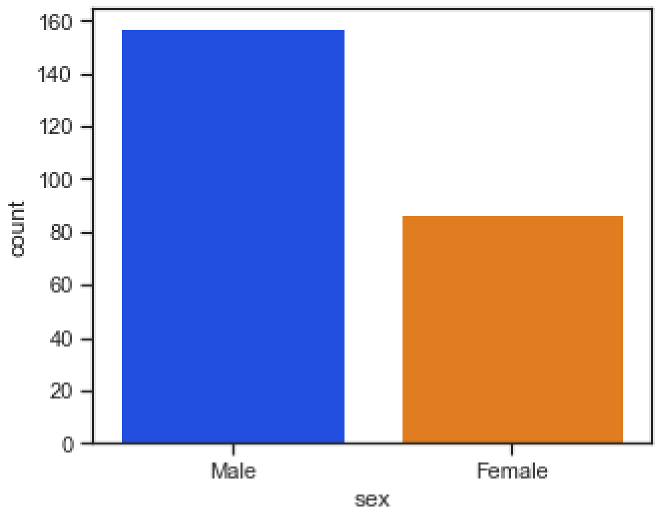
- We can use matplotlib's **plt.figure(figsize=(width,height)** to change the size of most seaborn plots.
- We can control the size and aspect ratio of most seaborn grid plots by passing in parameters: size, and aspect.

In [124]:

```
# Non Grid Plot

plt.figure(figsize=(5,4))
sns.countplot(x='sex', data=tips)
```

Out[124]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf77d5c788>
```

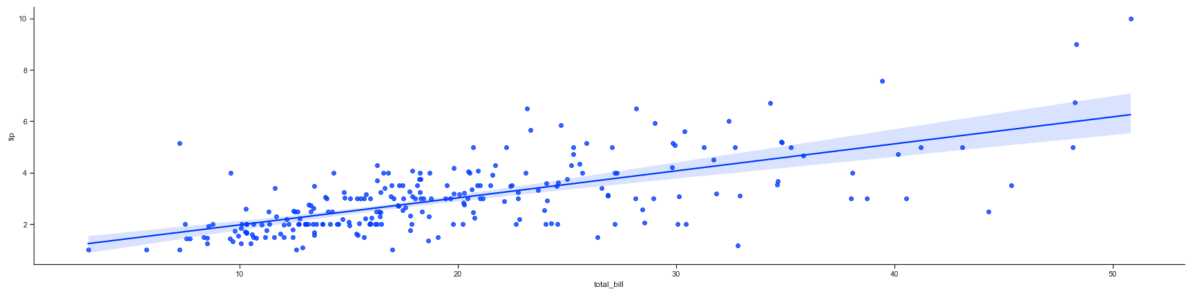


In [133]:

```
# Grid Type plot

sns.lmplot(x='total_bill', y='tip', data=tips, height=6, aspect= 4 )
```

Out[133]:

```
<seaborn.axisgrid.FacetGrid at 0x1cf72bd4108>
```

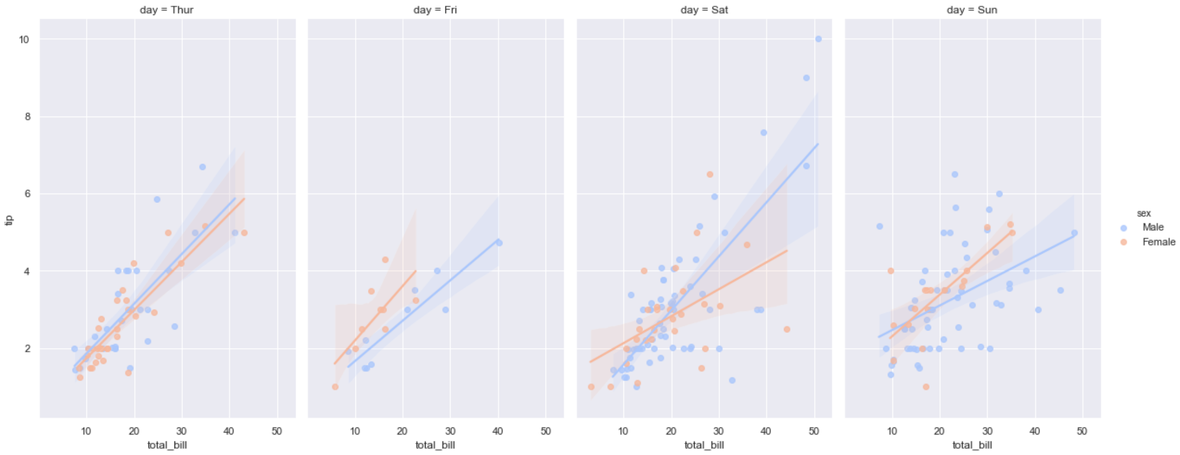


In [306]:

```
sns.lmplot(x='total_bill', y='tip', data=tips, col='day', hue='sex', palette='coolwarm', he
```

Out[306]:

```
<seaborn.axisgrid.FacetGrid at 0x1cf16844e08>
```



Scale and Context

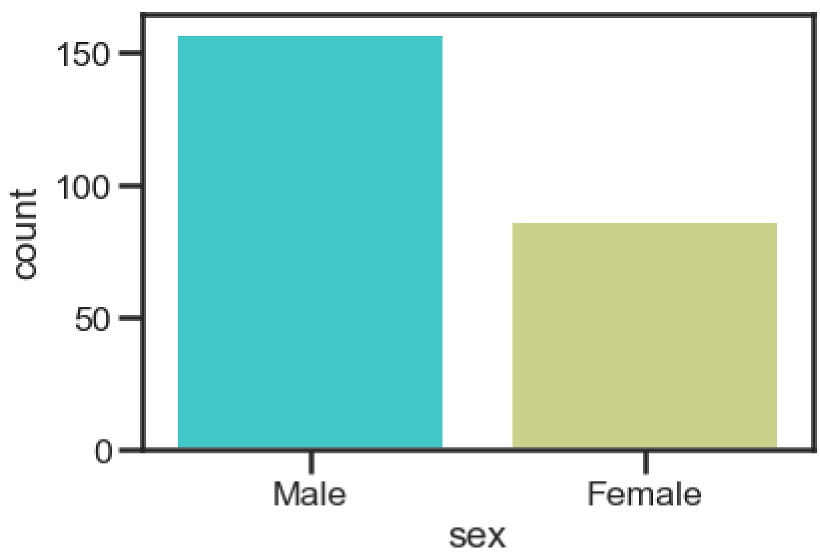
The set_context() allow us to override default parameters

In [144]:

```
sns.set_context('poster',font_scale=0.8)
sns.countplot(x='sex', data=tips, palette='rainbow')
```

Out[144]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cf7ff2fa48>
```



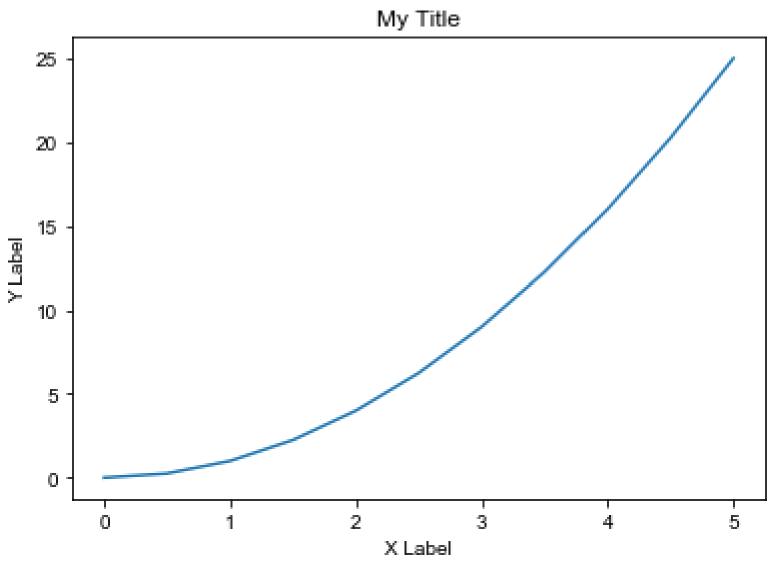
Check out the documentation page for more info on these topics: https://stanford.edu/~mwaskom/software/seaborn/tutorial/aesthetics.html (https://stanford.edu/~mwaskom/software/seaborn/tutorial/aesthetics.html)

Adding Secondary Plot

In [150]:

```
fig
```

Out[150]:



In [164]:

```
sns.set_style('ticks')
sns.set_context('paper',font_scale=1)
```

In [165]:

```
fig1 = plt.figure()

axes_1 = fig1.add_axes([0.1,0.1,0.8,0.8])
axes_2 = fig1.add_axes([0.2,0.5,0.3,0.3])

axes_1.plot(x,y)
axes_1.set_title('Large')

axes_2.plot(y,x)
axes_2.set_title('Small')
```

Out[165]:

```
Text(0.5, 1.0, 'Small')
```

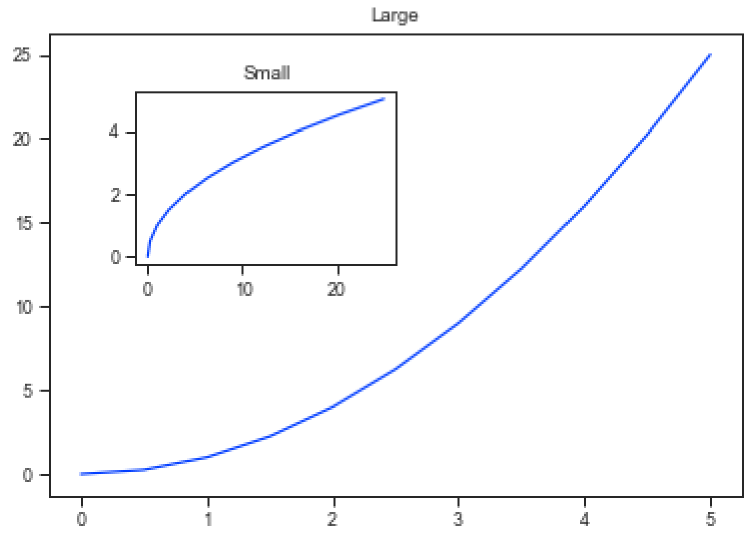
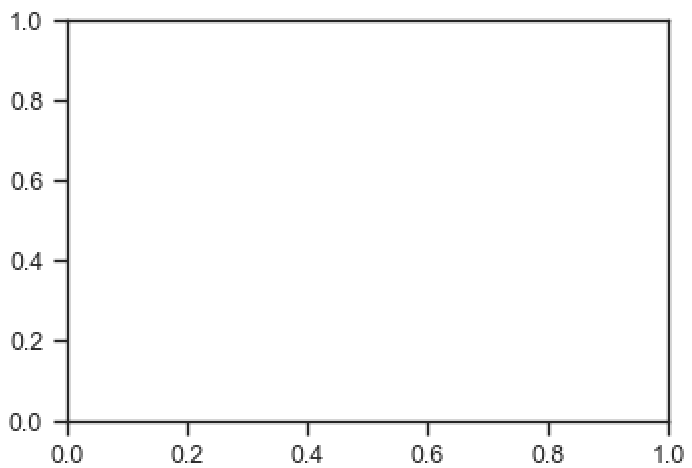


Figure Size and DPI (Dots per inch)

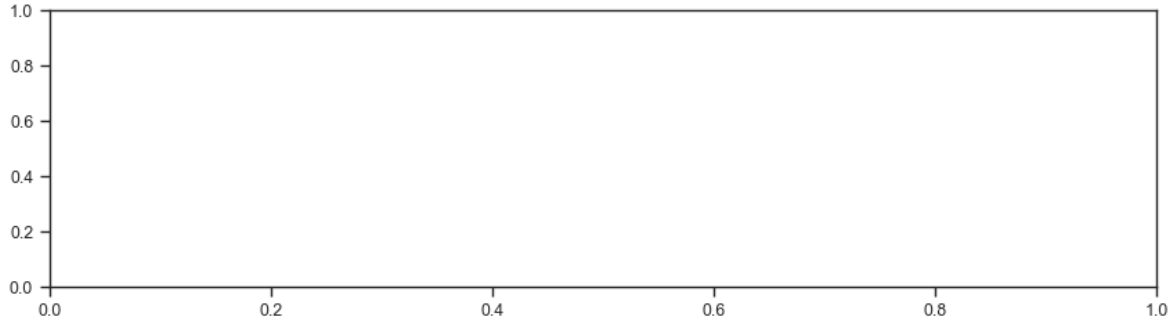
In [173]:

```
fig3 = plt.figure(figsize=(3,2), dpi=100)

ax = fig3.add_axes([0,0,1,1])
```



In [180]:

```
fig3 = plt.figure(figsize=(8,2), dpi=100)

ax = fig3.add_axes([0,0,1,1])
```

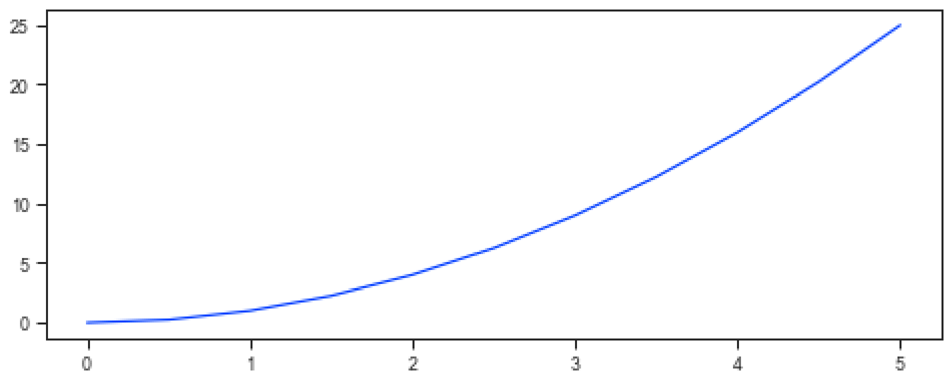


In [185]:

```
fig_sub1, ax1 = plt.subplots(figsize=(8,3))

ax1.plot(x,y)
```

Out[185]:

```
[<matplotlib.lines.Line2D at 0x1cf0333f3c8>]
```

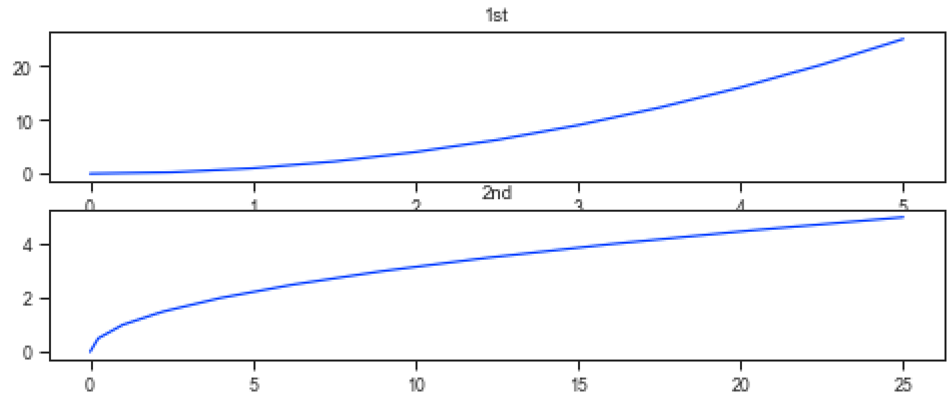


In [186]:

```
fig_sub2, ax2 = plt.subplots(nrows=2, ncols=1, figsize=(8,3))

ax2[0].plot(x,y)
ax2[0].set_title('1st')

ax2[1].plot(y,x)
ax2[1].set_title('2nd')
```

Out[186]:

```
Text(0.5, 1.0, '2nd')
```

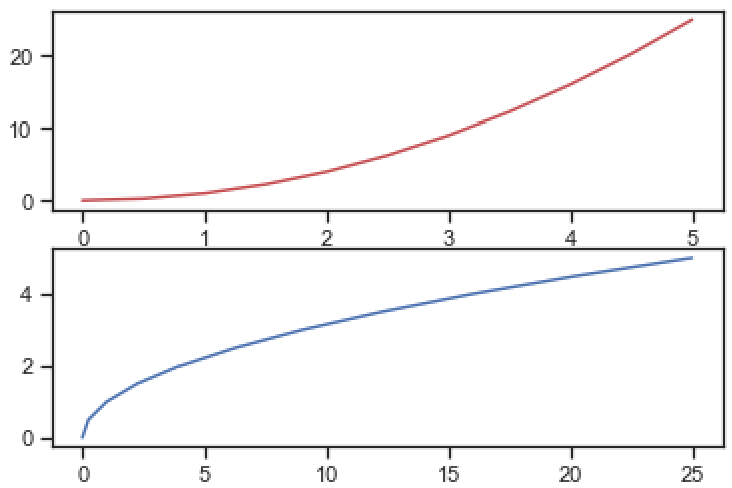


In [270]:

```
# Alternate way to create subplots

plt.subplot(2,1,1)
plt.plot(x,y,color='r')

plt.subplot(2,1,2)
plt.plot(y,x,color='b')
```

Out[270]:

```
[<matplotlib.lines.Line2D at 0x1cf047a48c8>]
```



In [189]:

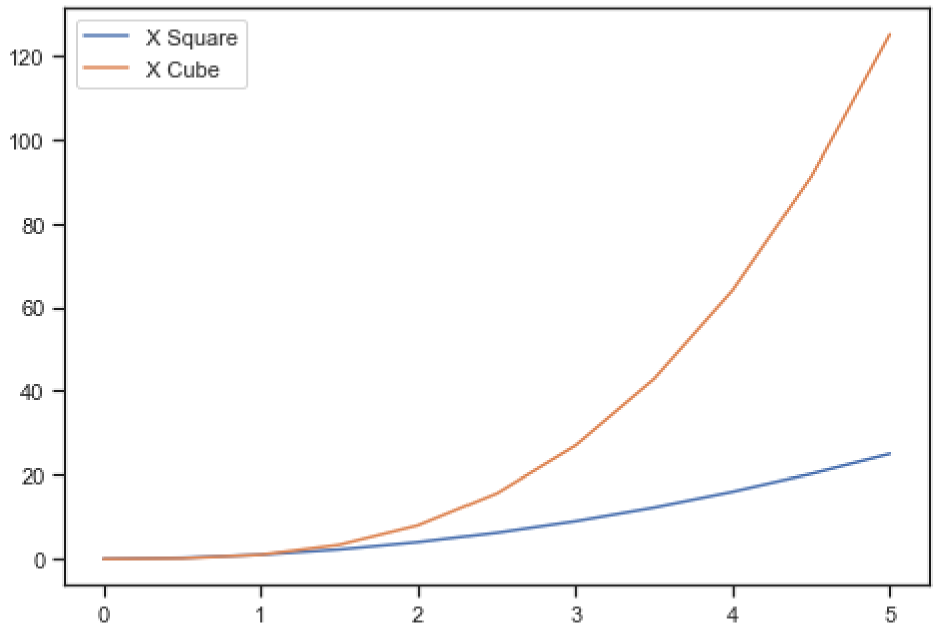
```
# Saving the the plot as Image file

fig1.savefig('My Chart.jpeg',dpi= 1200)
```

In [226]:

```
fig4 = plt.figure()

axes4 = fig4.add_axes([0,0,1,1])

axes4.plot(x, x**2, label = 'X Square')
axes4.plot(x, x**3, label = 'X Cube')

axes4.legend(loc = 0) # can set the location of the legend (0 to 10)

# Custom location of legend can be set by passing loc as axis notation is ()
# axes4.legend( loc = (0.1,0.1) )
```

Out[226]:

```
<matplotlib.legend.Legend at 0x1cf004231c8>
```



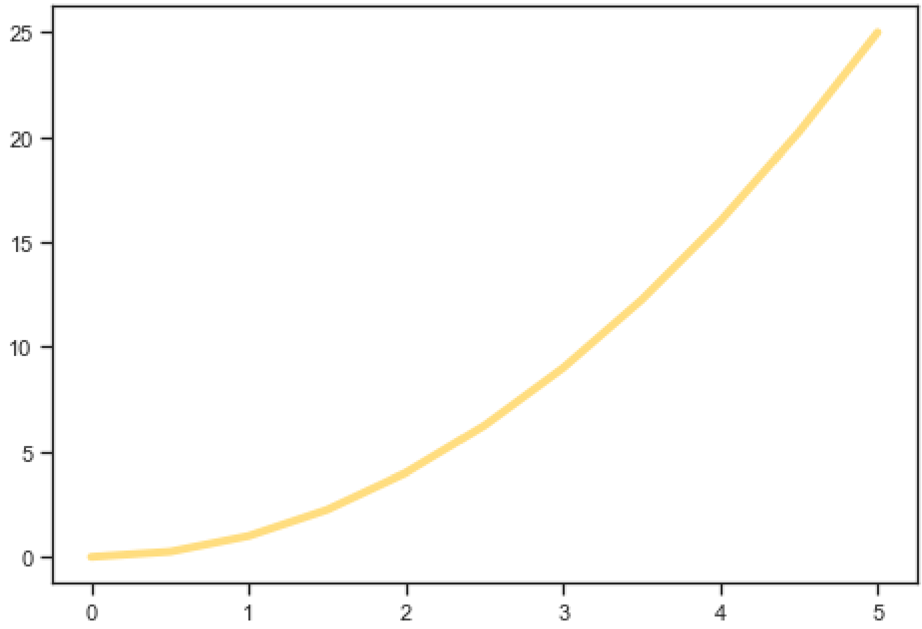
Plot Apperance

In [231]:

```
fig5 = plt.figure()

axes5 = fig5.add_axes([0,0,1,1])

axes5.plot(x, y, color='green', linewidth=4, alpha=0.5 ) # alpha is transperency

# axes5.plot(x, y, color='green', lw=4, alpha=0.5 ) # lw is an alias for linewidth
# axes5.plot(x, y, color='#FFBC00', lw=4, alpha=0.5 ) # RBG Hex Code
```

Out[231]:

```
[<matplotlib.lines.Line2D at 0x1cf72c1fd88>]
```

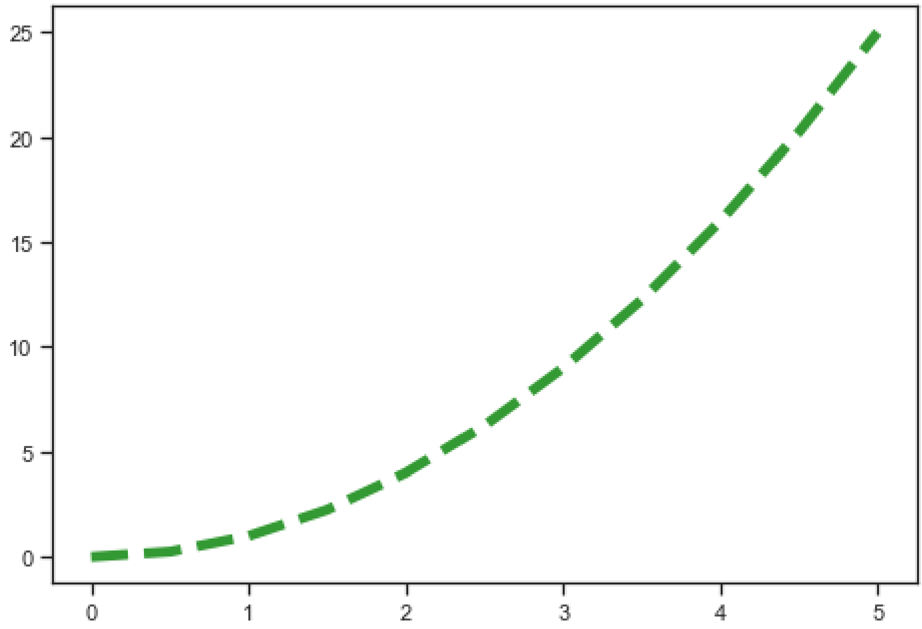


In [238]:

```
fig6 = plt.figure()

axes6 = fig6.add_axes([0,0,1,1])

axes6.plot(x, y, color='green', lw = 5, linestyle= '--', alpha= 0.8 )

# axes6.plot(x, y, color='green', lw=5, ls= '-.' ) # ls is an alias for linestyle
# axes6.plot(x, y, color='green', lw=5, ls= 'steps' )
```

Out[238]:

```
[<matplotlib.lines.Line2D at 0x1cf00434d48>]
```

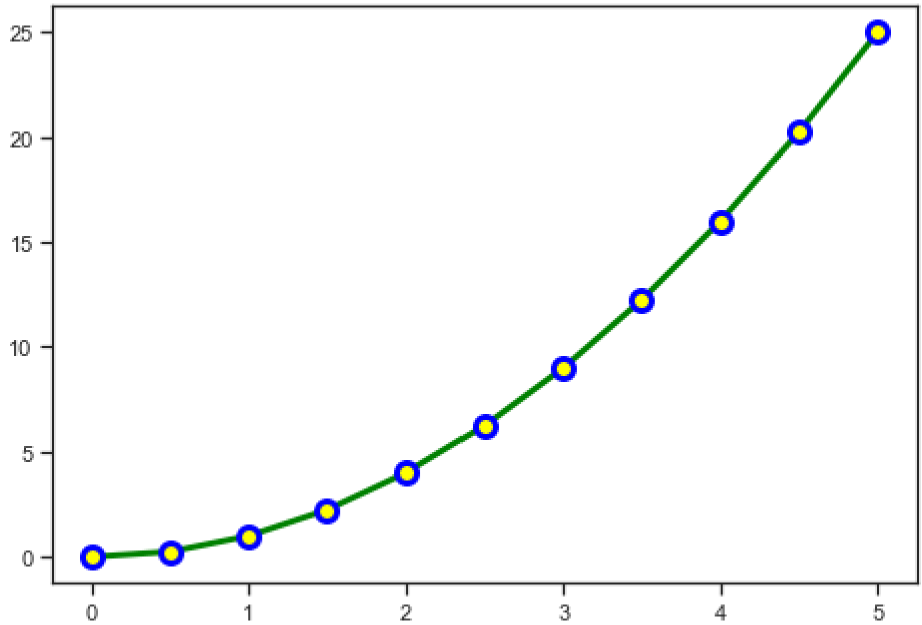


In [257]:

```
fig7= plt.figure()

axes7 = fig7.add_axes([0,0,1,1])

axes7.plot(x, y, color='green', lw=3 , ls='-', marker='o', markersize=10, markerfacecolor='
           markeredgewidth=3, markeredgecolor='blue' )

#axes7.plot(x,y,color='green', lw=3, ls = '-', marker = 'o')
#axes7.plot(x,y,color='green', lw=3, ls = '-', marker = '*')
#axes7.plot(x,y,color='green', lw=3, ls = '-', marker = 'o', markersize = 5)
```

Out[257]:

```
[<matplotlib.lines.Line2D at 0x1cf04210b08>]
```

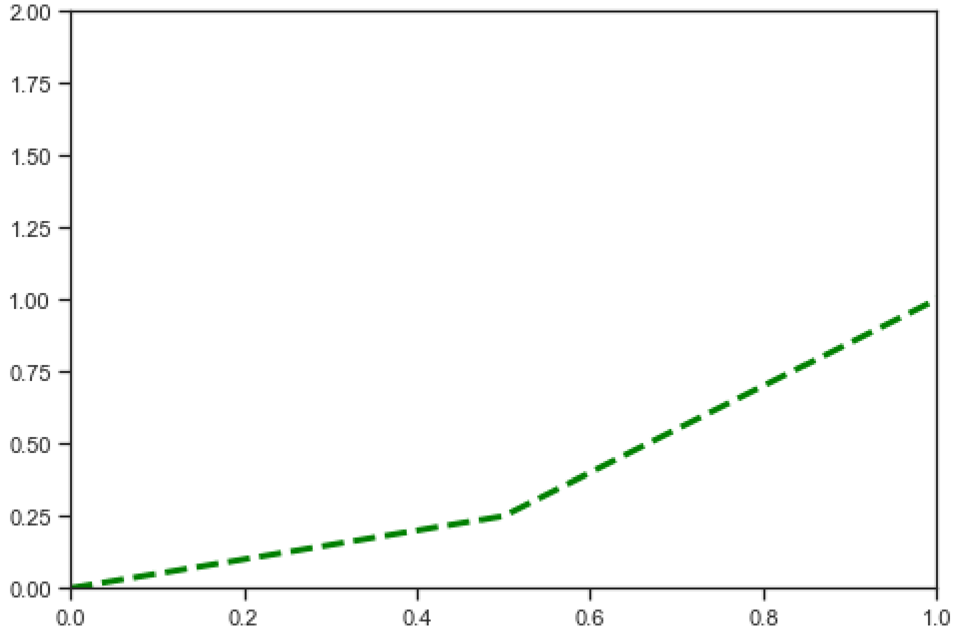


In [264]:

```
fig8 = plt.figure()

axes8 = fig8.add_axes([0,0,1,1])

axes8.plot(x, y, color='green', lw=3, ls='--' )

axes8.set_xlim([0,1])
axes8.set_ylim([0,2])
```

Out[264]:

```
(0, 2)
```



Visualization Exercise

In [84]:

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

%matplotlib inline
```

In [2]:

```
chipo = pd.read_table('chipotle.tsv.txt')
chipo.head()
```

Out[2]:

	order_id	quantity	item_name	choice_description	item_price
0	1	1	Chips and Fresh Tomato Salsa	NaN	$2.39
1	1	1	Izze	[Clementine]	$3.39
2	1	1	Nantucket Nectar	[Apple]	$3.39
3	1	1	Chips and Tomatillo-Green Chili Salsa	NaN	$2.39
4	2	2	Chicken Bowl	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	$16.98

In [3]:

```
chipo.dtypes
```

Out[3]:

```
order_id               int64
quantity               int64
item_name             object
choice_description    object
item_price            object
dtype: object
```

In [4]:

```
# Changing Data type of 'item_price' to Float

chipo['item_price'] = [float(x[1:-1]) for x in chipo['item_price']]
chipo.dtypes
```

Out[4]:

```
order_id                int64
quantity                int64
item_name              object
choice_description     object
item_price            float64
dtype: object
```

Q. Creating a histogram of the top 5 items bought

In [12]:

```
import collections

# Collections library have a class called Counter which helps us create a dictinary with ke
```

In [8]:

```
# Creating a series of item_name

s11 = chipo['item_name']
```

In [16]:

```
# Using counter class from dictionary from Collections library

item_count = collections.Counter(s11)
```

In [65]:

```
# Convert the Dictionary to DataFrame

df11 = pd.DataFrame.from_dict(item_count, orient='index',columns=['count'])
```

In [74]:

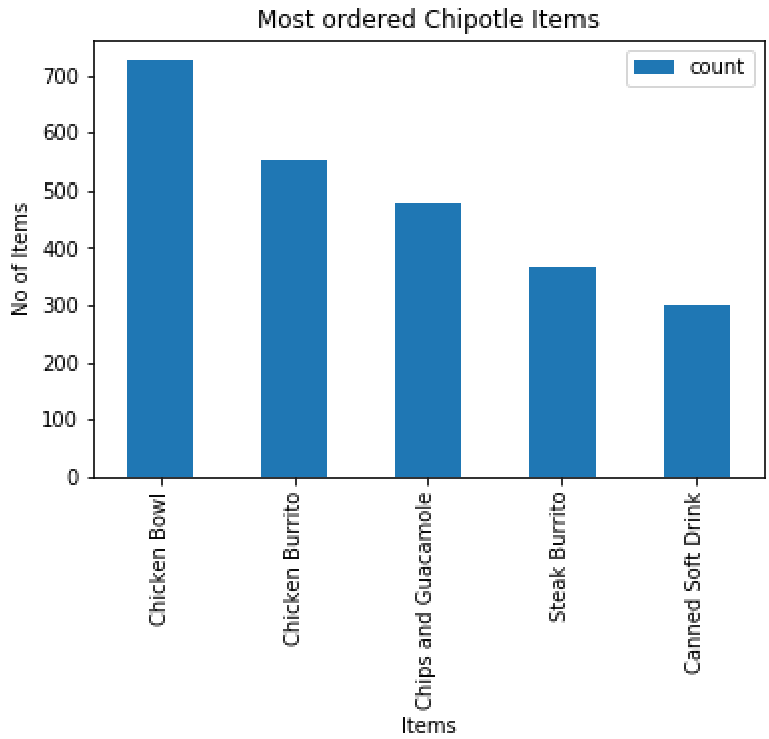
```
df11_top5 = df11.sort_values('count',ascending=False).head()
df11_top5
```

Out[74]:

	count
Chicken Bowl	726
Chicken Burrito	553
Chips and Guacamole	479
Steak Burrito	368
Canned Soft Drink	301

In [97]:

```
df11_top5.plot(kind='bar')

plt.xlabel('Items')
plt.ylabel('No of Items')
plt.title('Most ordered Chipotle Items')

plt.show()
```



Q. Create a scatterplot with the number of items orderered per order price

In [98]:

```
orders = chipo.groupby('order_id').sum()
```

In [99]:

```
plt.scatter(x=orders['item_price'], y=orders['quantity'], s=50, c='g',marker='o')

plt.xlabel('Item Price')
plt.ylabel('Items Ordered')
plt.title('Items Ordered per Item Price')
plt.ylim(0)

plt.show()
```