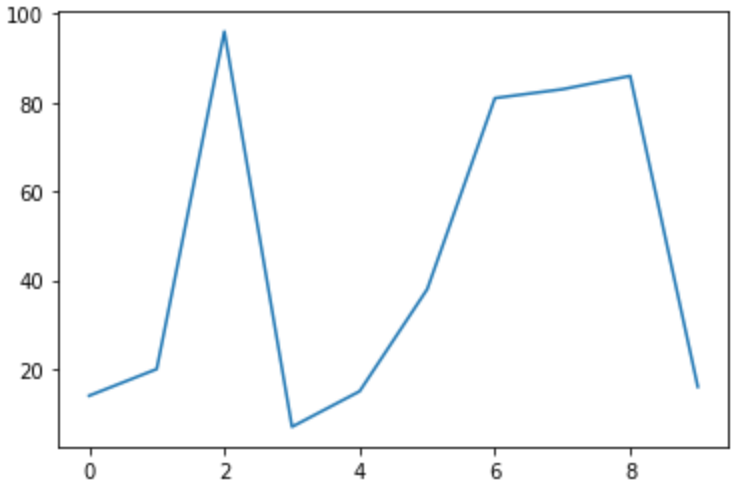
Analysis Univariate Analysis-Histogram, Line Chart, Boxplot, displot Bi- Variate Analysis - Scatter plot, line chart, bar chart, regression plot, join plot Multivariate Analysis- Heatmap, pie chart, area chart, tree map, pair plot

In [3]:
```
# import the library
import matplotlib.pyplot as plt
%matplotlib inline
```

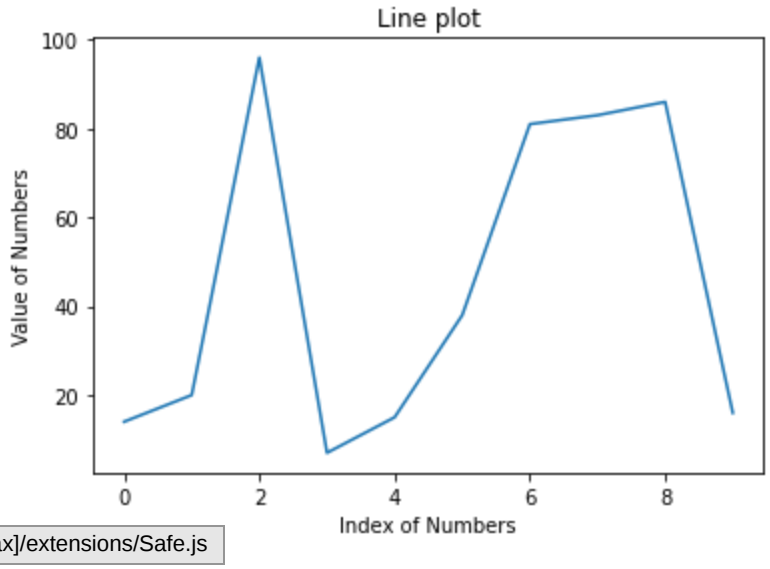
In [5]:
```
# plot function
import numpy as np
random_data=np.random.randint(0,100,10)
random_data
```

Out[5]:
```
array([14, 20, 96,  7, 15, 38, 81, 83, 86, 16])
```

In [6]:
```
# plot function
plt.plot(random_data)
plt.show()
```



In [9]:
```
# plot function
plt.plot(random_data)
plt.xlabel("Index of Numbers")
plt.ylabel("Value of Numbers")
plt.title("Line plot")
plt.show()
```



In [10]:

```
help(plt.plot)
```

```
Help on function plot in module matplotlib.pyplot:

plot(*args, scalex=True, scaley=True, data=None, **kwargs)
    Plot y versus x as lines and/or markers.

    Call signatures::

        plot([x], y, [fmt], *, data=None, **kwargs)
        plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)

    The coordinates of the points or line nodes are given by *x*, *y*.

    The optional parameter *fmt* is a convenient way for defining basic
    formatting like color, marker and linestyle. It's a shortcut string
    notation described in the *Notes* section below.

    >>> plot(x, y)        # plot x and y using default line style and color
    >>> plot(x, y, 'bo')  # plot x and y using blue circle markers
    >>> plot(y)           # plot y using x as index array 0..N-1
    >>> plot(y, 'r+')     # ditto, but with red plusses

    You can use `.Line2D` properties as keyword arguments for more
    control on the appearance. Line properties and *fmt* can be mixed.
    The following two calls yield identical results:

    >>> plot(x, y, 'go--', linewidth=2, markersize=12)
    >>> plot(x, y, color='green', marker='o', linestyle='dashed',
    ...      linewidth=2, markersize=12)

    When conflicting with *fmt*, keyword arguments take precedence.


    **Plotting labelled data**

    There's a convenient way for plotting objects with labelled data (i.e.
    data that can be accessed by index ``obj['y']``). Instead of giving
    the data in *x* and *y*, you can provide the object in the *data*
    parameter and just give the labels for *x* and *y*::

    >>> plot('xlabel', 'ylabel', data=obj)

    All indexable objects are supported. This could e.g. be a `dict`, a
    `pandas.DataFrame` or a structured numpy array.


    **Plotting multiple sets of data**

    There are various ways to plot multiple sets of data.

    - The most straight forward way is just to call `plot` multiple times.
      Example:

      >>> plot(x1, y1, 'bo')
      >>> plot(x2, y2, 'go')

    - If *x* and/or *y* are 2D arrays a separate data set will be drawn
      for every column. If both *x* and *y* are 2D, they must have the
      same shape. If only one of them is 2D with shape (N, m) the other
      must have length N and will be used for every data set m.

      Example:
```

```
  >>> x = [1, 2, 3]
  >>> y = np.array([[1, 2], [3, 4], [5, 6]])
  >>> plot(x, y)

  is equivalent to:

  >>> for col in range(y.shape[1]):
  ...     plot(x, y[:, col])

- The third way is to specify multiple sets of *[x]*, *y*, *[fmt]*
  groups::

  >>> plot(x1, y1, 'g^', x2, y2, 'g-')

  In this case, any additional keyword argument applies to all
  datasets. Also this syntax cannot be combined with the *data*
  parameter.

By default, each line is assigned a different style specified by a
'style cycle'. The *fmt* and line property parameters are only
necessary if you want explicit deviations from these defaults.
Alternatively, you can also change the style cycle using
:rc:`axes.prop_cycle`.


Parameters
----------
x, y : array-like or scalar
    The horizontal / vertical coordinates of the data points.
    *x* values are optional and default to ``range(len(y))``.

    Commonly, these parameters are 1D arrays.

    They can also be scalars, or two-dimensional (in that case, the
    columns represent separate data sets).

    These arguments cannot be passed as keywords.

fmt : str, optional
    A format string, e.g. 'ro' for red circles. See the *Notes*
    section for a full description of the format strings.

    Format strings are just an abbreviation for quickly setting
    basic line properties. All of these and more can also be
    controlled by keyword arguments.

    This argument cannot be passed as keyword.

data : indexable object, optional
    An object with labelled data. If given, provide the label names to
    plot in *x* and *y*.

    .. note::
        Technically there's a slight ambiguity in calls where the
        second label is a valid *fmt*. ``plot('n', 'o', data=obj)``
        could be ``plt(x, y)`` or ``plt(y, fmt)``. In such cases,
        the former interpretation is chosen, but a warning is issued.
        You may suppress the warning by adding an empty format string
        ``plot('n', 'o', '', data=obj)``.

Returns
-------
list of `.Line2D`
```

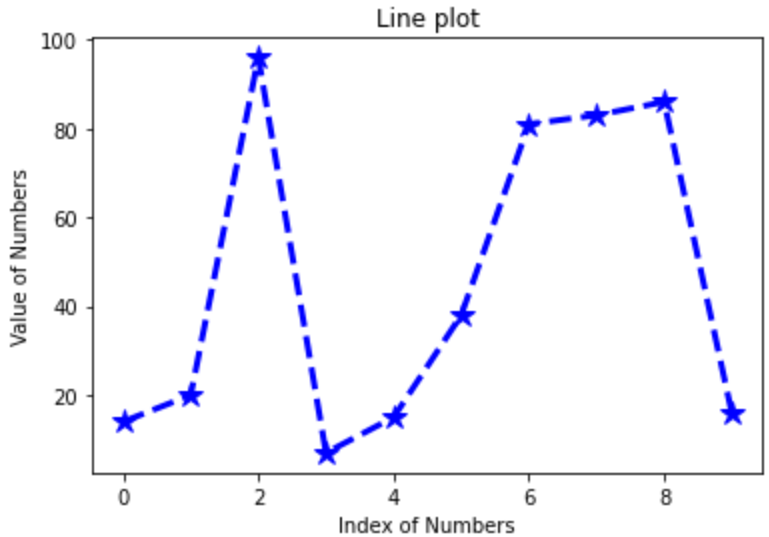
```
        A list of lines representing the plotted data.

    Other Parameters
    ----------------
    scalex, scaley : bool, default: True
        These parameters determine if the view limits are adapted to the
        data limits. The values are passed on to `autoscale_view`.

    **kwargs : `.Line2D` properties, optional
        *kwargs* are used to specify properties like a line label (for
        auto legends), linewidth, antialiasing, marker face color.
        Example::

        >>> plot([1, 2, 3], [1, 2, 3], 'go-', label='line 1', linewidth=2)
        >>> plot([1, 2, 3], [1, 4, 9], 'rs', label='line 2')

        If you specify multiple lines with one plot call, the kwargs apply
        to all those lines. In case the label object is iterable, each
        element is used as labels for each set of data.

        Here is a list of available `.Line2D` properties:

        Properties:
        agg_filter: a filter function, which takes a (m, n, 3) float array and a dpi valu
e, and returns a (m, n, 3) array
        alpha: scalar or None
        animated: bool
        antialiased or aa: bool
        clip_box: `.Bbox`
        clip_on: bool
        clip_path: Patch or (Path, Transform) or None
        color or c: color
        contains: unknown
        dash_capstyle: `.CapStyle` or {'butt', 'projecting', 'round'}
        dash_joinstyle: `.JoinStyle` or {'miter', 'round', 'bevel'}
        dashes: sequence of floats (on/off ink in points) or (None, None)
        data: (2, N) array or two 1D arrays
        drawstyle or ds: {'default', 'steps', 'steps-pre', 'steps-mid', 'steps-post'}, def
ault: 'default'
        figure: `.Figure`
        fillstyle: {'full', 'left', 'right', 'bottom', 'top', 'none'}
        gid: str
        in_layout: bool
        label: object
        linestyle or ls: {'-', '--', '-.', ':', '', (offset, on-off-seq), ...}
        linewidth or lw: float
        marker: marker style string, `~.path.Path` or `~.markers.MarkerStyle`
        markeredgecolor or mec: color
        markeredgewidth or mew: float
        markerfacecolor or mfc: color
        markerfacecoloralt or mfcalt: color
        markersize or ms: float
        markevery: None or int or (int, int) or slice or list[int] or float or (float, flo
at) or list[bool]
        path_effects: `.AbstractPathEffect`
        picker: float or callable[[Artist, Event], tuple[bool, dict]]
        pickradius: float
        rasterized: bool
        sketch_params: (scale: float, length: float, randomness: float)
        snap: bool or None
        solid_capstyle: `.CapStyle` or {'butt', 'projecting', 'round'}
        solid_joinstyle: `.JoinStyle` or {'miter', 'round', 'bevel'}
        transform: `matplotlib.transforms.Transform`
        url: str
```

```
    visible: bool
    xdata: 1D array
    ydata: 1D array
    zorder: float

See Also
--------
scatter : XY scatter plot with markers of varying size and/or color (
    sometimes also called bubble chart).

Notes
-----
**Format Strings**

A format string consists of a part for color, marker and line::

    fmt = '[marker][line][color]'

Each of them is optional. If not provided, the value from the style
cycle is used. Exception: If ``line`` is given, but no ``marker``,
the data will be a line without markers.

Other combinations such as ``[color][marker][line]`` are also
supported, but note that their parsing may be ambiguous.

**Markers**

=============   ===============================
character       description
=============   ===============================
``'.'``         point marker
``','``         pixel marker
``'o'``         circle marker
``'v'``         triangle_down marker
``'^'``         triangle_up marker
``'<'``         triangle_left marker
``'>'``         triangle_right marker
``'1'``         tri_down marker
``'2'``         tri_up marker
``'3'``         tri_left marker
``'4'``         tri_right marker
``'8'``         octagon marker
``'s'``         square marker
``'p'``         pentagon marker
``'P'``         plus (filled) marker
``'*'``         star marker
``'h'``         hexagon1 marker
``'H'``         hexagon2 marker
``'+'``         plus marker
``'x'``         x marker
``'X'``         x (filled) marker
``'D'``         diamond marker
``'d'``         thin_diamond marker
``'|'``         vline marker
``'_'``         hline marker
=============   ===============================

**Line Styles**

=============    ===============================
character        description
=============    ===============================
``'-'``          solid line style
``'--'``         dashed line style
```

```
``'-.'``          dash-dot line style
``':'``           dotted line style
=============    ================================

Example format strings::

    'b'    # blue markers with default shape
    'or'   # red circles
    '-g'   # green solid line
    '--'   # dashed line with default color
    '^k:'  # black triangle_up markers connected by a dotted line

**Colors**

The supported color abbreviations are the single letter codes

=============    ================================
character        color
=============    ================================
``'b'``          blue
``'g'``          green
``'r'``          red
``'c'``          cyan
``'m'``          magenta
``'y'``          yellow
``'k'``          black
``'w'``          white
=============    ================================

and the ``'CN'`` colors that index into the default property cycle.

If the color is the only part of the format string, you can
additionally use any  `matplotlib.colors` spec, e.g. full names
(``'green'``) or hex strings (``'#008000'``).
```

In [12]:

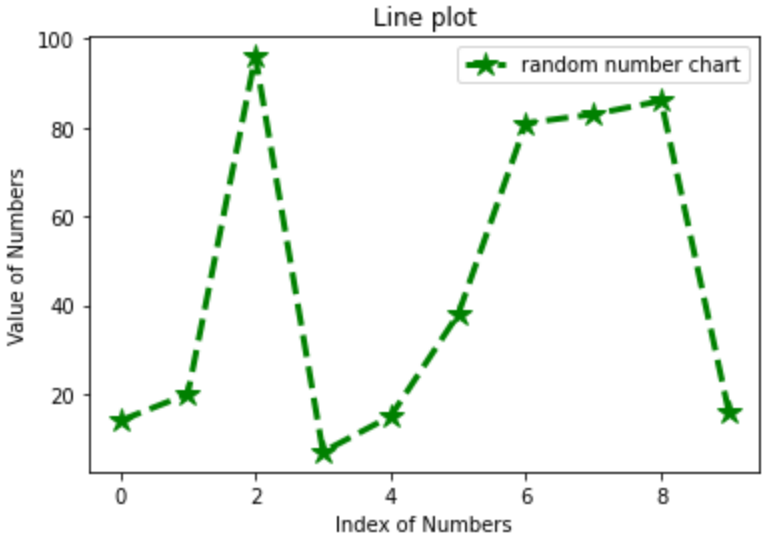
```
# plot function edited
plt.plot(random_data,color='b',linewidth=3,linestyle='--',marker='*',markersize=12)
plt.xlabel("Index of Numbers")
plt.ylabel("Value of Numbers")
plt.title("Line plot")
plt.show()
```



In [13]:

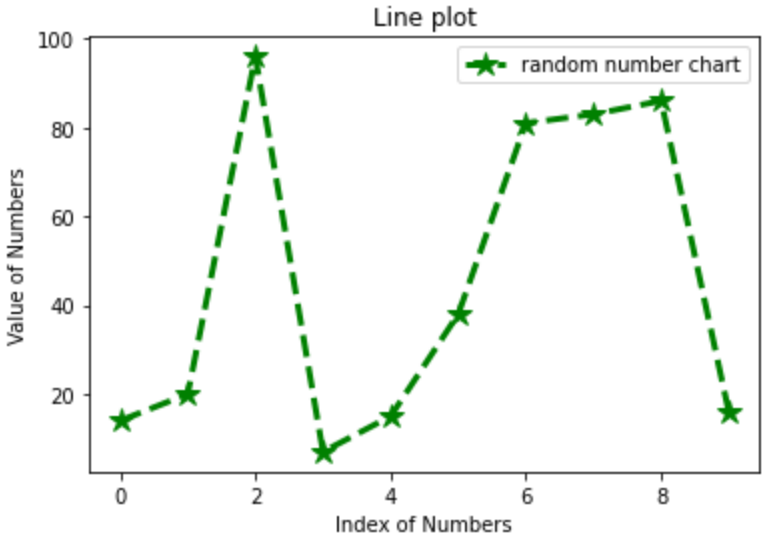
```
# plot function    legend
om_data,color='green',linewidth=3,linestyle='--',marker='*',markersize=12,lab
```

```
plt.xlabel("Index of Numbers")
plt.ylabel("Value of Numbers")
plt.title("Line plot")
plt.legend()
plt.show()
```



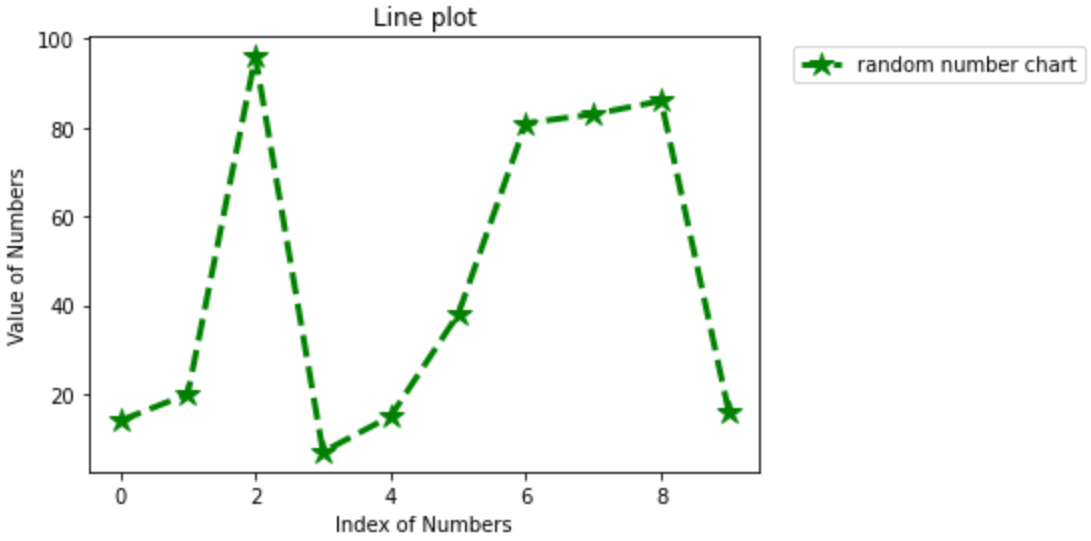
In [14]:

```
# position of legend
plt.plot(random_data,color='green',linewidth=3,linestyle='--',marker='*',markersize=12,lab
plt.xlabel("Index of Numbers")
plt.ylabel("Value of Numbers")
plt.title("Line plot")
plt.legend(loc='upper right')
plt.show()
```



In [15]:

```
plt.plot(random_data,color='green',linewidth=3,linestyle='--',marker='*',markersize=12,lab
plt.xlabel("Index of Numbers")
plt.ylabel("Value of Numbers")
plt.title("Line plot")
plt.legend(loc='upper right', bbox_to_anchor=(1.5, 1))
plt.show()
```

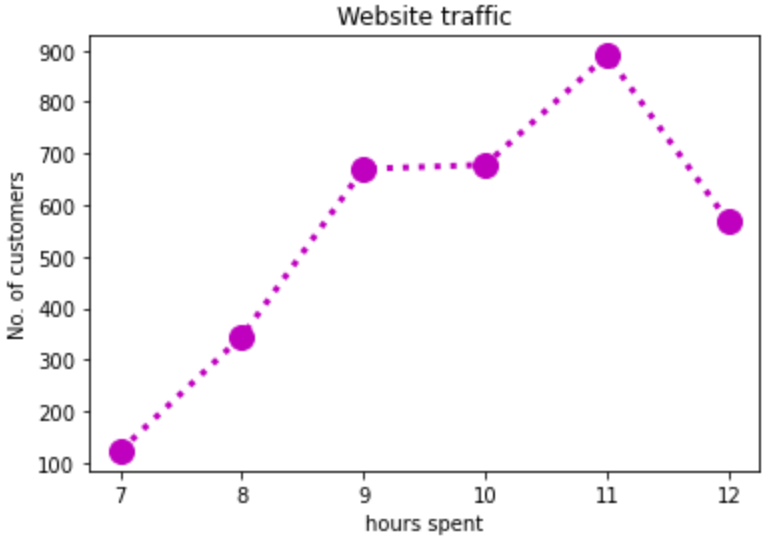


In [16]:

```
# line chart of 2 variable
# Line chart of two variables
web_customers=[123,345,670,678,890,567]
time_hrs=[7,8,9,10,11,12]
```

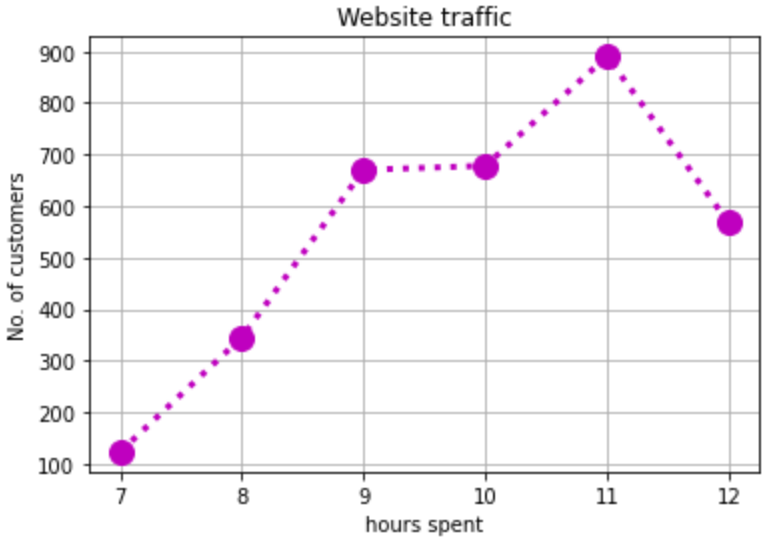
In [20]:

```
# line chart
# plot function
plt.plot(time_hrs,web_customers,color='m',linewidth=3,linestyle=':',marker='o',markersize=
plt.xlabel("hours spent")
plt.ylabel("No. of customers")
plt.title("Website traffic")
plt.show()
```



In [21]:

```
#grid
# line chart
# plot function
plt.plot(time_hrs,web_customers,color='m',linewidth=3,linestyle=':',marker='o',markersize=
plt.xlabel("hours spent")
plt.ylabel("No. of customers")
plt.grid(True)
plt.title("Website traffic")
plt.show()
```

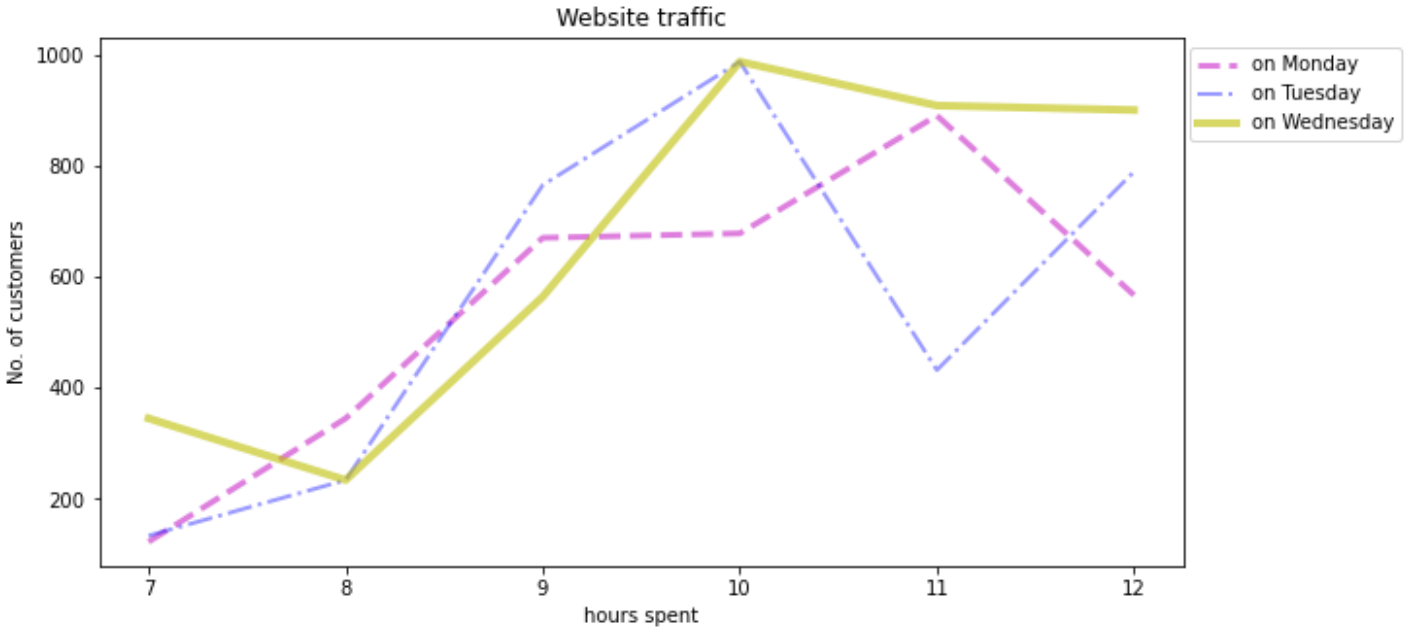


In [22]:

```
web_Monday=[123,345,670,678,890,567]
web_Tuesday=[134,234,765,987,432,789]
web_Wednesday=[345,234,564,987,908,900]
time_hrs=[7,8,9,10,11,12]
```

In [33]:

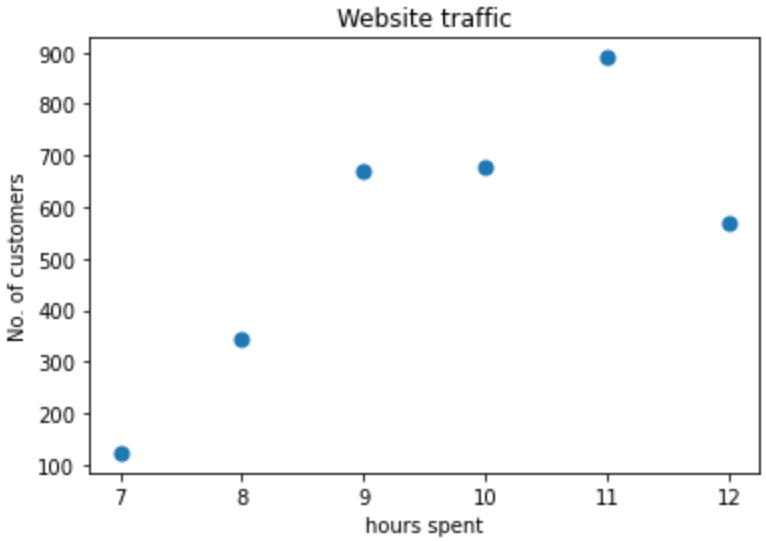
```
#create the plot
plt.figure(figsize=(10,5))
plt.plot(time_hrs,web_Monday,color='m',linewidth=3,linestyle='--',alpha=0.5,label="on Mond
plt.plot(time_hrs,web_Tuesday,color='b',linewidth=2,linestyle='-.',alpha=0.4,label="on Tue
# alpha is used for darkness
plt.plot(time_hrs,web_Wednesday,color='y',linewidth=4,linestyle='-',alpha=0.6,label="on We
plt.xlabel("hours spent")
plt.ylabel("No. of customers")
plt.title("Website traffic")
plt.legend(loc='upper right', bbox_to_anchor=(1.21, 1))
plt.show()
```



In [36]:

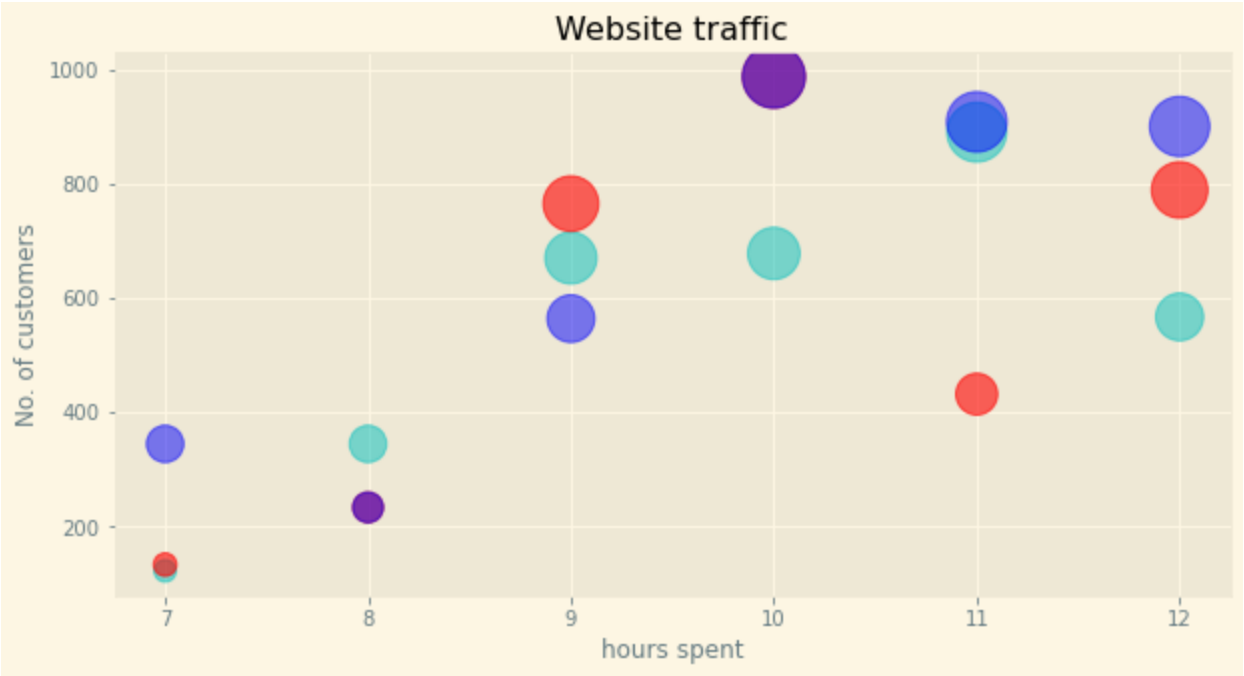
```
# scatter plot
web_customers=[123,345,670,678,890,567]
time_hrs=[7,8,9,10,11,12]
plt.scatter(time_hrs,web_customers,s=50)
plt.xlabel("hours spent")
plt.ylabel("No. of customers")
```

```
plt.title("Website traffic")
plt.show()
```



In [41]:

```
#create the plot
plt.figure(figsize=(10,5))

plt.scatter(time_hrs,web_Monday,color='c',alpha=0.5,s=web_Monday)

plt.scatter(time_hrs,web_Tuesday,color='r',alpha=0.6,s=web_Tuesday)

plt.scatter(time_hrs,web_Wednesday,color='b',alpha=0.5,s=web_Wednesday)
plt.xlabel("hours spent")
plt.ylabel("No. of customers")
plt.title("Website traffic")
plt.grid(True)
plt.show() # alpha- transperancy
```



In [38]:

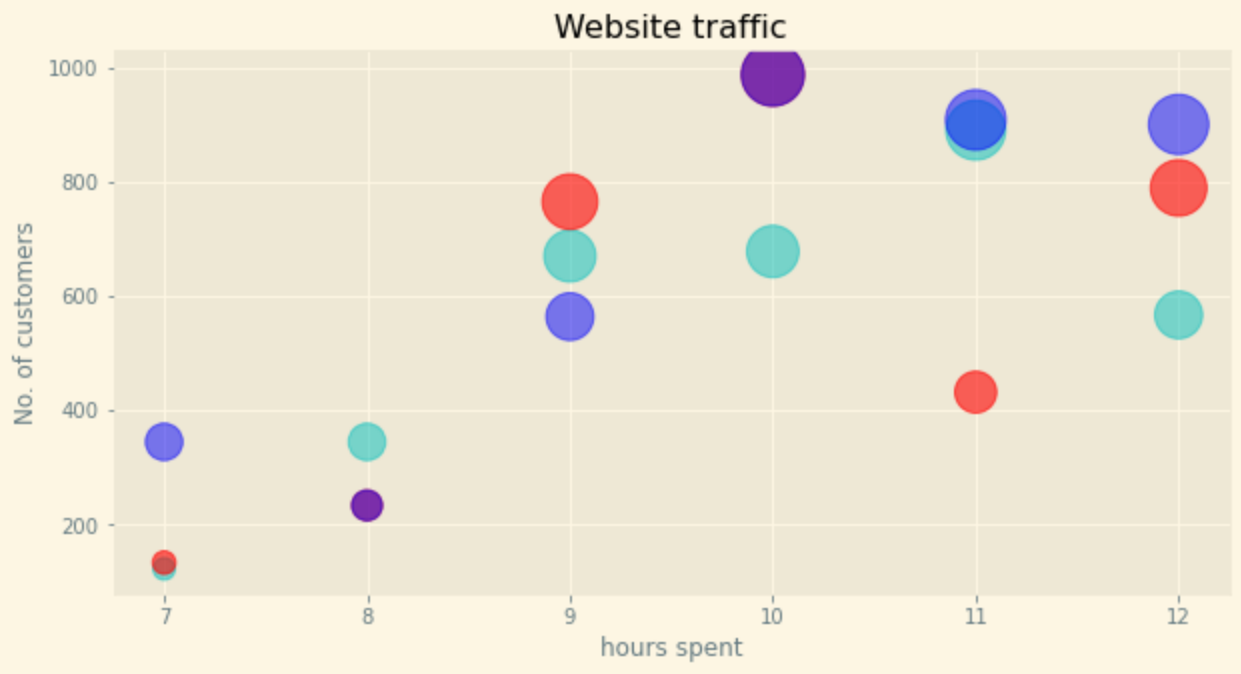
```
from matplotlib import style
print(plt.style.available)
```

```
['Solarize_Light2', '_classic_test_patch', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn', 'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark-palette', 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-mute
```

```
d', 'seaborn-notebook', 'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-tal
k', 'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'tableau-colorblind10']
```

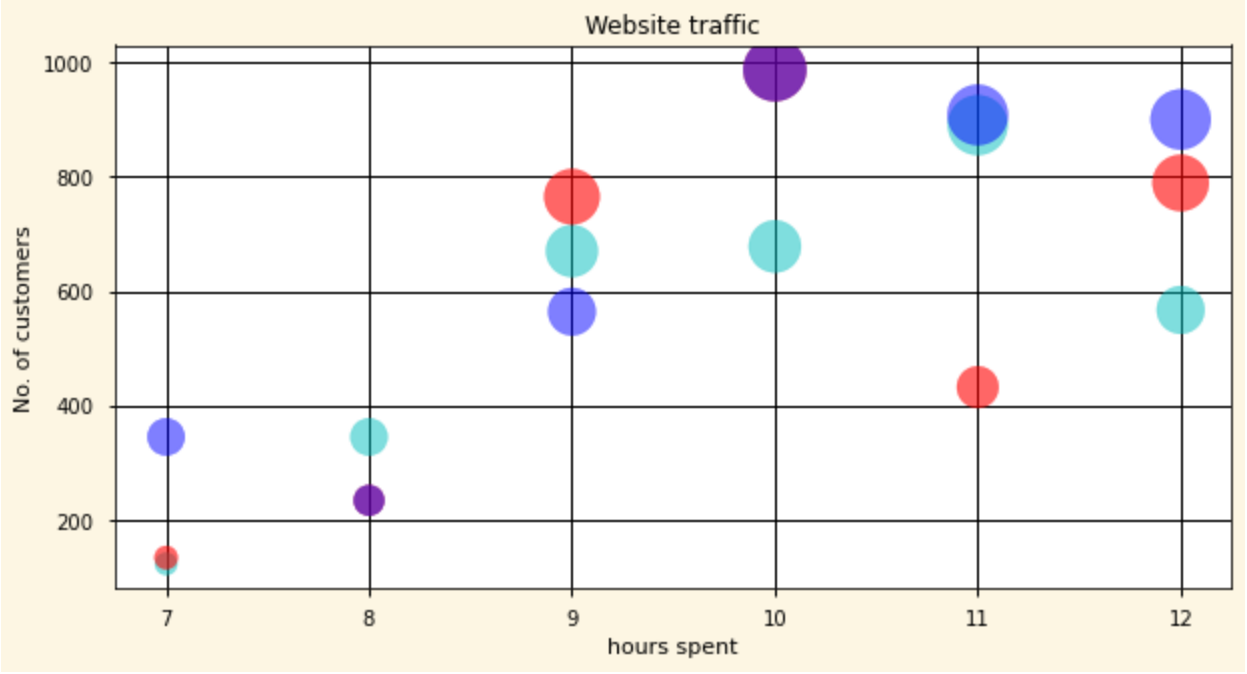
In [42]:

```
#create the plot
plt.figure(figsize=(10,5))
style.use('Solarize_Light2')
plt.scatter(time_hrs,web_Monday,color='c',alpha=0.5,s=web_Monday)

plt.scatter(time_hrs,web_Tuesday,color='r',alpha=0.6,s=web_Tuesday)

plt.scatter(time_hrs,web_Wednesday,color='b',alpha=0.5,s=web_Wednesday)
plt.xlabel("hours spent")
plt.ylabel("No. of customers")
plt.title("Website traffic")
plt.grid(True)
plt.show() # alpha- transperancy
```



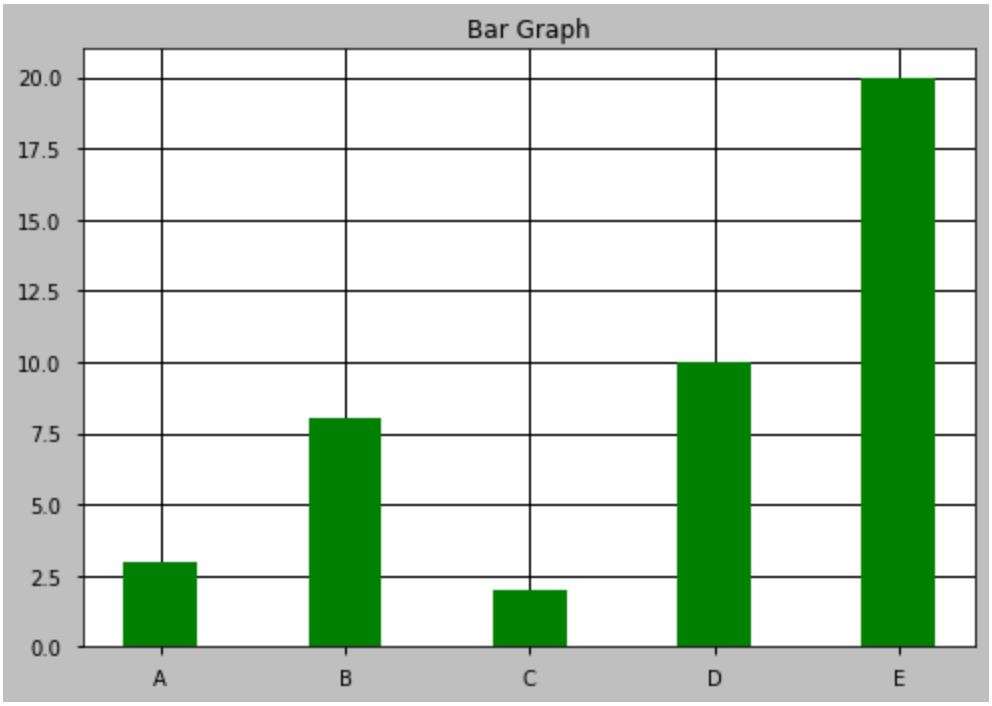
In [45]:

```
#create the plot
plt.figure(figsize=(10,5))
style.use('grayscale')
plt.scatter(time_hrs,web_Monday,color='c',alpha=0.5,s=web_Monday)

plt.scatter(time_hrs,web_Tuesday,color='r',alpha=0.6,s=web_Tuesday)

plt.scatter(time_hrs,web_Wednesday,color='b',alpha=0.5,s=web_Wednesday)
plt.xlabel("hours spent")
plt.ylabel("No. of customers")
plt.title("Website traffic")
plt.grid(True)
plt.show() # alpha- transperancy
```



In [46]:

```
# Bar plot
X=np.array(['A','B','C','D','E'])
Y=np.array([3,8,2,10,20])
plt.bar(X,Y,color='g',width=0.4)
plt.title("Bar Graph")
plt.show()
```



In [47]:

```
# Create Histogram
x_rand=np.random.randn(50)
x_rand
```

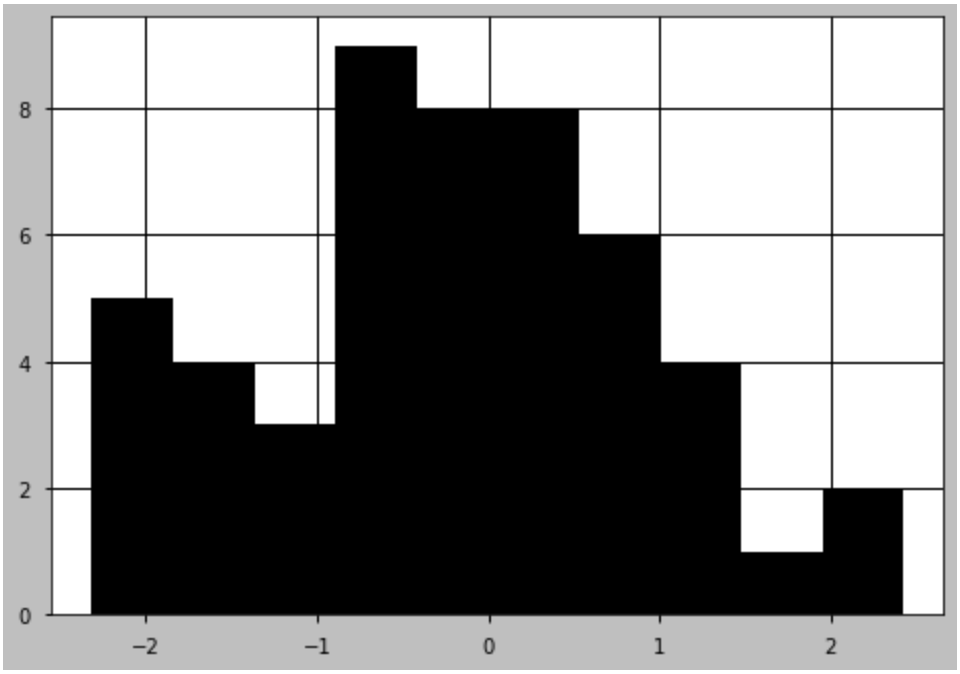
Out[47]:

```
array([-0.72020995, -0.94812865, -0.40486828,  0.88634421, -1.6448569 ,
       -0.25307971,  0.72493715,  0.81874883, -2.31226838, -1.52819908,
        0.26534665,  1.46018505,  0.23801219, -0.45388689,  0.30684697,
       -0.66530458, -0.56274453,  0.68451215, -1.46136816,  1.8499696 ,
       -0.82917039,  1.12988641,  2.42357016, -2.18284187,  0.0287505 ,
       -1.10666081, -0.35747044,  0.23683887, -0.62432983, -1.26040977,
       -0.87450205, -1.87287387,  0.50443593, -0.59987932,  1.98746318,
       -0.1081, -0.13364463, -1.69023618, -2.19349927,  0.07100418,
```

```
       0.40046242, -1.85353037, -0.65561501,  1.09981469,  0.01358706,
       1.12405838,  0.75930503, -0.34894011,  0.71754337,  0.0195008 ])
```

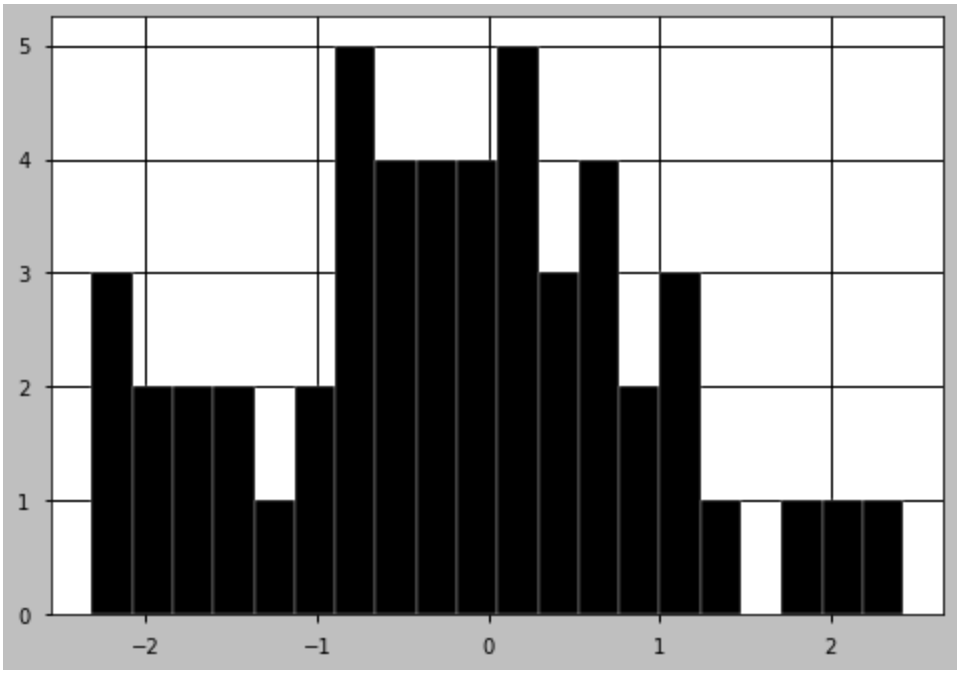
In [48]:

```
plt.hist(x_rand)
plt.show()
```



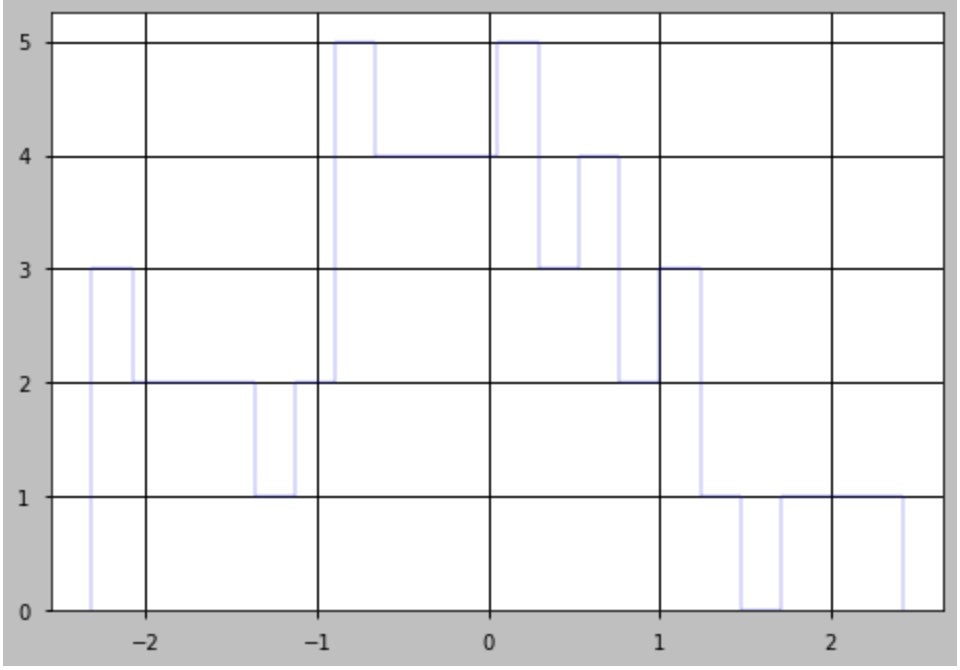
In [49]:

```
plt.hist(x_rand,bins=20,edgecolor='white')
plt.show()
```



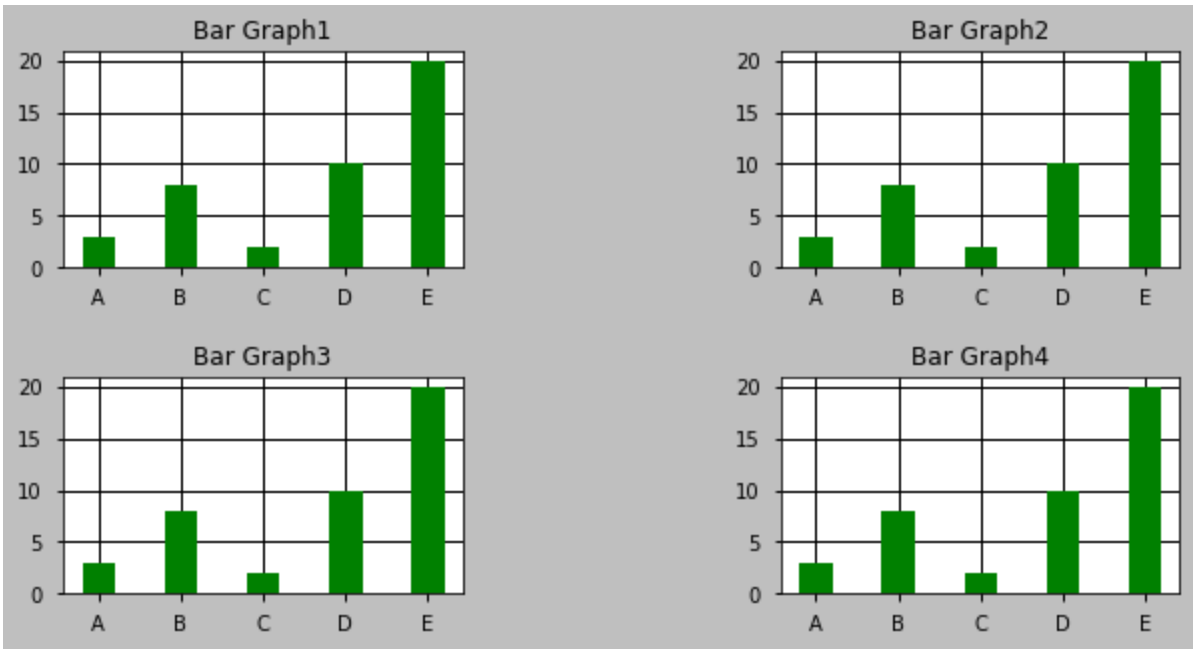
In [51]:

```
plt.hist(x_rand,bins=20,edgecolor='blue',histtype='step')
plt.show()
```



In [53]:

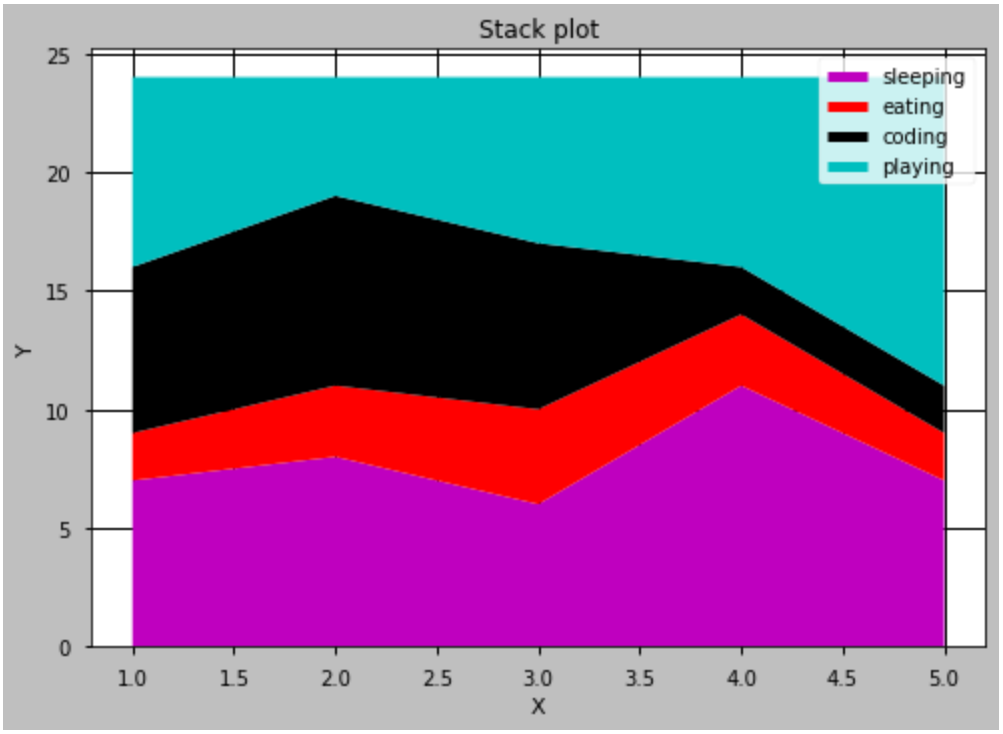
```
plt.figure(figsize=(10,5)) # size of graph
plt.subplots_adjust(hspace=0.5, wspace=0.8) # hspace- space between rows, wspace= columns
# 2 plot in 2 rows
plt.subplot(2,2,1)
plt.bar(X,Y,color='g',width=0.4)
plt.title("Bar Graph1")
plt.subplot(2,2,2)
plt.bar(X,Y,color='g',width=0.4)
plt.title("Bar Graph2")
plt.subplot(2,2,3)
plt.bar(X,Y,color='g',width=0.4)
plt.title("Bar Graph3")
plt.subplot(2,2,4)
plt.bar(X,Y,color='g',width=0.4)
plt.title("Bar Graph4")
plt.savefig('subplot.png') # to save figure
plt.show()
```



In [55]:

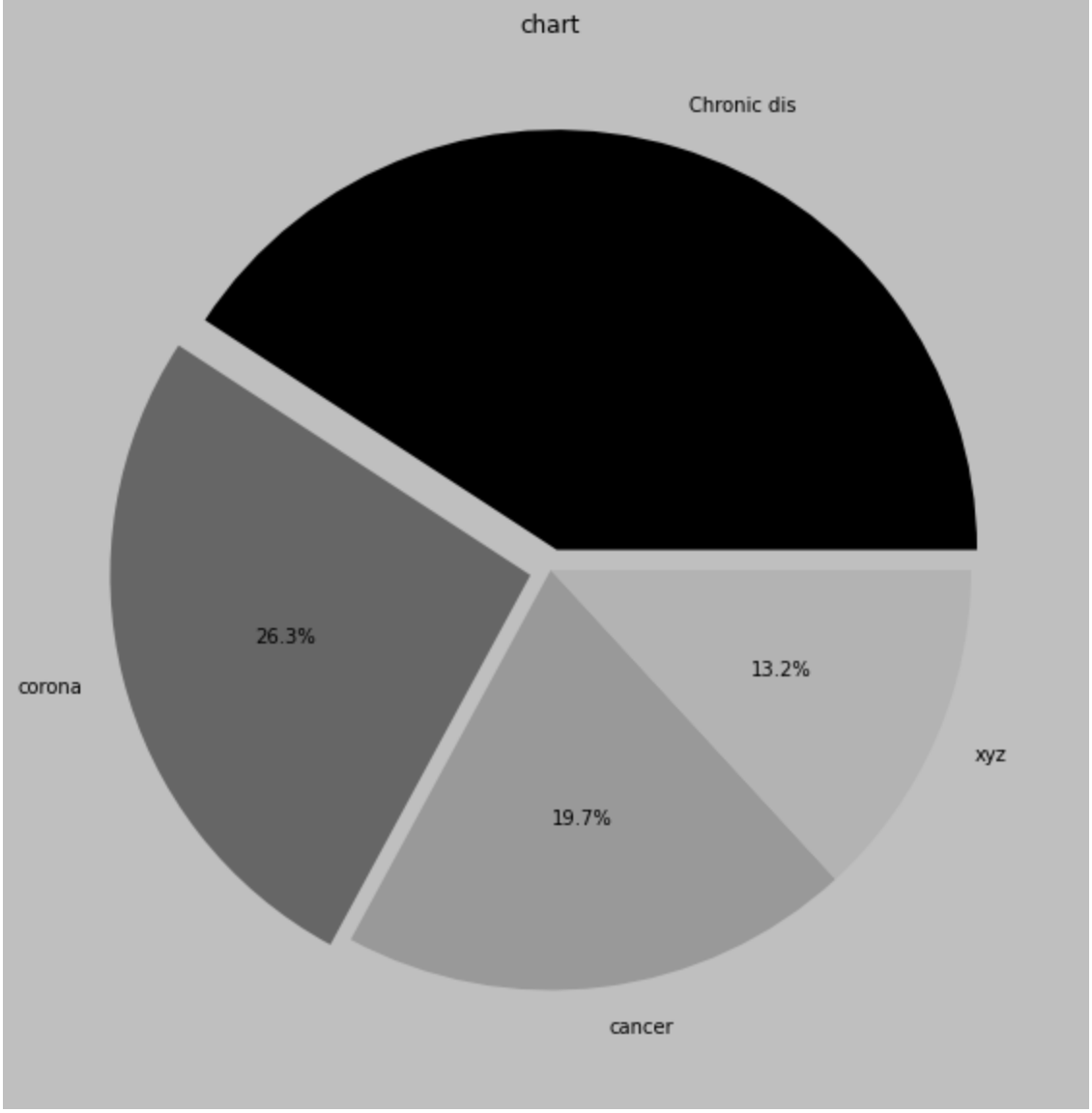
```
# days - sleep, code, playing ,eating
days=[1,2,3,4,5]
,6,11,7]
```

```
eating=[2,3,4,3,2]
coding=[7,8,7,2,2]
playing=[8,5,7,8,13]
plt.plot([],[],color='m',label='sleeping',linewidth=5)
plt.plot([],[],color='r',label='eating',linewidth=5)
plt.plot([],[],color='k',label='coding',linewidth=5)
plt.plot([],[],color='c',label='playing',linewidth=5)
plt.stackplot(days,sleeping,eating,coding,playing,colors=['m','r','k','c'])
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Stack plot")
plt.legend()
plt.show()
```



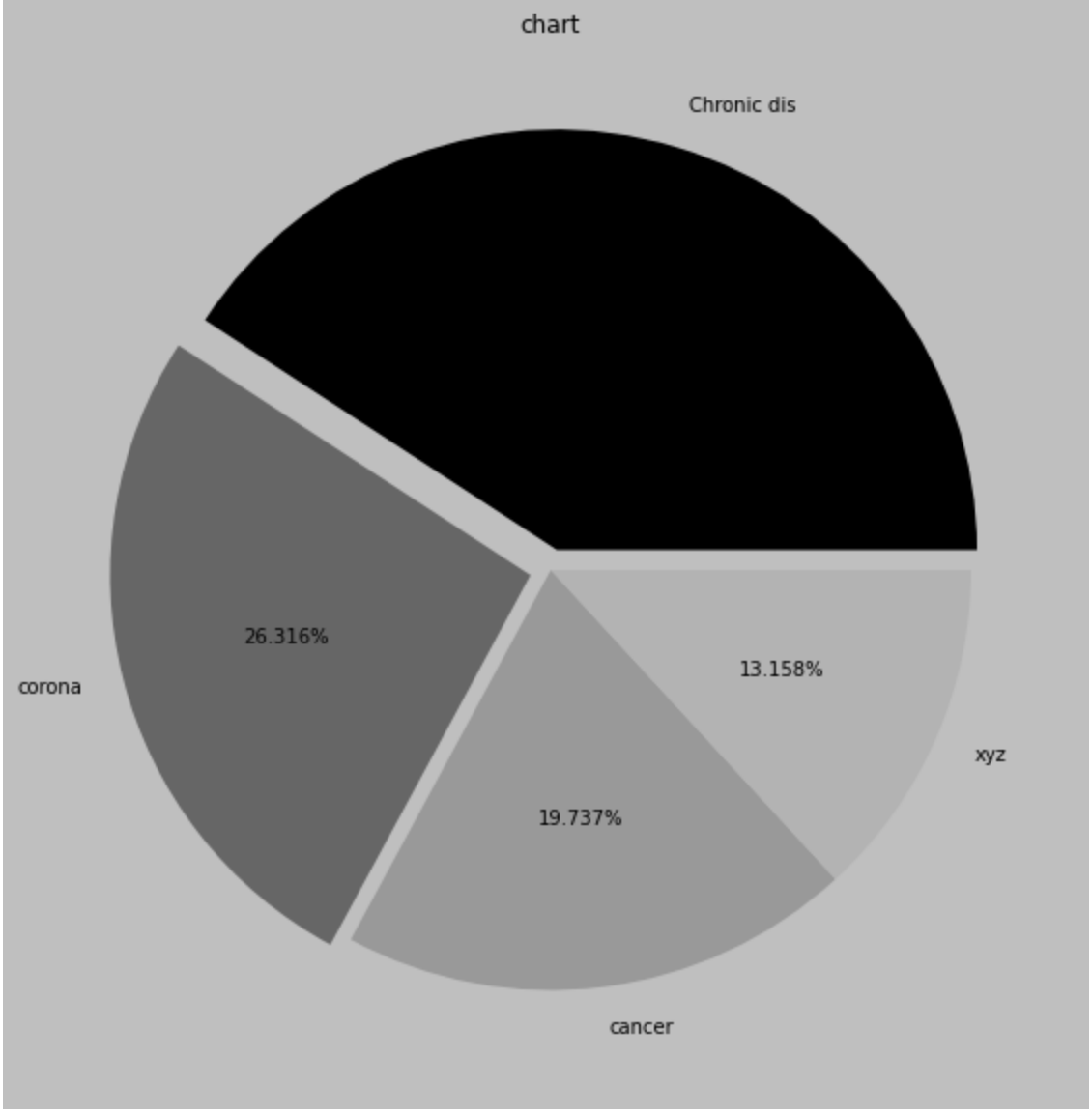
In [58]:

```
# pie chart
cause=['Chronic dis','corona','cancer','xyz']
percent=[62,40,30,20]
plt.figure(figsize=(10,10))
plt.pie(percent,labels=cause,explode=(0.05,0.05,0,0),autopct='%1.1f%%')# explode is margi
plt.title('chart')# autopct is for calculating % of data to creat pie chart
plt.show()
```



In [61]:

```
plt.figure(figsize=(10,10))
plt.pie(percent,labels=cause,explode=(0.05,0.05,0,0),autopct='%1.3f%%')# explode is
#merging slice with eachother
plt.title('chart')# autopct is for calculating % of data to creat pie chart, f for float
plt.show()
```



In [62]:
```
import seaborn as sns
```

In [63]:
```
# seaborn already inbuild dataset
data=sns.load_dataset("tips")
data
```

Out[63]:

	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
...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

244 rows × 7 columns

In [64]:
```
sns.get_dataset_names() # dataset available already
```

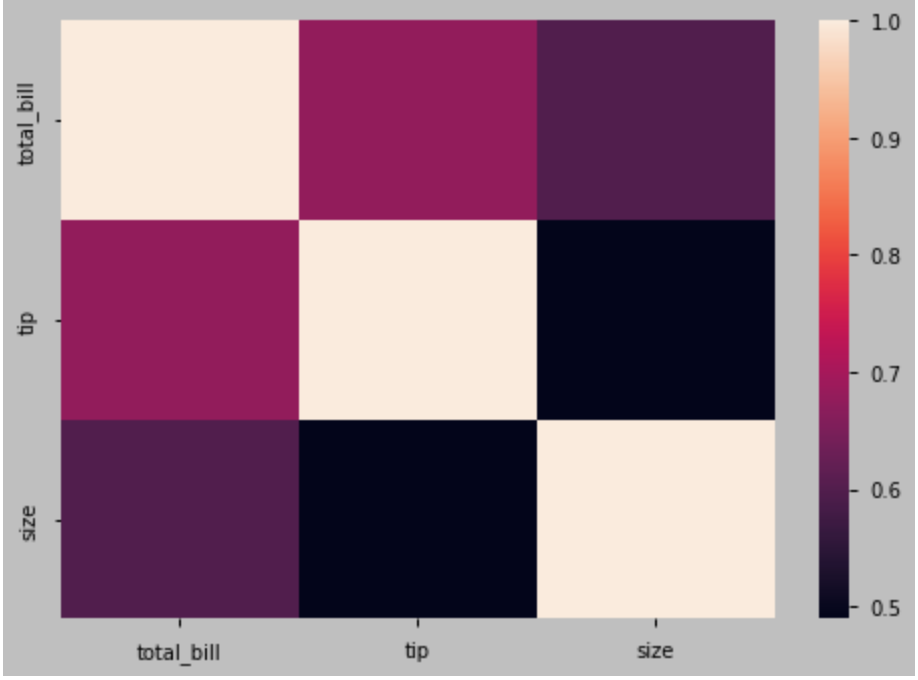
Out[64]:
```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'exercise',
 'flights',
 'fmri',
 'gammas',
 'geyser',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'taxis',
 'tips',
 'titanic']
```

In [65]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   total_bill  244 non-null    float64 
 1   tip         244 non-null    float64 
 2   sex         244 non-null    category
 3   smoker      244 non-null    category
 4   day         244 non-null    category
 5   time        244 non-null    category
 6   size        244 non-null    int64   
dtypes: category(4), float64(2), int64(1)
memory usage: 7.4 KB
```

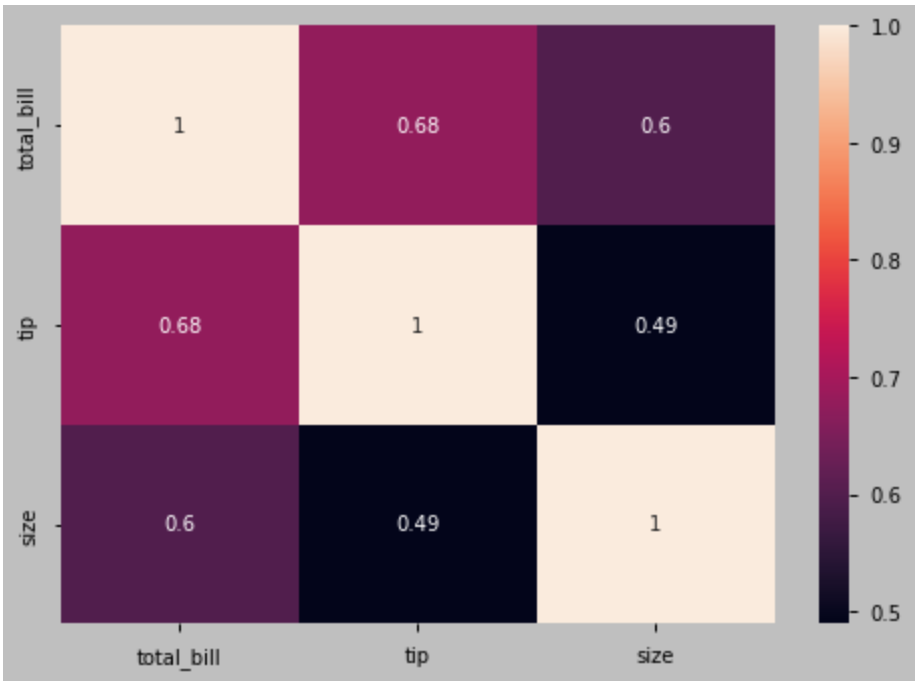
In [67]:
```
#Heatmap
data.corr()
sns.heatmap(data.corr())
```

Out[67]:
```
<AxesSubplot:>
```



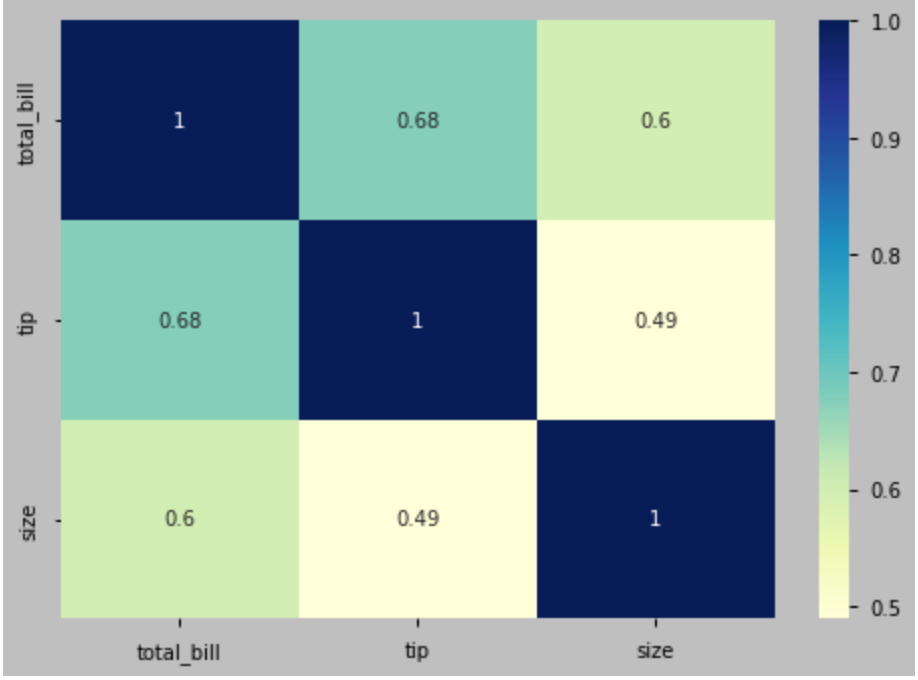
In [68]:
```
sns.heatmap(data.corr(),annot=True)
```

Out[68]:
```
<AxesSubplot:>
```



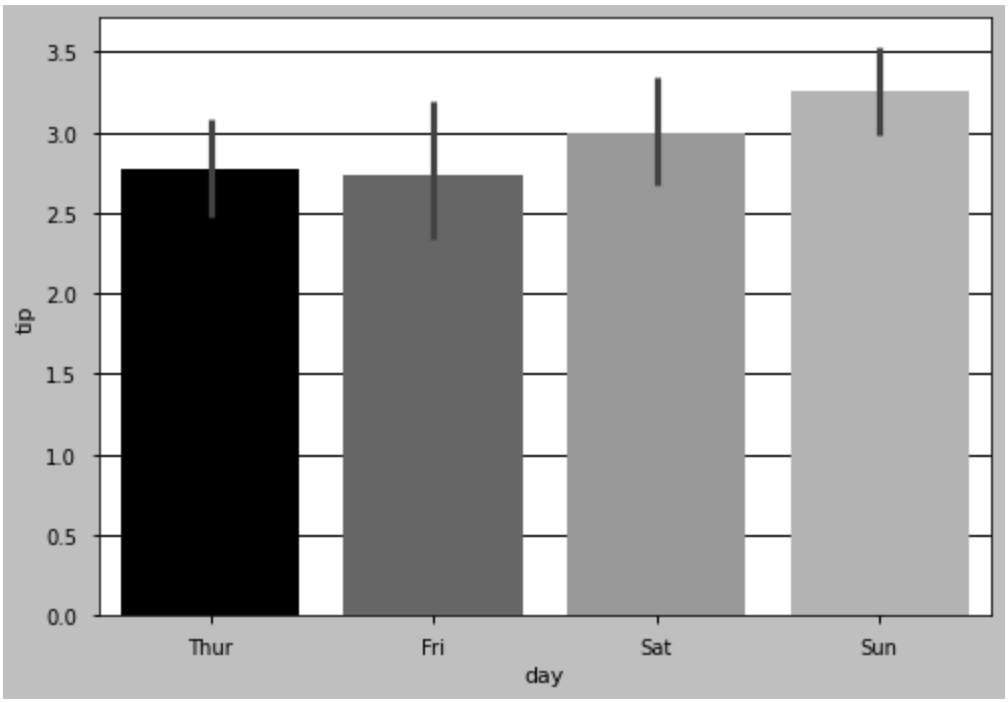
In [69]:
```
sns.heatmap(data.corr(),annot=True,cmap='YlGnBu')
```

Out[69]:
```
<AxesSubplot:>
```



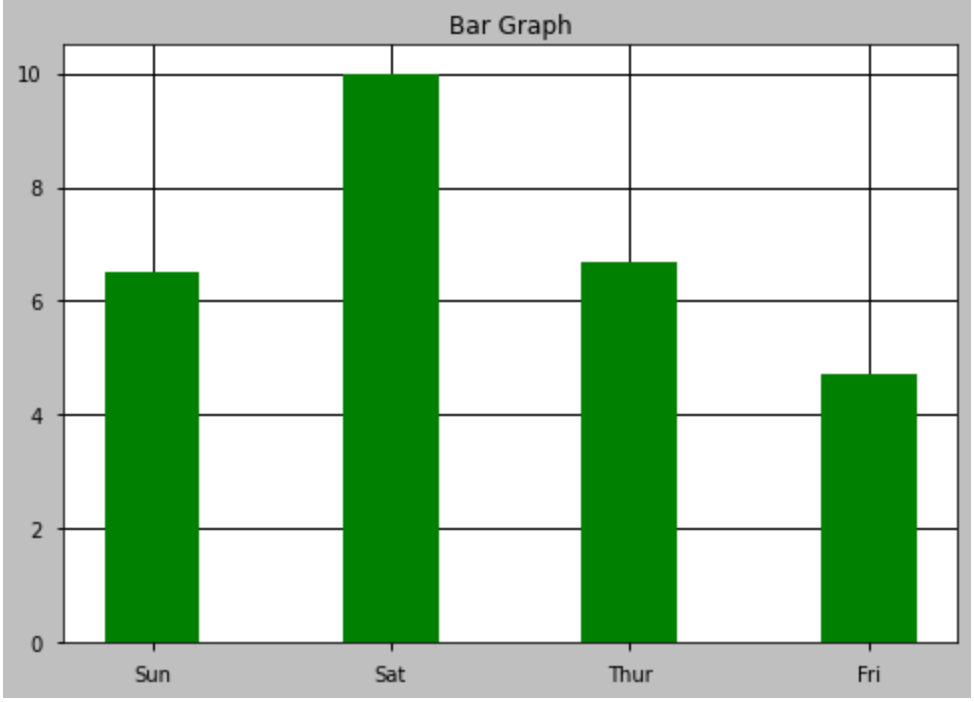
In [70]:

```
# Bar PLot
sns.barplot(x='day',y='tip',data=data) # black line in o/p defines error like outlier
plt.show()
```



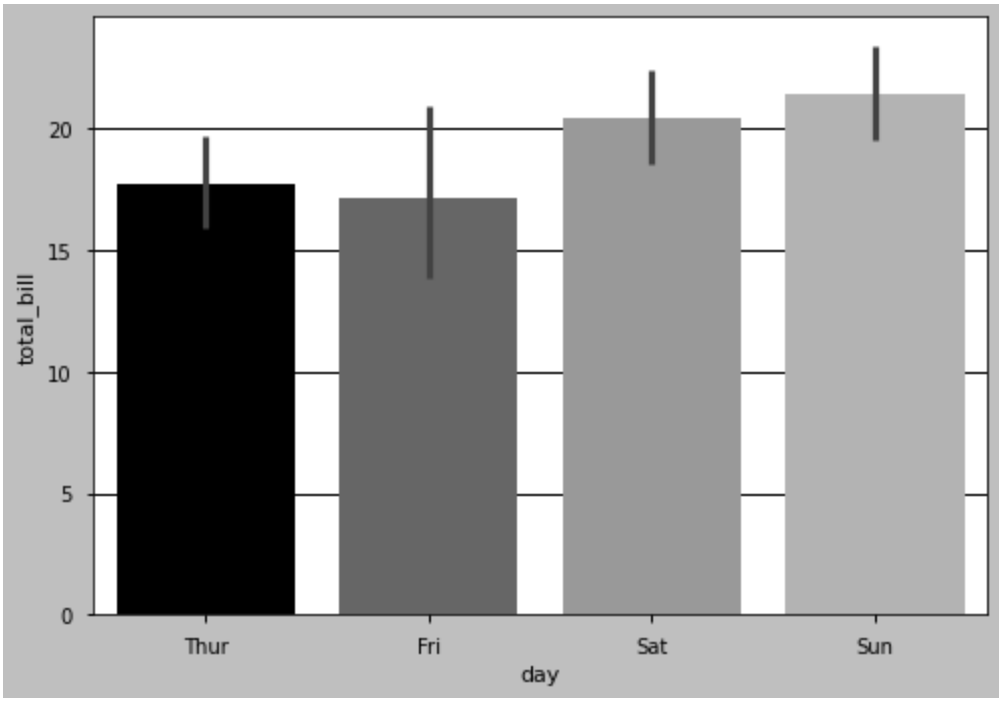
In [71]:

```
plt.bar(data['day'],data['tip'],color='g',width=0.4)
plt.title("Bar Graph")
plt.show()
```



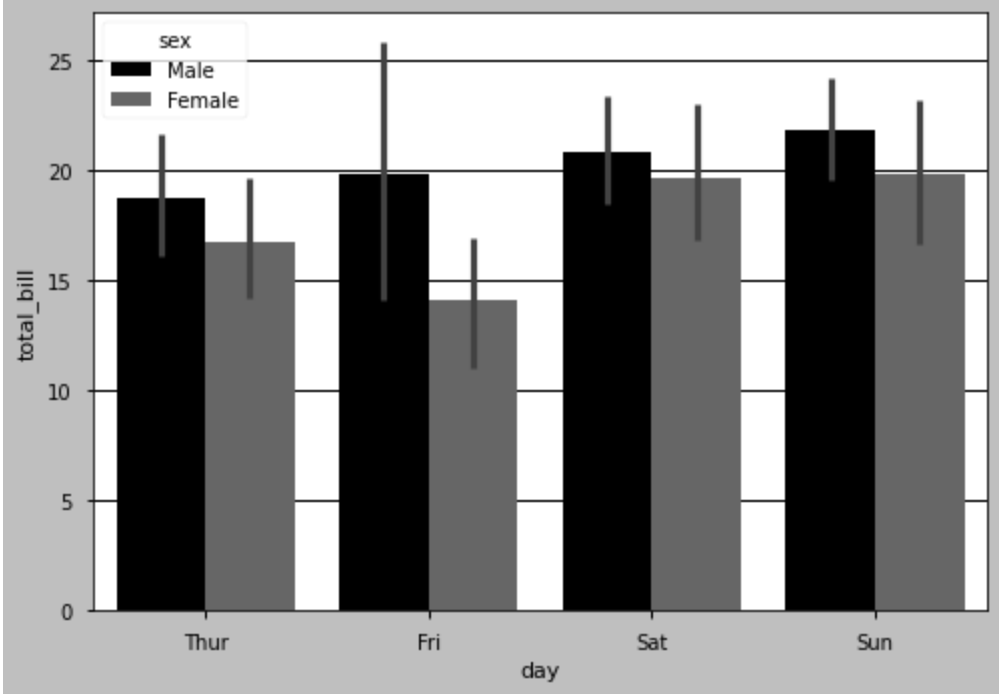
In [72]:

```
sns.barplot(x='day',y='total_bill',data=data) # black line in o/p defines error like outli
plt.show()
```



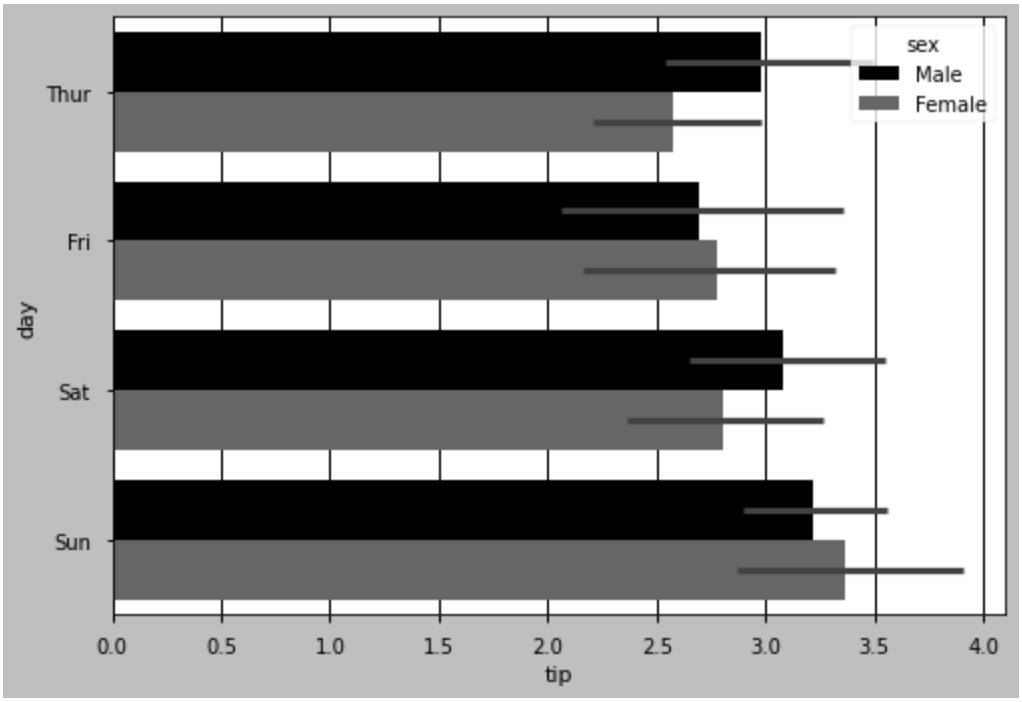
In [73]:

```
# create a bar plot using seaborn between day & total bill
sns.barplot(x='day',y='total_bill',hue='sex',data=data)
plt.show()
```



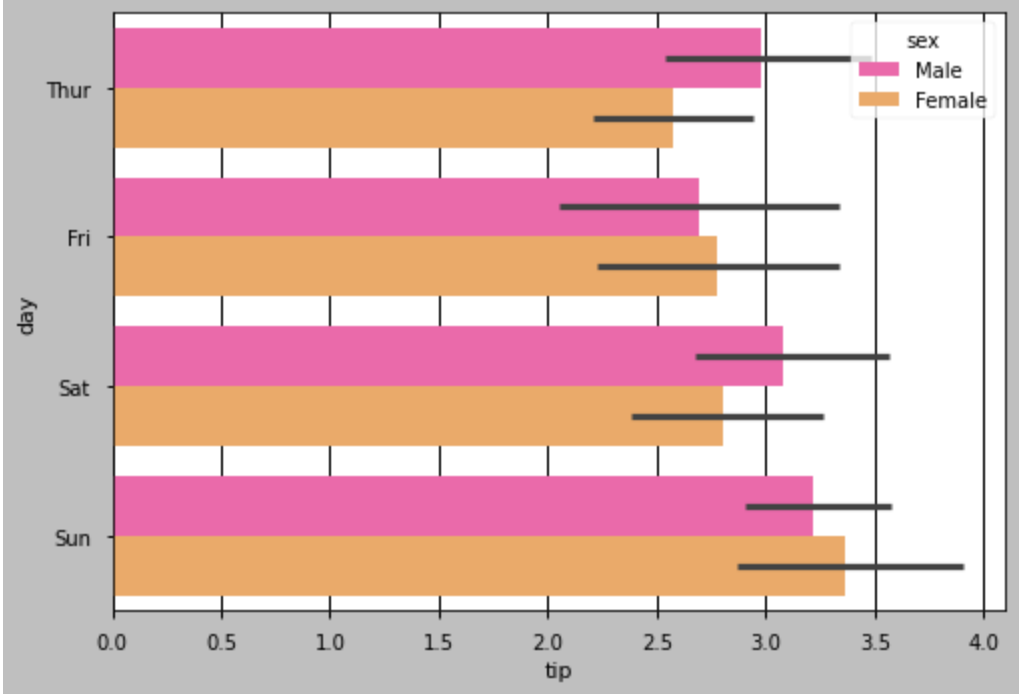
In [75]:

```
sns.barplot(x='tip',y='day',hue='sex',data=data)
plt.show()
```



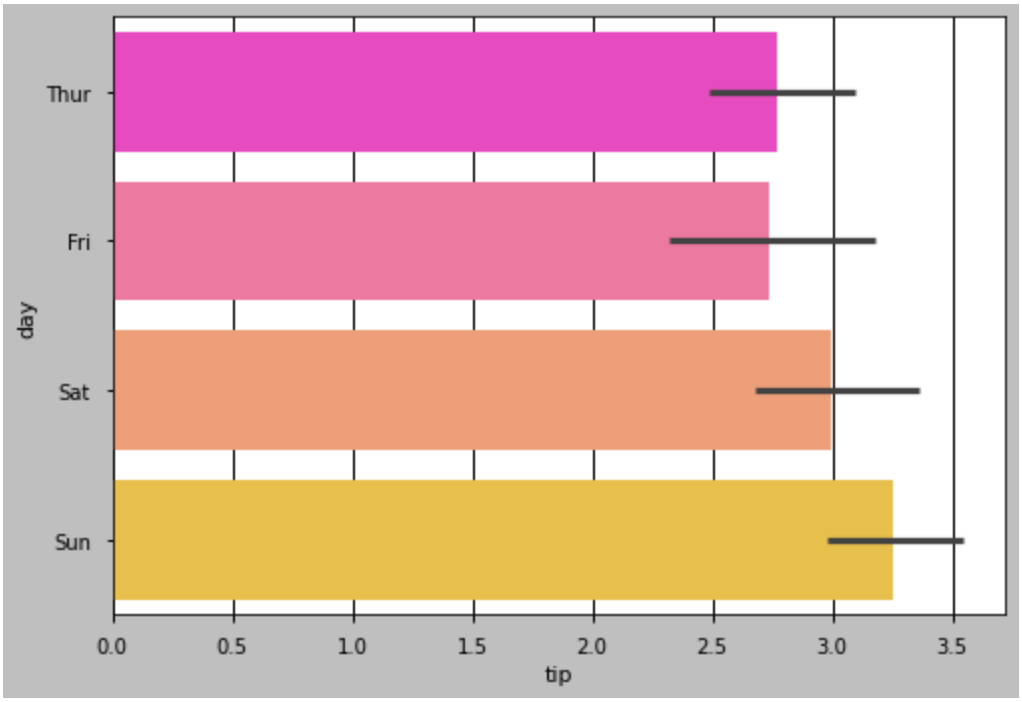
In [76]:

```
# day vs tip
sns.barplot(x='tip',y='day',hue='sex',data=data,palette='spring')
plt.show()
```



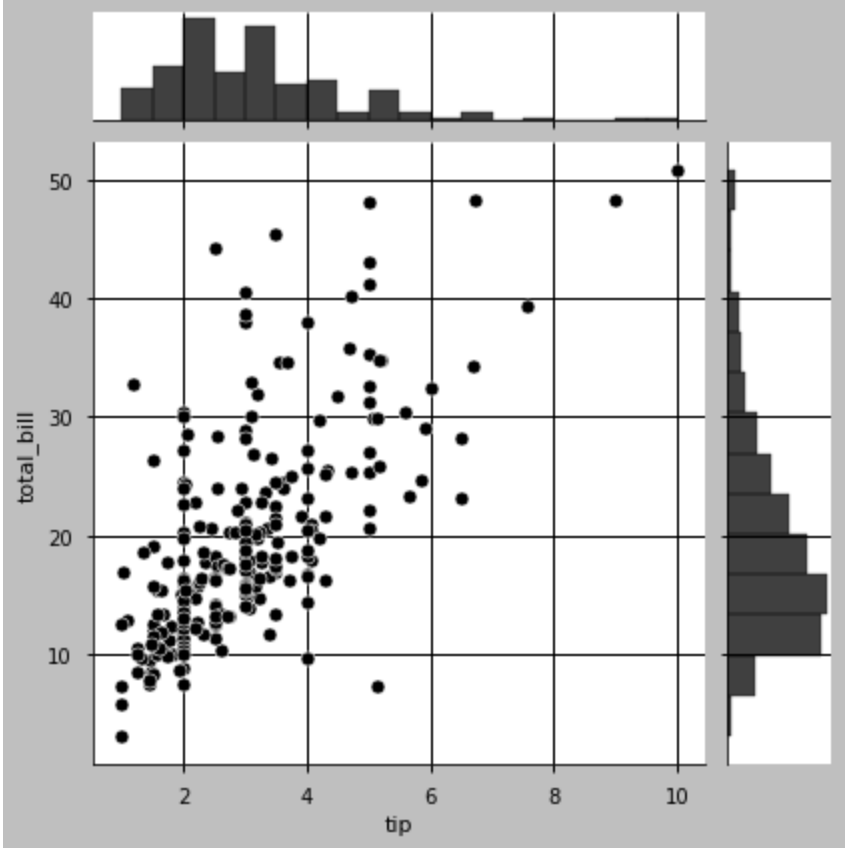
In [77]:

```
# day vs tip
sns.barplot(x='tip',y='day',data=data,palette='spring')
plt.show()
```



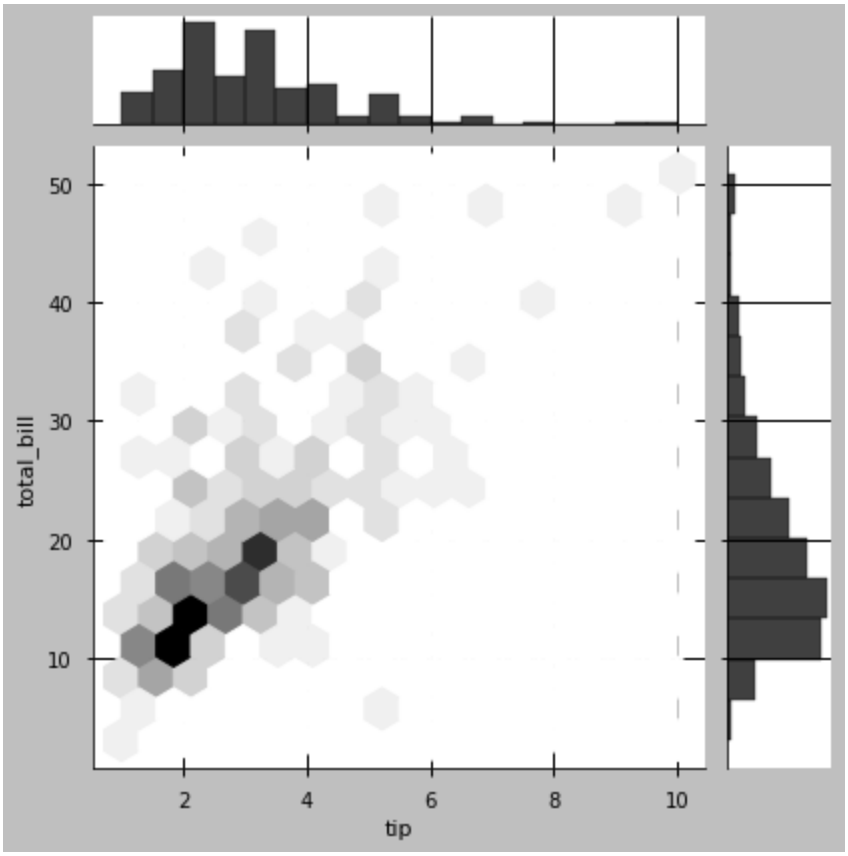
In [79]:

```
sns.jointplot(x='tip',y='total_bill',palette='spring',data=data)
plt.show()
```



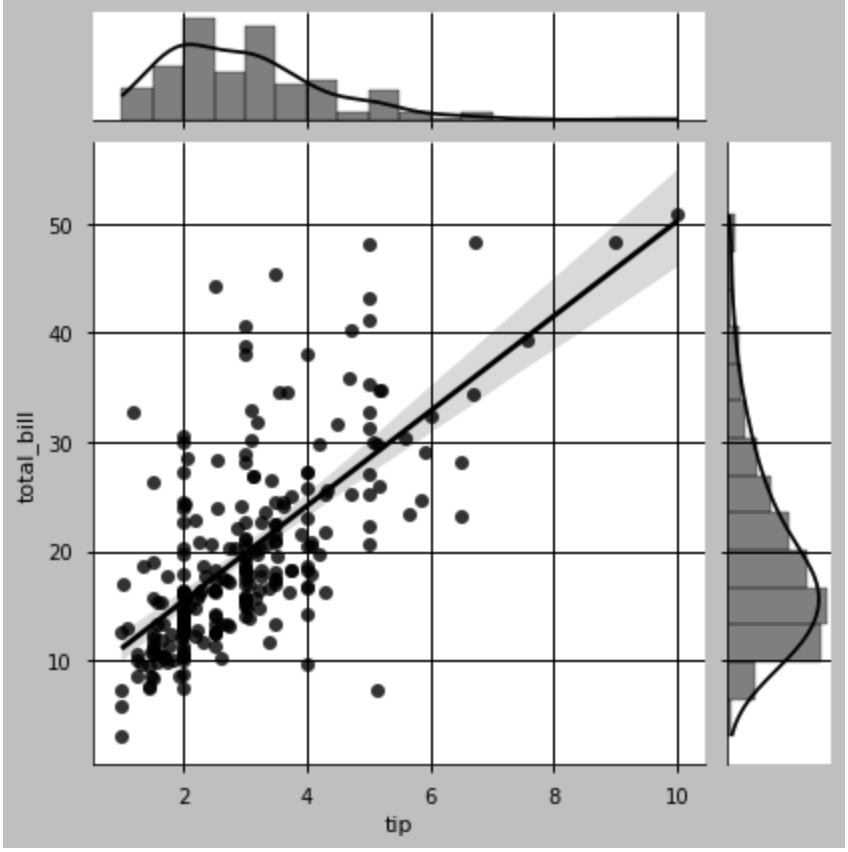
In [80]:

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



In [81]:

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

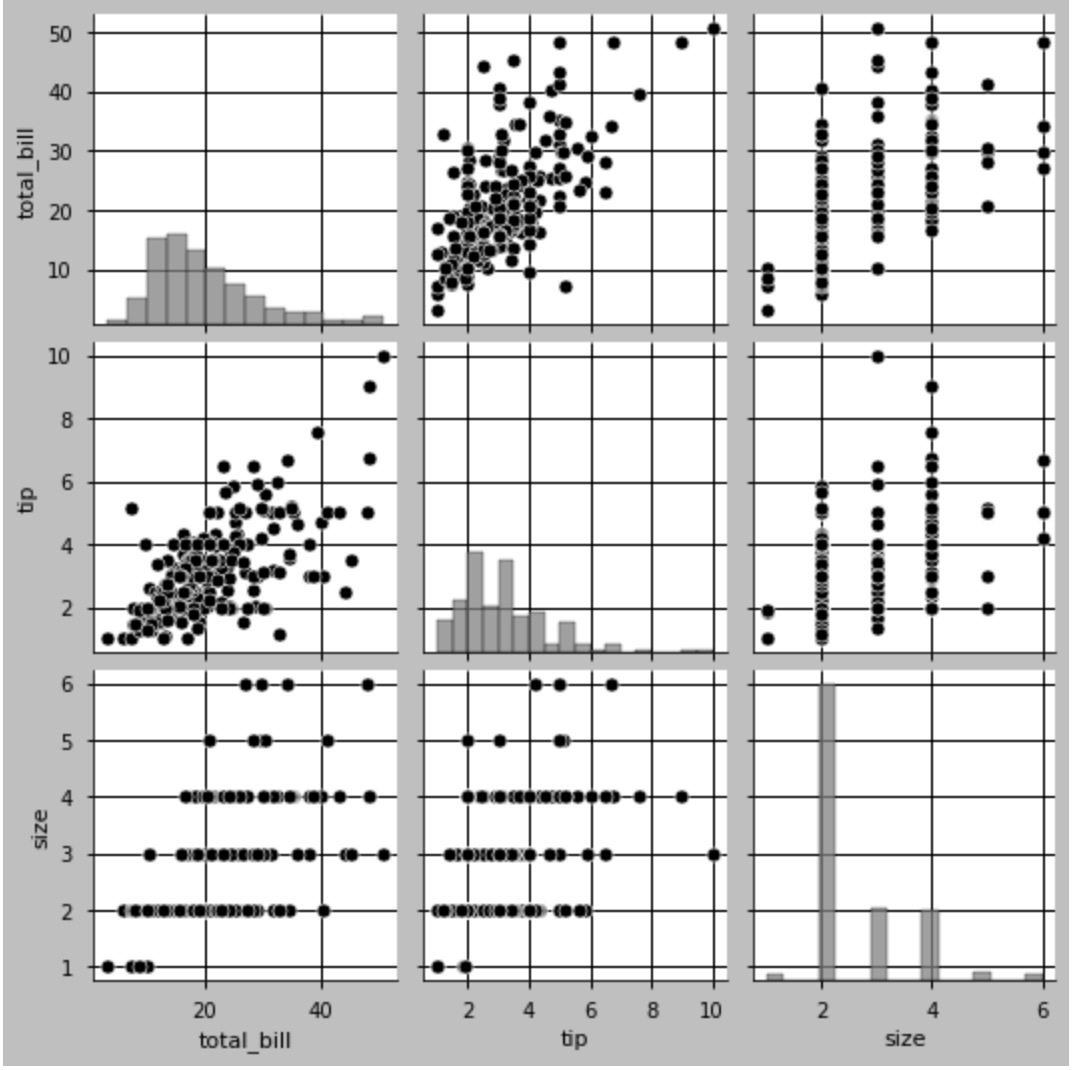


In [82]:

```
# pairplot()
# numerical data
sns.pairplot(data)
```

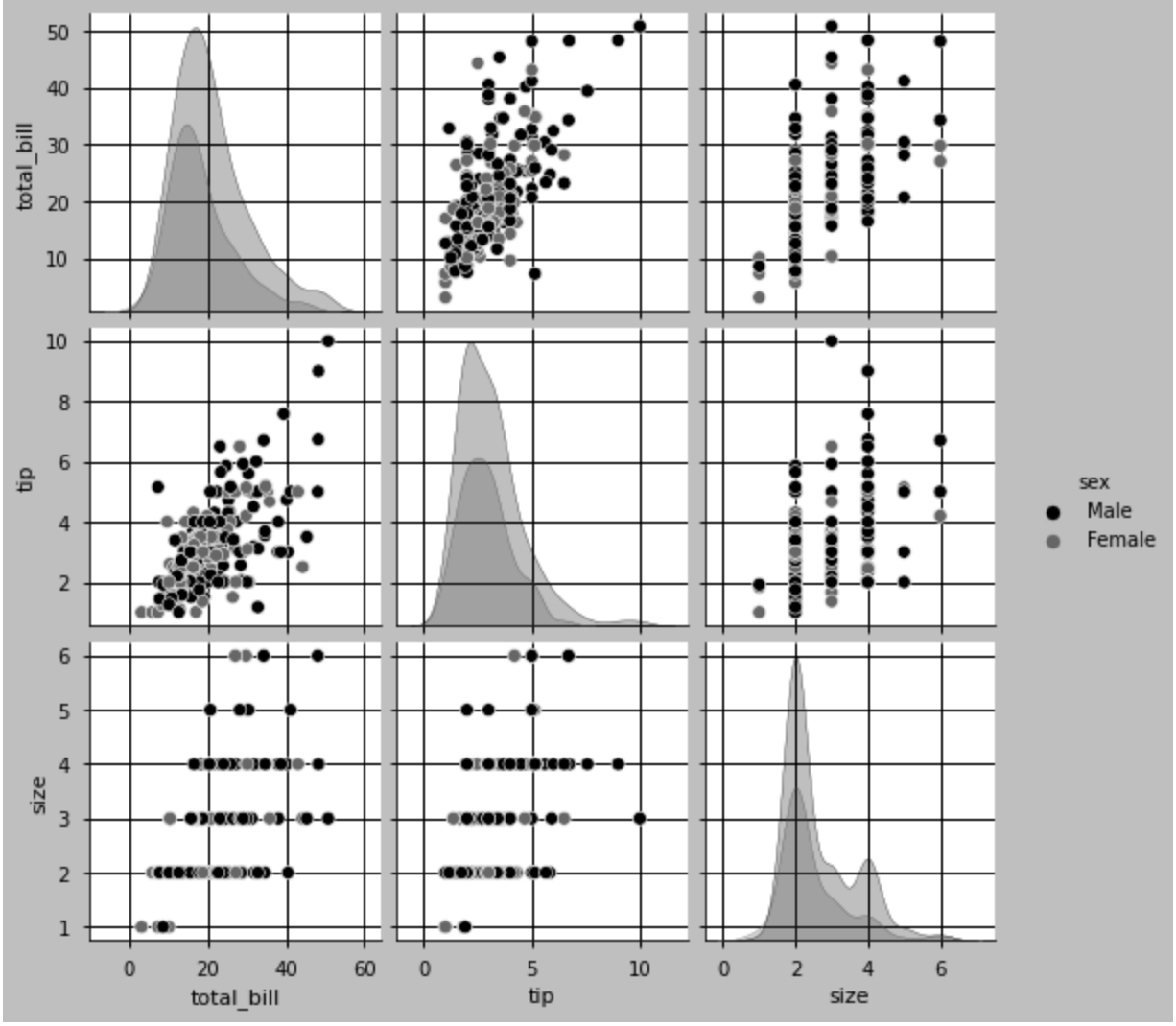
Out[82]:

```
<seaborn.axisgrid.PairGrid at 0x25724b1a9d0>
```



In [83]:
```
sns.pairplot(data,hue='sex')
```

Out[83]:
```
<seaborn.axisgrid.PairGrid at 0x2572360b730>
```



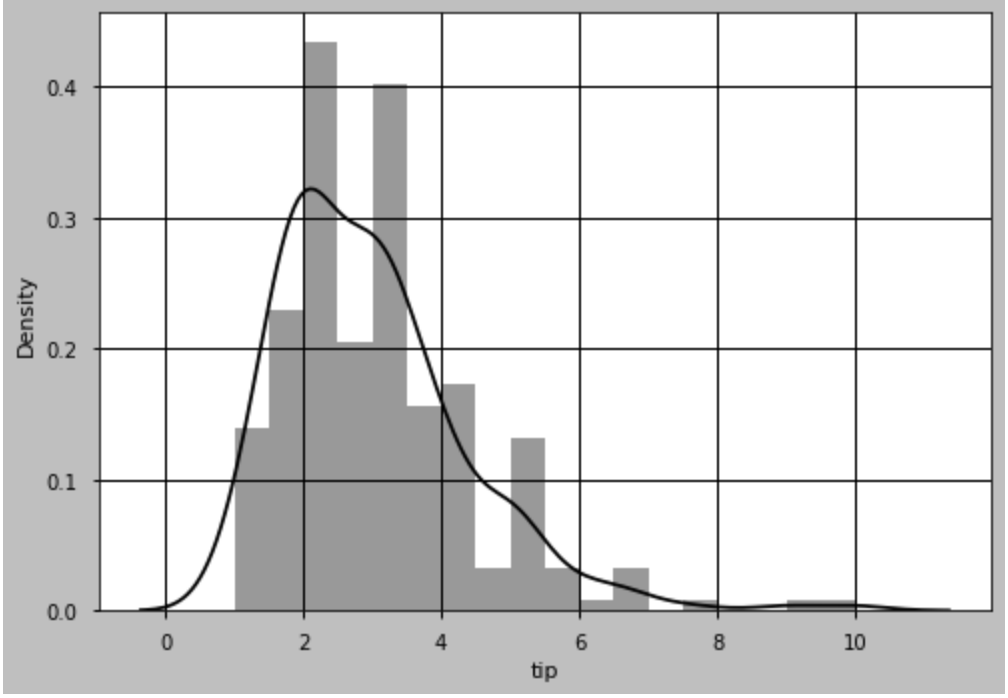
In [84]:

```
# displot()
sns.distplot(data['tip'])
```

```
C:\Users\91956\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[84]:

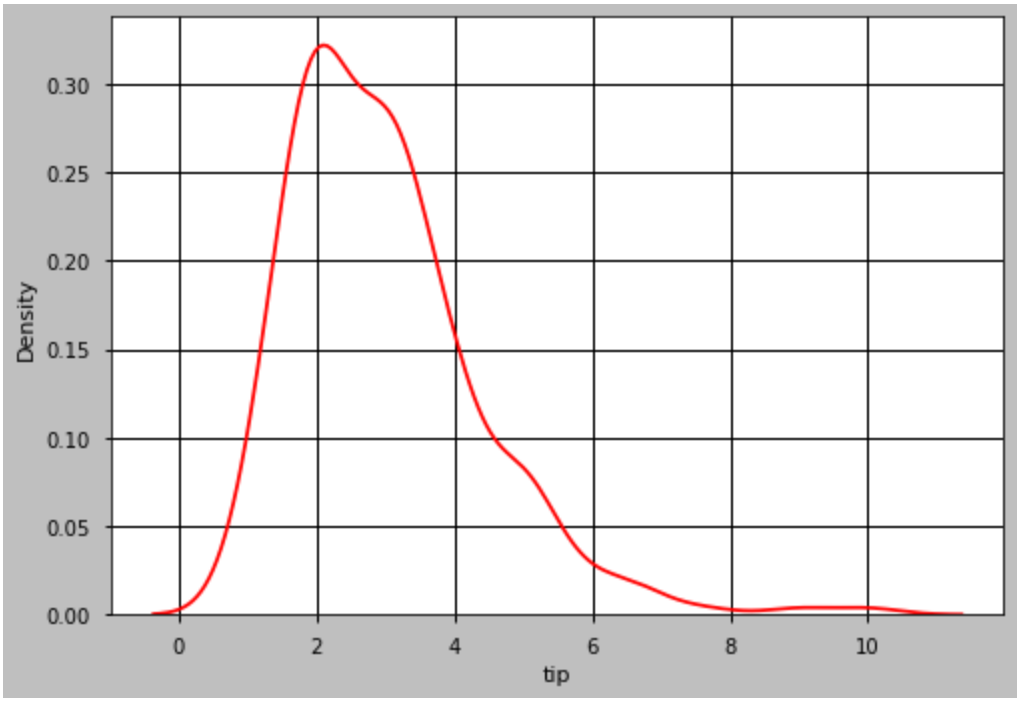
```
<AxesSubplot:xlabel='tip', ylabel='Density'>
```



In [85]:
```
# displot()
sns.distplot(data['tip'],hist=False,color='red')
```

```
C:\Users\91956\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `kd
eplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```

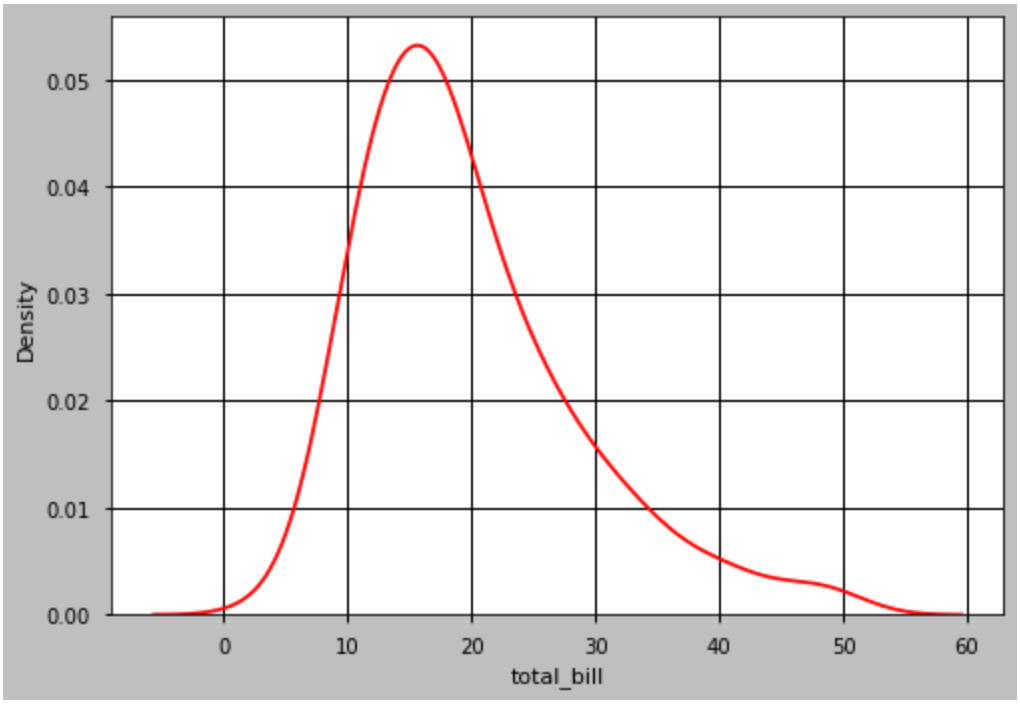
Out[85]:
```
<AxesSubplot:xlabel='tip', ylabel='Density'>
```



In [87]:
```
# displot()
sns.distplot(data['total_bill'],hist=False,color='red') # Use to find frequency distributi
```

```
C:\Users\91956\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt y
our code to use either `displot` (a figure-level function with similar flexibility) or `kd
eplot` (an axes-level function for kernel density plots).
n(msg, FutureWarning)
```

Out[87]: <AxesSubplot:xlabel='total_bill', ylabel='Density'>



In [ ]: