Data Cleaning: ¶

- missing data[replace,dropna,fillna]
- using sklearn
- duplicate data

In [1]:
```
import numpy as np
import pandas as pd
```

In [14]:
```
a = np.array([[1,2,np.nan,3,4],
              [10,12,56,78,34],
              [34,85,12,45,30],
             [np.nan,52,89,np.nan,67],
              [89,np.nan,34,67,89]]
            )
a
```

Out[14]:
```
array([[ 1.,  2., nan,  3.,  4.],
       [10., 12., 56., 78., 34.],
       [34., 85., 12., 45., 30.],
       [nan, 52., 89., nan, 67.],
       [89., nan, 34., 67., 89.]])
```

In [15]:
```
df = pd.DataFrame(a,columns = ["first","second","third","fourth","fivth"],
                  index = ["a","b","c","d","e"])
```

In [16]:
```
df
```

Out[16]:

	first	second	third	fourth	fivth
a	1.0	2.0	NaN	3.0	4.0
b	10.0	12.0	56.0	78.0	34.0
c	34.0	85.0	12.0	45.0	30.0
d	NaN	52.0	89.0	NaN	67.0
e	89.0	NaN	34.0	67.0	89.0

In [17]:
```
df.isnull()
```

Out[17]:

	first	second	third	fourth	fivth
a	False	False	True	False	False
b	False	False	False	False	False
c	False	False	False	False	False
d	True	False	False	True	False
e	False	True	False	False	False

In [18]:
```
df.isnull().sum()
```

Out[18]:
```
first     1
second    1
third     1
fourth    1
fivth     0
dtype: int64
```

In [ ]:
```
# we can handle nan value using two ways
# dropna
# fillna
```

In [19]:
```
df.dropna()
```

Out[19]:

	first	second	third	fourth	fivth
b	10.0	12.0	56.0	78.0	34.0
c	34.0	85.0	12.0	45.0	30.0

In [20]:
```
df["first"].replace(np.nan,0)
```

Out[20]:
```
a     1.0
b    10.0
c    34.0
d     0.0
e    89.0
Name: first, dtype: float64
```

In [21]:
```
df
```

Out[21]:

	first	second	third	fourth	fivth
a	1.0	2.0	NaN	3.0	4.0
b	10.0	12.0	56.0	78.0	34.0
c	34.0	85.0	12.0	45.0	30.0
d	NaN	52.0	89.0	NaN	67.0
e	89.0	NaN	34.0	67.0	89.0

In [22]:
```
df["first"]=df["first"].replace(np.nan,0)
```

In [23]:
```
df
```

Out[23]:

	first	second	third	fourth	fivth
a	1.0	2.0	NaN	3.0	4.0
b	10.0	12.0	56.0	78.0	34.0
c	34.0	85.0	12.0	45.0	30.0
d	0.0	52.0	89.0	NaN	67.0
e	89.0	NaN	34.0	67.0	89.0

In [24]:
```
df["second"].mean()
```

Out[24]: 37.75

In [25]:
```
df["second"]=df["second"].fillna(df["second"].mean())
```

In [26]:
```
df
```

Out[26]:

	first	second	third	fourth	fivth
a	1.0	2.00	NaN	3.0	4.0
b	10.0	12.00	56.0	78.0	34.0
c	34.0	85.00	12.0	45.0	30.0
d	0.0	52.00	89.0	NaN	67.0
e	89.0	37.75	34.0	67.0	89.0

In [27]:
```
df["third"]=df["third"].fillna(df["third"].median())
```

In [28]:
```
df
```

Out[28]:

	first	second	third	fourth	fivth
a	1.0	2.00	45.0	3.0	4.0
b	10.0	12.00	56.0	78.0	34.0
c	34.0	85.00	12.0	45.0	30.0
d	0.0	52.00	89.0	NaN	67.0
e	89.0	37.75	34.0	67.0	89.0

In [29]:
```
df.fillna(method = "ffill")
```

Out[29]:

	first	second	third	fourth	fivth
a	1.0	2.00	45.0	3.0	4.0
b	10.0	12.00	56.0	78.0	34.0
c	34.0	85.00	12.0	45.0	30.0
d	0.0	52.00	89.0	45.0	67.0
e	89.0	37.75	34.0	67.0	89.0

In [30]:
```
df.fillna(method = "bfill")
```

Out[30]:

	first	second	third	fourth	fivth
a	1.0	2.00	45.0	3.0	4.0
b	10.0	12.00	56.0	78.0	34.0
c	34.0	85.00	12.0	45.0	30.0
d	0.0	52.00	89.0	67.0	67.0
e	89.0	37.75	34.0	67.0	89.0

In [31]:
```
a = np.array([[1,2,np.nan,3,4],
              [10,12,56,78,34],
              [34,85,12,45,30],
             [np.nan,52,89,np.nan,67],
              [89,np.nan,34,67,89]]
            )
a
```

Out[31]:
```
array([[ 1.,  2., nan,  3.,  4.],
       [10., 12., 56., 78., 34.],
       [34., 85., 12., 45., 30.],
       [nan, 52., 89., nan, 67.],
       [89., nan, 34., 67., 89.]])
```

In [32]:
```
df = pd.DataFrame(a,columns = ["first","second","third","fourth","fivth"],
                  index = ["a","b","c","d","e"])
```

In [33]:
```
df
```

Out[33]:

	first	second	third	fourth	fivth
a	1.0	2.0	NaN	3.0	4.0
b	10.0	12.0	56.0	78.0	34.0
c	34.0	85.0	12.0	45.0	30.0
d	NaN	52.0	89.0	NaN	67.0
e	89.0	NaN	34.0	67.0	89.0

In [34]:
```
from sklearn.impute import SimpleImputer
```

In [36]:
```
help(SimpleImputer)
```

```
Help on class SimpleImputer in module sklearn.impute:

class SimpleImputer(sklearn.base.BaseEstimator, sklearn.base.TransformerMixi
n)
 |  SimpleImputer(missing_values=nan, strategy='mean', fill_value=None, verbo
se=0, copy=True)
 |
 |  Imputation transformer for completing missing values.
 |
 |  Read more in the :ref:`User Guide <impute>`.
 |
 |  Parameters
 |  ----------
 |  missing_values : number, string, np.nan (default) or None
 |      The placeholder for the missing values. All occurrences of
 |      `missing_values` will be imputed.
 |
 |  strategy : string, optional (default="mean")
 |      The imputation strategy.
 |
 |      - If "mean", then replace missing values using the mean along
 |        each column. Can only be used with numeric data.
 |      - If "median", then replace missing values using the median along
 |        each column. Can only be used with numeric data.
 |      - If "most_frequent", then replace missing using the most frequent
 |        value along each column. Can be used with strings or numeric data.
 |      - If "constant", then replace missing values with fill_value. Can be
 |        used with strings or numeric data.
 |
 |      .. versionadded:: 0.20
 |         strategy="constant" for fixed value imputation.
 |
 |  fill_value : string or numerical value, optional (default=None)
 |      When strategy == "constant", fill_value is used to replace all
 |      occurrences of missing_values.
 |      If left to the default, fill_value will be 0 when imputing numerical
 |      data and "missing_value" for strings or object data types.
 |
 |  verbose : integer, optional (default=0)
 |      Controls the verbosity of the imputer.
 |
 |  copy : boolean, optional (default=True)
 |      If True, a copy of X will be created. If False, imputation will
 |      be done in-place whenever possible. Note that, in the following case
s,
 |      a new copy will always be made, even if `copy=False`:
 |
 |      - If X is not an array of floating values;
 |      - If X is encoded as a CSR matrix.
 |
 |  Attributes
 |  ----------
 |  statistics_ : array of shape (n_features,)
 |      The imputation fill value for each feature.
```

```
|
|  Examples
|  --------
|  >>> import numpy as np
|  >>> from sklearn.impute import SimpleImputer
|  >>> imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
|  >>> imp_mean.fit([[7, 2, 3], [4, np.nan, 6], [10, 5, 9]])
|  ... # doctest: +NORMALIZE_WHITESPACE
|  SimpleImputer(copy=True, fill_value=None, missing_values=nan,
|         strategy='mean', verbose=0)
|  >>> X = [[np.nan, 2, 3], [4, np.nan, 6], [10, np.nan, 9]]
|  >>> print(imp_mean.transform(X))
|  ... # doctest: +NORMALIZE_WHITESPACE
|  [[ 7.   2.   3. ]
|   [ 4.   3.5  6. ]
|   [10.   3.5  9. ]]
|
|  Notes
|  -----
|  Columns which only contained missing values at `fit` are discarded upon
|  `transform` if strategy is not "constant".
|
|  Method resolution order:
|      SimpleImputer
|      sklearn.base.BaseEstimator
|      sklearn.base.TransformerMixin
|      builtins.object
|
|  Methods defined here:
|
|  __init__(self, missing_values=nan, strategy='mean', fill_value=None, verb
ose=0, copy=True)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  fit(self, X, y=None)
|      Fit the imputer on X.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix}, shape (n_samples, n_features)
|          Input data, where ``n_samples`` is the number of samples and
|          ``n_features`` is the number of features.
|
|      Returns
|      -------
|      self : SimpleImputer
|
|  transform(self, X)
|      Impute all missing values in X.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix}, shape (n_samples, n_features)
|          The input data to complete.
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.BaseEstimator:
```

```
|
|  __getstate__(self)
|
|  __repr__(self)
|      Return repr(self).
|
|  __setstate__(self, state)
|
|  get_params(self, deep=True)
|      Get parameters for this estimator.
|
|      Parameters
|      ----------
|      deep : boolean, optional
|          If True, will return the parameters for this estimator and
|          contained subobjects that are estimators.
|
|      Returns
|      -------
|      params : mapping of string to any
|          Parameter names mapped to their values.
|
|  set_params(self, **params)
|      Set the parameters of this estimator.
|
|      The method works on simple estimators as well as on nested objects
|      (such as pipelines). The latter have parameters of the form
|      ``<component>__<parameter>`` so that it's possible to update each
|      component of a nested object.
|
|      Returns
|      -------
|      self
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from sklearn.base.BaseEstimator:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.TransformerMixin:
|
|  fit_transform(self, X, y=None, **fit_params)
|      Fit to data, then transform it.
|
|      Fits transformer to X and y with optional parameters fit_params
|      and returns a transformed version of X.
|
|      Parameters
|      ----------
|      X : numpy array of shape [n_samples, n_features]
|          Training set.
|
```

```
|       y : numpy array of shape [n_samples]
|           Target values.
|
|       Returns
|       -------
|       X_new : numpy array of shape [n_samples, n_features_new]
|           Transformed array.
```

In [38]:
```
s = SimpleImputer(strategy='mean')
filldata = s.fit_transform(df)
filldata
```

Out[38]:
```
array([[ 1.  ,  2.  , 47.75,  3.  ,  4.  ],
       [10.  , 12.  , 56.  , 78.  , 34.  ],
       [34.  , 85.  , 12.  , 45.  , 30.  ],
       [33.5 , 52.  , 89.  , 48.25, 67.  ],
       [89.  , 37.75, 34.  , 67.  , 89.  ]])
```

In [41]:
```
pd.DataFrame(filldata,columns = df.columns,index=df.index)
```

Out[41]:

	first	second	third	fourth	fivth
a	1.0	2.00	47.75	3.00	4.0
b	10.0	12.00	56.00	78.00	34.0
c	34.0	85.00	12.00	45.00	30.0
d	33.5	52.00	89.00	48.25	67.0
e	89.0	37.75	34.00	67.00	89.0

In [40]:
```
df.columns
```

Out[40]:
```
Index(['first', 'second', 'third', 'fourth', 'fivth'], dtype='object')
```

In [43]:
```
df
```

Out[43]:

	first	second	third	fourth	fivth
a	1.0	2.0	NaN	3.0	4.0
b	10.0	12.0	56.0	78.0	34.0
c	34.0	85.0	12.0	45.0	30.0
d	NaN	52.0	89.0	NaN	67.0
e	89.0	NaN	34.0	67.0	89.0

In [42]:
```
s = SimpleImputer(strategy='median')
filldata1 = s.fit_transform(df)#particular column df[column name]
filldata1
```

Out[42]:
```
array([[ 1.,  2., 45.,  3.,  4.],
       [10., 12., 56., 78., 34.],
       [34., 85., 12., 45., 30.],
       [22., 52., 89., 56., 67.],
       [89., 32., 34., 67., 89.]])
```

In [44]:
```
# drop duplicate values
```

In [45]:
```
d = pd.DataFrame({"sno":[1,2,2,3,3,4,5,6,7],
                  "names":["a","b","b","c","c","d","e","f","g"]})
```

In [46]:
```
d
```

Out[46]:

	sno	names
0	1	a
1	2	b
2	2	b
3	3	c
4	3	c
5	4	d
6	5	e
7	6	f
8	7	g

In [47]:
```
d.duplicated()
```

Out[47]:
```
0    False
1    False
2     True
3    False
4     True
5    False
6    False
7    False
8    False
dtype: bool
```

In [48]:
```
d[d.duplicated()]
```

Out[48]:

	sno	names
2	2	b
4	3	c

In [49]:
```
d
```

Out[49]:

	sno	names
0	1	a
1	2	b
2	2	b
3	3	c
4	3	c
5	4	d
6	5	e
7	6	f
8	7	g

In [51]:
```
d["sno"]# accessing the particular column
```

Out[51]:
```
0    1
1    2
2    2
3    3
4    3
5    4
6    5
7    6
8    7
Name: sno, dtype: int64
```

In [54]:
```
d.drop("sno",inplace=True,axis=1)
```

In [55]:
```
d
```

Out[55]:

	names
0	a
1	b
2	b
3	c
4	c
5	d
6	e
7	f
8	g

In [56]:
```
d = pd.read_csv("complaints_processed.csv")
d.head()
```

Out[56]:

	Unnamed: 0	product	narrative
0	0	credit_card	purchase order day shipping amount receive pro...
1	1	credit_card	forwarded message date tue subject please inve...
2	2	retail_banking	forwarded message cc sent friday pdt subject f...
3	3	credit_reporting	payment history missing credit report speciali...
4	4	credit_reporting	payment history missing credit report made mis...

In [57]:
```
d.shape
```

Out[57]:
```
(162421, 3)
```

In [59]:
```
d.isna().sum()
```

Out[59]:
```
Unnamed: 0     0
product        0
narrative     10
dtype: int64
```

In [62]:
```
d["narrative"].dtype
```

Out[62]:
```
dtype('O')
```

Visulization

- Matplotlib

- Seaborn

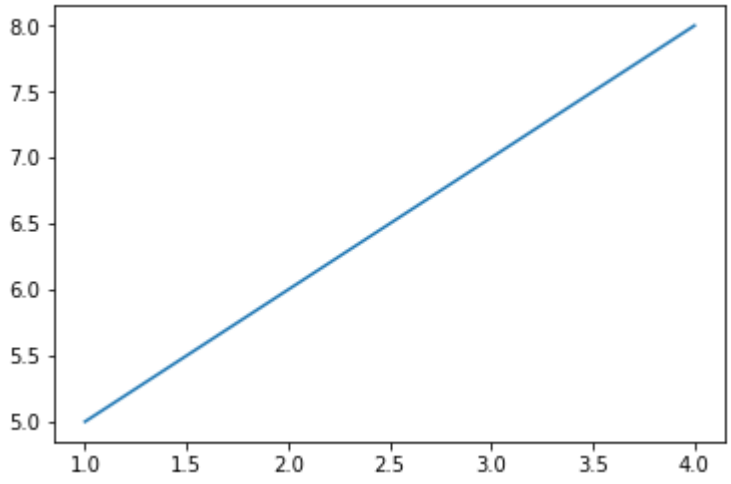
In [63]:
```
import matplotlib.pyplot as plt
```

In [64]:
```
print(dir(plt))
```

```
['Annotation', 'Arrow', 'Artist', 'AutoLocator', 'Axes', 'Button', 'Circle', 'F
igure', 'FigureCanvasBase', 'FixedFormatter', 'FixedLocator', 'FormatStrFormatt
er', 'Formatter', 'FuncFormatter', 'GridSpec', 'IndexLocator', 'Line2D', 'Linea
rLocator', 'Locator', 'LogFormatter', 'LogFormatterExponent', 'LogFormatterMath
text', 'LogLocator', 'MaxNLocator', 'MultipleLocator', 'Normalize', 'NullFormat
ter', 'NullLocator', 'Number', 'PolarAxes', 'Polygon', 'Rectangle', 'ScalarForm
atter', 'Slider', 'Subplot', 'SubplotTool', 'Text', 'TickHelper', 'Widget', '_I
NSTALL_FIG_OBSERVER', '_IP_REGISTERED', '__builtins__', '__cached__', '__doc_
_', '__file__', '__loader__', '__name__', '__package__', '__spec__', '_auto_dra
w_if_interactive', '_autogen_docstring', '_backend_mod', '_get_running_interact
ive_framework', '_interactive_bk', '_log', '_pylab_helpers', '_setp', '_setup_p
yplot_info_docstrings', '_show', '_string_to_bool', 'acorr', 'angle_spectrum',
'annotate', 'arrow', 'autoscale', 'autumn', 'axes', 'axhline', 'axhspan', 'axi
s', 'axvline', 'axvspan', 'bar', 'barbs', 'barh', 'bone', 'box', 'boxplot', 'br
oken_barh', 'cla', 'clabel', 'clf', 'clim', 'close', 'cm', 'cohere', 'colorba
r', 'colormaps', 'connect', 'contour', 'contourf', 'cool', 'copper', 'csd', 'cy
cler', 'dedent', 'delaxes', 'deprecated', 'disconnect', 'docstring', 'draw', 'd
raw_all', 'draw_if_interactive', 'errorbar', 'eventplot', 'figaspect', 'figimag
e', 'figlegend', 'fignum_exists', 'figtext', 'figure', 'fill', 'fill_between',
'fill_betweenx', 'findobj', 'flag', 'gca', 'gcf', 'gci', 'get', 'get_backend',
'get_cmap', 'get_current_fig_manager', 'get_figlabels', 'get_fignums', 'get_plo
t_commands', 'get_scale_docs', 'get_scale_names', 'getp', 'ginput', 'gray', 'gr
id', 'hexbin', 'hist', 'hist2d', 'hlines', 'hot', 'hsv', 'importlib', 'imread',
'imsave', 'imshow', 'inferno', 'inspect', 'install_repl_displayhook', 'interact
ive', 'ioff', 'ion', 'isinteractive', 'jet', 'legend', 'locator_params', 'loggi
ng', 'loglog', 'magma', 'magnitude_spectrum', 'margins', 'matplotlib', 'matsho
w', 'minorticks_off', 'minorticks_on', 'mlab', 'new_figure_manager', 'nipy_spec
tral', 'np', 'pause', 'pcolor', 'pcolormesh', 'phase_spectrum', 'pie', 'pink',
'plasma', 'plot', 'plot_date', 'plotfile', 'plotting', 'polar', 'prism', 'psd',
'pylab_setup', 'quiver', 'quiverkey', 'rc', 'rcParams', 'rcParamsDefault', 'rcP
aramsOrig', 'rc_context', 'rcdefaults', 'rcsetup', 're', 'register_cmap', 'rgri
ds', 'savefig', 'sca', 'scatter', 'sci', 'semilogx', 'semilogy', 'set_cmap', 's
etp', 'show', 'silent_list', 'specgram', 'spring', 'spy', 'stackplot', 'stem',
'step', 'streamplot', 'style', 'subplot', 'subplot2grid', 'subplot_tool', 'subp
lots', 'subplots_adjust', 'summer', 'suptitle', 'switch_backend', 'sys', 'tabl
e', 'text', 'thetagrids', 'tick_params', 'ticklabel_format', 'tight_layout', 't
ime', 'title', 'tricontour', 'tricontourf', 'tripcolor', 'triplot', 'twinx', 't
winy', 'uninstall_repl_displayhook', 'violinplot', 'viridis', 'vlines', 'waitfo
rbuttonpress', 'warn_deprecated', 'warnings', 'winter', 'xcorr', 'xkcd', 'xlabe
l', 'xlim', 'xscale', 'xticks', 'ylabel', 'ylim', 'yscale', 'yticks']
```

In [65]:
```
# lineplot
# scatter plot
# bar plot
# box plot
# pie plot
```

In [67]:
```
x = [1,2,3,4]
y = [5,6,7,8]
plt.plot(x,y)
plt.show()
```



In [69]:
```
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.DataFame` 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')
```

```
- Alternatively, if your data is already a 2d array, you can pass it
  directly to *x*, *y*. A separate data set will be drawn for every
  column.

  Example: an array ``a`` where the first column represents the *x*
  values and the other columns are the *y* columns::

  >>> plot(a[0], a[1:])

- 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 the
'axes.prop_cycle' rcParam.

Parameters
----------
x, y : array-like or scalar
    The horizontal / vertical coordinates of the data points.
    *x* values are optional. If not given, they default to
    ``[0, ..., N-1]``.

    Commonly, these parameters are arrays of length N. However,
    scalars are supported as well (equivalent to an array with
    constant value).

    The parameters can also be 2-dimensional. Then, the columns
    represent separate data sets.

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.

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)`.
```

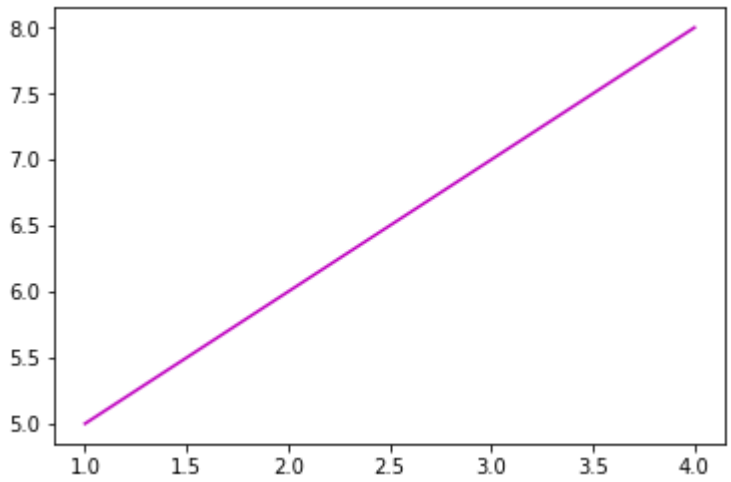
```
    Other Parameters
    ----------------
    scalex, scaley : bool, optional, default: True
        These parameters determined if the view limits are adapted to
        the data limits. The values are passed on to `autoscale_view`.

    **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 make multiple lines with one plot command, the kwargs
        apply to all those lines.

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

          agg_filter: a filter function, which takes a (m, n, 3) float array an
d a dpi value, and returns a (m, n, 3) array
      alpha: float
      animated: bool
      antialiased: bool
      clip_box: `.Bbox`
      clip_on: bool
      clip_path: [(`~matplotlib.path.Path`, `.Transform`) | `.Patch` | None]
      color: color
      contains: callable
      dash_capstyle: {'butt', 'round', 'projecting'}
      dash_joinstyle: {'miter', 'round', 'bevel'}
      dashes: sequence of floats (on/off ink in points) or (None, None)
      drawstyle: {'default', 'steps', 'steps-pre', 'steps-mid', 'steps-post'}
      figure: `.Figure`
      fillstyle: {'full', 'left', 'right', 'bottom', 'top', 'none'}
      gid: str
      in_layout: bool
      label: object
      linestyle: {'-', '--', '-.', ':', '', (offset, on-off-seq), ...}
      linewidth: float
      marker: unknown
      markeredgecolor: color
      markeredgewidth: float
      markerfacecolor: color
      markerfacecoloralt: color
      markersize: float
      markevery: unknown
      path_effects: `.AbstractPathEffect`
      picker: float or callable[[Artist, Event], Tuple[bool, dict]]
      pickradius: float
      rasterized: bool or None
      sketch_params: (scale: float, length: float, randomness: float)
      snap: bool or None
      solid_capstyle: {'butt', 'round', 'projecting'}
      solid_joinstyle: {'miter', 'round', 'bevel'}
```

```
  transform: matplotlib.transforms.Transform
  url: str
  visible: bool
  xdata: 1D array
  ydata: 1D array
  zorder: float

Returns
-------
lines
    A list of `.Line2D` objects representing the plotted data.


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

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.

**Colors**

The following color abbreviations are supported:

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

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'``).

**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
    ``'s'``          square marker
    ``'p'``          pentagon marker
    ``'*'``          star marker
    ``'h'``          hexagon1 marker
    ``'H'``          hexagon2 marker
    ``'+'``          plus marker
    ``'x'``          x 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
        'ro'   # red circles
        'g-'   # green solid line
        '--'   # dashed line with default color
        'k^:'  # black triangle_up markers connected by a dotted line

    .. note::
        In addition to the above described arguments, this function can take a
        **data** keyword argument. If such a **data** argument is given, the
        following arguments are replaced by **data[<arg>]**:

        * All arguments with the following names: 'x', 'y'.

        Objects passed as **data** must support item access (``data[<arg>]``) a
nd
        membership test (``<arg> in data``).
```

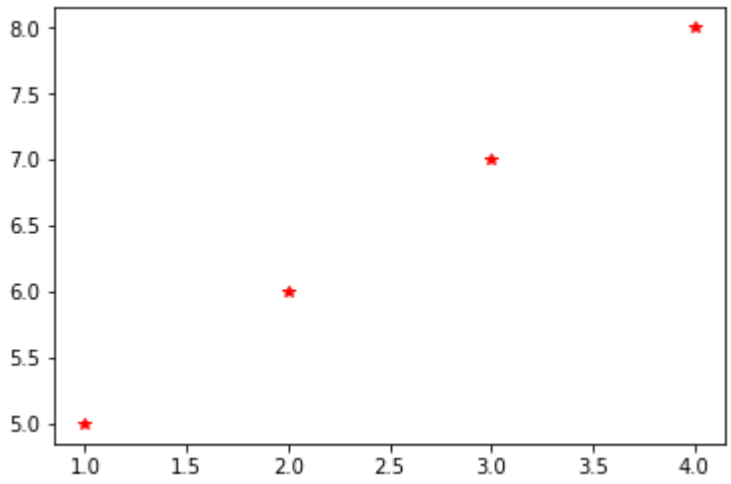
In [70]:
```
plt.plot(x,y,color = "m")
```

Out[70]:
```
[<matplotlib.lines.Line2D at 0x2ad71db3978>]
```



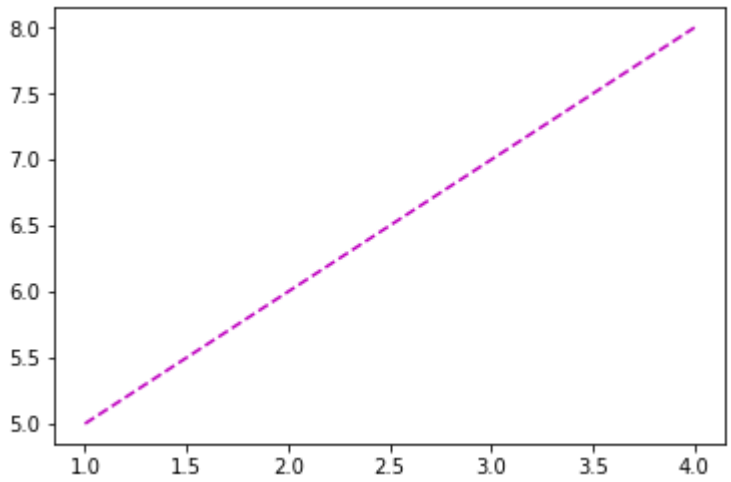
In [73]:
```
plt.plot(x,y,'r*')
```

Out[73]:
```
[<matplotlib.lines.Line2D at 0x2ad720249b0>]
```



In [74]:
```
plt.plot(x,y,"m--")
```

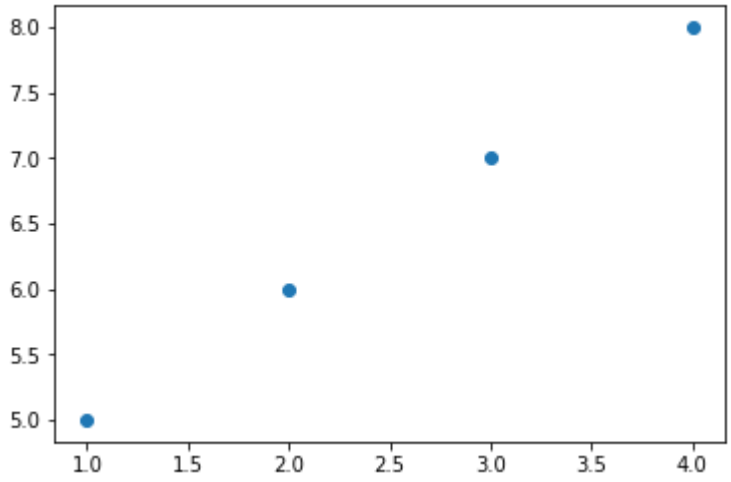
Out[74]:
```
[<matplotlib.lines.Line2D at 0x2ad72087278>]
```



scatter plot

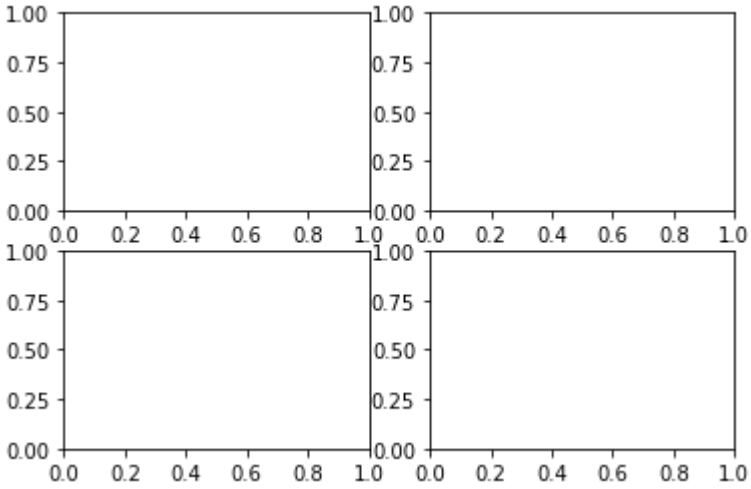
- display the data in points

In [76]:
```
plt.scatter(x,y)
plt.show()
```



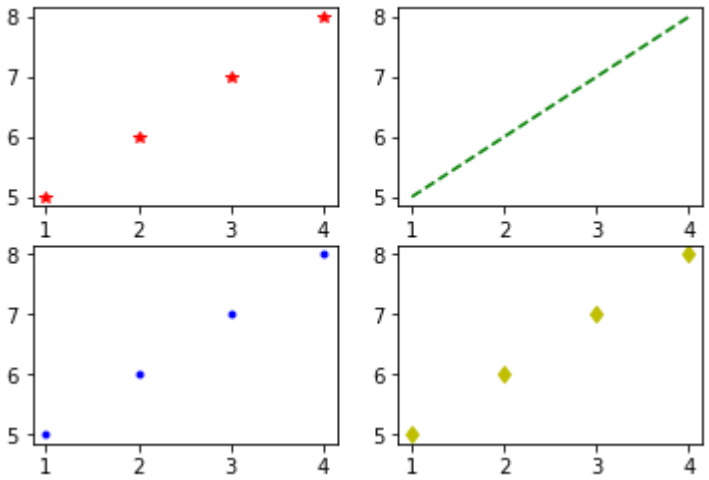
In [ ]:
```
## subplots
# plt.subplots(rows,columns,position)
```

In [78]:
```
plt.subplots(2,2)
plt.show()
```



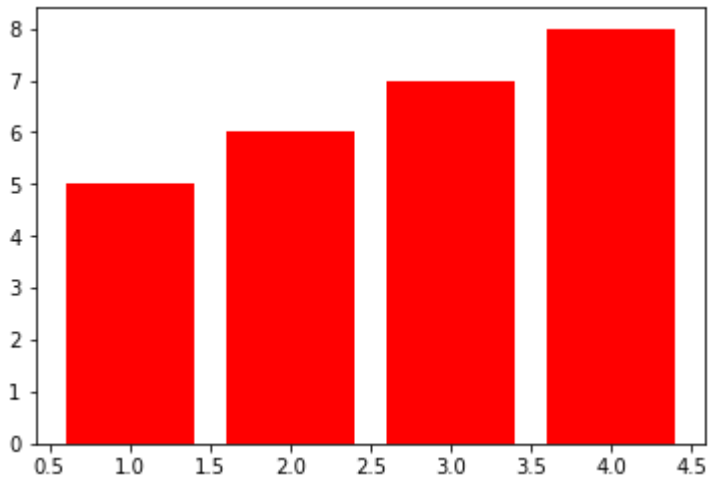
In [80]:
```
plt.subplot(2,2,1)
plt.plot(x,y,'r*')
plt.subplot(2,2,2)
plt.plot(x,y,'g--')
plt.subplot(2,2,3)
plt.plot(x,y,'b.')
plt.subplot(2,2,4)
plt.plot(x,y,'yd')
```

Out[80]:
```
[<matplotlib.lines.Line2D at 0x2ad72363c88>]
```



In [81]:
```
# bar plots
```

In [85]:
```
plt.bar(x,y,color = "r")
plt.show()
```



In [84]:
```
help(plt.bar)
```

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

bar(x, height, width=0.8, bottom=None, *, align='center', data=None, **kwargs)
    Make a bar plot.

    The bars are positioned at *x* with the given *align*\ment. Their
    dimensions are given by *width* and *height*. The vertical baseline
    is *bottom* (default 0).

    Each of *x*, *height*, *width*, and *bottom* may either be a scalar
    applying to all bars, or it may be a sequence of length N providing a
    separate value for each bar.

    Parameters
    ----------
    x : sequence of scalars
        The x coordinates of the bars. See also *align* for the
        alignment of the bars to the coordinates.

    height : scalar or sequence of scalars
        The height(s) of the bars.

    width : scalar or array-like, optional
        The width(s) of the bars (default: 0.8).

    bottom : scalar or array-like, optional
        The y coordinate(s) of the bars bases (default: 0).

    align : {'center', 'edge'}, optional, default: 'center'
        Alignment of the bars to the *x* coordinates:

        - 'center': Center the base on the *x* positions.
        - 'edge': Align the left edges of the bars with the *x* positions.

        To align the bars on the right edge pass a negative *width* and
        ``align='edge'``.

    Returns
    -------
    container : `.BarContainer`
        Container with all the bars and optionally errorbars.

    Other Parameters
    ----------------
    color : scalar or array-like, optional
        The colors of the bar faces.

    edgecolor : scalar or array-like, optional
        The colors of the bar edges.

    linewidth : scalar or array-like, optional
        Width of the bar edge(s). If 0, don't draw edges.

    tick_label : string or array-like, optional
```

```
        The tick labels of the bars.
        Default: None (Use default numeric labels.)

    xerr, yerr : scalar or array-like of shape(N,) or shape(2,N), optional
        If not *None*, add horizontal / vertical errorbars to the bar tips.
        The values are +/- sizes relative to the data:

        - scalar: symmetric +/- values for all bars
        - shape(N,): symmetric +/- values for each bar
        - shape(2,N): Separate - and + values for each bar. First row
            contains the lower errors, the second row contains the
            upper errors.
        - *None*: No errorbar. (Default)

        See :doc:`/gallery/statistics/errorbar_features`
        for an example on the usage of ``xerr`` and ``yerr``.

    ecolor : scalar or array-like, optional, default: 'black'
        The line color of the errorbars.

    capsize : scalar, optional
       The length of the error bar caps in points.
       Default: None, which will take the value from
       :rc:`errorbar.capsize`.

    error_kw : dict, optional
        Dictionary of kwargs to be passed to the `~.Axes.errorbar`
        method. Values of *ecolor* or *capsize* defined here take
        precedence over the independent kwargs.

    log : bool, optional, default: False
        If *True*, set the y-axis to be log scale.

    orientation : {'vertical',  'horizontal'}, optional
        *This is for internal use only.* Please use `barh` for
        horizontal bar plots. Default: 'vertical'.

    See also
    --------
    barh: Plot a horizontal bar plot.

    Notes
    -----
    The optional arguments *color*, *edgecolor*, *linewidth*,
    *xerr*, and *yerr* can be either scalars or sequences of
    length equal to the number of bars.  This enables you to use
    bar as the basis for stacked bar charts, or candlestick plots.
    Detail: *xerr* and *yerr* are passed directly to
    :meth:`errorbar`, so they can also have shape 2xN for
    independent specification of lower and upper errors.

    Other optional kwargs:

      agg_filter: a filter function, which takes a (m, n, 3) float array and a
dpi value, and returns a (m, n, 3) array
      alpha: float or None
      animated: bool
```

```
        antialiased: unknown
        capstyle: {'butt', 'round', 'projecting'}
        clip_box: `.Bbox`
        clip_on: bool
        clip_path: [(`~matplotlib.path.Path`, `.Transform`) | `.Patch` | None]
        color: color
        contains: callable
        edgecolor: color or None or 'auto'
        facecolor: color or None
        figure: `.Figure`
        fill: bool
        gid: str
        hatch: {'/', '\\', '|', '-', '+', 'x', 'o', 'O', '.', '*'}
        in_layout: bool
        joinstyle: {'miter', 'round', 'bevel'}
        label: object
        linestyle: {'-', '--', '-.', ':', '', (offset, on-off-seq), ...}
        linewidth: float or None for default
        path_effects: `.AbstractPathEffect`
        picker: None or bool or float or callable
        rasterized: bool or None
        sketch_params: (scale: float, length: float, randomness: float)
        snap: bool or None
        transform: `.Transform`
        url: str
        visible: bool
        zorder: float

    .. note::
        In addition to the above described arguments, this function can take a
        **data** keyword argument. If such a **data** argument is given, the
        following arguments are replaced by **data[<arg>]**:

        * All arguments with the following names: 'bottom', 'color', 'ecolor',
'edgecolor', 'height', 'left', 'linewidth', 'tick_label', 'width', 'x', 'xerr',
'y', 'yerr'.
        * All positional arguments.

        Objects passed as **data** must support item access (``data[<arg>]``) a
nd
        membership test (``<arg> in data``).
```

Seaborn

In [86]:
```
import seaborn as sns
```

In [87]:
```
sns.get_dataset_names()
```

```
C:\Users\Alekhya\Anaconda3\lib\site-packages\seaborn\utils.py:376: UserWarning:
No parser was explicitly specified, so I'm using the best available HTML parser
for this system ("lxml"). This usually isn't a problem, but if you run this cod
e on another system, or in a different virtual environment, it may use a differ
ent parser and behave differently.

The code that caused this warning is on line 376 of the file C:\Users\Alekhya\A
naconda3\lib\site-packages\seaborn\utils.py. To get rid of this warning, pass t
he additional argument 'features="lxml"' to the BeautifulSoup constructor.

  gh_list = BeautifulSoup(http)
```

Out[87]:
```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'exercise',
 'flights',
 'fmri',
 'gammas',
 'geyser',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'tips',
 'titanic']
```

In [88]:
```
print(dir(sns))
```

```
['FacetGrid', 'JointGrid', 'PairGrid', '__builtins__', '__cached__', '__doc__',
'__file__', '__loader__', '__name__', '__package__', '__path__', '__spec__', '_
_version__', '_orig_rc_params', 'algorithms', 'axes_style', 'axisgrid', 'barplo
t', 'blend_palette', 'boxenplot', 'boxplot', 'categorical', 'catplot', 'choose_
colorbrewer_palette', 'choose_cubehelix_palette', 'choose_dark_palette', 'choos
e_diverging_palette', 'choose_light_palette', 'clustermap', 'cm', 'color_palett
e', 'colors', 'countplot', 'crayon_palette', 'crayons', 'cubehelix_palette', 'd
ark_palette', 'desaturate', 'despine', 'distplot', 'distributions', 'diverging_
palette', 'dogplot', 'external', 'factorplot', 'get_dataset_names', 'heatmap',
'hls_palette', 'husl_palette', 'jointplot', 'kdeplot', 'light_palette', 'linepl
ot', 'lmplot', 'load_dataset', 'lvplot', 'matrix', 'miscplot', 'mpl', 'mpl_pale
tte', 'pairplot', 'palettes', 'palplot', 'plotting_context', 'pointplot', 'rcmo
d', 'regplot', 'regression', 'relational', 'relplot', 'reset_defaults', 'reset_
orig', 'residplot', 'rugplot', 'saturate', 'scatterplot', 'set', 'set_color_cod
es', 'set_context', 'set_hls_values', 'set_palette', 'set_style', 'stripplot',
'swarmplot', 'timeseries', 'tsplot', 'utils', 'violinplot', 'widgets', 'xkcd_pa
lette', 'xkcd_rgb']
```

In [89]:
```
help(sns.colors)
```

```
Help on package seaborn.colors in seaborn:

NAME
    seaborn.colors

PACKAGE CONTENTS
    crayons
    xkcd_rgb

DATA
    crayons = {'Almond': '#EFDECD', 'Antique Brass': '#CD9575', 'Apricot':...
    xkcd_rgb = {'acid green': '#8ffe09', 'adobe': '#bd6c48', 'algae': '#54...

FILE
    c:\users\alekhya\anaconda3\lib\site-packages\seaborn\colors\__init__.py
```

In [90]:
```
sns.axes_style()
```

Out[90]:
```
{'axes.facecolor': 'white',
 'axes.edgecolor': 'black',
 'axes.grid': False,
 'axes.axisbelow': 'line',
 'axes.labelcolor': 'black',
 'figure.facecolor': (1, 1, 1, 0),
 'grid.color': '#b0b0b0',
 'grid.linestyle': '-',
 'text.color': 'black',
 'xtick.color': 'black',
 'ytick.color': 'black',
 'xtick.direction': 'out',
 'ytick.direction': 'out',
 'lines.solid_capstyle': 'projecting',
 'patch.edgecolor': 'black',
 'image.cmap': 'viridis',
 'font.family': ['sans-serif'],
 'font.sans-serif': ['DejaVu Sans',
  'Bitstream Vera Sans',
  'Computer Modern Sans Serif',
  'Lucida Grande',
  'Verdana',
  'Geneva',
  'Lucid',
  'Arial',
  'Helvetica',
  'Avant Garde',
  'sans-serif'],
 'patch.force_edgecolor': False,
 'xtick.bottom': True,
 'xtick.top': False,
 'ytick.left': True,
 'ytick.right': False,
 'axes.spines.left': True,
 'axes.spines.bottom': True,
 'axes.spines.right': True,
 'axes.spines.top': True}
```

In [92]:
```
sns.get_dataset_names()
```

```
C:\Users\Alekhya\Anaconda3\lib\site-packages\seaborn\utils.py:376: UserWarning:
No parser was explicitly specified, so I'm using the best available HTML parser
for this system ("lxml"). This usually isn't a problem, but if you run this cod
e on another system, or in a different virtual environment, it may use a differ
ent parser and behave differently.

The code that caused this warning is on line 376 of the file C:\Users\Alekhya\A
naconda3\lib\site-packages\seaborn\utils.py. To get rid of this warning, pass t
he additional argument 'features="lxml"' to the BeautifulSoup constructor.

  gh_list = BeautifulSoup(http)
```

Out[92]:
```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'exercise',
 'flights',
 'fmri',
 'gammas',
 'geyser',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'tips',
 'titanic']
```

In [94]:
```
a = sns.load_dataset("tips")
a.head()
```

Out[94]:

	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 [95]:
```
a.shape
```

Out[95]:
```
(244, 7)
```

In [96]:
```
a.columns
```

Out[96]:
```
Index(['total_bill', 'tip', 'sex', 'smoker', 'day', 'time', 'size'], dtype='obj
ect')
```

In [97]:
```
a.info()
```

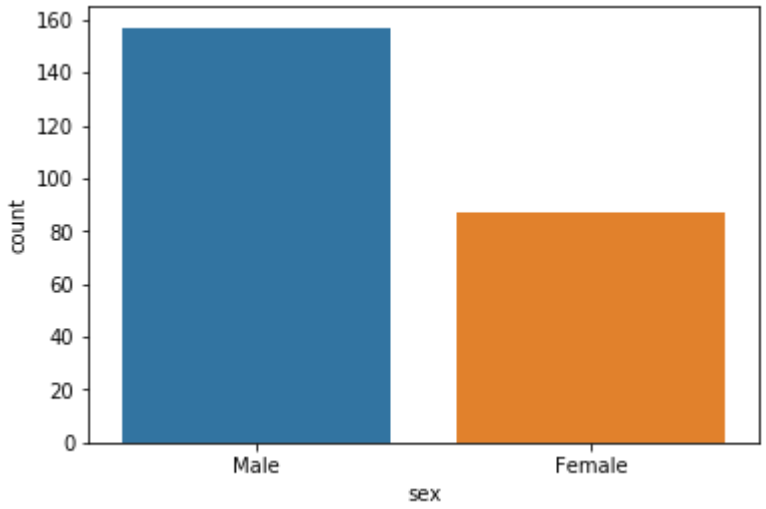
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
total_bill    244 non-null float64
tip           244 non-null float64
sex           244 non-null category
smoker        244 non-null category
day           244 non-null category
time          244 non-null category
size          244 non-null int64
dtypes: category(4), float64(2), int64(1)
memory usage: 7.2 KB
```

In [98]:
```
a.isna().sum()
```

Out[98]:
```
total_bill    0
tip           0
sex           0
smoker        0
day           0
time          0
size          0
dtype: int64
```

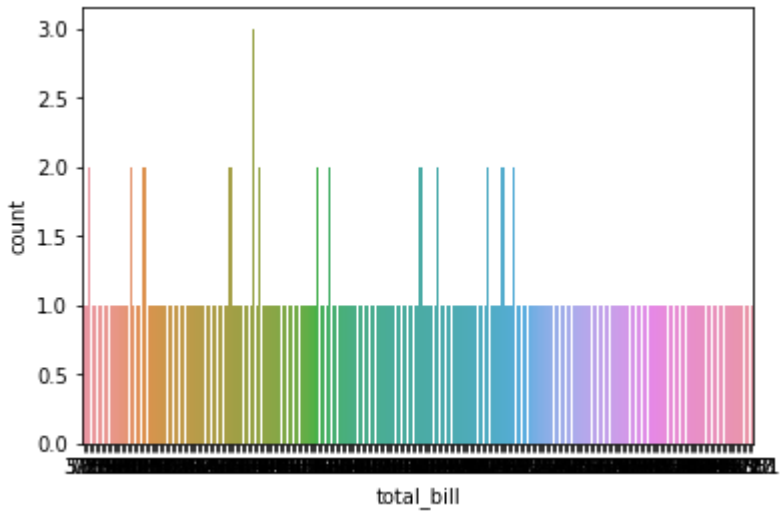
In [99]:
```
# count plot
sns.countplot(x = "sex",data =a)
```

Out[99]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2ad721fb6a0>
```



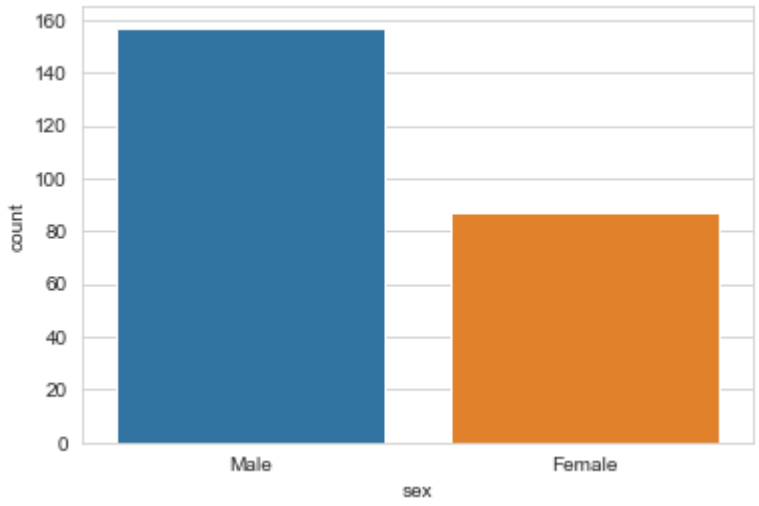
In [100]:
```
sns.countplot(x = "total_bill",data =a)
```

Out[100]: <matplotlib.axes._subplots.AxesSubplot at 0x2ad72c61e48>



In [101]:
```
sns.set_style("whitegrid")
sns.countplot(x="sex",data=a)
```

Out[101]: <matplotlib.axes._subplots.AxesSubplot at 0x2ad725835c0>



In [ ]: