Ch02-statlearn-lab

January 22, 2026

[1]:
```
print('fit a model with', 11, 'variables')
```

```
fit a model with 11 variables
```

[2]:
```
print?
```

```
Signature: print(*args, sep=' ', end='\n',
file=None, flush=False)
Docstring:
Prints the values to a stream, or to sys.stdout by default.

sep
  string inserted between values, default a space.
end
  string appended after the last value, default a newline.
file
  a file-like object (stream); defaults to the current sys.stdout.
flush
  whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

[3]:
```
3 + 5
```

[3]: 8

[4]:
```
"hello" + " " + "world"
```

[4]: 'hello world'

[5]:
```
x = [3, 4, 5]
x
```

[5]: [3, 4, 5]

[6]:
```
y = [4, 9, 7]
x + y
```

[6]: [3, 4, 5, 4, 9, 7]

[7]:
```
import numpy as np
```

[8]:
```
x = np.array([3, 4, 5])
y = np.array([4, 9, 7])
```

[9]:
```
x + y
```

[9]:
```
array([ 7, 13, 12])
```

[10]:
```
x = np.array([[1, 2], [3, 4]])
x
```

[10]:
```
array([[1, 2],
       [3, 4]])
```

[11]:
```
x.ndim
```

[11]:
```
2
```

[12]:
```
x.dtype
```

[12]:
```
dtype('int64')
```

[13]:
```
np.array([[1, 2], [3.0, 4]]).dtype
```

[13]:
```
dtype('float64')
```

[14]:
```
np.array?
```

```
Signature:
np.array(
    object,
    dtype=None,
    *,
    copy=True,
    order='K',
    subok=False,
    ndmin=0,
    ndmax=0,
    like=None,
)
Docstring:
array(object, dtype=None, *, copy=True, order='K', subok=False, ndmin=0,
      ndmax=0, like=None)

Create an array.

Parameters
----------
```

```
object : array_like
    An array, any object exposing the array interface, an object whose
    ``__array__`` method returns an array, or any (nested) sequence.
    If object is a scalar, a 0-dimensional array containing object is
    returned.
dtype : data-type, optional
    The desired data-type for the array. If not given, NumPy will try to use
    a default ``dtype`` that can represent the values (by applying promotion
    rules when necessary.)
copy : bool, optional
    If ``True`` (default), then the array data is copied. If ``None``,
    a copy will only be made if ``__array__`` returns a copy, if obj is
    a nested sequence, or if a copy is needed to satisfy any of the other
    requirements (``dtype``, ``order``, etc.). Note that any copy of
    the data is shallow, i.e., for arrays with object dtype, the new
    array will point to the same objects. See Examples for `ndarray.copy`.
    For ``False`` it raises a ``ValueError`` if a copy cannot be avoided.
    Default: ``True``.
order : {'K', 'A', 'C', 'F'}, optional
    Specify the memory layout of the array. If object is not an array, the
    newly created array will be in C order (row major) unless 'F' is
    specified, in which case it will be in Fortran order (column major).
    If object is an array the following holds.

    ===== ========= ===================================================
    order  no copy                     copy=True
    ===== ========= ===================================================
    'K'   unchanged F & C order preserved, otherwise most similar order
    'A'   unchanged F order if input is F and not C, otherwise C order
    'C'   C order   C order
    'F'   F order   F order
    ===== ========= ===================================================

    When ``copy=None`` and a copy is made for other reasons, the result is
    the same as if ``copy=True``, with some exceptions for 'A', see the
    Notes section. The default order is 'K'.
subok : bool, optional
    If True, then sub-classes will be passed-through, otherwise
    the returned array will be forced to be a base-class array (default).
ndmin : int, optional
    Specifies the minimum number of dimensions that the resulting
    array should have.  Ones will be prepended to the shape as
    needed to meet this requirement.
ndmax : int, optional
    Specifies the maximum number of dimensions to create when inferring
    shape from nested sequences. By default (ndmax=0), NumPy recurses
    through all nesting levels (up to the compile-time constant
    ``NPY_MAXDIMS``).
```

```
    Setting ``ndmax`` stops recursion at the specified depth, preserving
    deeper nested structures as objects instead of promoting them to
    higher-dimensional arrays. In this case, ``dtype=object`` is required.

    .. versionadded:: 2.4.0
like : array_like, optional
    Reference object to allow the creation of arrays which are not
    NumPy arrays. If an array-like passed in as ``like`` supports
    the ``__array_function__`` protocol, the result will be defined
    by it. In this case, it ensures the creation of an array object
    compatible with that passed in via this argument.

    .. versionadded:: 1.20.0

Returns
-------
out : ndarray
    An array object satisfying the specified requirements.

See Also
--------
empty_like : Return an empty array with shape and type of input.
ones_like : Return an array of ones with shape and type of input.
zeros_like : Return an array of zeros with shape and type of input.
full_like : Return a new array with shape of input filled with value.
empty : Return a new uninitialized array.
ones : Return a new array setting values to one.
zeros : Return a new array setting values to zero.
full : Return a new array of given shape filled with value.
copy : Return an array copy of the given object.


Notes
-----
When order is 'A' and ``object`` is an array in neither 'C' nor 'F' order,
and a copy is forced by a change in dtype, then the order of the result is
not necessarily 'C' as expected. This is likely a bug.

Examples
--------
>>> import numpy as np
>>> np.array([1, 2, 3])
array([1, 2, 3])

Upcasting:

>>> np.array([1, 2, 3.0])
array([ 1.,  2.,  3.])
```

```
More than one dimension:

>>> np.array([[1, 2], [3, 4]])
array([[1, 2],
       [3, 4]])

Minimum dimensions 2:

>>> np.array([1, 2, 3], ndmin=2)
array([[1, 2, 3]])

Type provided:

>>> np.array([1, 2, 3], dtype=complex)
array([ 1.+0.j,  2.+0.j,  3.+0.j])

Data-type consisting of more than one element:

>>> x = np.array([(1,2),(3,4)],dtype=[('a','<i4'),('b','<i4')])
>>> x['a']
array([1, 3], dtype=int32)

Creating an array from sub-classes:

>>> np.array(np.asmatrix('1 2; 3 4'))
array([[1, 2],
       [3, 4]])

>>> np.array(np.asmatrix('1 2; 3 4'), subok=True)
matrix([[1, 2],
        [3, 4]])

Limiting the maximum dimensions with ``ndmax``:

>>> a = np.array([[1, 2], [3, 4]], dtype=object, ndmax=2)
>>> a
array([[1, 2],
       [3, 4]], dtype=object)
>>> a.shape
(2, 2)

>>> b = np.array([[1, 2], [3, 4]], dtype=object, ndmax=1)
>>> b
array([list([1, 2]), list([3, 4])], dtype=object)
>>> b.shape
(2,)
Type:      builtin_function_or_method
```

[15]:
```
np.array([[1, 2], [3, 4]], float).dtype
```

[15]:
```
dtype('float64')
```

[16]:
```
x.shape
```

[16]:
```
(2, 2)
```

[17]:
```
x = np.array([1, 2, 3, 4])
x.sum()
```

[17]:
```
np.int64(10)
```

[18]:
```
x = np.array([1, 2, 3, 4])
np.sum(x)
```

[18]:
```
np.int64(10)
```

[19]:
```
x = np.array([1, 2, 3, 4, 5, 6])
print('beginning x:\n', x)
x_reshape = x.reshape((2, 3))
print('reshaped x:\n', x_reshape)
```

```
beginning x:
 [1 2 3 4 5 6]
reshaped x:
 [[1 2 3]
 [4 5 6]]
```

[20]:
```
x_reshape[0, 0]
```

[20]:
```
np.int64(1)
```

[21]:
```
x_reshape[1, 2]
```

[21]:
```
np.int64(6)
```

[22]:
```
print('x before we modify x_reshape:\n', x)
print('x_reshape before we modify x_reshape:\n', x_reshape)
x_reshape[0, 0] = 5
print('x_reshape after we modify its top left element:\n', x_reshape)
print('x after we modify top left element of x_reshape:\n', x)
```

```
x before we modify x_reshape:
 [1 2 3 4 5 6]
x_reshape before we modify x_reshape:
 [[1 2 3]
 [4 5 6]]
x_reshape after we modify its top left element:
```

```
 [[5 2 3]
 [4 5 6]]
x after we modify top left element of x_reshape:
 [5 2 3 4 5 6]
```

[23]:
```
my_tuple = (3, 4, 5)
my_tuple[0] = 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[23], line 2
      1 my_tuple = (3, 4, 5)
----> 2 my_tuple[0] = 2

TypeError: 'tuple' object does not support item assignment
```

[ ]:
```
x_reshape.shape, x_reshape.ndim, x_reshape.T
```

[ ]:
```
((2, 3),
 2,
 array([[5, 4],
        [2, 5],
        [3, 6]]))
```

[ ]:
```
np.sqrt(x)
```

[ ]:
```
array([2.23606798, 1.41421356, 1.73205081, 2.        , 2.23606798,
       2.44948974])
```

[ ]:
```
x**2
```

[ ]:
```
array([25,  4,  9, 16, 25, 36])
```

[ ]:
```
x**0.5
```

[ ]:
```
array([2.23606798, 1.41421356, 1.73205081, 2.        , 2.23606798,
       2.44948974])
```

[ ]:
```
x = np.random.normal(size=50)
x
```

[ ]:
```
array([ 0.63214394,  0.34288365,  0.85005043, -0.47385094,  0.32864279,
       -0.78796854,  0.51125391, -1.12947776, -0.71391649, -1.92367418,
       -0.42609273, -0.7309404 ,  0.83090301, -0.47570241, -0.5852834 ,
        0.89788488, -1.27638353,  2.59421427,  1.25973331, -0.61486026,
        1.3979914 ,  0.79368659, -1.14506316, -2.00869422,  0.19502313,
       -0.45956661,  0.65944298,  0.69723473, -1.09211872,  0.15972909,
```

```
       -0.22530418,  0.32276968,  0.50374192,  0.80506415, -0.48224507,
       -0.53214642, -1.09662644, -1.52722433, -2.10241781, -0.42403843,
        1.32310673,  2.00932483, -0.16726754,  2.30113584, -0.68976675,
       -0.67621251, -0.78371139, -0.62709974, -1.39453964,  0.25195549])
```

[ ]:
```
y = x + np.random.normal(loc=50, scale=1, size=50)
```

[ ]:
```
np.corrcoef(x, y)
```

[ ]:
```
array([[1.        , 0.71781515],
       [0.71781515, 1.        ]])
```

[ ]:
```
print(np.random.normal(scale=5, size=2))
print(np.random.normal(scale=5, size=2))
```

```
[-0.97181857  3.2315884 ]
[ 0.73365791 -7.23683363]
```

[ ]:
```
rng = np.random.default_rng(1303)
print(rng.normal(scale=5, size=2))
rng2 = np.random.default_rng(1303)
print(rng2.normal(scale=5, size=2))
```

```
[ 4.09482632 -1.07485605]
[ 4.09482632 -1.07485605]
```

[ ]:
```
rng = np.random.default_rng(3)
y = rng.standard_normal(10)
np.mean(y), y.mean()
```

[ ]:
```
(-0.1126795190952861, -0.1126795190952861)
```

[ ]:
```
np.var(y), y.var(), np.mean((y - y.mean())**2)
```

[ ]:
```
(2.7243406406465125, 2.7243406406465125, 2.7243406406465125)
```

[ ]:
```
np.sqrt(np.var(y)), np.std(y)
```

[ ]:
```
(1.6505576756498128, 1.6505576756498128)
```

[ ]:
```
X = rng.standard_normal((10, 3))
X
```

[ ]:
```
array([[ 0.22578661, -0.35263079, -0.28128742],
       [-0.66804635, -1.05515055, -0.39080098],
       [ 0.48194539, -0.23855361,  0.9577587 ],
       [-0.19980213,  0.02425957,  1.54582085],
       [ 0.54510552, -0.50522874, -0.18283897],
       [ 0.54052513,  1.93508803, -0.26962033],
```

```
       [-0.24355868,  1.0023136 , -0.88645994],
       [-0.29172023,  0.88253897,  0.58035002],
       [ 0.0915167 ,  0.67010435, -2.82816231],
       [ 1.02130682, -0.95964476, -1.66861984]])
```

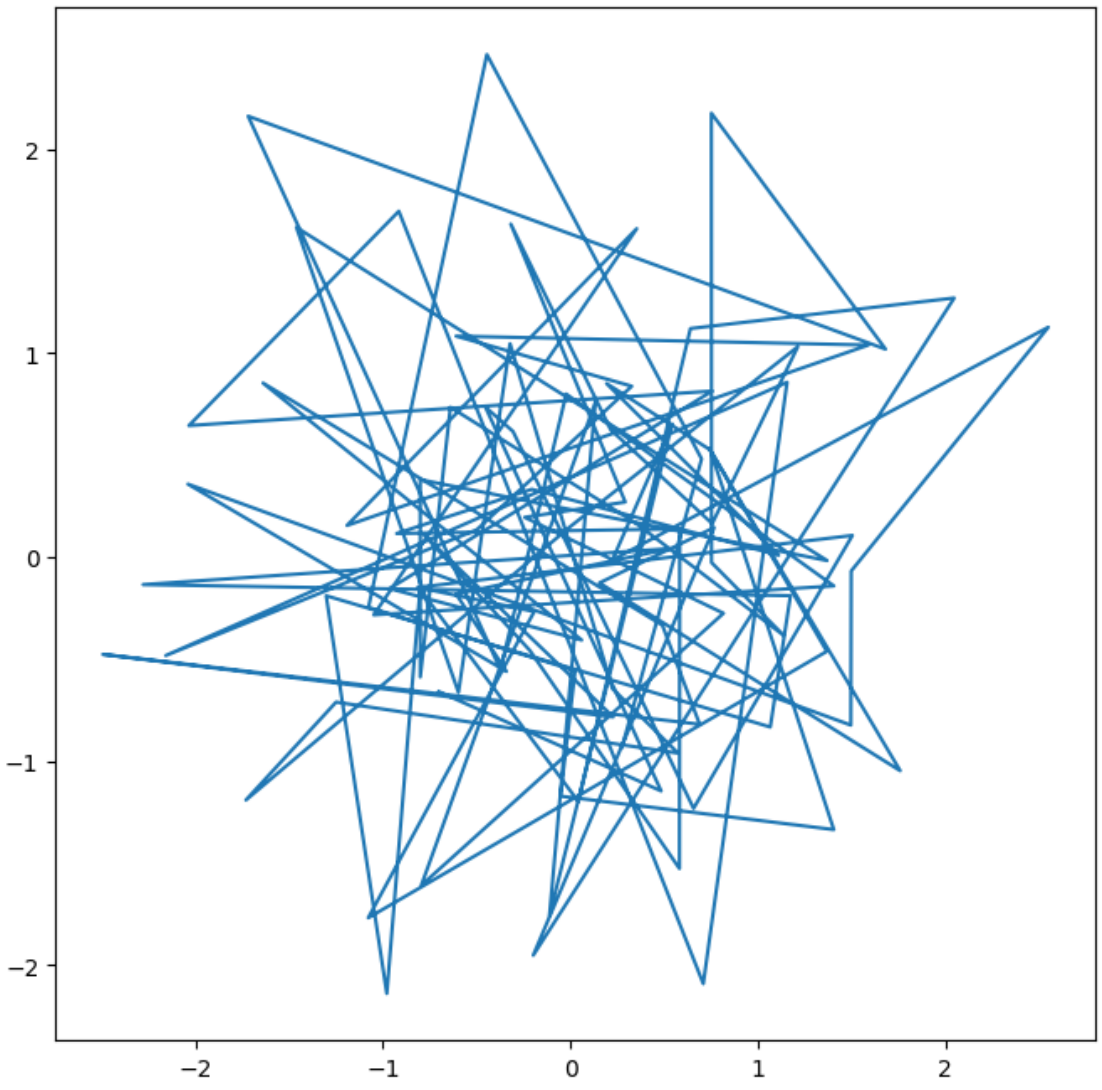
[ ]:
```
X.mean(axis=0)
```

[ ]:
```
array([ 0.15030588,  0.14030961, -0.34238602])
```

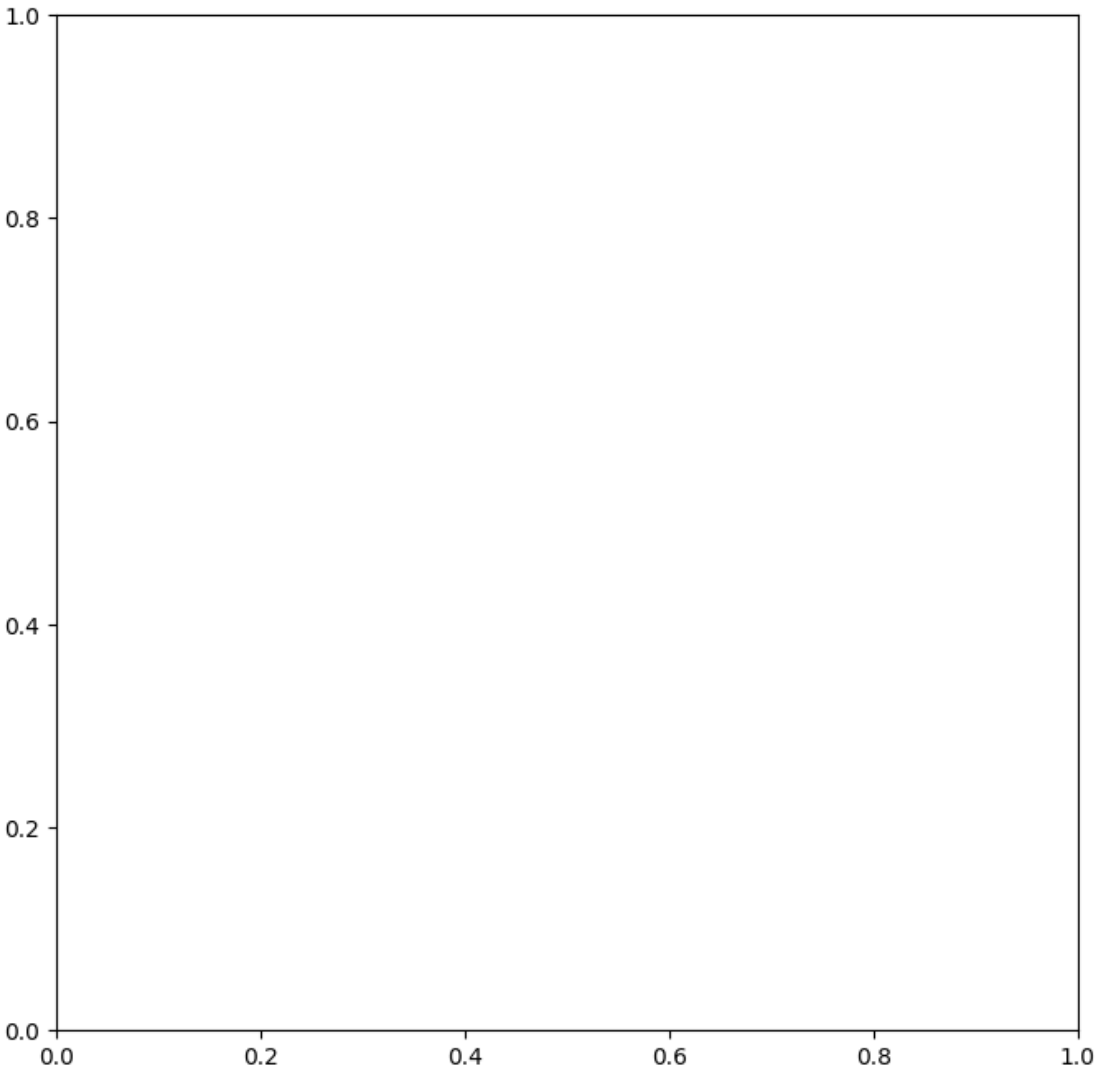
[ ]:
```
X.mean(0)
```

[ ]:
```
array([ 0.15030588,  0.14030961, -0.34238602])
```

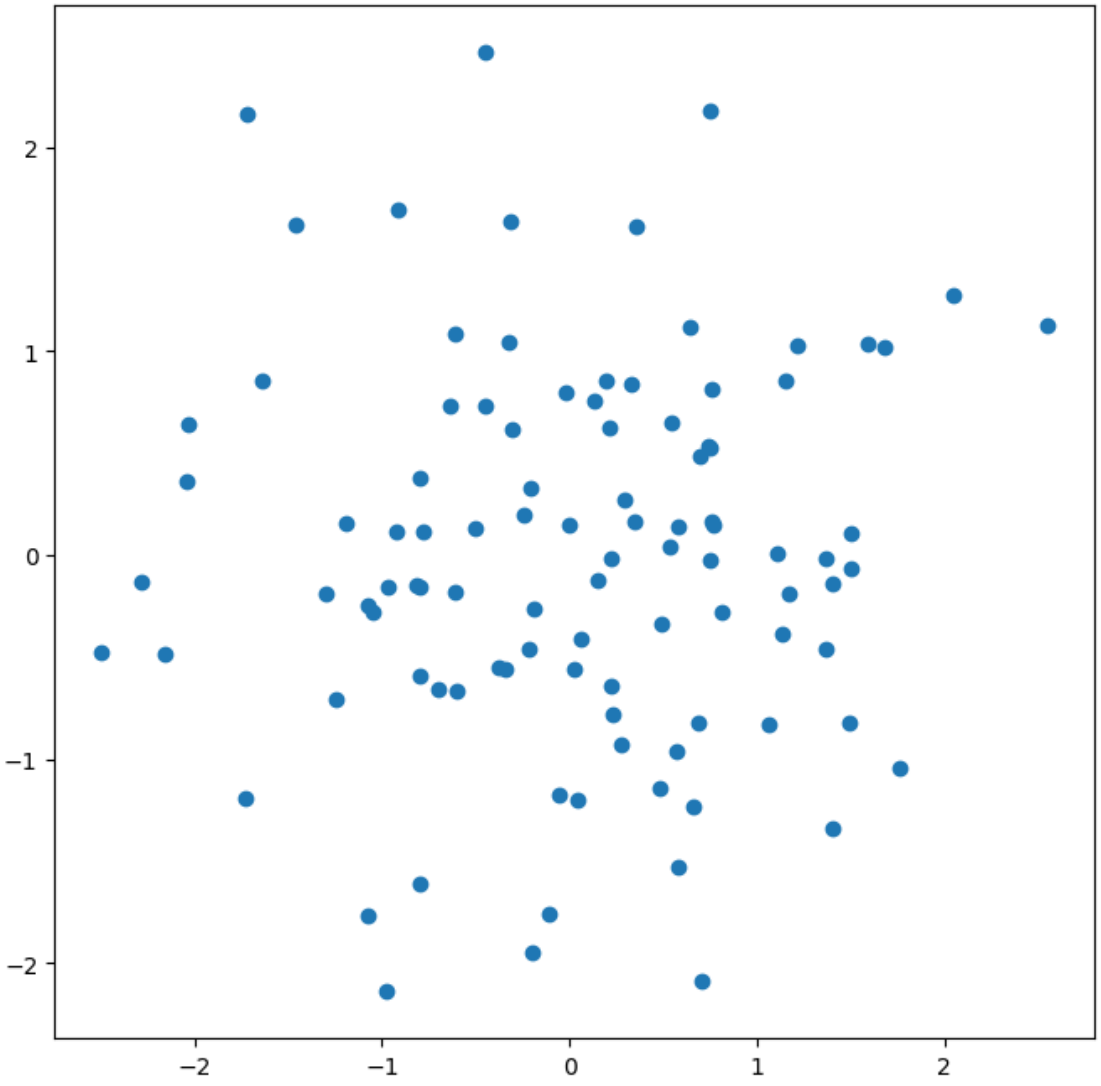
[ ]:
```
from matplotlib.pyplot import subplots
fig, ax = subplots(figsize=(8, 8))
x = rng.standard_normal(100)
y = rng.standard_normal(100)
ax.plot(x, y);
```



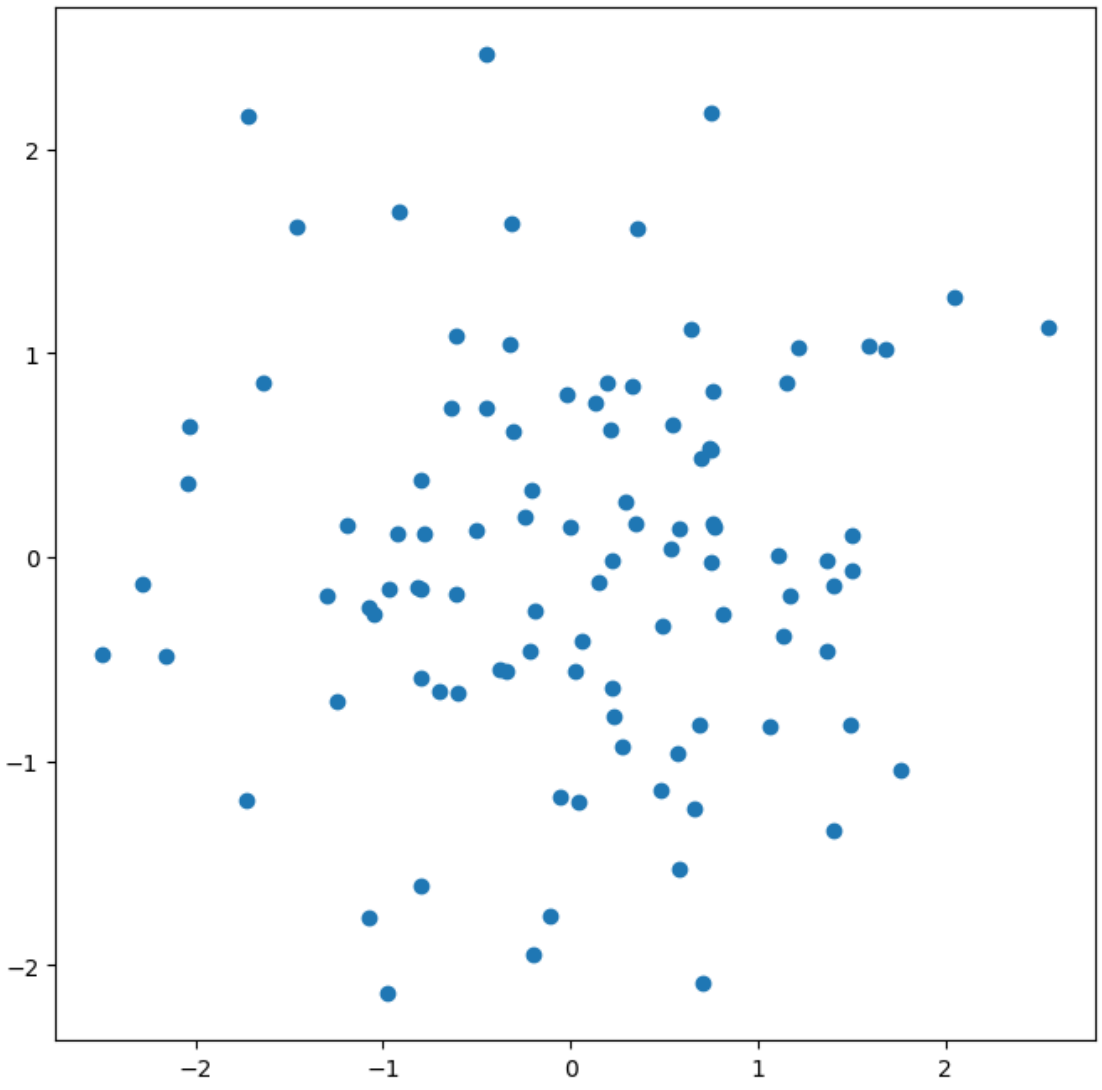
[ ]:
```
output = subplots(figsize=(8, 8))
fig = output[0]
ax = output[1]
```



[ ]:
```
fig, ax = subplots(figsize=(8, 8))
ax.plot(x, y, 'o');
```

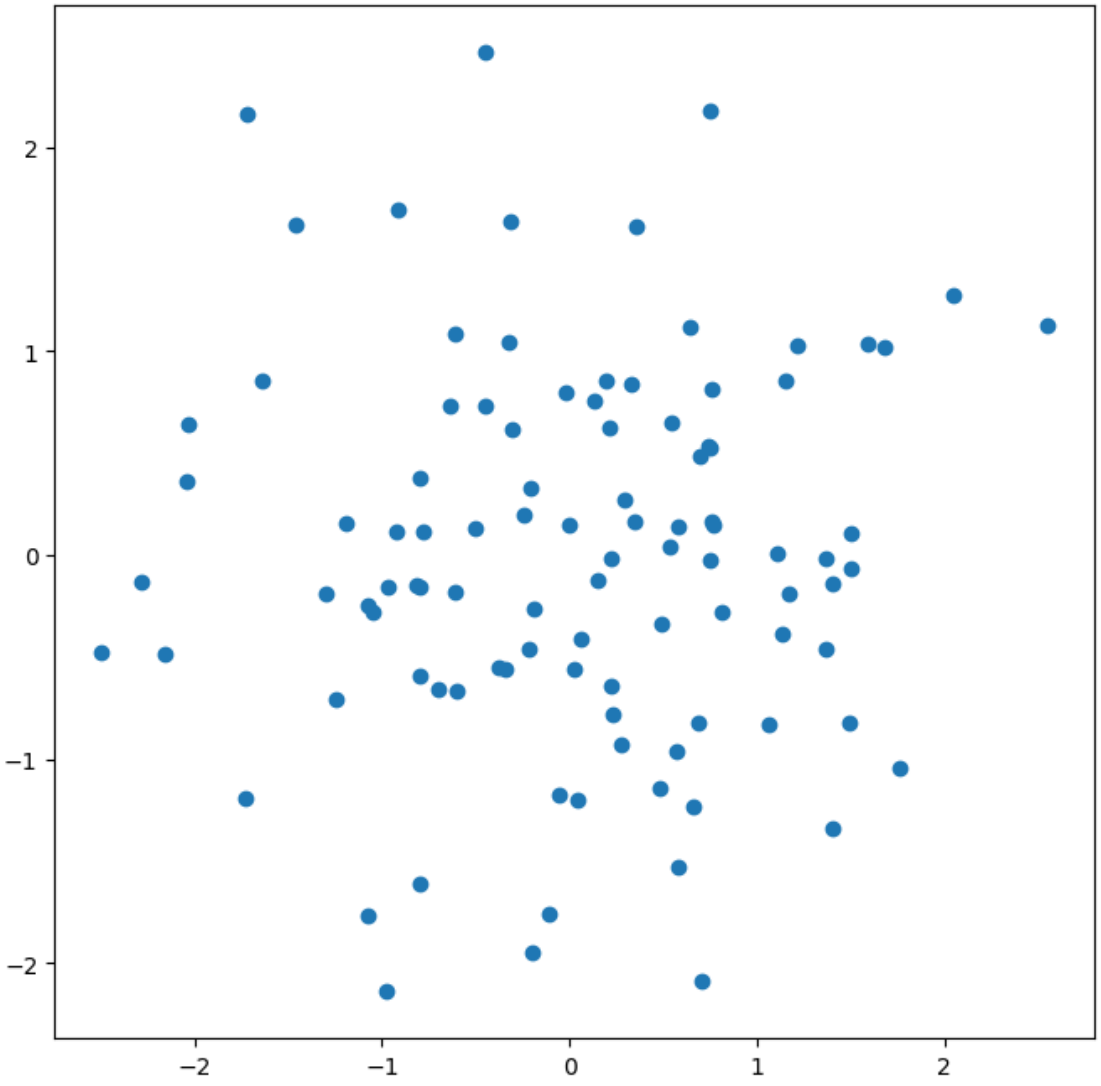


[ ]:
```
fig, ax = subplots(figsize=(8, 8))
ax.scatter(x, y, marker='o');
```

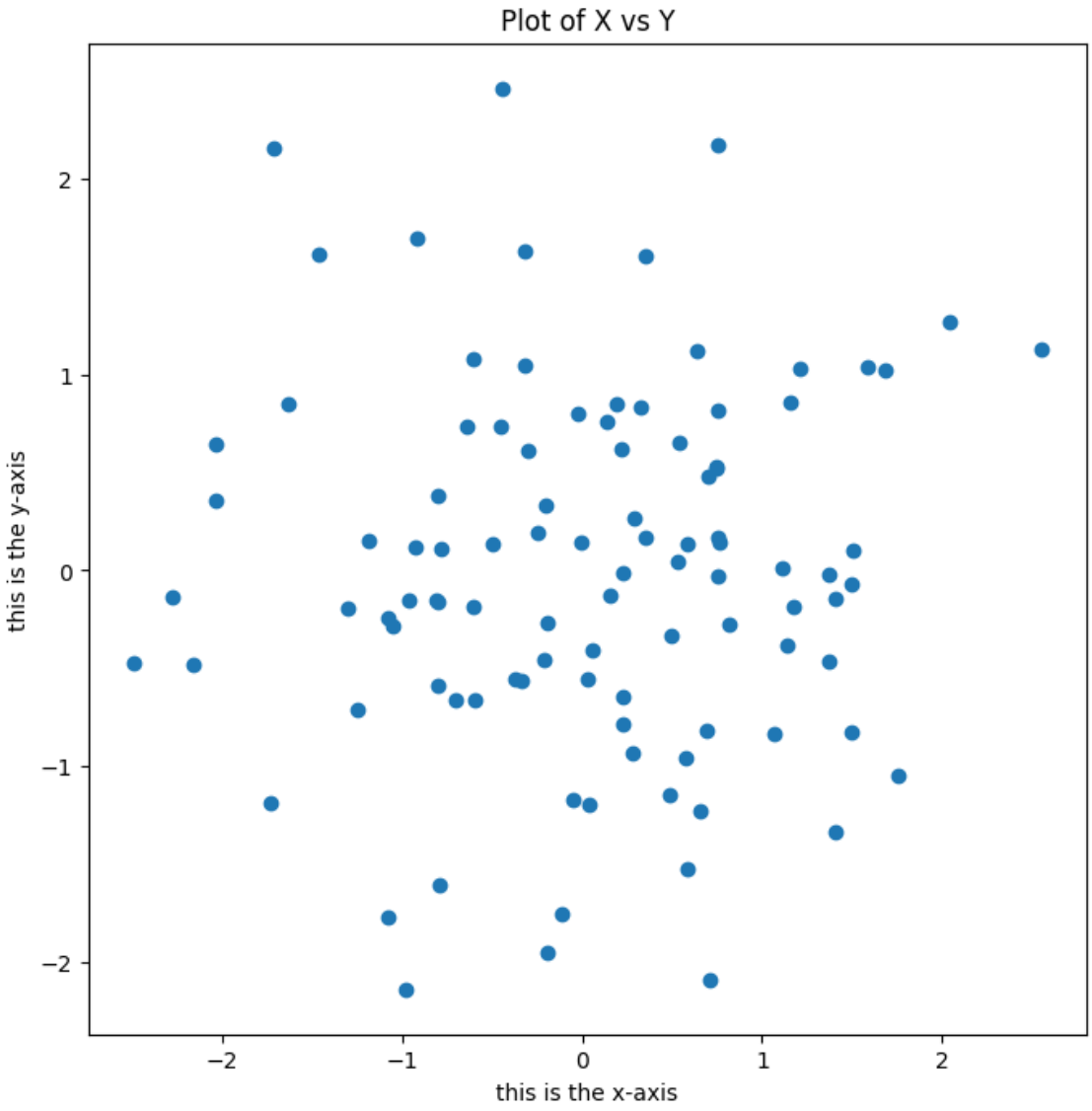


[ ]:
```
fig, ax = subplots(figsize=(8, 8))
ax.scatter(x, y, marker='o')
```

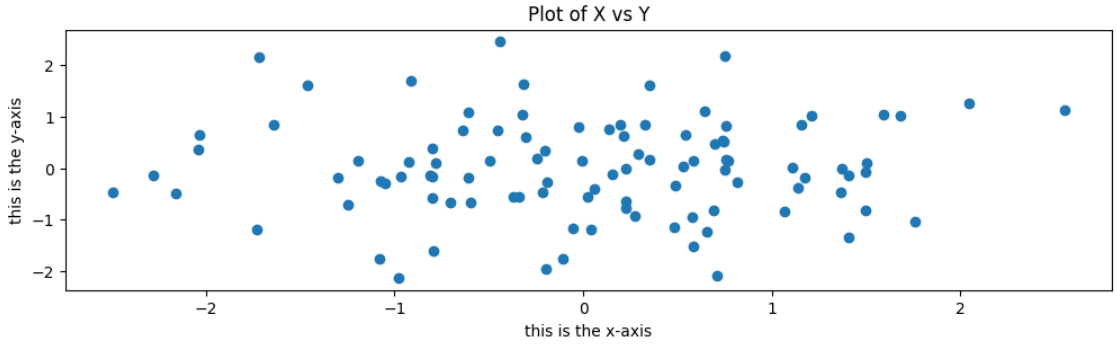
[ ]: <matplotlib.collections.PathCollection at 0x122230260>



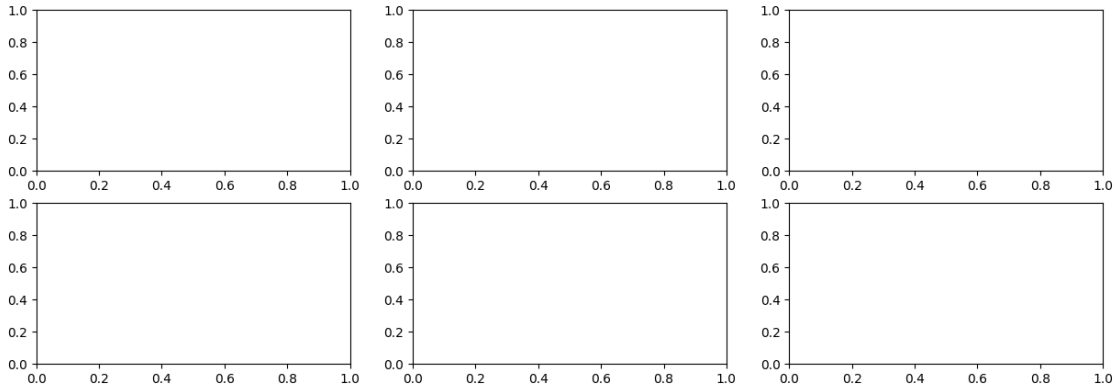
[ ]:
```
fig, ax = subplots(figsize=(8, 8))
ax.scatter(x, y, marker='o')
ax.set_xlabel("this is the x-axis")
ax.set_ylabel("this is the y-axis")
ax.set_title("Plot of X vs Y");
```



[ ]:
```
fig.set_size_inches(12,3)
fig
```

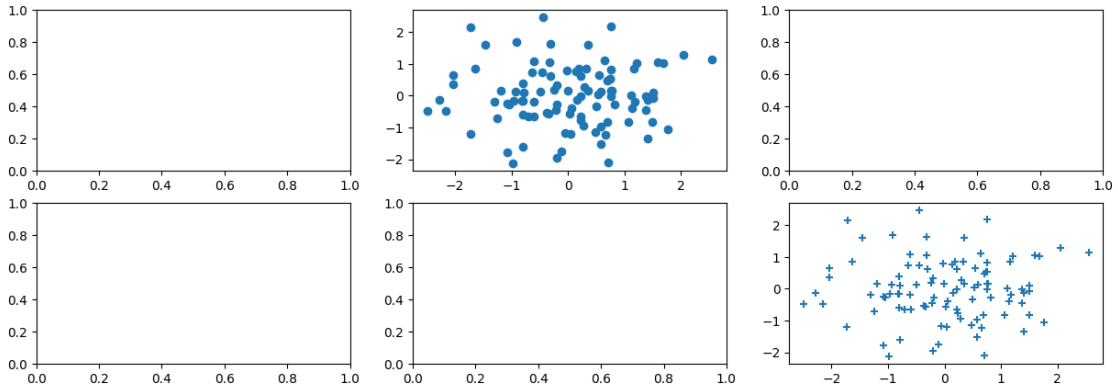
[ ]:


[ ]:
```
fig, axes = subplots(nrows=2,
                     ncols=3,
                     figsize=(15, 5))
```



[ ]:
```
axes[0,1].plot(x, y, 'o')
axes[1,2].scatter(x, y, marker='+')
fig
```

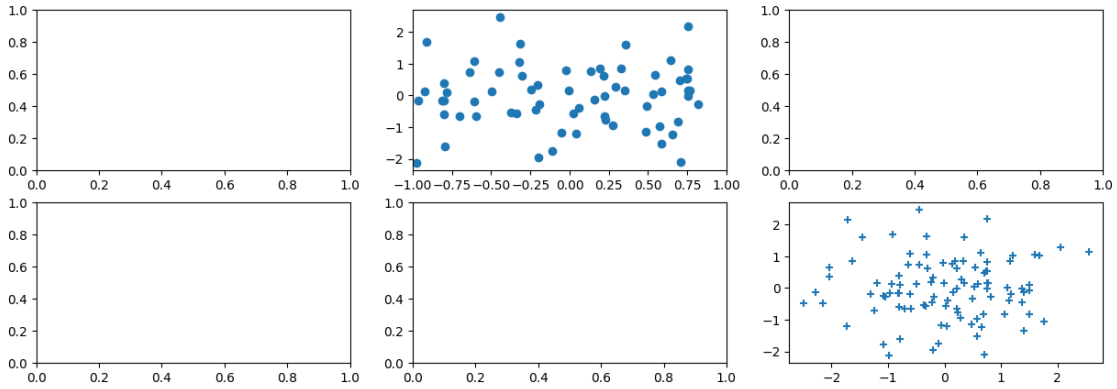
[ ]:



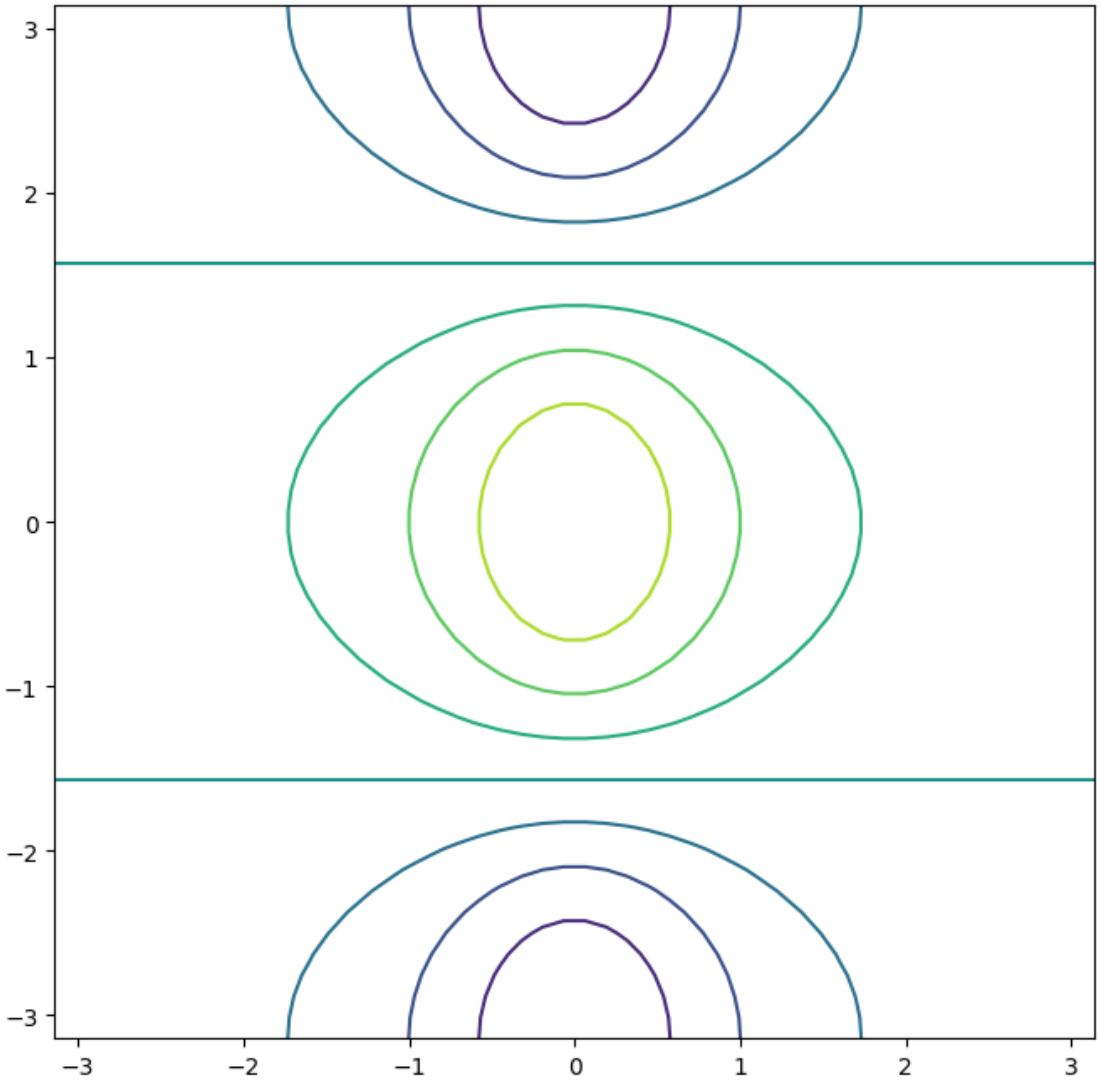
[ ]:
```
fig.savefig("Figure.png", dpi=400)
fig.savefig("Figure.pdf", dpi=200);
```

[ ]:
```
axes[0,1].set_xlim([-1,1])
fig.savefig("Figure_updated.jpg")
fig
```

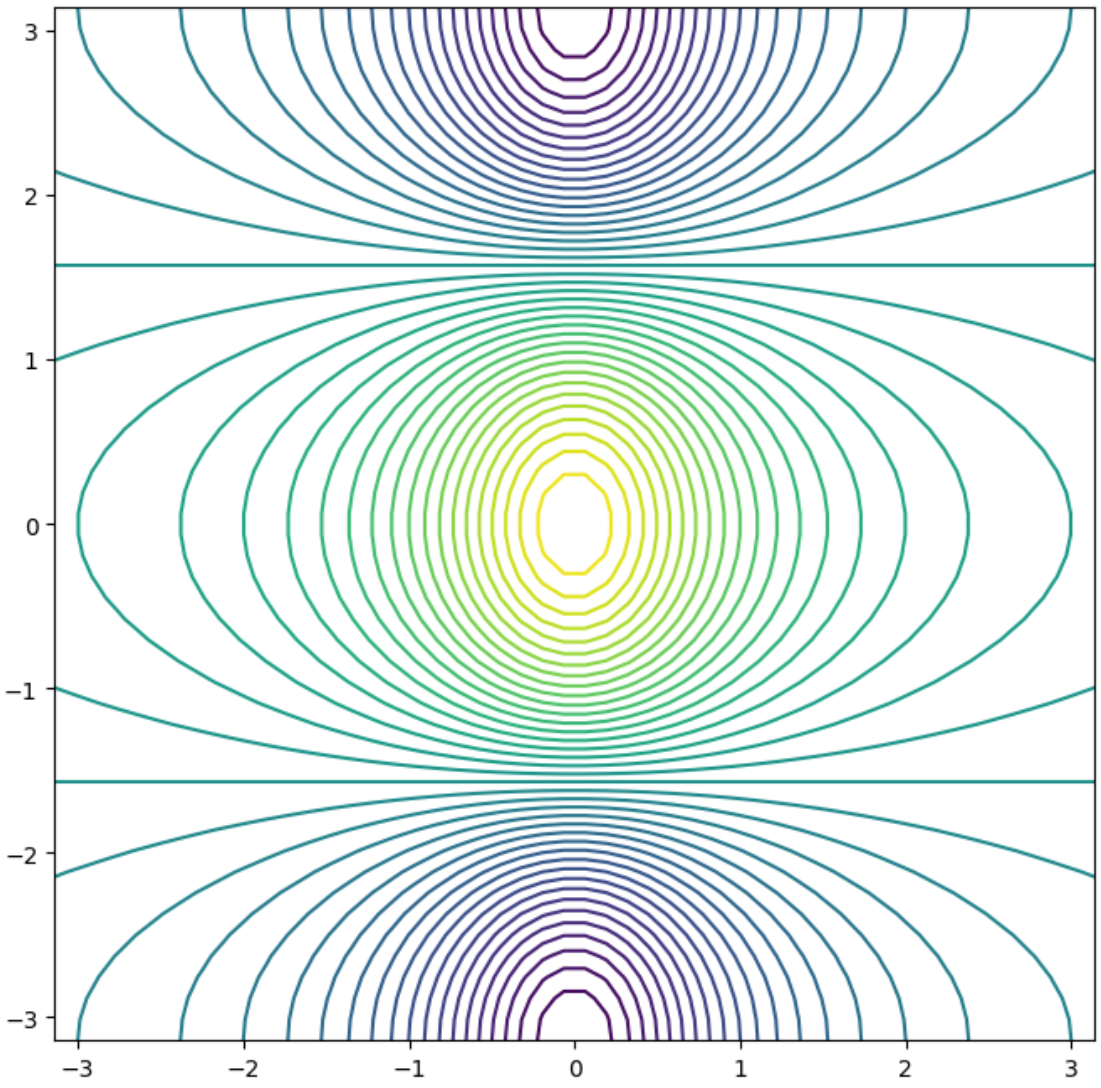
[ ]:



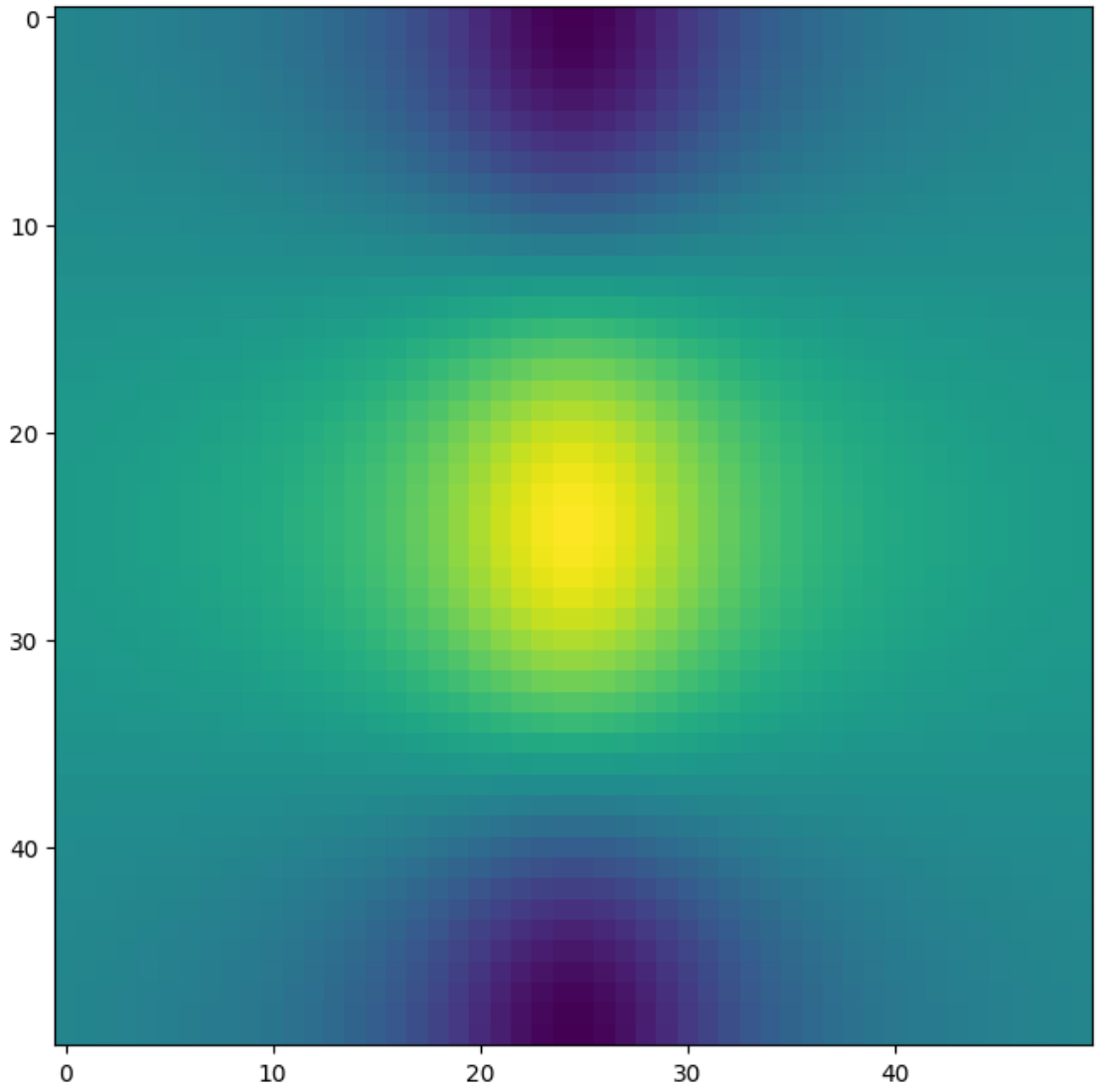
[ ]:
```
fig, ax = subplots(figsize=(8, 8))
x = np.linspace(-np.pi, np.pi, 50)
y = x
f = np.multiply.outer(np.cos(y), 1 / (1 + x**2))
ax.contour(x, y, f);
```



[ ]:
```
fig, ax = subplots(figsize=(8, 8))
ax.contour(x, y, f, levels=45);
```



[ ]:
```
fig, ax = subplots(figsize=(8, 8))
ax.imshow(f);
```



[ ]:
```
seq1 = np.linspace(0, 10, 11)
seq1
```

[ ]:
```
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

[ ]:
```
seq2 = np.arange(0, 10)
seq2
```

[ ]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[ ]:
```
"hello world"[3:6]
```

[ ]:
```
'lo '
```

[ ]:
```
"hello world"[slice(3,6)]
```

[ ]:
```
'lo '
```

[ ]:
```
A = np.array(np.arange(16)).reshape((4, 4))
A
```

[ ]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

[ ]:
```
A[1,2]
```

[ ]:
```
6
```

[ ]:
```
A[[1,3]]
```

[ ]:
```
array([[ 4,  5,  6,  7],
       [12, 13, 14, 15]])
```

[ ]:
```
A[:,[0,2]]
```

[ ]:
```
array([[ 0,  2],
       [ 4,  6],
       [ 8, 10],
       [12, 14]])
```

[ ]:
```
A[[1,3],[0,2]]
```

[ ]:
```
array([ 4, 14])
```

[ ]:
```
np.array([A[1,0],A[3,2]])
```

[ ]:
```
array([ 4, 14])
```

[ ]:
```
A[[1,3],[0,2,3]]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[63], line 1
----> 1 A[[1,3],[0,2,3]]

IndexError: shape mismatch: indexing arrays could not be broadcast together wit
 ↪shapes (2,) (3,)
```

[ ]:
```
A[[1,3]][:,[0,2]]
```

[ ]:
```
array([[ 4,  6],
       [12, 14]])
```

[ ]:
```
idx = np.ix_([1,3],[0,2,3])
A[idx]
```

[ ]:
```
array([[ 4,  6,  7],
       [12, 14, 15]])
```

[ ]:
```
A[1:4:2,0:3:2]
```

[ ]:
```
array([[ 4,  6],
       [12, 14]])
```

[ ]:
```
keep_rows = np.zeros(A.shape[0], bool)
keep_rows
```

[ ]:
```
array([False, False, False, False])
```

[ ]:
```
keep_rows[[1,3]] = True
keep_rows
```

[ ]:
```
array([False,  True, False,  True])
```

[ ]:
```
np.all(keep_rows == np.array([0,1,0,1]))
```

[ ]:
```
True
```

[ ]:
```
A[np.array([0,1,0,1])]
```

[ ]:
```
array([[0, 1, 2, 3],
       [4, 5, 6, 7],
       [0, 1, 2, 3],
       [4, 5, 6, 7]])
```

[ ]:
```
A[keep_rows]
```

[ ]:
```
array([[ 4,  5,  6,  7],
       [12, 13, 14, 15]])
```

[ ]:
```
keep_cols = np.zeros(A.shape[1], bool)
keep_cols[[0, 2, 3]] = True
idx_bool = np.ix_(keep_rows, keep_cols)
A[idx_bool]
```

[ ]:
```
array([[ 4,  6,  7],
       [12, 14, 15]])
```

```
[ ]: idx_mixed = np.ix_([1,3], keep_cols)
     A[idx_mixed]
```

```
[ ]: array([[ 4,  6,  7],
            [12, 14, 15]])
```

```
[ ]: import pandas as pd
     Auto = pd.read_csv('Auto.csv')
     Auto
```

```
[ ]:       mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
     0    18.0          8         307.0         130    3504          12.0    70
     1    15.0          8         350.0         165    3693          11.5    70
     2    18.0          8         318.0         150    3436          11.0    70
     3    16.0          8         304.0         150    3433          12.0    70
     4    17.0          8         302.0         140    3449          10.5    70
     ..    …          …             …           …      …             …     …
     387  27.0          4         140.0          86    2790          15.6    82
     388  44.0          4          97.0          52    2130          24.6    82
     389  32.0          4         135.0          84    2295          11.6    82
     390  28.0          4         120.0          79    2625          18.6    82
     391  31.0          4         119.0          82    2720          19.4    82

          origin                       name
     0         1  chevrolet chevelle malibu
     1         1          buick skylark 320
     2         1         plymouth satellite
     3         1              amc rebel sst
     4         1                ford torino
     ..      …                          …
     387       1            ford mustang gl
     388       2                  vw pickup
     389       1              dodge rampage
     390       1                ford ranger
     391       1                 chevy s-10

     [392 rows x 9 columns]
```

```
[ ]: Auto = pd.read_csv('Auto.data', delim_whitespace=True)
```

```
[ ]: Auto['horsepower']
```

```
[ ]: 0      130.0
     1      165.0
     2      150.0
     3      150.0
     4      140.0
```

```
       …
392    86.00
393    52.00
394    84.00
395    79.00
396    82.00
Name: horsepower, Length: 397, dtype: object
```

[ ]:
```
np.unique(Auto['horsepower'])
```

[ ]:
```
array(['100.0', '102.0', '103.0', '105.0', '107.0', '108.0', '110.0',
       '112.0', '113.0', '115.0', '116.0', '120.0', '122.0', '125.0',
       '129.0', '130.0', '132.0', '133.0', '135.0', '137.0', '138.0',
       '139.0', '140.0', '142.0', '145.0', '148.0', '149.0', '150.0',
       '152.0', '153.0', '155.0', '158.0', '160.0', '165.0', '167.0',
       '170.0', '175.0', '180.0', '190.0', '193.0', '198.0', '200.0',
       '208.0', '210.0', '215.0', '220.0', '225.0', '230.0', '46.00',
       '48.00', '49.00', '52.00', '53.00', '54.00', '58.00', '60.00',
       '61.00', '62.00', '63.00', '64.00', '65.00', '66.00', '67.00',
       '68.00', '69.00', '70.00', '71.00', '72.00', '74.00', '75.00',
       '76.00', '77.00', '78.00', '79.00', '80.00', '81.00', '82.00',
       '83.00', '84.00', '85.00', '86.00', '87.00', '88.00', '89.00',
       '90.00', '91.00', '92.00', '93.00', '94.00', '95.00', '96.00',
       '97.00', '98.00', '?'], dtype=object)
```

[ ]:
```
Auto = pd.read_csv('Auto.data',
                   na_values=['?'],
                   delim_whitespace=True)
Auto['horsepower'].sum()
```

[ ]:
```
40952.0
```

[ ]:
```
Auto.shape
```

[ ]:
```
(397, 9)
```

[ ]:
```
Auto_new = Auto.dropna()
Auto_new.shape
```

[ ]:
```
(392, 9)
```

[ ]:
```
Auto = Auto_new # overwrite the previous value
Auto.columns
```

[ ]:
```
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'year', 'origin', 'name'],
      dtype='object')
```

[ ]:
```
Auto[:3]
```

[ ]:
```
    mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0  18.0          8         307.0       130.0  3504.0          12.0    70
1  15.0          8         350.0       165.0  3693.0          11.5    70
2  18.0          8         318.0       150.0  3436.0          11.0    70

   origin                       name
0       1  chevrolet chevelle malibu
1       1          buick skylark 320
2       1         plymouth satellite
```

[ ]:
```
idx_80 = Auto['year'] > 80
Auto[idx_80]
```

[ ]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
338  27.2          4         135.0        84.0  2490.0          15.7    81
339  26.6          4         151.0        84.0  2635.0          16.4    81
340  25.8          4         156.0        92.0  2620.0          14.4    81
341  23.5          6         173.0       110.0  2725.0          12.6    81
342  30.0          4         135.0        84.0  2385.0          12.9    81
343  39.1          4          79.0        58.0  1755.0          16.9    81
344  39.0          4          86.0        64.0  1875.0          16.4    81
345  35.1          4          81.0        60.0  1760.0          16.1    81
346  32.3          4          97.0        67.0  2065.0          17.8    81
347  37.0          4          85.0        65.0  1975.0          19.4    81
348  37.7          4          89.0        62.0  2050.0          17.3    81
349  34.1          4          91.0        68.0  1985.0          16.0    81
350  34.7          4         105.0        63.0  2215.0          14.9    81
351  34.4          4          98.0        65.0  2045.0          16.2    81
352  29.9          4          98.0        65.0  2380.0          20.7    81
353  33.0          4         105.0        74.0  2190.0          14.2    81
355  33.7          4         107.0        75.0  2210.0          14.4    81
356  32.4          4         108.0        75.0  2350.0          16.8    81
357  32.9          4         119.0       100.0  2615.0          14.8    81
358  31.6          4         120.0        74.0  2635.0          18.3    81
359  28.1          4         141.0        80.0  3230.0          20.4    81
360  30.7          6         145.0        76.0  3160.0          19.6    81
361  25.4          6         168.0       116.0  2900.0          12.6    81
362  24.2          6         146.0       120.0  2930.0          13.8    81
363  22.4          6         231.0       110.0  3415.0          15.8    81
364  26.6          8         350.0       105.0  3725.0          19.0    81
365  20.2          6         200.0        88.0  3060.0          17.1    81
366  17.6          6         225.0        85.0  3465.0          16.6    81
367  28.0          4         112.0        88.0  2605.0          19.6    82
368  27.0          4         112.0        88.0  2640.0          18.6    82
369  34.0          4         112.0        88.0  2395.0          18.0    82
```

370	31.0	4	112.0	85.0	2575.0	16.2	82
371	29.0	4	135.0	84.0	2525.0	16.0	82
372	27.0	4	151.0	90.0	2735.0	18.0	82
373	24.0	4	140.0	92.0	2865.0	16.4	82
374	36.0	4	105.0	74.0	1980.0	15.3	82
375	37.0	4	91.0	68.0	2025.0	18.2	82
376	31.0	4	91.0	68.0	1970.0	17.6	82
377	38.0	4	105.0	63.0	2125.0	14.7	82
378	36.0	4	98.0	70.0	2125.0	17.3	82
379	36.0	4	120.0	88.0	2160.0	14.5	82
380	36.0	4	107.0	75.0	2205.0	14.5	82
381	34.0	4	108.0	70.0	2245.0	16.9	82
382	38.0	4	91.0	67.0	1965.0	15.0	82
383	32.0	4	91.0	67.0	1965.0	15.7	82
384	38.0	4	91.0	67.0	1995.0	16.2	82
385	25.0	6	181.0	110.0	2945.0	16.4	82
386	38.0	6	262.0	85.0	3015.0	17.0	82
387	26.0	4	156.0	92.0	2585.0	14.5	82
388	22.0	6	232.0	112.0	2835.0	14.7	82
389	32.0	4	144.0	96.0	2665.0	13.9	82
390	36.0	4	135.0	84.0	2370.0	13.0	82
391	27.0	4	151.0	90.0	2950.0	17.3	82
392	27.0	4	140.0	86.0	2790.0	15.6	82
393	44.0	4	97.0	52.0	2130.0	24.6	82
394	32.0	4	135.0	84.0	2295.0	11.6	82
395	28.0	4	120.0	79.0	2625.0	18.6	82
396	31.0	4	119.0	82.0	2720.0	19.4	82

	origin	name
338	1	plymouth reliant
339	1	buick skylark
340	1	dodge aries wagon (sw)
341	1	chevrolet citation
342	1	plymouth reliant
343	3	toyota starlet
344	1	plymouth champ
345	3	honda civic 1300
346	3	subaru
347	3	datsun 210 mpg
348	3	toyota tercel
349	3	mazda glc 4
350	1	plymouth horizon 4
351	1	ford escort 4w
352	1	ford escort 2h
353	2	volkswagen jetta
355	3	honda prelude
356	3	toyota corolla

```
357       3                      datsun 200sx
358       3                         mazda 626
359       2           peugeot 505s turbo diesel
360       2                      volvo diesel
361       3                    toyota cressida
362       3                  datsun 810 maxima
363       1                      buick century
364       1              oldsmobile cutlass ls
365       1                    ford granada gl
366       1             chrysler lebaron salon
367       1                 chevrolet cavalier
368       1           chevrolet cavalier wagon
369       1           chevrolet cavalier 2-door
370       1          pontiac j2000 se hatchback
371       1                     dodge aries se
372       1                    pontiac phoenix
373       1                ford fairmont futura
374       2                volkswagen rabbit l
375       3                mazda glc custom l
376       3                  mazda glc custom
377       1             plymouth horizon miser
378       1                     mercury lynx l
379       3                  nissan stanza xe
380       3                      honda accord
381       3                    toyota corolla
382       3                       honda civic
383       3                 honda civic (auto)
384       3                      datsun 310 gx
385       1               buick century limited
386       1  oldsmobile cutlass ciera (diesel)
387       1          chrysler lebaron medallion
388       1                     ford granada l
389       3                   toyota celica gt
390       1                  dodge charger 2.2
391       1                   chevrolet camaro
392       1                   ford mustang gl
393       2                         vw pickup
394       1                    dodge rampage
395       1                      ford ranger
396       1                       chevy s-10
```

[ ]:
```
Auto[['mpg', 'horsepower']]
```

[ ]:
```
      mpg  horsepower
0    18.0       130.0
1    15.0       165.0
2    18.0       150.0
```

```
3    16.0       150.0
4    17.0       140.0
..    …          …
392  27.0        86.0
393  44.0        52.0
394  32.0        84.0
395  28.0        79.0
396  31.0        82.0

[392 rows x 2 columns]
```

[ ]:
```
Auto.index
```

[ ]:
```
Index([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
       …
       387, 388, 389, 390, 391, 392, 393, 394, 395, 396],
      dtype='int64', length=392)
```

[ ]:
```
Auto_re = Auto.set_index('name')
Auto_re
```

[ ]:
```
                           mpg  cylinders  displacement  horsepower  weight  \
name
chevrolet chevelle malibu  18.0          8         307.0       130.0  3504.0
buick skylark 320          15.0          8         350.0       165.0  3693.0
plymouth satellite         18.0          8         318.0       150.0  3436.0
amc rebel sst              16.0          8         304.0       150.0  3433.0
ford torino                17.0          8         302.0       140.0  3449.0
…                           …          …           …          …      …
ford mustang gl            27.0          4         140.0        86.0  2790.0
vw pickup                  44.0          4          97.0        52.0  2130.0
dodge rampage              32.0          4         135.0        84.0  2295.0
ford ranger                28.0          4         120.0        79.0  2625.0
chevy s-10                 31.0          4         119.0        82.0  2720.0

                           acceleration  year  origin
name
chevrolet chevelle malibu          12.0    70       1
buick skylark 320                  11.5    70       1
plymouth satellite                 11.0    70       1
amc rebel sst                      12.0    70       1
ford torino                        10.5    70       1
…                                   …      …       …
ford mustang gl                    15.6    82       1
vw pickup                          24.6    82       2
dodge rampage                      11.6    82       1
ford ranger                        18.6    82       1
```

```
chevy s-10                        19.4    82       1

[392 rows x 8 columns]
```

[ ]:
```
Auto_re.columns
```

[ ]:
```
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'year', 'origin'],
      dtype='object')
```

[ ]:
```
rows = ['amc rebel sst', 'ford torino']
Auto_re.loc[rows]
```

[ ]:
```
                mpg  cylinders  displacement  horsepower  weight  \
name
amc rebel sst  16.0          8         304.0       150.0  3433.0
ford torino    17.0          8         302.0       140.0  3449.0

               acceleration  year  origin
name
amc rebel sst          12.0    70       1
ford torino            10.5    70       1
```

[ ]:
```
Auto_re.iloc[[3,4]]
```

[ ]:
```
                mpg  cylinders  displacement  horsepower  weight  \
name
amc rebel sst  16.0          8         304.0       150.0  3433.0
ford torino    17.0          8         302.0       140.0  3449.0

               acceleration  year  origin
name
amc rebel sst          12.0    70       1
ford torino            10.5    70       1
```

[ ]:
```
Auto_re.iloc[:,[0,2,3]]
```

[ ]:
```
                            mpg  displacement  horsepower
name
chevrolet chevelle malibu  18.0         307.0       130.0
buick skylark 320          15.0         350.0       165.0
plymouth satellite         18.0         318.0       150.0
amc rebel sst              16.0         304.0       150.0
ford torino                17.0         302.0       140.0
…                           …             …          …
ford mustang gl            27.0         140.0        86.0
vw pickup                  44.0          97.0        52.0
dodge rampage              32.0         135.0        84.0
```

```
ford ranger                28.0         120.0        79.0
chevy s-10                 31.0         119.0        82.0

[392 rows x 3 columns]
```

```
[ ]: Auto_re.iloc[[3,4],[0,2,3]]
```

```
[ ]:                mpg  displacement  horsepower
     name
     amc rebel sst  16.0         304.0       150.0
     ford torino    17.0         302.0       140.0
```

```
[ ]: Auto_re.loc['ford galaxie 500', ['mpg', 'origin']]
```

```
[ ]:                    mpg  origin
     name
     ford galaxie 500  15.0       1
     ford galaxie 500  14.0       1
     ford galaxie 500  14.0       1
```

```
[ ]: idx_80 = Auto_re['year'] > 80
     Auto_re.loc[idx_80, ['weight', 'origin']]
```

```
[ ]:                                  weight  origin
     name
     plymouth reliant                 2490.0       1
     buick skylark                    2635.0       1
     dodge aries wagon (sw)           2620.0       1
     chevrolet citation               2725.0       1
     plymouth reliant                 2385.0       1
     toyota starlet                   1755.0       3
     plymouth champ                   1875.0       1
     honda civic 1300                 1760.0       3
     subaru                           2065.0       3
     datsun 210 mpg                   1975.0       3
     toyota tercel                    2050.0       3
     mazda glc 4                      1985.0       3
     plymouth horizon 4               2215.0       1
     ford escort 4w                   2045.0       1
     ford escort 2h                   2380.0       1
     volkswagen jetta                 2190.0       2
     honda prelude                    2210.0       3
     toyota corolla                   2350.0       3
     datsun 200sx                     2615.0       3
     mazda 626                        2635.0       3
     peugeot 505s turbo diesel        3230.0       2
     volvo diesel                     3160.0       2
```

```
toyota cressida                    2900.0       3
datsun 810 maxima                  2930.0       3
buick century                      3415.0       1
oldsmobile cutlass ls              3725.0       1
ford granada gl                    3060.0       1
chrysler lebaron salon             3465.0       1
chevrolet cavalier                 2605.0       1
chevrolet cavalier wagon           2640.0       1
chevrolet cavalier 2-door          2395.0       1
pontiac j2000 se hatchback         2575.0       1
dodge aries se                     2525.0       1
pontiac phoenix                    2735.0       1
ford fairmont futura               2865.0       1
volkswagen rabbit l                1980.0       2
mazda glc custom l                 2025.0       3
mazda glc custom                   1970.0       3
plymouth horizon miser             2125.0       1
mercury lynx l                     2125.0       1
nissan stanza xe                   2160.0       3
honda accord                       2205.0       3
toyota corolla                     2245.0       3
honda civic                        1965.0       3
honda civic (auto)                 1965.0       3
datsun 310 gx                      1995.0       3
buick century limited              2945.0       1
oldsmobile cutlass ciera (diesel)  3015.0       1
chrysler lebaron medallion         2585.0       1
ford granada l                     2835.0       1
toyota celica gt                   2665.0       3
dodge charger 2.2                  2370.0       1
chevrolet camaro                   2950.0       1
ford mustang gl                    2790.0       1
vw pickup                          2130.0       2
dodge rampage                      2295.0       1
ford ranger                        2625.0       1
chevy s-10                         2720.0       1
```

[ ]:
```
Auto_re.loc[lambda df: df['year'] > 80, ['weight', 'origin']]
```

[ ]:
```
                                   weight  origin
name
plymouth reliant                   2490.0       1
buick skylark                      2635.0       1
dodge aries wagon (sw)             2620.0       1
chevrolet citation                 2725.0       1
plymouth reliant                   2385.0       1
toyota starlet                     1755.0       3
```

plymouth champ	1875.0	1
honda civic 1300	1760.0	3
subaru	2065.0	3
datsun 210 mpg	1975.0	3
toyota tercel	2050.0	3
mazda glc 4	1985.0	3
plymouth horizon 4	2215.0	1
ford escort 4w	2045.0	1
ford escort 2h	2380.0	1
volkswagen jetta	2190.0	2
honda prelude	2210.0	3
toyota corolla	2350.0	3
datsun 200sx	2615.0	3
mazda 626	2635.0	3
peugeot 505s turbo diesel	3230.0	2
volvo diesel	3160.0	2
toyota cressida	2900.0	3
datsun 810 maxima	2930.0	3
buick century	3415.0	1
oldsmobile cutlass ls	3725.0	1
ford granada gl	3060.0	1
chrysler lebaron salon	3465.0	1
chevrolet cavalier	2605.0	1
chevrolet cavalier wagon	2640.0	1
chevrolet cavalier 2-door	2395.0	1
pontiac j2000 se hatchback	2575.0	1
dodge aries se	2525.0	1
pontiac phoenix	2735.0	1
ford fairmont futura	2865.0	1
volkswagen rabbit l	1980.0	2
mazda glc custom l	2025.0	3
mazda glc custom	1970.0	3
plymouth horizon miser	2125.0	1
mercury lynx l	2125.0	1
nissan stanza xe	2160.0	3
honda accord	2205.0	3
toyota corolla	2245.0	3
honda civic	1965.0	3
honda civic (auto)	1965.0	3
datsun 310 gx	1995.0	3
buick century limited	2945.0	1
oldsmobile cutlass ciera (diesel)	3015.0	1
chrysler lebaron medallion	2585.0	1
ford granada l	2835.0	1
toyota celica gt	2665.0	3
dodge charger 2.2	2370.0	1
chevrolet camaro	2950.0	1

```
ford mustang gl                    2790.0       1
vw pickup                          2130.0       2
dodge rampage                      2295.0       1
ford ranger                        2625.0       1
chevy s-10                         2720.0       1
```

[ ]:
```
Auto_re.loc[lambda df: (df['year'] > 80) & (df['mpg'] > 30),
            ['weight', 'origin']
           ]
```

[ ]:
```
                                   weight  origin
name
toyota starlet                     1755.0       3
plymouth champ                     1875.0       1
honda civic 1300                   1760.0       3
subaru                             2065.0       3
datsun 210 mpg                     1975.0       3
toyota tercel                      2050.0       3
mazda glc 4                        1985.0       3
plymouth horizon 4                 2215.0       1
ford escort 4w                     2045.0       1
volkswagen jetta                   2190.0       2
honda prelude                      2210.0       3
toyota corolla                     2350.0       3
datsun 200sx                       2615.0       3
mazda 626                          2635.0       3
volvo diesel                       3160.0       2
chevrolet cavalier 2-door          2395.0       1
pontiac j2000 se hatchback         2575.0       1
volkswagen rabbit l                1980.0       2
mazda glc custom l                 2025.0       3
mazda glc custom                   1970.0       3
plymouth horizon miser             2125.0       1
mercury lynx l                     2125.0       1
nissan stanza xe                   2160.0       3
honda accord                       2205.0       3
toyota corolla                     2245.0       3
honda civic                        1965.0       3
honda civic (auto)                 1965.0       3
datsun 310 gx                      1995.0       3
oldsmobile cutlass ciera (diesel)  3015.0       1
toyota celica gt                   2665.0       3
dodge charger 2.2                  2370.0       1
vw pickup                          2130.0       2
dodge rampage                      2295.0       1
chevy s-10                         2720.0       1
```

[ ]:
```
Auto_re.loc[lambda df: (df['displacement'] < 300)
                       & (df.index.str.contains('ford')
                       | df.index.str.contains('datsun')),
            ['weight', 'origin']
           ]
```

[ ]:

name	weight	origin
ford maverick	2587.0	1
datsun pl510	2130.0	3
datsun pl510	2130.0	3
ford torino 500	3302.0	1
ford mustang	3139.0	1
datsun 1200	1613.0	3
ford pinto runabout	2226.0	1
ford pinto (sw)	2395.0	1
datsun 510 (sw)	2288.0	3
ford maverick	3021.0	1
datsun 610	2379.0	3
ford pinto	2310.0	1
datsun b210	1950.0	3
ford pinto	2451.0	1
datsun 710	2003.0	3
ford maverick	3158.0	1
ford pinto	2639.0	1
datsun 710	2545.0	3
ford pinto	2984.0	1
ford maverick	3012.0	1
ford granada ghia	3574.0	1
datsun b-210	1990.0	3
ford pinto	2565.0	1
datsun f-10 hatchback	1945.0	3
ford granada	3525.0	1
ford mustang ii 2+2	2755.0	1
datsun 810	2815.0	3
ford fiesta	1800.0	1
datsun b210 gx	2070.0	3
ford fairmont (auto)	2965.0	1
ford fairmont (man)	2720.0	1
datsun 510	2300.0	3
datsun 200-sx	2405.0	3
ford fairmont 4	2890.0	1
datsun 210	2020.0	3
datsun 310	2019.0	3
ford fairmont	2870.0	1
datsun 510 hatchback	2434.0	3
datsun 210	2110.0	3

```
datsun 280-zx          2910.0       3
datsun 210 mpg         1975.0       3
ford escort 4w         2045.0       1
ford escort 2h         2380.0       1
datsun 200sx           2615.0       3
datsun 810 maxima      2930.0       3
ford granada gl        3060.0       1
ford fairmont futura   2865.0       1
datsun 310 gx          1995.0       3
ford granada l         2835.0       1
ford mustang gl        2790.0       1
ford ranger            2625.0       1
```

[ ]:
```
total = 0
for value in [3,2,19]:
    total += value
print('Total is: {0}'.format(total))
```

```
Total is: 24
```

[ ]:
```
total = 0
for value in [2,3,19]:
    for weight in [3, 2, 1]:
        total += value * weight
print('Total is: {0}'.format(total))
```

```
Total is: 144
```

[ ]:
```
total = 0
for value, weight in zip([2,3,19],
                         [0.2,0.3,0.5]):
    total += weight * value
print('Weighted average is: {0}'.format(total))
```

```
Weighted average is: 10.8
```

[ ]:
```
rng = np.random.default_rng(1)
A = rng.standard_normal((127, 5))
M = rng.choice([0, np.nan], p=[0.8,0.2], size=A.shape)
A += M
D = pd.DataFrame(A, columns=['food',
                             'bar',
                             'pickle',
                             'snack',
                             'popcorn'])
D[:3]
```

[ ]:
```
       food       bar    pickle     snack   popcorn
0  0.345584  0.821618  0.330437 -1.303157       NaN
```

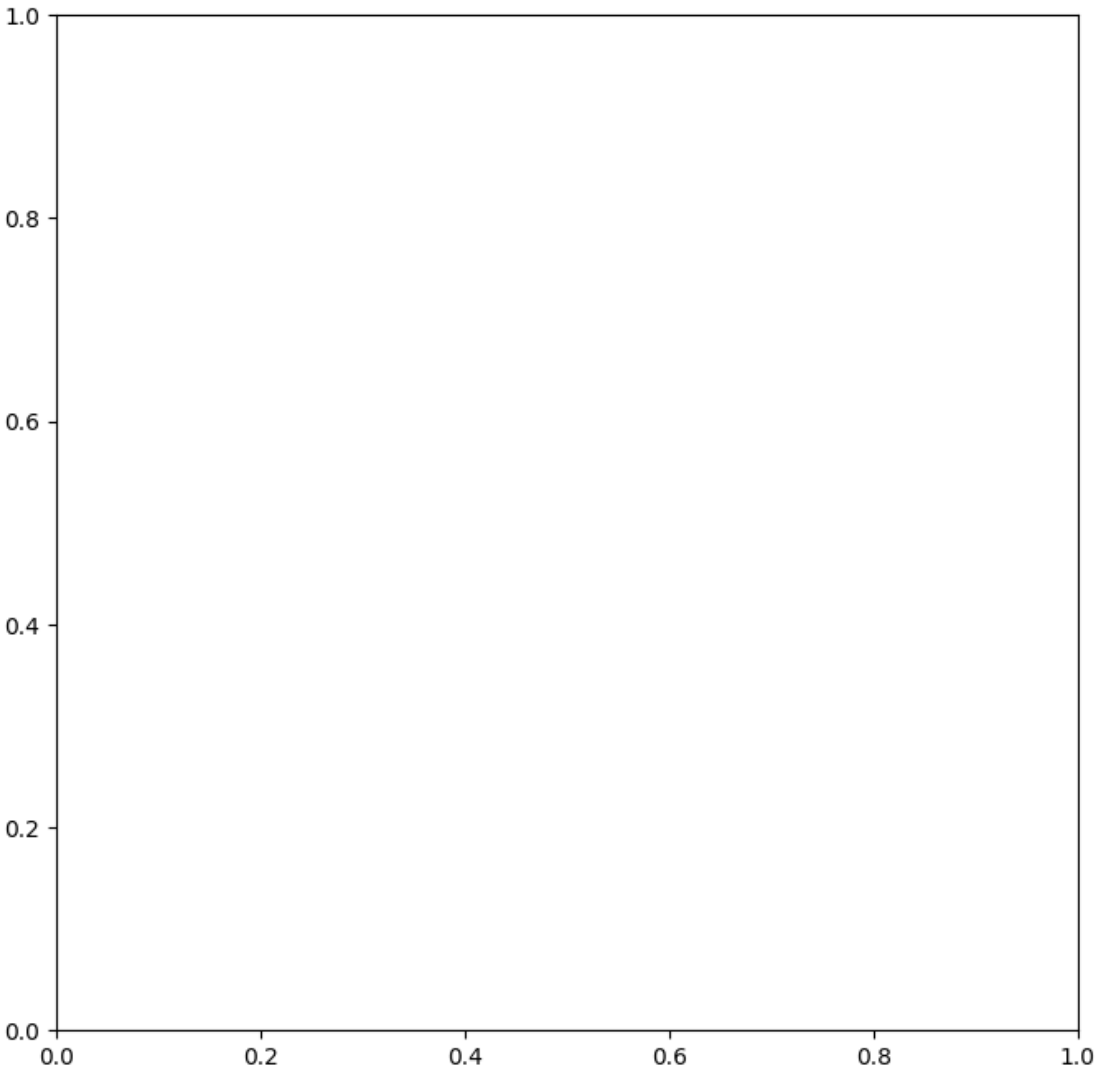
```
1       NaN -0.536953  0.581118  0.364572  0.294132
2       NaN  0.546713       NaN -0.162910 -0.482119
```

[ ]:
```
for col in D.columns:
    template = 'Column "{0}" has {1:.2%} missing values'
    print(template.format(col,
          np.isnan(D[col]).mean()))
```

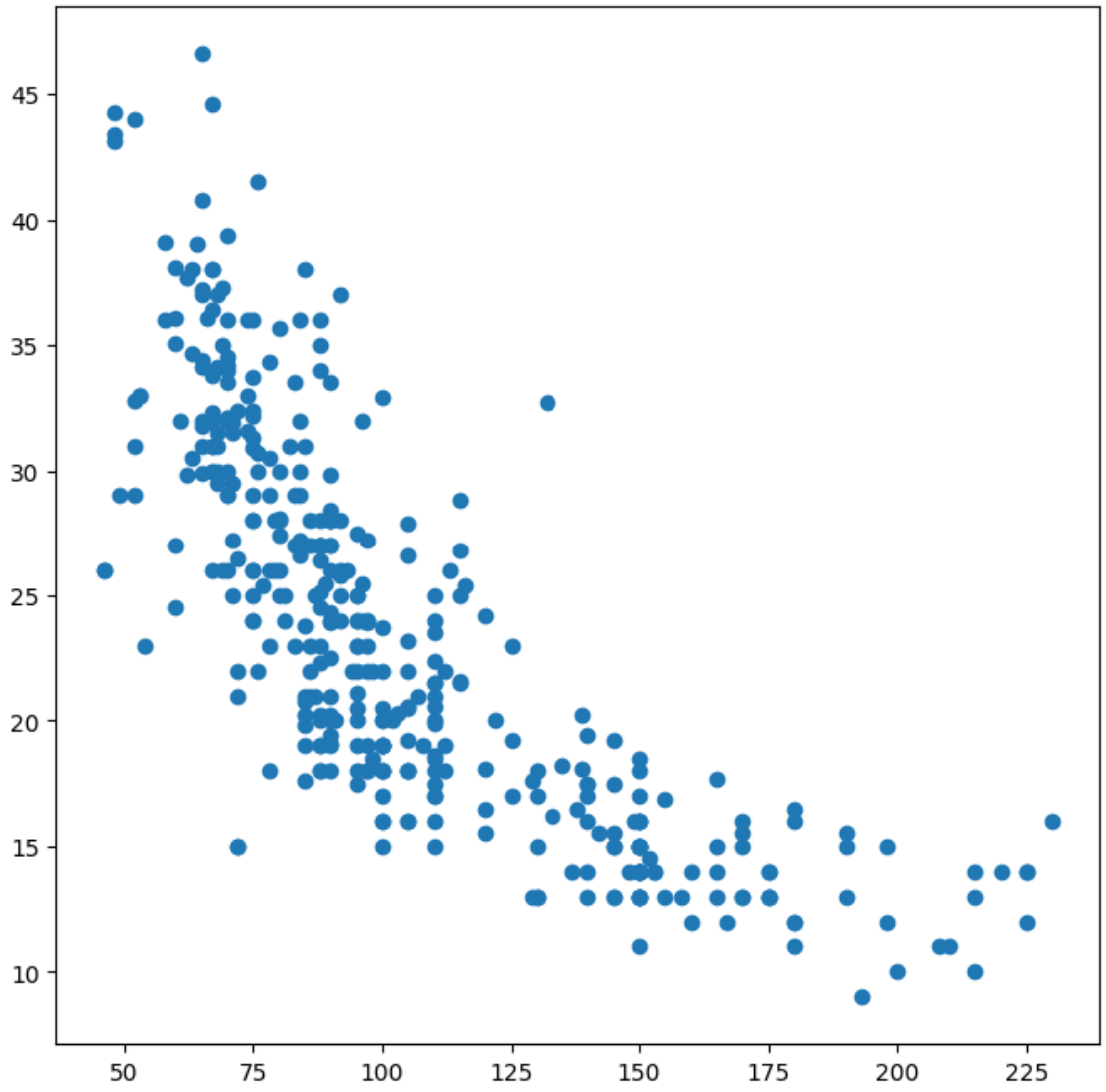
```
Column "food" has 16.54% missing values
Column "bar" has 25.98% missing values
Column "pickle" has 29.13% missing values
Column "snack" has 21.26% missing values
Column "popcorn" has 22.83% missing values
```

[ ]:
```
fig, ax = subplots(figsize=(8, 8))
ax.plot(horsepower, mpg, 'o');
```

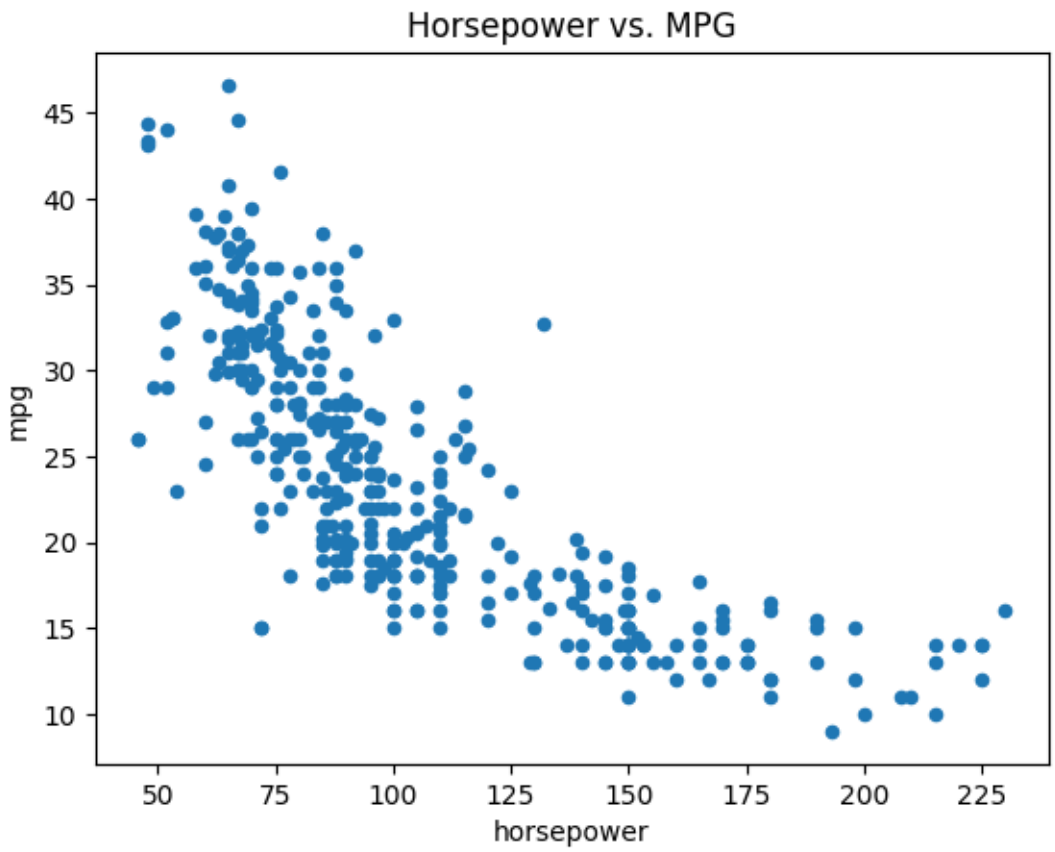
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[102], line 2
      1 fig, ax = subplots(figsize=(8, 8))
----> 2 ax.plot(horsepower, mpg, 'o');

NameError: name 'horsepower' is not defined
```



[ ]:
```
fig, ax = subplots(figsize=(8, 8))
ax.plot(Auto['horsepower'], Auto['mpg'], 'o');
```

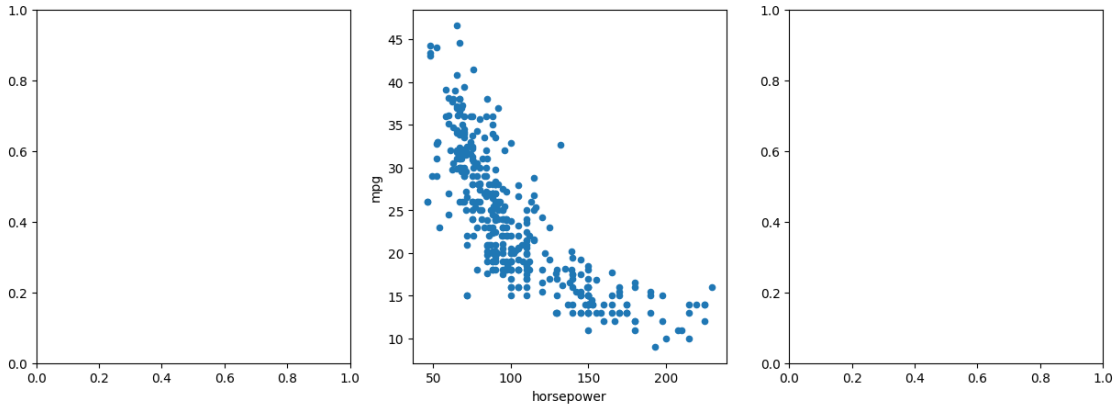



```
[ ]: ax = Auto.plot.scatter('horsepower', 'mpg')
     ax.set_title('Horsepower vs. MPG');
```



[ ]:
```
fig = ax.figure
fig.savefig('horsepower_mpg.png');
```

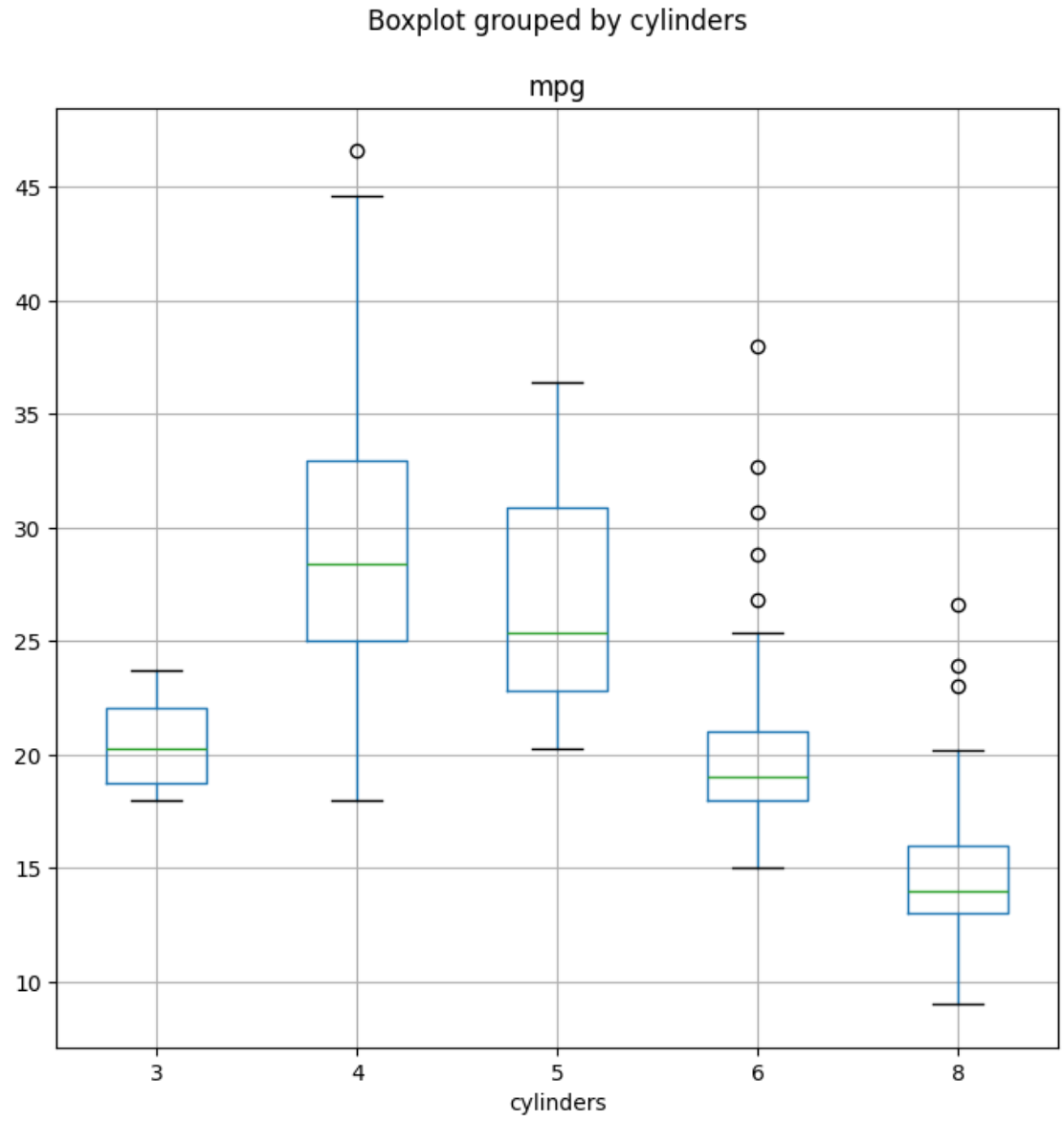
[ ]:
```
fig, axes = subplots(ncols=3, figsize=(15, 5))
Auto.plot.scatter('horsepower', 'mpg', ax=axes[1]);
```



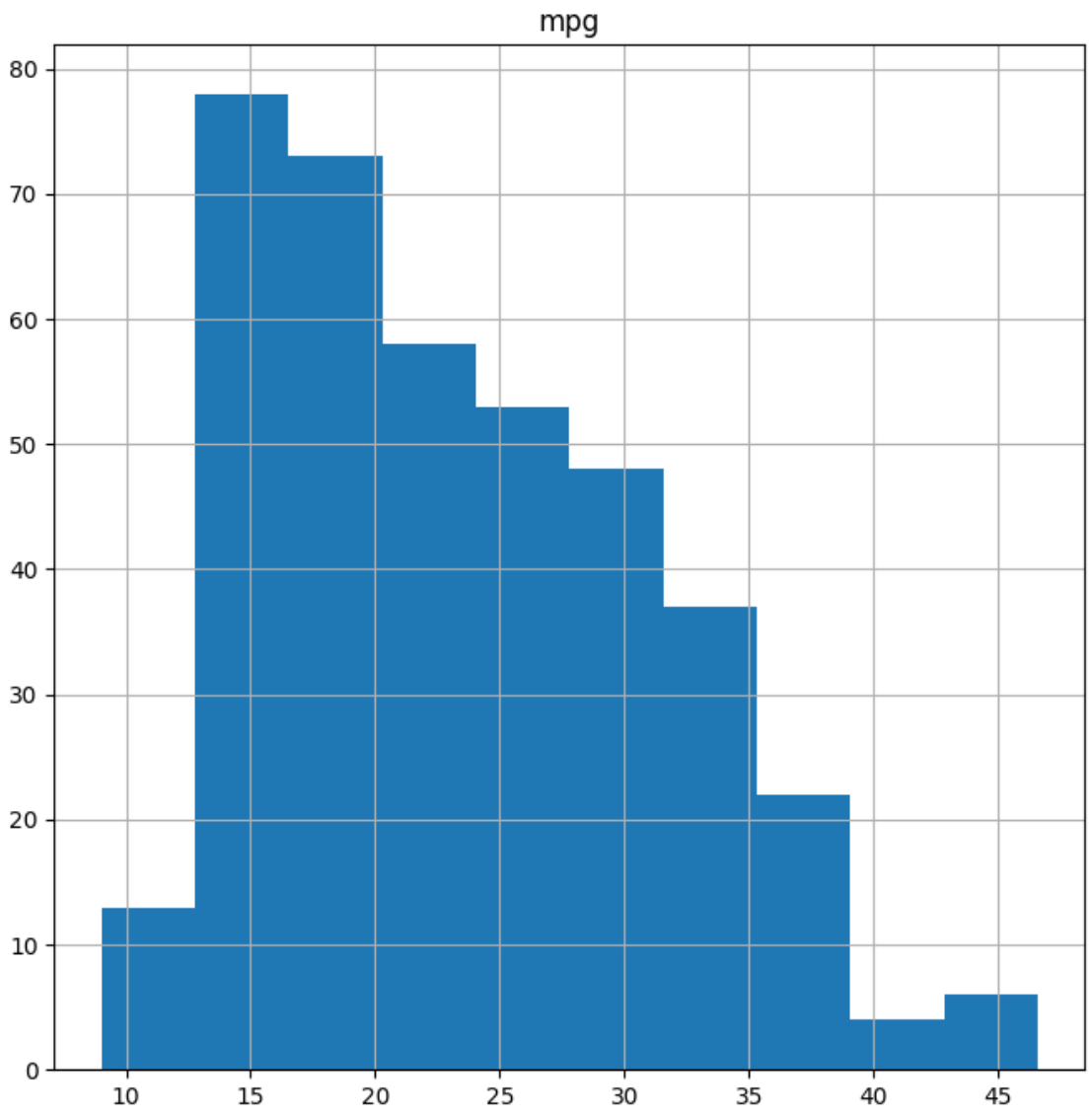
[ ]:
```
Auto.cylinders = pd.Series(Auto.cylinders, dtype='category')
Auto.cylinders.dtype
```

[ ]:
```
CategoricalDtype(categories=[3, 4, 5, 6, 8], ordered=False,
categories_dtype=int64)
```

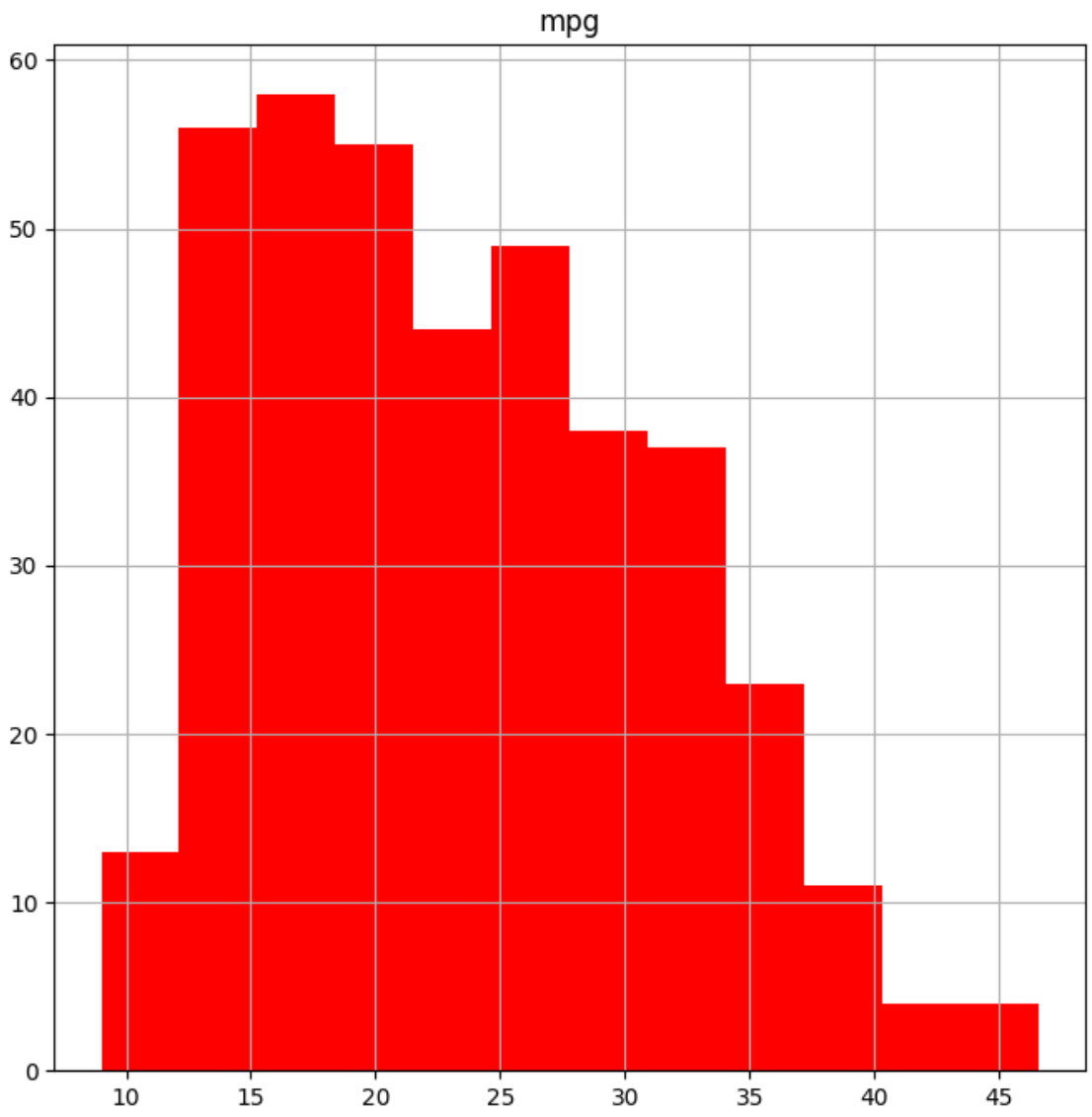
[ ]:
```
fig, ax = subplots(figsize=(8, 8))
Auto.boxplot('mpg', by='cylinders', ax=ax);
```



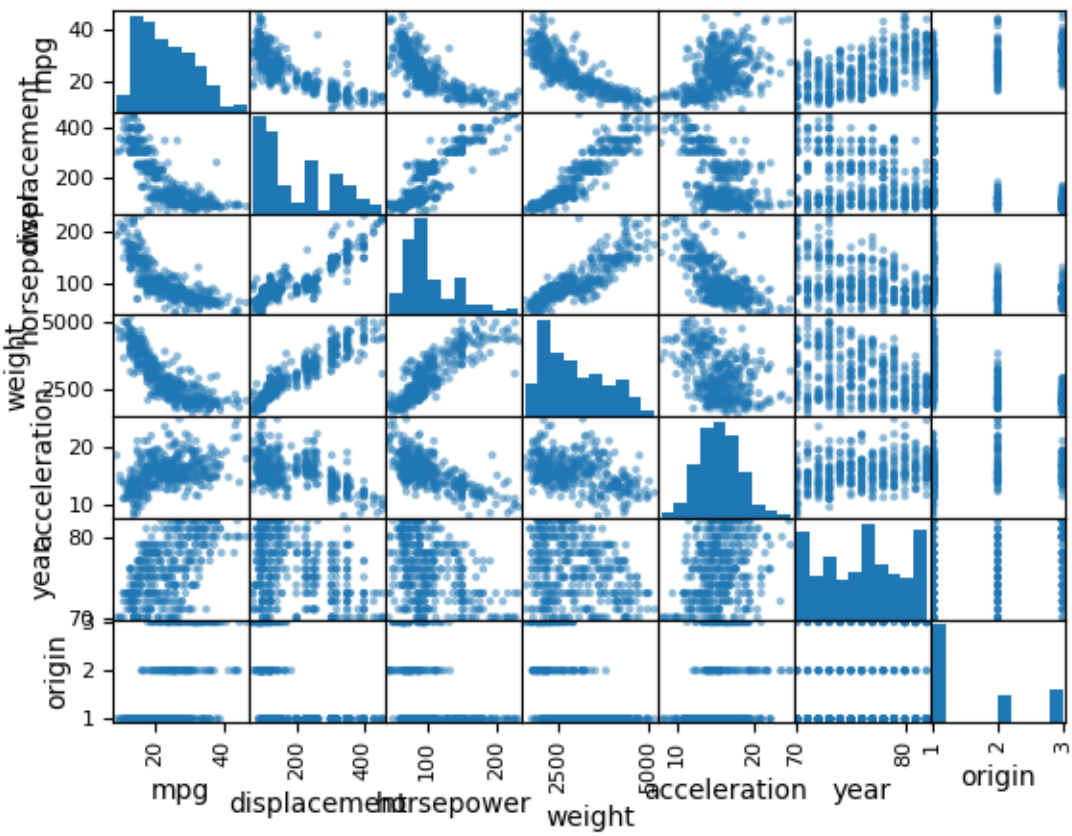
[ ]:
```
fig, ax = subplots(figsize=(8, 8))
Auto.hist('mpg', ax=ax);
```



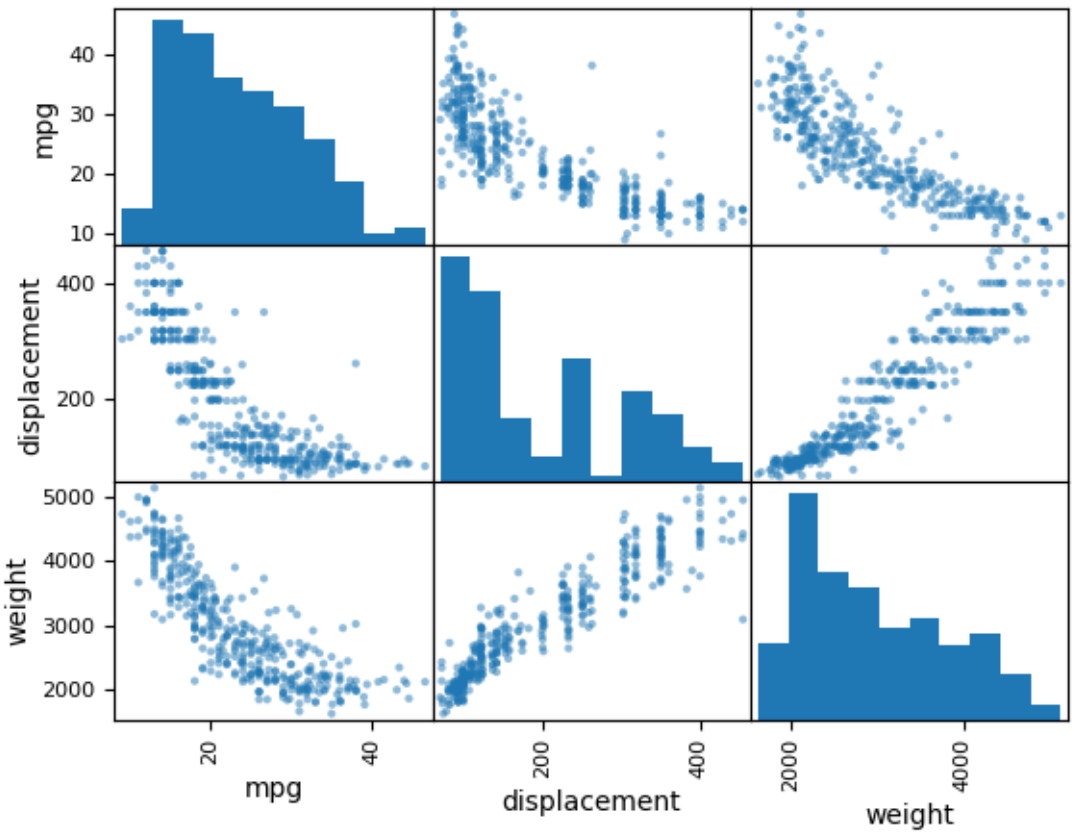
[ ]:
```
fig, ax = subplots(figsize=(8, 8))
Auto.hist('mpg', color='red', bins=12, ax=ax);
```



[ ]:
```
pd.plotting.scatter_matrix(Auto);
```



[ ]:
```
pd.plotting.scatter_matrix(Auto[['mpg',
                                 'displacement',
                                 'weight']]);
```




```
[ ]: Auto[['mpg', 'weight']].describe()
```

```
[ ]:              mpg       weight
     count  392.000000   392.000000
     mean    23.445918  2977.584184
     std      7.805007   849.402560
     min      9.000000  1613.000000
     25%     17.000000  2225.250000
     50%     22.750000  2803.500000
     75%     29.000000  3614.750000
     max     46.600000  5140.000000
```

```
[ ]: Auto['cylinders'].describe()
     Auto['mpg'].describe()
```

```
[ ]: count    392.000000
     mean      23.445918
     std        7.805007
     min        9.000000
     25%       17.000000
```

```
50%       22.750000
75%       29.000000
max       46.600000
Name: mpg, dtype: float64
```