A_worksheet

February 2, 2023

1 Worksheet A

variationalform https://variationalform.github.io/

Just Enough: progress at pace https://variationalform.github.io/

https://github.com/variationalform

Simon Shaw https://www.brunel.ac.uk/people/simon-shaw.

This document uses python

and also makes use of LaTeX

in Markdown

1.1 What this is about:

This worksheet is based on the material in the notebooks

- intro
- vectors

Note that while the 'lecture' notebooks are prefixed with `1_`, `2_` and so on, to indicate the order in which they should be studied, the worksheets are prefixed with `A_`, `B_`, …

Refer back to the section where we introduced the use of `numpy` for vector calculations.

There we had set up the vectors $\boldsymbol{a}$ and $\boldsymbol{p}$ and illustrated arithmetic like this:

[1]:
```
import numpy as np
a = np.array([3, -2, 1])
p = np.array([5, 2, -10])
g = a-p
print(g)
a = g+p
print(a)
```

```
[-2 -4 11]
[ 3 -2  1]
```

Exercises

1. Repeat the above calculation but with a and p as `numpy` column arrays.
2. Use `np.sqrt(16)` to print out $\sqrt{16}$ (e.g. https://numpy.org/doc/stable/reference/generated/numpy.sqrt.html)
3. Use `np.power(2,3)` to print out 2^3 (e.g. https://numpy.org/doc/stable/reference/generated/numpy.power.html).
4. Use `np.pi` to print out the area of a circle with radius 5. (e.g. https://numpy.org/doc/stable/reference/constants.html)
5. Define these as `numpy` row arrays.

$$\boldsymbol{x} = \begin{pmatrix} 3 \\ -5 \\ \pi \\ 7.2 \\ \sqrt{9} \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{y} = \begin{pmatrix} -3 \\ 16 \\ 1 \\ 1089 \\ 15 \end{pmatrix}$$

6. Print out $\boldsymbol{x} + 2\boldsymbol{y}$.
7. Repeat 5 and 6 but with x and y as `numpy` column arrays.

[2]:
```
# put your working in here - make new cells if you like
```

Hints

```
a = np.array([[3], [-2], [1]])
p = np.array([[5], [2], [-10]])
g = a-p
print(g)
a = g+p
print(a)

print('sqrt(16) = ', np.sqrt(16))
print('2 cubed = ', np.power(2,3))
print('pi*r-squared =', np.pi*5**2)
x = np.array([3, -5, np.pi, 7.2, np.sqrt(9)])
y = np.array([-3, 16, 1, 1089, 15])
print ('x + 2y = ', x + 2*y)

x.shape = (5,1)
y.shape = (5,1)
print ('x + 2y = \n', x + 2*y)
```

1.2 1089

What is special about 1089?

Take three integers from $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and use them to make a three digit number. Reverse the digits and subtract the smaller from the larger to get a new three digit number (put a zero in front if it is only two digits). Add this number to its own reversal.

Example, 2, 6, 8 gives 268. Reversing and subtracting gives $862 - 268 = 594$. Reversing and adding then gives $594 + 495 =$?

Try this for other numbers - is it always the same?

See e.g. https://en.wikipedia.org/wiki/1089_(number)

[3]:
```
print(862-268)
print(594+495)
```

```
594
1089
```

1.2.1 Vector norms

We saw the vector p-norm for any $p \geq 1$ given by

$$\|\boldsymbol{v}\|_p = \begin{cases} \sqrt[p]{|v_1|^p + |v_2|^p + \cdots + |v_n|^p}, & \text{if } 1 \leq p < \infty; \\ \max\{|v_k| \colon k = 1, 2, \ldots, n\}, & \text{if } p = \infty. \end{cases}$$

Although $p < 1$ is not allowed here, we do sometimes use *phoney norms* for cases when $p < 1$

Exercises - see e.g. https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html Using `numpy.linalg.norm()` with,

$$\boldsymbol{w} = \big(3, 0, -5, 4, \pi^2, 0, -23, 7, -99\big)^T$$

calculate these quantities and check your answers:

1. $\|\boldsymbol{w}\|_2$
2. $\|\boldsymbol{w}\|_1$
3. $\|\boldsymbol{w}\|_\infty$
4. $\|\boldsymbol{w}\|_3$
5. $\|\boldsymbol{w}\|_{\ln 7}$
6. $\|\boldsymbol{w}\|_{23}$

Hint: https://numpy.org/doc/stable/reference/generated/numpy.log.html

Furthermore, use `numpy` to calculate these *phoney norms*:

1. $\|\boldsymbol{w}\|_{\sqrt{2}-1}$
2. $\|\boldsymbol{w}\|_0$

Hint: https://numpy.org/doc/stable/reference/generated/numpy.sqrt.html

[4]:
```
# put your working in here - make new cells if you like
```

Hints

```
import numpy as np
w = np.array([3, 0, -5, 4, np.pi**2, 0, -23, 7, -99])
print(w)
print('||w||_2   = ', np.linalg.norm(w,2))
print('||w||_1   = ', np.linalg.norm(w,1))
print('||w||_inf = ', np.linalg.norm(w,np.inf))  # note how we denote infinity
print('||w||_3   = ', np.linalg.norm(w,3))
print('||w||_ln7 = ', np.linalg.norm(w,np.log(7)))
print('||w||_23  = ', np.linalg.norm(w,23))
print('||w||_0.414...   = ', np.linalg.norm(w,np.sqrt(2)-1))
print('||w||_0   = ', np.linalg.norm(w,0))
```

1.3 Data as vectors

We look back at some of the data and learn a bit more about python and how to manipulate raw data.

We start with the `taxis` data sets and see how we can convert some num-numerica data into numeric.

[5]:
```
import seaborn as sns
dft = sns.load_dataset('taxis')
dft.head(9)
```

[5]:

	pickup	dropoff	passengers	distance	fare	tip	\
0	2019-03-23 20:21:09	2019-03-23 20:27:24	1	1.60	7.0	2.15	
1	2019-03-04 16:11:55	2019-03-04 16:19:00	1	0.79	5.0	0.00	
2	2019-03-27 17:53:01	2019-03-27 18:00:25	1	1.37	7.5	2.36	
3	2019-03-10 01:23:59	2019-03-10 01:49:51	1	7.70	27.0	6.15	
4	2019-03-30 13:27:42	2019-03-30 13:37:14	3	2.16	9.0	1.10	
5	2019-03-11 10:37:23	2019-03-11 10:47:31	1	0.49	7.5	2.16	
6	2019-03-26 21:07:31	2019-03-26 21:17:29	1	3.65	13.0	2.00	
7	2019-03-22 12:47:13	2019-03-22 12:58:17	0	1.40	8.5	0.00	
8	2019-03-23 11:48:50	2019-03-23 12:06:14	1	3.63	15.0	1.00	

	tolls	total	color	payment	pickup_zone	\
0	0.0	12.95	yellow	credit card	Lenox Hill West	
1	0.0	9.30	yellow	cash	Upper West Side South	
2	0.0	14.16	yellow	credit card	Alphabet City	
3	0.0	36.95	yellow	credit card	Hudson Sq	
4	0.0	13.40	yellow	credit card	Midtown East	
5	0.0	12.96	yellow	credit card	Times Sq/Theatre District	
6	0.0	18.80	yellow	credit card	Battery Park City	
7	0.0	11.80	yellow	NaN	Murray Hill	
8	0.0	19.30	yellow	credit card	East Harlem South	

	dropoff_zone	pickup_borough	dropoff_borough

```
0      UN/Turtle Bay South      Manhattan       Manhattan
1    Upper West Side South      Manhattan       Manhattan
2             West Village      Manhattan       Manhattan
3          Yorkville West      Manhattan       Manhattan
4          Yorkville West      Manhattan       Manhattan
5            Midtown East      Manhattan       Manhattan
6  Two Bridges/Seward Park      Manhattan       Manhattan
7                 Flatiron      Manhattan       Manhattan
8          Midtown Center      Manhattan       Manhattan
```

Exercises In the following the question mark denotes values that you have to determine.

1. use `dft.shape` to determine and print the number of rows and columns in the data set.
2. use `dft.shape[?]` to determine and print the number of rows in the data set.
3. use `dft.shape[?]` to determine and print the number of columns in the data set.
4. Use `dft.iat[?,?]` to print out the contents of the fourth column of the third row.
5. Use `dft.loc[?].iat[?]` to print out the contents of the twelfth column of the sixth row.
6. Use `dft.loc[?]` to print out the contents of ninth row.

[6]:
```
# put your working in here - make new cells if you like
```

Hints

```
print('The number of rows and columns are: ', dft.shape)
print('The number of rows is: ', dft.shape[0])
print('The number of columns is: ', dft.shape[1])
print('dft.iat[2,3]      = ', dft.iat[2,3])
print('dft.loc[5].iat[11] = ', dft.loc[5].iat[11],'\n')
print('dft.loc[8] = ')
```

1.4 Working with `datetime`

We have already seen that the first column in the `dft.head()` output can be ignored - that is just a label for each observation and has nothing to do with the taxi ride data.

The pickup and dropoff columns are dates and times, and we ignored them earlier when we first worked with this data set because they aren't simple numerical values that can be put into a vector.

However, if we think of those as the number of seconds since 1 January 1970 then they are also just numbers (integers if we ignore fractions of a second). These two numbers can then go in a vector, just as those values in columns three to eight did.

Let's see how to convert the dates and times to simpler integers.

The following material on date-time conversion was adapted from that at https://docs.python.org/3/library/datetime.html and https://stackoverflow.com/questions/11743019/convert-python-datetime-to-epoch-with-strftime - it uses the `datetime` module.

We'll remind ourself of the first few data points, and then import `datetime`.

[7]:
```
dft.head(3)
```

[7]:
```
                 pickup              dropoff  passengers  distance  fare   tip  \
0  2019-03-23 20:21:09  2019-03-23 20:27:24           1      1.60   7.0  2.15
1  2019-03-04 16:11:55  2019-03-04 16:19:00           1      0.79   5.0  0.00
2  2019-03-27 17:53:01  2019-03-27 18:00:25           1      1.37   7.5  2.36

   tolls  total   color      payment            pickup_zone  \
0    0.0  12.95  yellow  credit card        Lenox Hill West
1    0.0   9.30  yellow         cash  Upper West Side South
2    0.0  14.16  yellow  credit card          Alphabet City

             dropoff_zone pickup_borough dropoff_borough
0    UN/Turtle Bay South      Manhattan       Manhattan
1  Upper West Side South      Manhattan       Manhattan
2           West Village      Manhattan       Manhattan
```

[8]:
```
# here is a useful alternative
dft[0:3]
```

[8]:
```
                 pickup              dropoff  passengers  distance  fare   tip  \
0  2019-03-23 20:21:09  2019-03-23 20:27:24           1      1.60   7.0  2.15
1  2019-03-04 16:11:55  2019-03-04 16:19:00           1      0.79   5.0  0.00
2  2019-03-27 17:53:01  2019-03-27 18:00:25           1      1.37   7.5  2.36

   tolls  total   color      payment            pickup_zone  \
0    0.0  12.95  yellow  credit card        Lenox Hill West
1    0.0   9.30  yellow         cash  Upper West Side South
2    0.0  14.16  yellow  credit card          Alphabet City

             dropoff_zone pickup_borough dropoff_borough
0    UN/Turtle Bay South      Manhattan       Manhattan
1  Upper West Side South      Manhattan       Manhattan
2           West Village      Manhattan       Manhattan
```

Exercise Show only rows 5 to 9 (inclusive) of the data set.

Hint: do you think that `[4:9]` refers to rows four to nine? If not, then what?

[9]:
```
# put your working in here - make new cells if you like
```

[10]:
```
# import the module
from datetime import datetime
# and show the first few of rows again
dft[0:3]
```

[10]:
```
                 pickup             dropoff  passengers  distance  fare   tip  \
0  2019-03-23 20:21:09  2019-03-23 20:27:24           1      1.60   7.0  2.15
1  2019-03-04 16:11:55  2019-03-04 16:19:00           1      0.79   5.0  0.00
2  2019-03-27 17:53:01  2019-03-27 18:00:25           1      1.37   7.5  2.36

   tolls  total   color      payment            pickup_zone  \
0    0.0  12.95  yellow  credit card        Lenox Hill West
1    0.0   9.30  yellow         cash  Upper West Side South
2    0.0  14.16  yellow  credit card          Alphabet City

             dropoff_zone pickup_borough dropoff_borough
0    UN/Turtle Bay South      Manhattan       Manhattan
1  Upper West Side South      Manhattan       Manhattan
2           West Village      Manhattan       Manhattan
```

[11]:
```
# use the pickup time from the first data point
put = datetime(2019,3,23,20,21,9)
print('the data and time are ', put)
# the elapsed time since 1 January 1970 is
print('elapsed time since 1 January 1970 = ', (put - datetime(1970,1,1)).
 ↪total_seconds() )
# a quicker way to get this is
print('timestamp = ', put.timestamp())
```

```
the data and time are  2019-03-23 20:21:09
elapsed time since 1 January 1970 =  1553372469.0
timestamp =  1553372469.0
```

Now, we don't want to be typing all of this ourselves so let's start to automate the process.

First, we use `dft.iat[0,0]` to get the first element in the first row of the data-frame. We print it out and see that it is just a character string giving the date and time we saw above.

[12]:
```
print(dft.iat[0,0])
```

```
2019-03-23 20:21:09
```

This string is `dft.iat[0,0])` which we interpret as `%Y-%m-%d %H:%M:%S` and then we can use `strptime()` to get the `datetime` variable. This https://www.digitalocean.com/community/tutorials/python-string-to-datetime-strptime was a useful source for writing these notes.

Once done, it's then easy to get the timestamp - just as above.

[13]:
```
dt = datetime.strptime(str(dft.iat[0,0]), '%Y-%m-%d %H:%M:%S' )
print('The date and time are ', dt, ' with timestamp ', dt.timestamp())
```

```
The date and time are  2019-03-23 20:21:09  with timestamp  1553372469.0
```

Note that we had to write `str(dft.iat[0,0])` rather that just `dft.iat[0,0]`. This is because (at least at the time of writing) if we try to run this notebook in binder we will get an error.

The `str()` function converts its argument into a string. The next two commands show the `type` of `dft.iat[0,0]`. If you see `str` (i.e. string) for both then you don't need to replace `dft.iat[0,0]` with `str(dft.iat[0,0])`.

In binder these don't both give `str`, and so we have to force it to be a string.

[14]:
```
type( str(dft.iat[0,0]) )
```

[14]:
```
str
```

[15]:
```
type(dft.iat[0,0])
```

[15]:
```
str
```

Let's remind ourselves of the first line of the data, and then get the timestamps for the first two entries:

[16]:
```
dft[0:1]
```

[16]:
```
                 pickup              dropoff  passengers  distance  fare   tip  \
0  2019-03-23 20:21:09  2019-03-23 20:27:24           1       1.6   7.0  2.15

   tolls  total   color      payment      pickup_zone         dropoff_zone  \
0    0.0  12.95  yellow  credit card  Lenox Hill West  UN/Turtle Bay South

  pickup_borough dropoff_borough
0      Manhattan       Manhattan
```

[17]:
```
print(datetime.strptime( str(dft.iat[0,0]), '%Y-%m-%d %H:%M:%S' ).timestamp())
print(datetime.strptime( str(dft.iat[0,1]), '%Y-%m-%d %H:%M:%S' ).timestamp())
```

```
1553372469.0
1553372844.0
```

The time taken between pickup and dropoff, `2019-03-23 20:21:09` and `2019-03-23 20:27:24` is 6 minutes and 15 seconds. This is

[18]:
```
print(6*60+15,' seconds')
```

```
375  seconds
```

Now let's look at the difference between the two time stamps

[19]:
```
ts1 = datetime.strptime( str(dft.iat[0,0]), '%Y-%m-%d %H:%M:%S' ).timestamp()
ts2 = datetime.strptime( str(dft.iat[0,1]), '%Y-%m-%d %H:%M:%S' ).timestamp()
print('ts 1 = ', ts1)
print('ts 2 = ', ts2)
print('ts2-ts1 = ', ts2-ts1)
```

```
ts 1 =  1553372469.0
ts 2 =  1553372844.0
ts2-ts1 =  375.0
```

and so the timestamp is just the conversion of a date to seconds.

You may wonder, then, *when was zero seconds*? Well, it was 1 January 1970 UTC (Coordinated Universal Time). To see this, we need to make sure we specify the correct timezone and then:

[20]:
```
from datetime import timezone
print(datetime(1970,1,1,0,0,0,tzinfo=timezone.utc).timestamp())
```

```
0.0
```

So, here then is our vector for the zero-th taxi data. Note that it now includes the first two values as well as the six we had in the lecture.

Note that we use `concatenate` to join two `numpy` arrays.

[21]:
```
r0 = np.array(dft.iloc[0,2:8])
print(r0)
r0 = np.concatenate(([ts1, ts2], np.array(dft.iloc[0,2:8])), axis=None)
print(r0)
```

```
[1 1.6 7.0 2.15 0.0 12.95]
[1553372469.0 1553372844.0 1 1.6 7.0 2.15 0.0 12.95]
```

Exercises In the following the question mark denotes values that you have to determine.

1. Use dft[?:?] to access rows 253 - 256 of the `taxis` datasets.
2. What do `%` and `//` do in `python`? (Hint: `print(10//7,10%7)`).
3. Use `datetime` to find the elapsed time in seconds for row 256.
4. Convert that time in seconds to minutes and seconds.
5. Print out row 14.
6. Create a vector of numbers that holds the first eight values from row 14.

For the last, use `timestamp()` to represent the date-time values. Also, look at what these commands do. They might help you join two arrays without explicitly using `concatenate`.

```
A = np.array([1,2])
B = np.array([3,4,5,6,7,8])
print(A,B)
C = np.r_[A,B]
print(C)
```

[22]:
```
# put your working in here - make new cells if you like
```

[23]:
```
# put your working in here - make new cells if you like
```

[24]:
```
# put your working in here - make new cells if you like
```

[25]:
```
# put your working in here - make new cells if you like
```

Hints

```
print(10//7,10%7)

print('dft[252:256] = ')
dft[252:256]


ts1 = datetime.strptime(dft.iat[255,0], '%Y-%m-%d %H:%M:%S' ).timestamp()
ts2 = datetime.strptime(dft.iat[255,1], '%Y-%m-%d %H:%M:%S' ).timestamp()
print('elaspsed time in seconds: ', ts2-ts1)
mins = (ts2-ts1) % 60
print('elaspsed time in min:secs: ', (ts2-ts1) // 60, ':', (ts2-ts1) % 60)


dft[13:14]

ts1 = datetime.strptime(dft.iat[13,0], '%Y-%m-%d %H:%M:%S' ).timestamp()
ts2 = datetime.strptime(dft.iat[13,1], '%Y-%m-%d %H:%M:%S' ).timestamp()
ts12 = np.array([ts1,ts2])
print(ts12)
row = np.array(dft.iloc[13,2:8])
print(row)
row = np.r_[ts12,r3]
print(row)
```

THINK ABOUT What information could be lost as a result of converting the date-time to a number?

Exercises For the `taxis` data set:

1. Produce a scatterplot of "dropoff_borough" vs. "tip"
2. Plot the dependence of fare on distance.

```
dft = sns.load_dataset('taxis')
dft.head()

sns.scatterplot(data=dft, x="dropoff_borough", y="tip")
sns.scatterplot(data=dft, x="distance", y="fare")
```

[26]:
```
# put your working in here - make new cells if you like
```

[27]:
```
# put your working in here - make new cells if you like
```

[28]:
```
# put your working in here - make new cells if you like
```

Exercises For the `tips` data set:

1. What is the standard deviation of the tips?
2. Plot the scatter of tip against the total bill
3. Plot the scatter of total bill against day
4. Plot the scatter of tip against gender

```
dftp = sns.load_dataset('tips')
dftp.describe()
sns.scatterplot(data=dftp, x="total_bill", y="tip")
sns.scatterplot(data=dftp, x="day", y="total_bill")
sns.scatterplot(data=dftp, x="sex", y="tip")
```

[29]:
```
# put your working in here - make new cells if you like
```

[30]:
```
# put your working in here - make new cells if you like
```

[31]:
```
# put your working in here - make new cells if you like
```

[32]:
```
# put your working in here - make new cells if you like
```

1.5 Self-Study and Homework

Work through the following material and have a go at the questions at the end. Make a note of anything you don't understand, and ask in the next session.

1.5.1 The `anscombe` data set

As discussed in the lectures, this is pretty famous. There was a lot to take in during the walk-through of the lecture notebook so this is another opportunity to slow things down and read at your own pace. See https://en.wikipedia.org/wiki/Anscombe%27s_quartet

Image Credit: `https://upload.wikimedia.org/wikipedia/commons/7/7e/Julia-anscombe-plot-1.png`

[33]:
```
dfa = sns.load_dataset('anscombe')
# look at how we get an apostrophe...
print("The size of Anscombe's data set is:", dfa.shape)
```

```
The size of Anscombe's data set is: (44, 3)
```

Let's take a look at the data set - we can look at the head and tail of the table just as we did above.

[34]:
```
dfa.head()
```

[34]:

	dataset	x	y
0	I	10.0	8.04
1	I	8.0	6.95
2	I	13.0	7.58
3	I	9.0	8.81
4	I	11.0	8.33

[35]:
```
dfa.tail()
```

[35]:

	dataset	x	y
39	IV	8.0	5.25
40	IV	19.0	12.50
41	IV	8.0	5.56
42	IV	8.0	7.91
43	IV	8.0	6.89

It looks like the four data sets are in the `dataset` column. How can we extract them as separate items?

Well, one way is to print the whole dataset and see which rows correspond to each dataset. Like this…

[36]:
```
print(dfa)
```

```
   dataset     x      y
0        I  10.0   8.04
1        I   8.0   6.95
2        I  13.0   7.58
3        I   9.0   8.81
4        I  11.0   8.33
5        I  14.0   9.96
6        I   6.0   7.24
7        I   4.0   4.26
8        I  12.0  10.84
9        I   7.0   4.82
10       I   5.0   5.68
11      II  10.0   9.14
12      II   8.0   8.14
13      II  13.0   8.74
14      II   9.0   8.77
15      II  11.0   9.26
16      II  14.0   8.10
17      II   6.0   6.13
18      II   4.0   3.10
19      II  12.0   9.13
20      II   7.0   7.26
21      II   5.0   4.74
22     III  10.0   7.46
23     III   8.0   6.77
24     III  13.0  12.74
25     III   9.0   7.11
26     III  11.0   7.81
27     III  14.0   8.84
28     III   6.0   6.08
29     III   4.0   5.39
```

```
30     III  12.0   8.15
31     III   7.0   6.42
32     III   5.0   5.73
33      IV   8.0   6.58
34      IV   8.0   5.76
35      IV   8.0   7.71
36      IV   8.0   8.84
37      IV   8.0   8.47
38      IV   8.0   7.04
39      IV   8.0   5.25
40      IV  19.0  12.50
41      IV   8.0   5.56
42      IV   8.0   7.91
43      IV   8.0   6.89
```

From this we can see that there are four data sets: I, II, III and IV. They each contain 11 pairs (x, y).

- The first set occupies rows $0, 1, 2, \ldots, 10$
- The second set occupies rows $11, 12, \ldots, 21$
- The third set occupies rows $22, 23, \ldots, 32$
- The fourth set occupies rows $33, 34, \ldots, 43$

However, this kind of technique is not going to be useful if we have a data set with millions of data points (rows). We certainly wont want to print them all like we did above.

Is there another way to determine the number of distinct feature values in a given column of the data frame?

Fortunately, yes. We want to know how many different values the `dataset` column has. We can do it like this.

[37]:
```
dfa.dataset.unique()
```

[37]:
```
array(['I', 'II', 'III', 'IV'], dtype=object)
```

We can count the number of different ones automatically too, by asking for the `shape` of the returned value. Here we go:

[38]:
```
dfa.dataset.unique().shape
```

[38]:
```
(4,)
```

This tell us that there are 4 items - as expected. Don't worry too much about it saying `(4,)` rather that just `4`. We've seen what `shape` refers to earlier.

Now, we want to extract each of the four datasets as separate data sets so we can work with them. We can do that by using `loc` to get the row-wise locations where each value of the `dataset` feature is the same. (Ref: <# https://stackoverflow.com/questions/17071871/how-do-i-select-rows-from-a-dataframe-based-on-column-values>)

For example, to get the data for the sub-data-set `I` we can do this:

[39]:
```
dfa.loc[dfa['dataset'] == 'I']
```

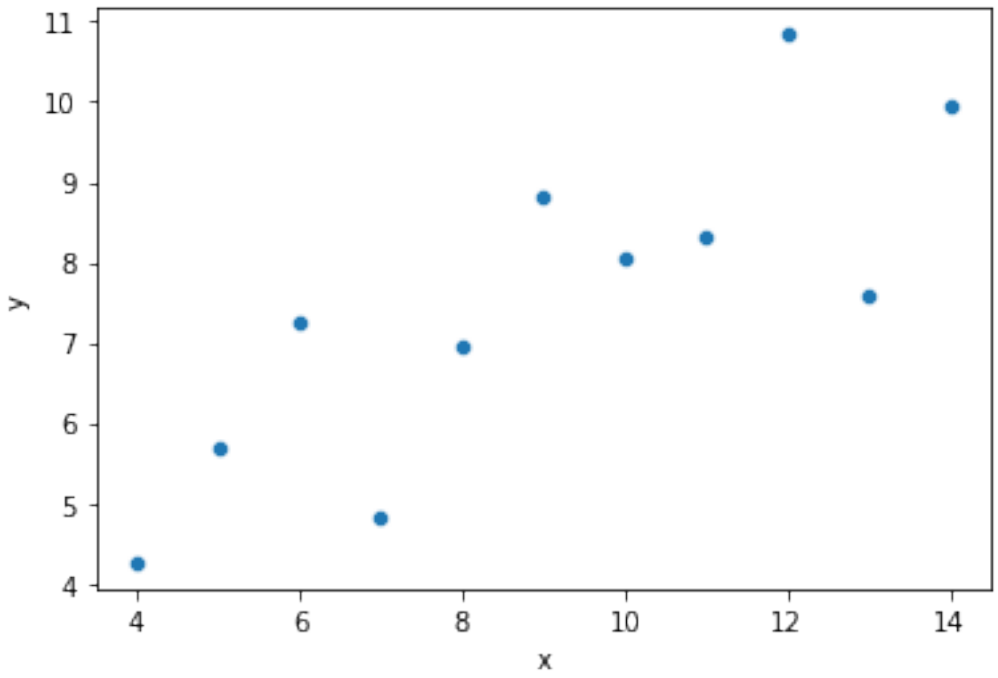
[39]:

	dataset	x	y
0	I	10.0	8.04
1	I	8.0	6.95
2	I	13.0	7.58
3	I	9.0	8.81
4	I	11.0	8.33
5	I	14.0	9.96
6	I	6.0	7.24
7	I	4.0	4.26
8	I	12.0	10.84
9	I	7.0	4.82
10	I	5.0	5.68

Now we have this subset of data we can examine it - with a scatter plot for example.

[40]:
```
sns.scatterplot(data=dfa.loc[dfa['dataset'] == 'I'], x="x", y="y")
```

[40]:
```
<AxesSubplot:xlabel='x', ylabel='y'>
```



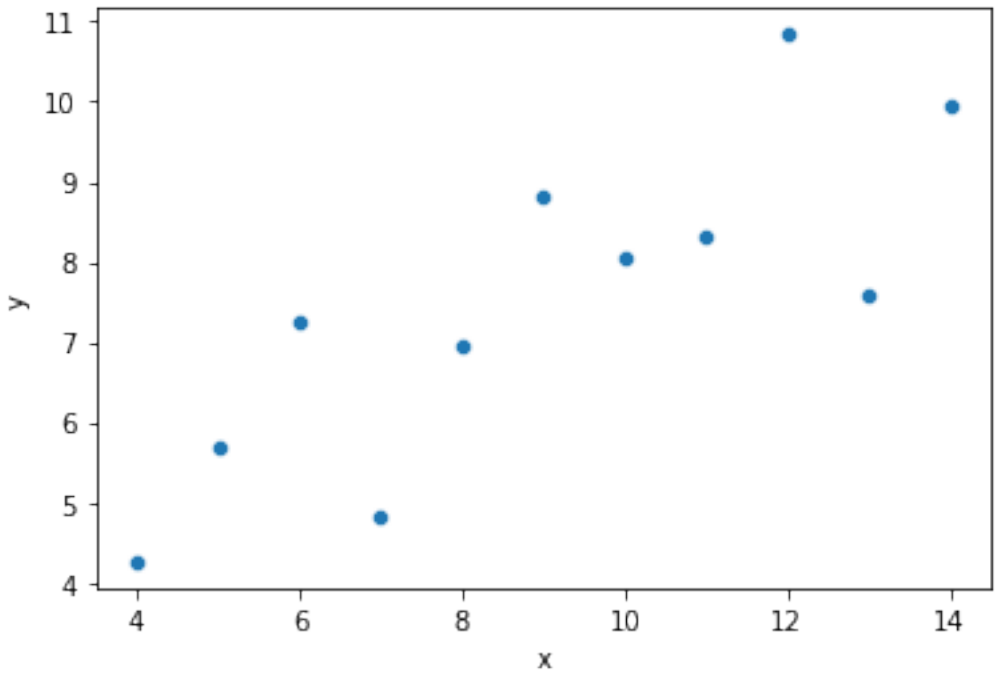
To really work properly with each subset we should extract them and give each of them a name that is meaningful.

[41]:
```
# On the other hand:
dfa1 = dfa.loc[dfa['dataset'] == 'I']
dfa2 = dfa.loc[dfa['dataset'] == 'II']
dfa3 = dfa.loc[dfa['dataset'] == 'III']
dfa4 = dfa.loc[dfa['dataset'] == 'IV']
```

[42]:
```
sns.scatterplot(data=dfa1, x="x", y="y")
dfa1.describe()
```

[42]:

	x	y
count	11.000000	11.000000
mean	9.000000	7.500909
std	3.316625	2.031568
min	4.000000	4.260000
25%	6.500000	6.315000
50%	9.000000	7.580000
75%	11.500000	8.570000
max	14.000000	10.840000

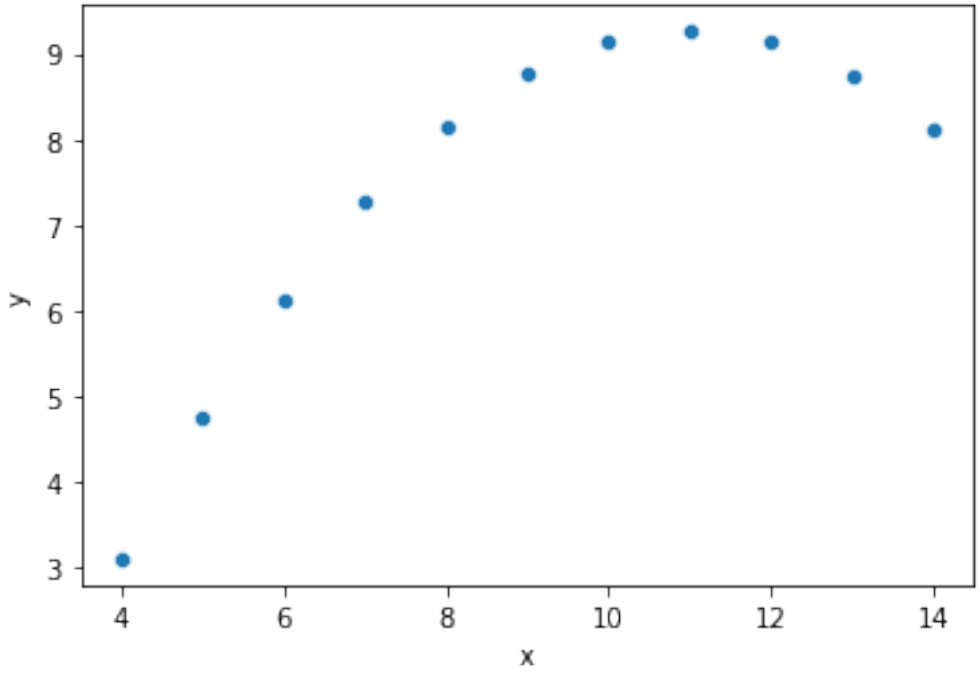


[43]:
```
sns.scatterplot(data=dfa2, x="x", y="y")
dfa2.describe()
```

[43]:

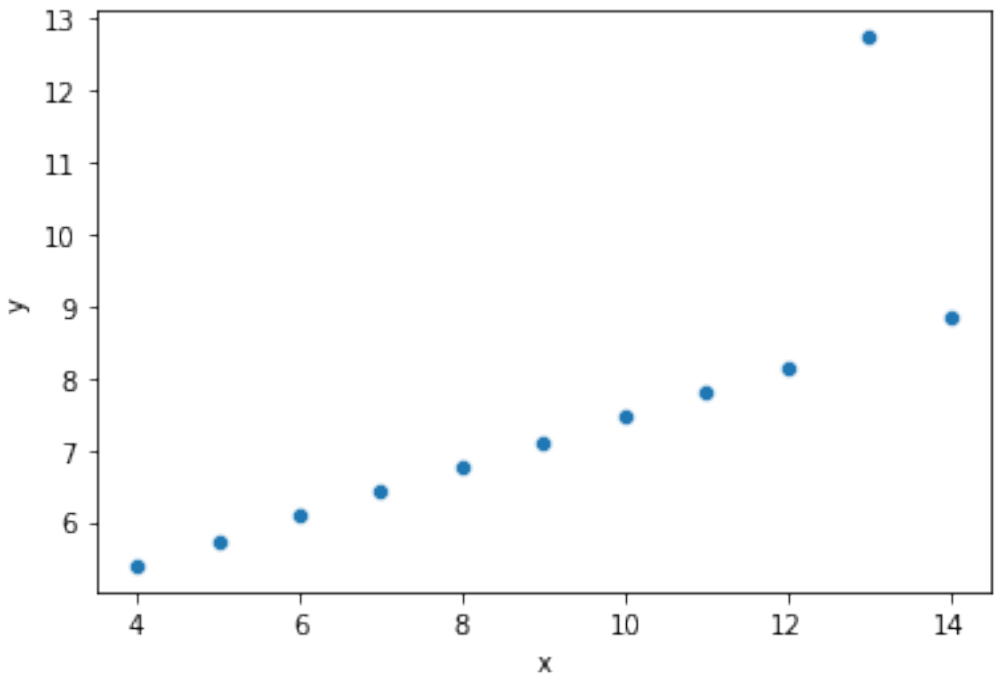
	x	y
count	11.000000	11.000000
mean	9.000000	7.500909

```
std      3.316625   2.031657
min      4.000000   3.100000
25%      6.500000   6.695000
50%      9.000000   8.140000
75%     11.500000   8.950000
max     14.000000   9.260000
```



[44]:
```
sns.scatterplot(data=dfa3, x="x", y="y")
dfa3.describe()
```

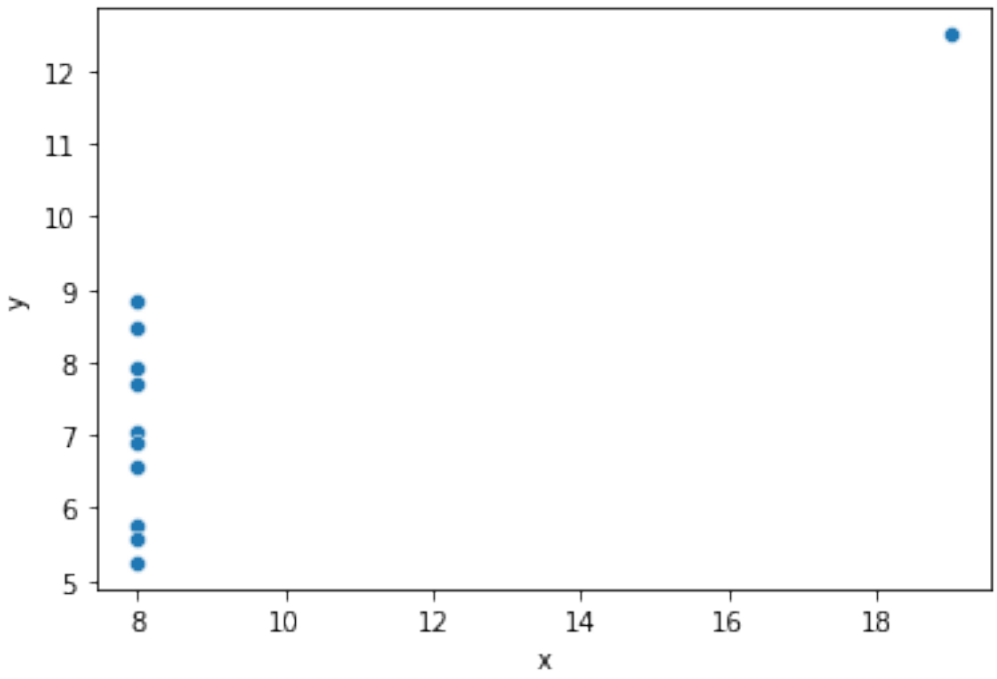
[44]:
```
               x          y
count  11.000000  11.000000
mean    9.000000   7.500000
std     3.316625   2.030424
min     4.000000   5.390000
25%     6.500000   6.250000
50%     9.000000   7.110000
75%    11.500000   7.980000
max    14.000000  12.740000
```



[45]:
```
sns.scatterplot(data=dfa4, x="x", y="y")
dfa4.describe()
```

[45]:

	x	y
count	11.000000	11.000000
mean	9.000000	7.500909
std	3.316625	2.030579
min	8.000000	5.250000
25%	8.000000	6.170000
50%	8.000000	7.040000
75%	8.000000	8.190000
max	19.000000	12.500000



1.6 Exercises

For the Anscombe data set:

1. Which of the summary statistics for x are the same or similar for each subset?
2. Which of the summary statistics for y are the same or similar for each subset?

Look at the `diamonds` data set

1. How many diamonds are listed there?
2. How many attributes, or features, does each have?
3. Create a scatter plot of price against carat.

```
1: dfd = sns.load_dataset('diamonds')
dfd.shape
53940 and 10
2: sns.scatterplot(data=dfd, x="carat", y="price")
```

1.7 Technical Notes, Production and Archiving

Ignore the material below. What follows is not relevant to the material being taught.

Production Workflow

- Finalise the notebook material above
- Clear and fresh run of entire notebook
- Create html slide show:
  - `jupyter nbconvert --to slides A_worksheet.ipynb`

- Set `OUTPUTTING=1` below
- Comment out the display of web-sourced diagrams
- Clear and fresh run of entire notebook
- Comment back in the display of web-sourced diagrams
- Clear all cell output
- Set `OUTPUTTING=0` below
- Save
- git add, commit and push to FML
- copy PDF, HTML etc to web site
  - git add, commit and push
- rebuild binder

Some of this originated from

https://stackoverflow.com/questions/38540326/save-html-of-a-jupyter-notebook-from-within-the-n

These lines create a back up of the notebook. They can be ignored.

At some point this is better as a bash script outside of the notebook

[46]:
```
%%bash
NBROOTNAME='A_worksheet'
OUTPUTTING=1

if [ $OUTPUTTING -eq 1 ]; then
  jupyter nbconvert --to html $NBROOTNAME.ipynb
  cp $NBROOTNAME.html ../backups/$(date +"%m_%d_%Y-%H%M%S")_$NBROOTNAME.html
  mv -f $NBROOTNAME.html ./formats/html/

  jupyter nbconvert --to pdf $NBROOTNAME.ipynb
  cp $NBROOTNAME.pdf ../backups/$(date +"%m_%d_%Y-%H%M%S")_$NBROOTNAME.pdf
  mv -f $NBROOTNAME.pdf ./formats/pdf/

  jupyter nbconvert --to script $NBROOTNAME.ipynb
  cp $NBROOTNAME.py ../backups/$(date +"%m_%d_%Y-%H%M%S")_$NBROOTNAME.py
  mv -f $NBROOTNAME.py ./formats/py/
else
  echo 'Not Generating html, pdf and py output versions'
fi
```

```
Not Generating html, pdf and py output versions
```

[ ]:
```

```