1_intro

January 13, 2023

1 MA5634: Fundamentals of Machine Learning

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:

You will be introduced to …

- fundamental techniques used in data science, like:
  - k-NN: k-Nearest Neighbours;
  - data reduction with SVD and PCA;
  - linear and polynomial regression;
  - perceptrons and support vector machines;
  - neural networks and deep learning.
- essential mathematical concepts: *just enough: progress at pace*
  - you are **not expected to be a mathematician** …
  - … but you will be expected to either recall or learn basic facts and techniques in
    - vectors, matrices, and differential calculus
- essential `python` programming: *just enough: progress at pace*
  - you are **not expected to be a computer scientist** …
  - … but `python` will be introduced and used as a tool
    - only the necessary `python` syntax, tools and techniques will be taught
    - our emphasis will be on *doing* rather than *proving*

1.2 Assessment

- 40% coursework (details to follow in a few weeks)
- 60% examination (revision and reflection time will be allocated)

1.3 Study Guide

The Quality Assurance Agency for Higher Education (QAA, https://www.qaa.ac.uk) defines one academic credit as nominally equal to 10 hours of study (see https://www.qaa.ac.uk/docs/qaa/quality-code/higher-education-credit-framework-for-england.pdf).

Therefore, this 15 credit block requires nominally 150 hours of your time. Although every one of us is different and may choose to spend our time in different ways, the following sketch of these 150 hours is fairly accurate.

There will be 33 hours spent on two lectures plus one seminar/lab in each of eleven weeks. There will be a two hour exam, to which you could assign 25 hours of preparation/revision time. This accounts for $33 + 2 + 25 = 60$ hours.

In addition there is assignment which you could allocate 20 hours to, making up to 80 hours. This leaves 70 of the 150 hours over. In each of 10 weeks of term there will be a requirement to engage in set tasks and problems, and to read sections of set books and sources in order to strengthen your understanding of imparted material as well as to prepare you for the next topics. These 70 hours average out to 7 hours per week over those 10 weeks.

Note that as a full-time equivalent student you study a $4 \times 15 = 60$ credit week. Using the figures above you can think of this as $4 \times 3 = 12$ contact hours plus $4 \times 7 = 28$ private study hours per week. This is a 40 hour week.

Note that engaging at this level does not guarantee any outcome, whether that be a bare pass or an A grade. It is a guideline only. If despite engaging at this level you are struggling to progress and achieve in the module then seek help and advice.

Further, these '40 hours' have to be high quality inquisitive engagement. Writing and re-writing notes, procrastinating, and looking at but not engaging with learning materials don't really count. You'll know when you're actually **working** - you'll feel it. Have a read of this https://en.wikipedia.org/wiki/Flow_(psychology) and make learning a daily rewarding habit.

1.4 Key Concepts: Glossary of Relevant Terms

The first few of these are debateable, evolving and subject to change and interpretation. It's worth searching and reading for yourself. These are a fast growing areas.

Data Science A blend of mathematics, computer science and statistics brought to bear with some form of domain expertise.

Data Analytics Systematic computational analysis of data, used typically to discover value and insights.

Data Engineering The stewardship, cleaning, warehousing and preparation of data to support its pipelining to its exploitation.

Artificial Intelligence The development and deployment of digital systems that can effectively substitute for humans in tasks beyond the routine application of fixed rules. When you talk to your home assistant, your phone, or your satellite TV receiver, or your car, or your laptop, and so on, it has no idea what you are going to say. It doesn't have a bank of pre-answered questions, but instead it responds dynamically to what it hears. It has been trained on data, and it has learned how to respond. Incidentally, how do you think these systems even understand what you said? As a child, it took you months to begin to understand human speech…

Machine Learning The development and deployment of algorithms that are able to learn from data without explicit instructions, and then analyze, predict, or otherwise draw inferences, from unseen data. These algorithms would typically be expected to add measurable value by their performance.

> *Consider for example an algorithm that predicted **tails** for every coin flip. It's right half the time* - but there's no value in that.

Learning Machine learning models do not have intrinsic knowledge but instead learn from data. Typically a data set comprises a list of items each of which has one or more *features* which correspond to a *label*. We'll see some examples of this below.

We think of the features as being inputs to the machine learning model, and the label as being the output. Typically we want to be able to feed in new features, and have the model predict the label.

To do this we need a **training data set** so that the model can learn how to map the features to the label: the *input to the output*.

There are three basic learning paradigms:

- **Supervised Learning**: Here the data is labelled. This means that for a given set of features, or inputs, we also know their labels, or outputs. Examples of this are where…
  - We could have a list of features of insured drivers, such as age, time since they passed their driving test, type of car, locality, and along with those features a monetary value on their accident claim. The task would be to learn how much of an insurance premium to charge to a new customer once those features have been determined.
  - We might have a bank of images of handwritten digits, and for each image we know what digit is represented. The MNIST database of handwritten digits, see http://yann.lecun.com/exdb/mnist/ or https://en.wikipedia.org/wiki/MNIST_database for example, is a well known example of this. The task is to learn how to predict what digit is captured by a new image. This could be used in ANPR systems for example, https://en.wikipedia.org/wiki/Automatic_number-plate_recognition.
- **Unsupervised Learning**: This is where we only know the features and we want to cluster the data in such a way that a set of similar features can be assiociated with some common characteristic (the label).
  - This can be used on data where the anlayst doesn't initially know what they are looking for. For example, a retailer might have a mass of data of customer age, locale, average spend, types of purchased item, time of day of purchase, day of week of purchase, time of year etc. What characteristics can be used to group these customers? How can advertising be targetted?

  - principal component analysis seeks to re-orient data so that its dominant statistical properties are revealed. We'll see this later.

- **Reinforcement Learning**: This seeks to strike a balance between the two above. There are no labels, but instead, as time progresses the learning algorithm has a *reward* variable which is increased when an action it has learned has resuilted in a measurable benefit. Over time the algorithm develops a policy to inform its actions.

This last is a major topic and will not be covered in these lectures. We will see examples of the first two.

Regression and Classification Our algorithms will be developed to perform one of the following tasks:

- **Regression:** here the output, the label, can take any value in a continuous set. For example, the height of a tree, given local climate, soil type, genus, age since planting, could be considered to be any non-negative real number (although not with equal probability).

- **Classification:** in this case the label will be deemed to be one of a certain class. For example, in the handwritten digits example above, the output will be one of the digits $\{0, 1, 2, 3, \ldots, 9\}$.

Some of the algorithms we study will be able to perform both the regression and clustering tasks, although we wont always delve deeply into both capabilities.

1.5 Reading List

For the data science, our main sources of information are as follows:

- MML: Mathematics for Machine Learning, by Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. Cambridge University Press. `https://mml-book.github.io`.
- MLFCES: Machine Learning: A First Course for Engineers and Scientists, by Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, Thomas B. Schön. Cambridge University Press. `http://smlbook.org`.
- FCLA: A First Course in Linear Algebra, by Ken Kuttler, `https://math.libretexts.org/Bookshelves/Linear_Algebra/A_First_Course_in_Linear_Algebra_(Kuttler)`
- AP: Applied Probability, by Paul Pfeiffer `https://stats.libretexts.org/Bookshelves/Probability_Theory/Applied_Probability_(Pfeiffer)`
- IPDS: Introduction to Probability for Data Science, by Stanley H. Chan, `https://probability4datascience.com`
- SVMS: Support Vector Machines Succinctly, by Alexandre Kowalczyk, `https://www.syncfusion.com/succinctly-free-ebooks/support-vector-machines-succinctly`
- VMLS: Introduction to Applied Linear Algebra - Vectors, Matrices, and Least Squares, by Stephen Boyd and Lieven Vandenberghe, `https://web.stanford.edu/~boyd/vmls/`

All of the above can be accessed legally and without cost.

There are also these useful references for coding:

- PT: `python`: `https://docs.python.org/3/tutorial`
- NP: `numpy`: `https://numpy.org/doc/stable/user/quickstart.html`
- MPL: `matplotlib`: `https://matplotlib.org`

The capitalized abbreviations will be used throughout to refer to these sources. For example, we could say *See [MLFCES, Chap 2, Sec. 1] for more discussion of* ***Supervised Learning****.* This would just be a quick way of saying

> Look in Section 1, of Chapter 2, of Machine Learning: A First Course for Engineers and Scientists, by Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, Thomas B. Schön, for more discussion of supervised learning.

There will be other sources shared as we go along. For now these will get us a long way.

1.6 Coding: `python` and some data sets

For each of our main topics we will see some example data, discuss a means of working with it, and then implement those means in code. We will develop enough theory so as to understand how the codes work, but our main focus will be the intution behind the method, and the effective problem solving using code.

We choose `python` because its use in both the commercial and academic data science arena seems to be pre-eminent.

The data science techniques and algorithms we will study, and the supporting technology like graphics and number crunching, are implemented in well-known and well-documented `python` libraries. These are the main ones we will use:

- `matplotlib`: used to create visualizations, plotting 2D graphs in particular.
- `numpy`: this is *numerical python*, it is used for array processing which for us will usually mean the numerical calculaitons involving vectors and matrices.
- `scikit-learn`: a set of well documented and easy to use tools for predictive data analysis.
- `pandas`: a data analysis tool, used for the storing and manipulation of data.
- `seaborn`: a data visualization library for attractive and informative statistical graphics.

There will be others, but these are the main ones. Let's look at some examples of how to use these

1.7 Binder, Anaconda, Jupyter - a first look at some data

Eventually we will use the anaconda distribution to access `python` and the libraries we need. The coding itself will be carried out in a Jupyter notebook. We'll go through this in an early lab session. We'll start though with Binder: click here:

https://mybinder.org/v2/gh/variationalform/FML.git/HEAD

Let's see some code and some data. In the following cell we import `seaborn` and look at the names of the built in data sets. The `seaborn` library, https://seaborn.pydata.org, is designed for data visualization. It uses `matplotlib`, https://matplotlib.org, which is a graphics library for `python`.

If you want to dig deeper, you can look at https://blog.enterprisedna.co/how-to-load-sample-datasets-in-python/ and https://github.com/mwaskom/seaborn-data for the background - but you don't need to.

[1]:
```python
import seaborn as sns
# we can now refer to the seaborn library functions using 'sns'
# note that you can use another character string - but 'sns' is standard.
```

```
# note that # is used to write 'comments'
# Now let's get the names of the built-in data sets.
sns.get_dataset_names()

# type SHIFT=RETURN to execute the highlighted (active) cell
```

[1]:
```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'dowjones',
 'exercise',
 'flights',
 'fmri',
 'geyser',
 'glue',
 'healthexp',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'seaice',
 'taxis',
 'tips',
 'titanic']
```

1.7.1 The taxis data set

[2]:
```
# let's take a look at 'taxis'
dft = sns.load_dataset('taxis')
# this just plots the first few lines of the data
dft.head()
```

[2]:
```
               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

   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

            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
```

[3]:
```
# this will plot the last few lines... There are 6433 records (Why?)
dft.tail()
```

[3]:
```
                   pickup             dropoff  passengers  distance  fare  \
6428  2019-03-31 09:51:53  2019-03-31 09:55:27           1      0.75   4.5
6429  2019-03-31 17:38:00  2019-03-31 18:34:23           1     18.74  58.0
6430  2019-03-23 22:55:18  2019-03-23 23:14:25           1      4.14  16.0
6431  2019-03-04 10:09:25  2019-03-04 10:14:29           1      1.12   6.0
6432  2019-03-13 19:31:22  2019-03-13 19:48:02           1      3.85  15.0

       tip  tolls  total  color      payment          pickup_zone  \
6428  1.06    0.0   6.36  green  credit card    East Harlem North
6429  0.00    0.0  58.80  green  credit card              Jamaica
6430  0.00    0.0  17.30  green         cash  Crown Heights North
6431  0.00    0.0   6.80  green  credit card        East New York
6432  3.36    0.0  20.16  green  credit card          Boerum Hill

                           dropoff_zone pickup_borough dropoff_borough
6428                Central Harlem North      Manhattan       Manhattan
6429  East Concourse/Concourse Village         Queens           Bronx
6430                      Bushwick North       Brooklyn        Brooklyn
6431        East Flatbush/Remsen Village       Brooklyn        Brooklyn
6432                     Windsor Terrace       Brooklyn        Brooklyn
```

What we are seeing here is a **data frame**. It is furnished by the `pandas` library: https://pandas.pydata.org which is used by the `seaborn` library to store its example data sets.

Each row of the data frame corresponds to a single **data point**, which we could also call an `observation` or `measurement` (depending on context).

Each column (except the left-most) corresponds to a **feature** of the data point. The first column is just an index giving the row number. Note that this index starts at zero - so, for example, the third row will be labelled/indexed as 2. Be careful of this - it can be confusing.

In this, the variable dft is a pandas data frame: dft = 'data frame taxis'

[4]:
```
# let's print the data frame...
print(dft)
```

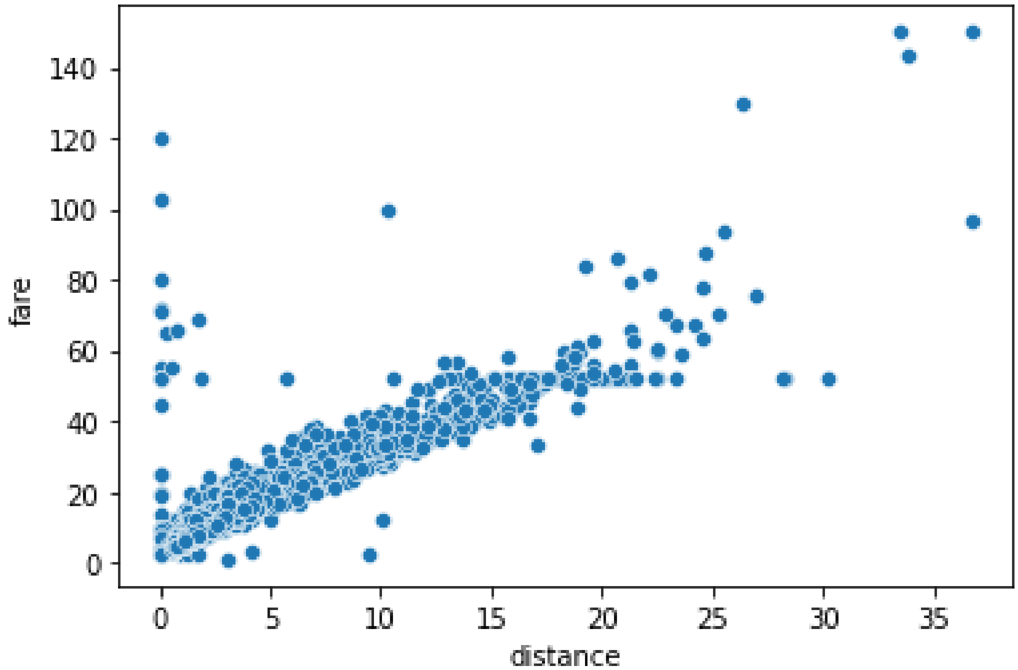
```
                  pickup              dropoff  passengers  distance  fare  \
0     2019-03-23 20:21:09  2019-03-23 20:27:24           1      1.60   7.0
1     2019-03-04 16:11:55  2019-03-04 16:19:00           1      0.79   5.0
2     2019-03-27 17:53:01  2019-03-27 18:00:25           1      1.37   7.5
3     2019-03-10 01:23:59  2019-03-10 01:49:51           1      7.70  27.0
4     2019-03-30 13:27:42  2019-03-30 13:37:14           3      2.16   9.0
…                     …                   …         …       …     …
6428  2019-03-31 09:51:53  2019-03-31 09:55:27           1      0.75   4.5
6429  2019-03-31 17:38:00  2019-03-31 18:34:23           1     18.74  58.0
6430  2019-03-23 22:55:18  2019-03-23 23:14:25           1      4.14  16.0
6431  2019-03-04 10:09:25  2019-03-04 10:14:29           1      1.12   6.0
6432  2019-03-13 19:31:22  2019-03-13 19:48:02           1      3.85  15.0

       tip  tolls  total   color      payment            pickup_zone  \
0     2.15    0.0  12.95  yellow  credit card        Lenox Hill West
1     0.00    0.0   9.30  yellow         cash  Upper West Side South
2     2.36    0.0  14.16  yellow  credit card          Alphabet City
3     6.15    0.0  36.95  yellow  credit card              Hudson Sq
4     1.10    0.0  13.40  yellow  credit card           Midtown East
…      …    …     …     …          …                    …
6428  1.06    0.0   6.36   green  credit card      East Harlem North
6429  0.00    0.0  58.80   green  credit card                Jamaica
6430  0.00    0.0  17.30   green         cash    Crown Heights North
6431  0.00    0.0   6.80   green  credit card          East New York
6432  3.36    0.0  20.16   green  credit card            Boerum Hill

                           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
…                                   …            …             …
6428                Central Harlem North      Manhattan       Manhattan
6429  East Concourse/Concourse Village         Queens           Bronx
6430                      Bushwick North       Brooklyn        Brooklyn
6431       East Flatbush/Remsen Village       Brooklyn        Brooklyn
6432                     Windsor Terrace       Brooklyn        Brooklyn

[6433 rows x 14 columns]
```

Visualization Rows and rows of numbers aren't that helpful.

seaborn makes visualization easy - here is a scatter plot of the data.

[5]:
```
sns.scatterplot(data=dft, x="distance", y="fare")
```

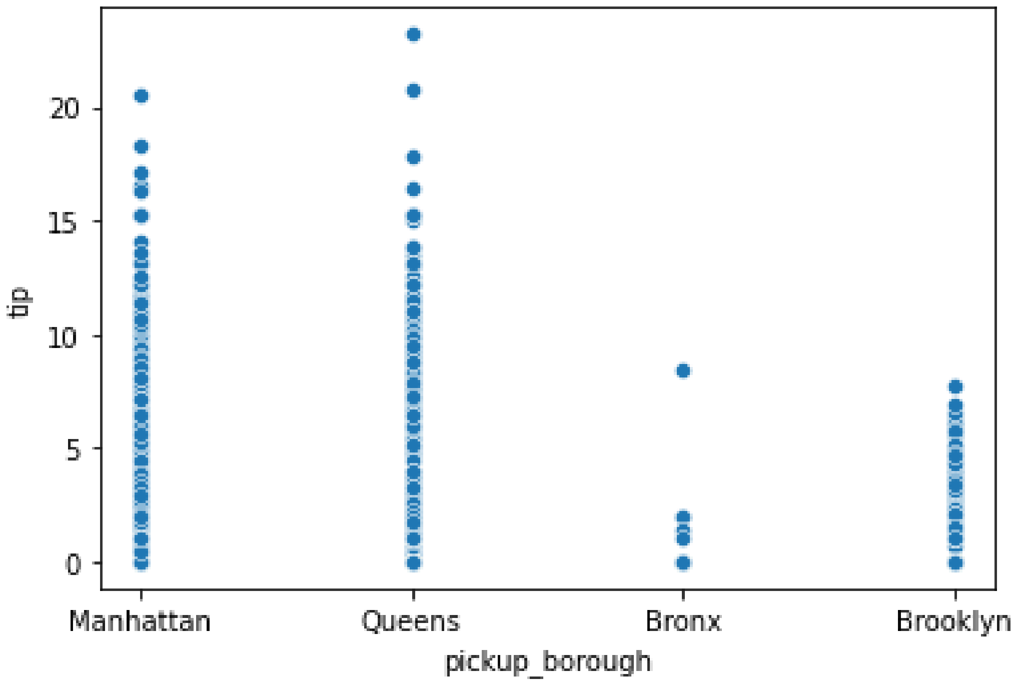
[5]:
```
<AxesSubplot:xlabel='distance', ylabel='fare'>
```



THINK ABOUT: it looks like fare is roughly proportional to distance. But what could cause the outliers?

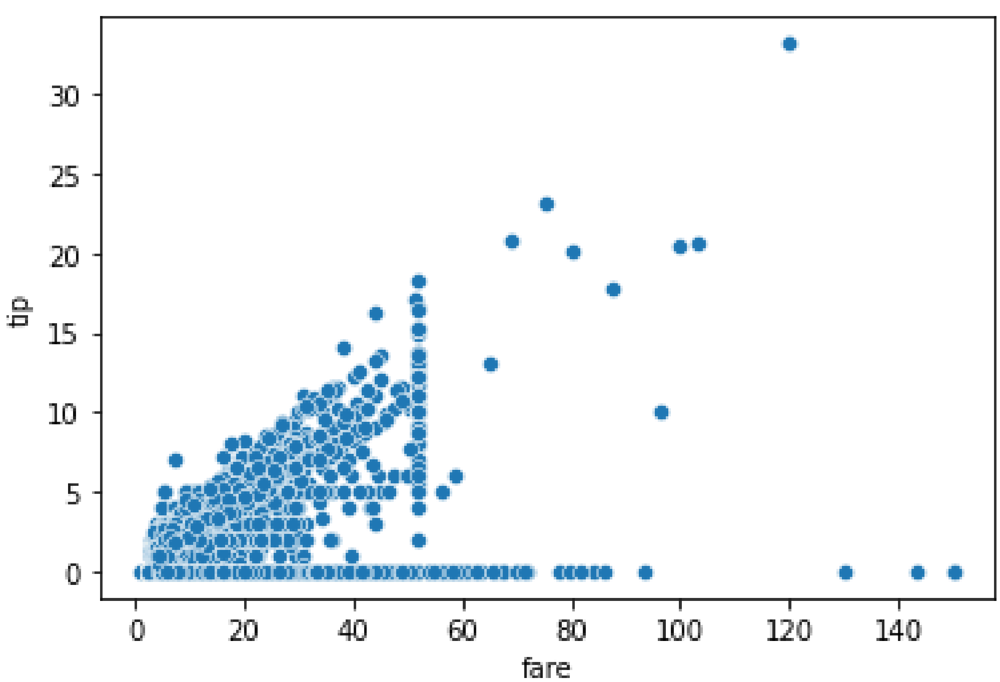
[6]:
```
# here's another example
sns.scatterplot(data=dft, x="pickup_borough", y="tip")
```

[6]: <AxesSubplot:xlabel='pickup_borough', ylabel='tip'>



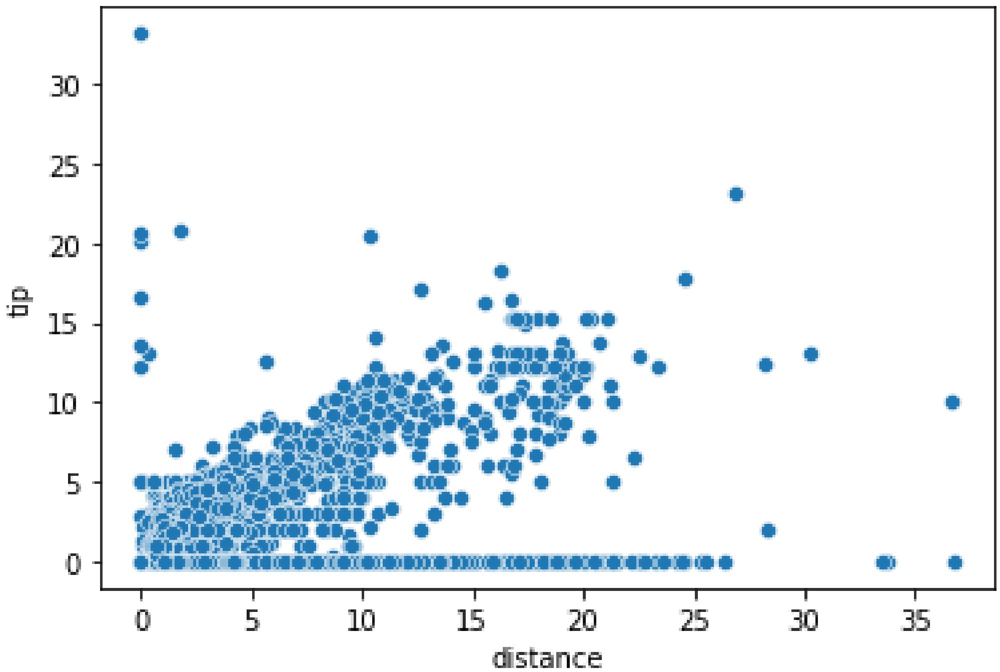
[7]:
```
# is the tip proportional to the fare?
sns.scatterplot(data=dft, x="fare", y="tip")
```

[7]: <AxesSubplot:xlabel='fare', ylabel='tip'>



```
[8]: # is the tip proportional to the distance?
     sns.scatterplot(data=dft, x="distance", y="tip")
```

```
[8]: <AxesSubplot:xlabel='distance', ylabel='tip'>
```



1.7.2 The tips data set

Let's look now at the `tips` data set. Along the way we'll see a few more ways we can use the data frame object

```
[9]: # load the data - dft: data frame tips
     # note that this overwrites the previous 'value/meaning' of dft
     dft = sns.load_dataset('tips')
     dft.head()
```

[9]:

	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

An extensive list of data frame methods/functions can be found here: https://pandas.pydata.

org/pandas-docs/stable/reference/api/pandas.DataFrame.html#pandas.DataFrame Let's look at some of them

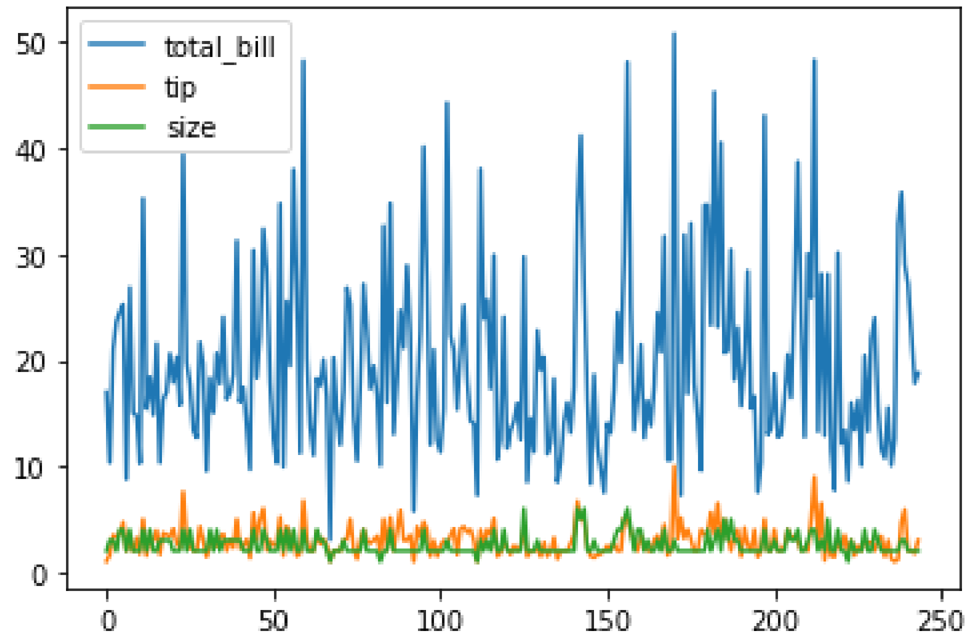
[10]:
```
print(dft.info)
print('The shape of the data frame is: ', dft.shape)
print('The size of the data frame is: ', dft.size)
print('Note that 244*7 =', 244*7)
```

```
<bound method DataFrame.info of      total_bill   tip     sex smoker   day
time  size
0         16.99  1.01  Female     No   Sun  Dinner     2
1         10.34  1.66    Male     No   Sun  Dinner     3
2         21.01  3.50    Male     No   Sun  Dinner     3
3         23.68  3.31    Male     No   Sun  Dinner     2
4         24.59  3.61  Female     No   Sun  Dinner     4
..          …     …       …      …     …       …     …
239       29.03  5.92    Male     No   Sat  Dinner     3
240       27.18  2.00  Female    Yes   Sat  Dinner     2
241       22.67  2.00    Male    Yes   Sat  Dinner     2
242       17.82  1.75    Male     No   Sat  Dinner     2
243       18.78  3.00  Female     No  Thur  Dinner     2

[244 rows x 7 columns]>
The shape of the data frame is:  (244, 7)
The size of the data frame is:  1708
Note that 244*7 = 1708
```

Visualization Again, numbers aren't always that helpful. Plots often give us more insight.

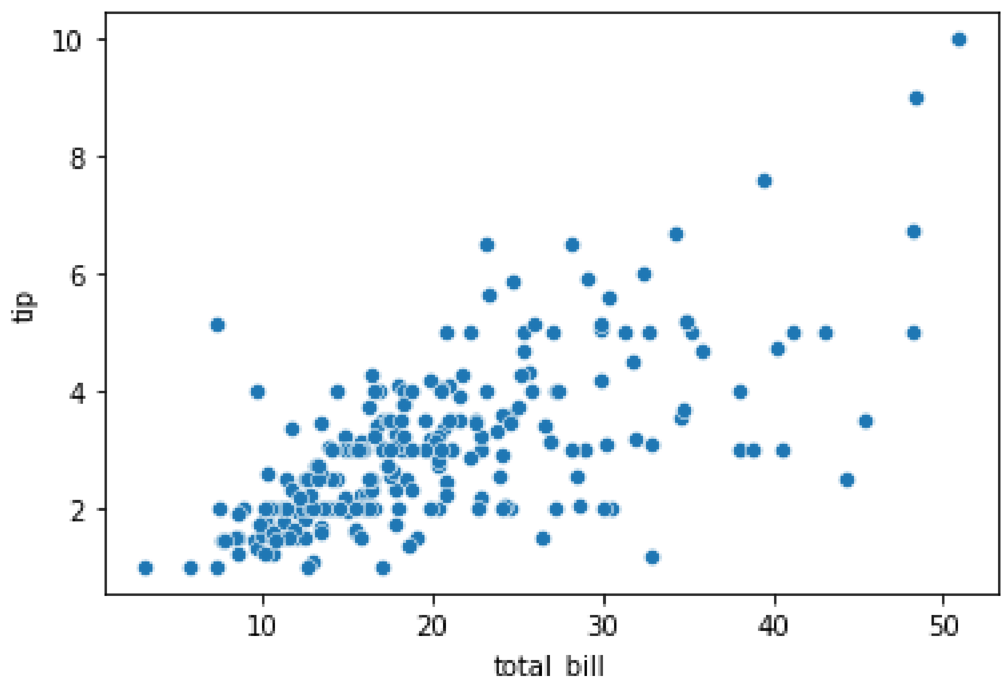
[11]:
```
dft.plot()
```

[11]: <AxesSubplot:>



[12]:
```
sns.scatterplot(data=dft, x="total_bill", y="tip")
```

[12]: <AxesSubplot:xlabel='total_bill', ylabel='tip'>



Statistics and Probability You're assumed to be familiar with basic terms and concepts in these areas, but we will revise and review those that we need later.

We can get some basic stats for our data set with the `describe()` method…

[13]:
```
# here are some descriptive statistics
dft.describe()
```

[13]:

	total_bill	tip	size
count	244.000000	244.000000	244.000000
mean	19.785943	2.998279	2.569672
std	8.902412	1.383638	0.951100
min	3.070000	1.000000	1.000000
25%	13.347500	2.000000	2.000000
50%	17.795000	2.900000	2.000000
75%	24.127500	3.562500	3.000000
max	50.810000	10.000000	6.000000

1.7.3 The anscombe data set

This is pretty famous. There are four sets of 11 coordinate pairs. When plotted they look completely different. But they have the same summary statistics (at least the common ones).

See https://en.wikipedia.org/wiki/Anscombe%27s_quartet

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

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

[14]:
```
dfa = sns.load_dataset('anscombe')
# look at how we get an apostrophe in the string...
print("The size of Anscombe's data set is:", dfa.shape)
```

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

[15]:
```
dfa.head()
```

[15]:

	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

[16]:
```
dfa.tail()
```

[16]:

	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…

[17]:
```python
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 and the `head` and `tail` output above we can infer 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:

[18]:
```
dfa.dataset.unique()
```

[18]:
```
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:

[19]:
```
dfa.dataset.unique().shape
```

[19]:
```
(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'll come to that later when we discuss `numpy` (Numerical python: https://numpy.org).

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.

For example, using the hints here https://stackoverflow.com/questions/17071871/how-do-i-select-rows-from-a-dataframe-based-on-column-values, to get the data for the sub-data-set `I` we can do this:

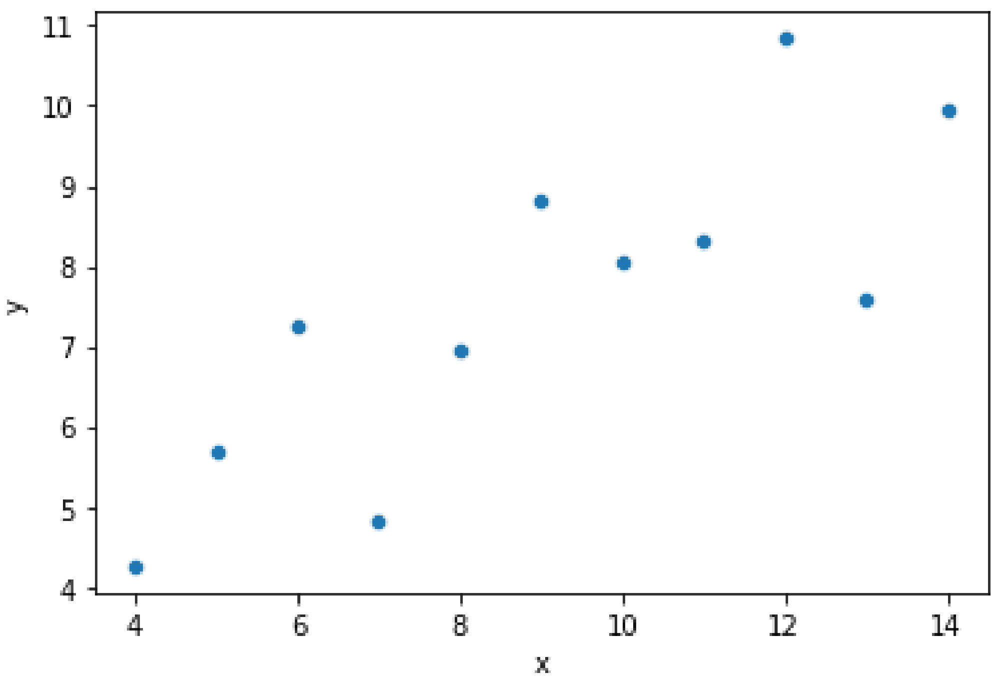
[20]:
```
dfa.loc[dfa['dataset'] == 'I']
```

[20]:
```
    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.

[21]:
```
sns.scatterplot(data=dfa.loc[dfa['dataset'] == 'I'], x="x", y="y")
```

[21]: <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.

[22]:
```
dfa1 = dfa.loc[dfa['dataset'] == 'I']
dfa2 = dfa.loc[dfa['dataset'] == 'II']
dfa3 = dfa.loc[dfa['dataset'] == 'III']
```

```
dfa4 = dfa.loc[dfa['dataset'] == 'IV']
```

Now let's look at each of the four data sets in a scatter plot, and use the `describe` method to examine the summary statistics.

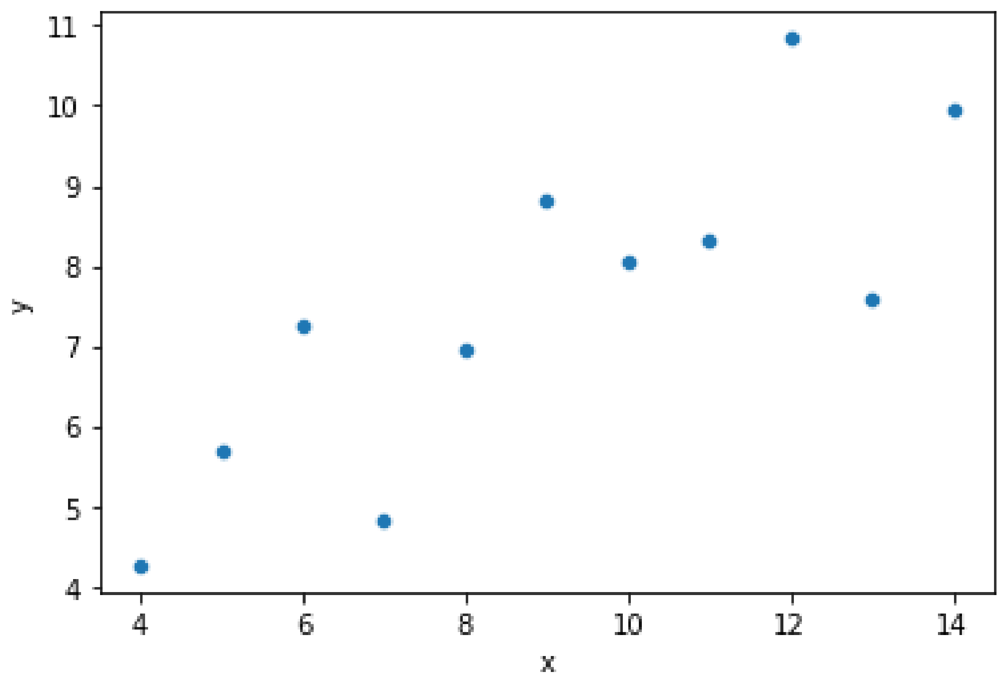
The outcome is quite surprising…

dataset 1

[23]:
```
sns.scatterplot(data=dfa1, x="x", y="y")
dfa1.describe()
```

[23]:

	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

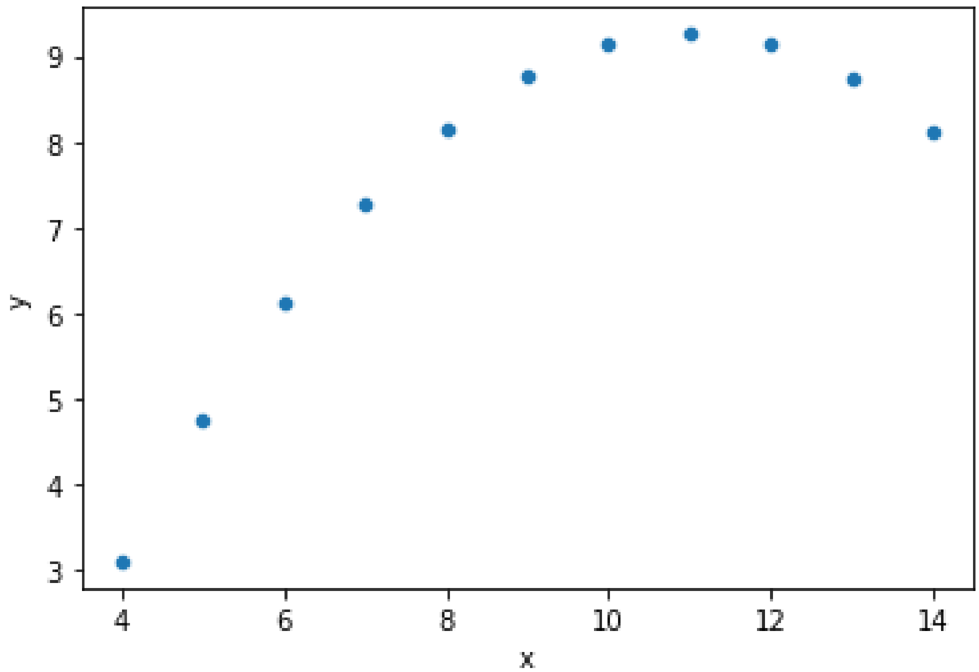


dataset 2

[24]:
```
sns.scatterplot(data=dfa2, x="x", y="y")
dfa2.describe()
```

[24]:

	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

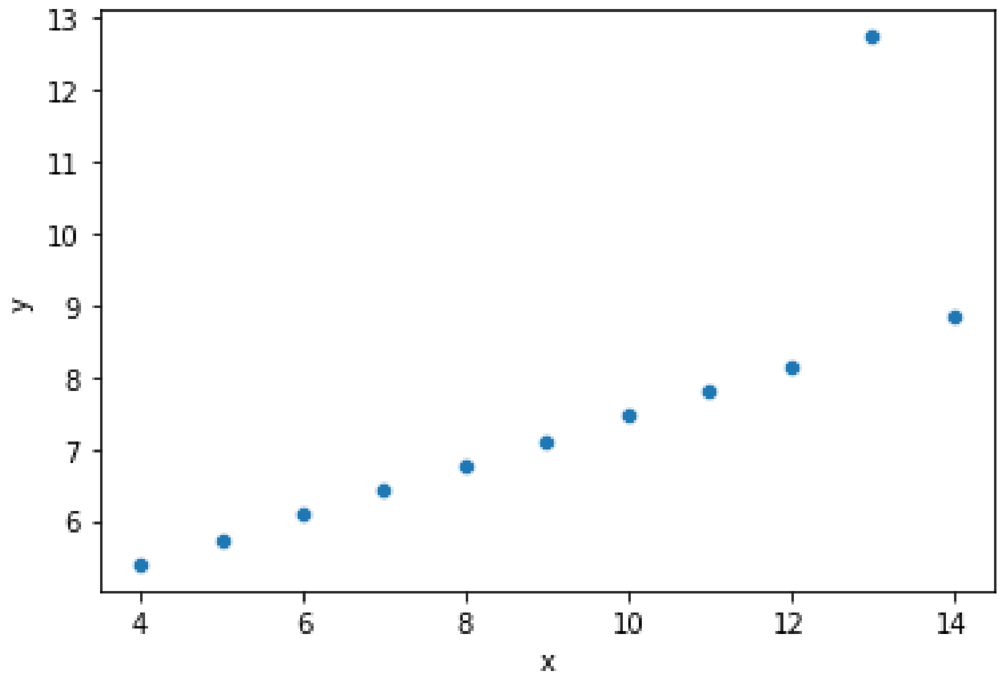


dataset 3

[25]:
```
sns.scatterplot(data=dfa3, x="x", y="y")
dfa3.describe()
```

[25]:

	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

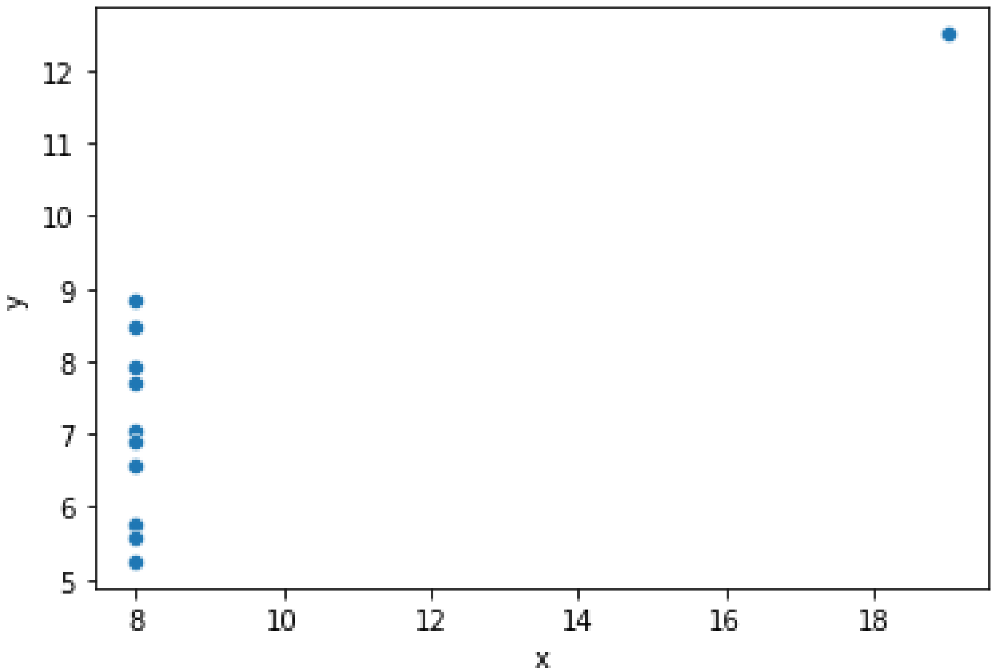


dataset 4

[26]:
```
sns.scatterplot(data=dfa4, x="x", y="y")
dfa4.describe()
```

[26]:

	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



1.8 Exercises

For the `taxis` data set:

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

```
1: sns.scatterplot(data=ds, x="dropoff_borough", y="tip")
2: sns.scatterplot(data=ds, x="distance", y="tip")
```

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

```
1: ds.describe()
2: sns.scatterplot(data=ds, x="total_bill", y="tip")
3: sns.scatterplot(data=ds, x="day", y="total_bill")
4: sns.scatterplot(data=ds, x="sex", y="tip")
```

1.9 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
- Set `OUTPUTTING=1` above
- Clear and fresh run of entire notebook
- Create html slide show:
  - `jupyter nbconvert --to slides 1_intro.ipynb`
- Clear all cell output
- Set `OUTPUTTING=0` above
- Save
- git add, commit and push to FML
- copy PDF, HTML etc to web site
  - git add, commit and push
- rebuild binder

Ignore this - it is done in 2_vectors

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? How many attributes does each have?
2. Scatter plot price against carat.

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

Some of this originated from

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

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

[27]:
```
%%bash
NBROOTNAME='1_intro'
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
```

```
[NbConvertApp] Converting notebook 1_intro.ipynb to html
[NbConvertApp] Writing 853220 bytes to 1_intro.html
[NbConvertApp] Converting notebook 1_intro.ipynb to pdf
[NbConvertApp] Support files will be in 1_intro_files/
[NbConvertApp] Making directory ./1_intro_files
[NbConvertApp] Making directory ./1_intro_files
[NbConvertApp] Making directory ./1_intro_files
[NbConvertApp] Making directory ./1_intro_files
[NbConvertApp] Making directory ./1_intro_files
[NbConvertApp] Making directory ./1_intro_files
[NbConvertApp] Making directory ./1_intro_files
[NbConvertApp] Making directory ./1_intro_files
[NbConvertApp] Making directory ./1_intro_files
[NbConvertApp] Making directory ./1_intro_files
[NbConvertApp] Making directory ./1_intro_files
[NbConvertApp] Writing 76939 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 241941 bytes to 1_intro.pdf
[NbConvertApp] Converting notebook 1_intro.ipynb to script
[NbConvertApp] Writing 24366 bytes to 1_intro.py
```

[ ]:

```
```