Your grade: 83.33%

Your latest: **83.33%** • Your highest: **83.33%** • To pass you need at least 80%. We keep your highest score.

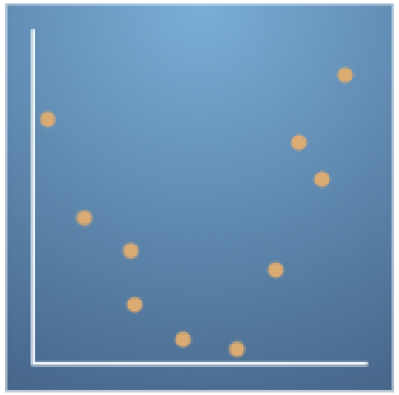
Next item →

1. In the previous video you saw how to fit a line $y = mx + c$ to linear data. In this quiz you will practise identifying data that is appropriate for linear regression, and initialise some fits yourself.

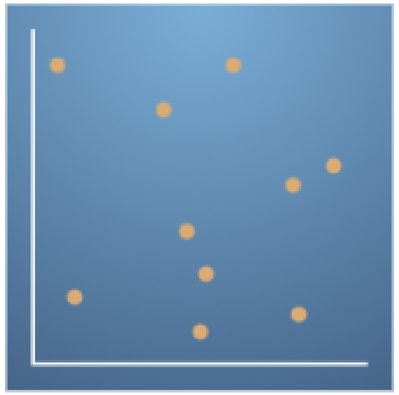
1 / 1 point

Which of the following figures looks like it contains sensible data for a linear fit?

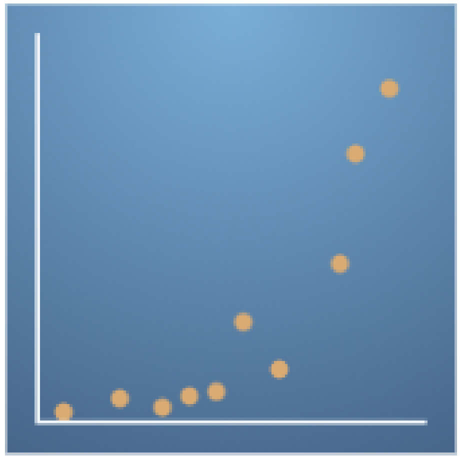
○



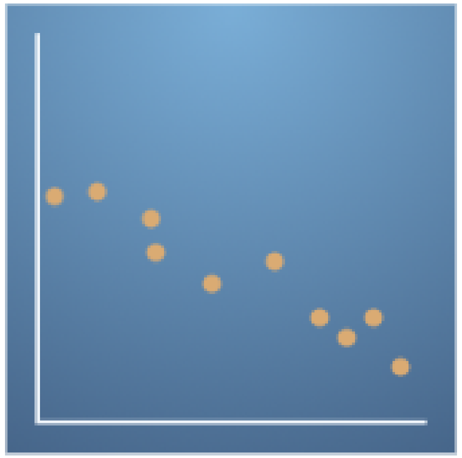
○



○



[✓ ●]



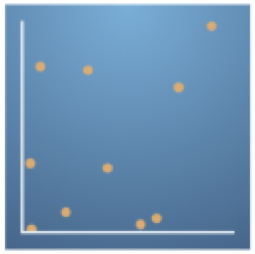
Correct

These data look linear, with negative gradient m

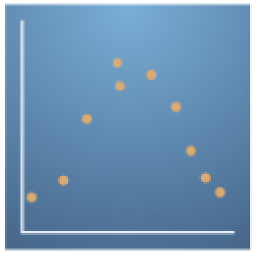
2. Which of the following figures looks like it contains sensible data for a linear fit?

1 / 1 point

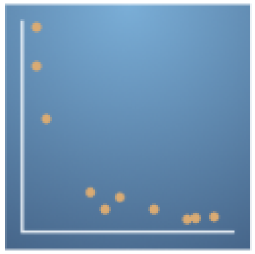
○



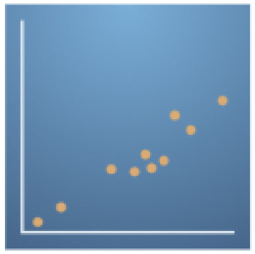
○



○



[✓]



Correct

3. Now that we've identified candidates for linear regression we can do some linear fitting ourselves. The code block below plots some predefined data points and a linear regression with the values $[m, c]$, where m is the gradient and c is the y-intercept. It also gives the χ^2 value discussed in the previous video, which is a measure of how good the fit is.

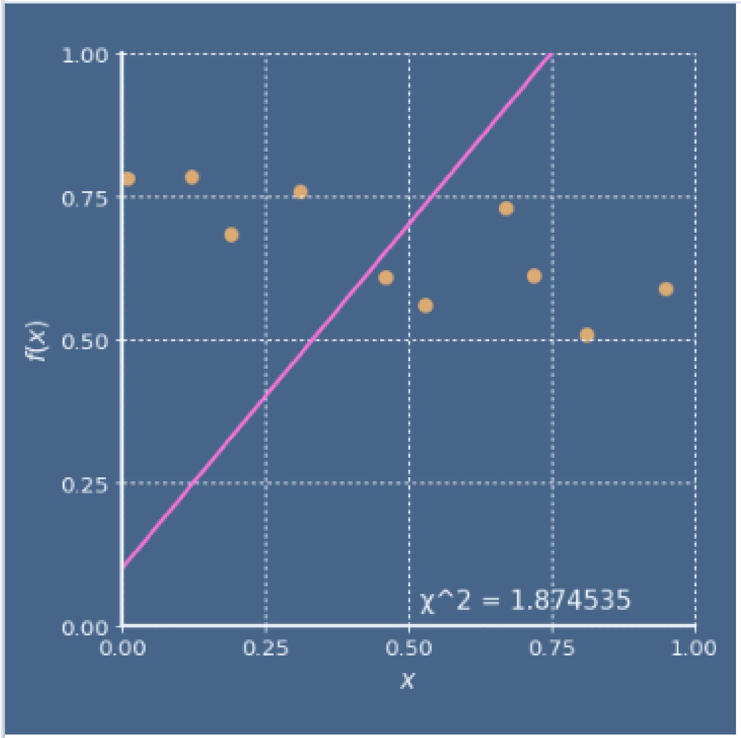
1 point

Play with the values of m and c to get a sense for how different linear fits affect χ^2, then try to find the best possible fit to the data.

```
# See what m and c do to the fit
m = 1.2 ; c = 0.1
p = [m,c]
line(p)
```

Run

Reset



The minimum χ^2 value is 0.03819 to 4 significant figures. Try to find a fit with $\chi^2 \leq 0.04$ and then input these values into the following code block:

```
# Replace m and c with values that minimise χ^2.
p = [1.2, 0.1]
```

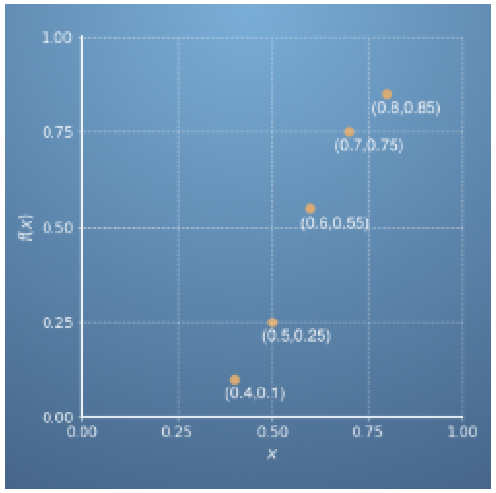
Run

Reset

4. Fitting by eye is not that easy even for small sets of data. Let's make some linear fits using the maths discussed in the previous video.

1 / 1 point

The following is a figure with 5 data points, labelled with their (x, y) coordinates:



Let's fit a linear regression to this small sample by hand. Recall that we can use χ^2 to measure how good our fit is, defined by

$$\chi^2 = \sum(y_i - mx_i - c)^2,$$

and that we can find the minimum of χ^2 by differentiating it and setting it to zero. This leads us to the equations for m and c,

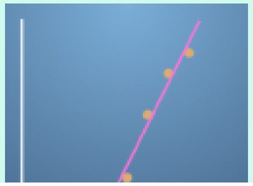
$$m = \frac{\sum(x_i - \bar{x})y_i}{\sum(x_i - \bar{x})^2}, \qquad c = \bar{y} - m\bar{x},$$

which minimise χ^2.

Use these equations to calculate the m and c which minimise χ^2 for the 5 data points given above and select the correct values below:

- ○ $m = -1.8, c = 0.7$
- ○ $m = 1.8, c = -0.6$
- ○ $m = 1.6, c = -0.5$
- [✓ $m = 2, c = -0.7$]

Correct
Here's the fit:



5. As you have seen it can be quite a lot of effort to fit even 5 data points when doing the maths by hand. Often it's necessary to work with much larger data sets, so let's consider a new example with 50 data points. Instead of doing it by hand we'll implement a function to do the maths for us.

1 / 1 point

Run the following code block first to see the data without any kind of linear fit. The function *linfit* is being defined inside the code block. Your task is to edit the definition so that *linfit* takes the array of x data, *xdat*, and the array of y data, *ydat*, and returns the correct m and c to create a linear fit which minimises χ^2.

The calculation for $\bar{x}$, *xbar*, and $\bar{y}$, *ybar*, is already given. As you can see *numpy* has been imported as *np*.

```python
# Here the function is defined
def linfit(xdat,ydat):
  # Here xbar and ybar are calculated
  xbar = np.sum(xdat)/len(xdat)
  ybar = np.sum(ydat)/len(ydat)
  m= np.sum((xdat-xbar)*ydat)/np.sum((xdat-xbar)**2)
  c = ybar - m*xbar
  # Insert calculation of m and c here. If nothing is here the data will be plotted with no lin

  # Return your values as [m, c]
  return [m, c]

# Produce the plot - don't put this in the next code block
line()
```

Run

Reset

Use the above code block to test your code. When you are confident that you have correctly defined the function, put it into the next codeblock and run it, being careful not to include *line()* in your answer.

```python
# Here the function is defined
def linfit(xdat,ydat):
  # Here xbar and ybar are calculated
  xbar = np.sum(xdat)/len(xdat)
  ybar = np.sum(ydat)/len(ydat)

  # Insert calculation of m and c below
  m = np.sum((xdat - xbar)*ydat)/np.sum((xdat - xbar)**2)
  c = ybar - m*xbar
  # Return your values as [m, c]
  return [m, c]

# Don't include line() in this answer box
```

Run

Reset

Correct

Well done, you have correctly found the values of m and c

6. While it is informative to write the code ourselves, as in the previous question, in practice functions which do various types of regression are implemented in lots of programming languages. There are several of these in python.

1 / 1 point

One such example is the *scipy.stats.linregress()* method, which takes arrays of x data and y data in exactly the same way as the *linfit()* function you defined in the previous question. As an output it gives the slope m and intercept c as well as a few useful statistical measures like the standard error.

In the following code block, the x data is again stored in the *xdat* array, and the y data in the *ydat* array. Call the method *stats.linregress()* with the data arguments, and then pass the output to *line()* to plot the regression.

```python
from scipy import stats

# Use the stats.linregress() method to evaluate regression
regression = stats.linregress(xdat,ydat)

line(regression)
```

Run

Reset

Hopefully it is clear that *linregress()* does everything *linfit()* did and more, without having to write it yourself!

Once you're happy that you've implemented things correctly in the above code block, repeat the same in the following code block without the last line to complete the question.

```python
from scipy import stats

# Use the stats.linregress() method to evaluate regression
regression = stats.linregress(xdat, ydat)

# Don't use line(regression) in this code box
```

Run

Reset

Correct

Well done,

you have assigned 'regression' correctly