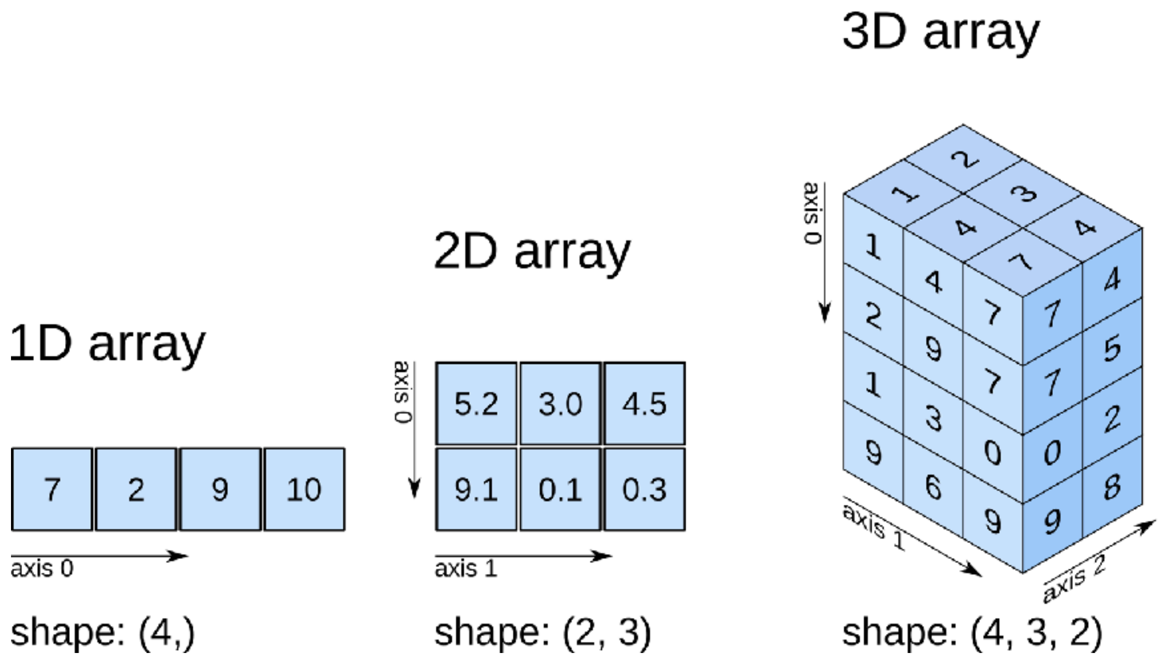
Digital World (2019)

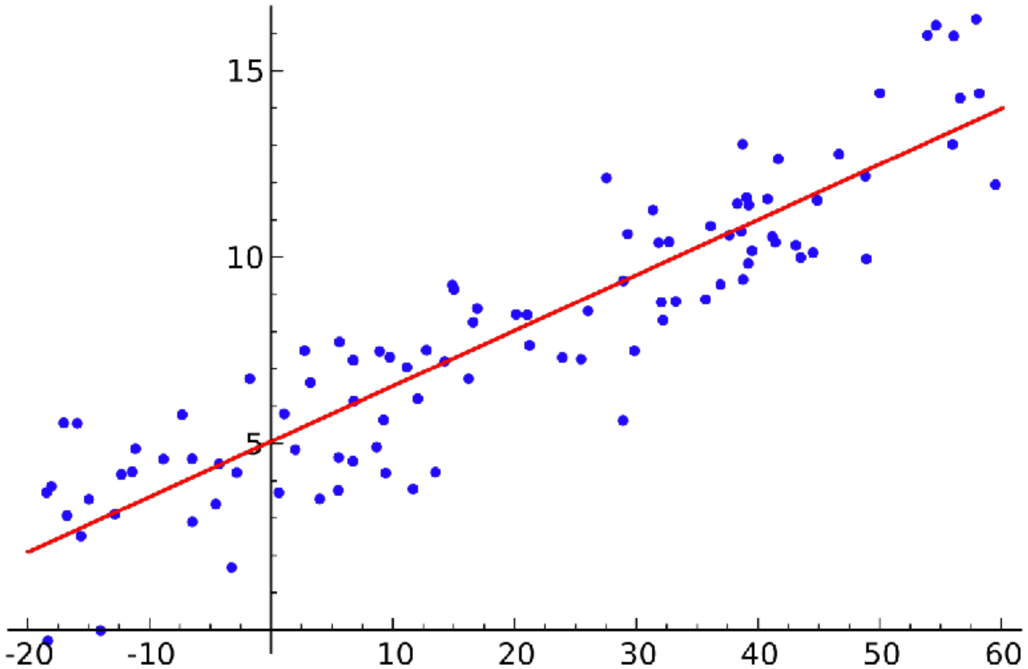
Week 10, S1: Linear Regression

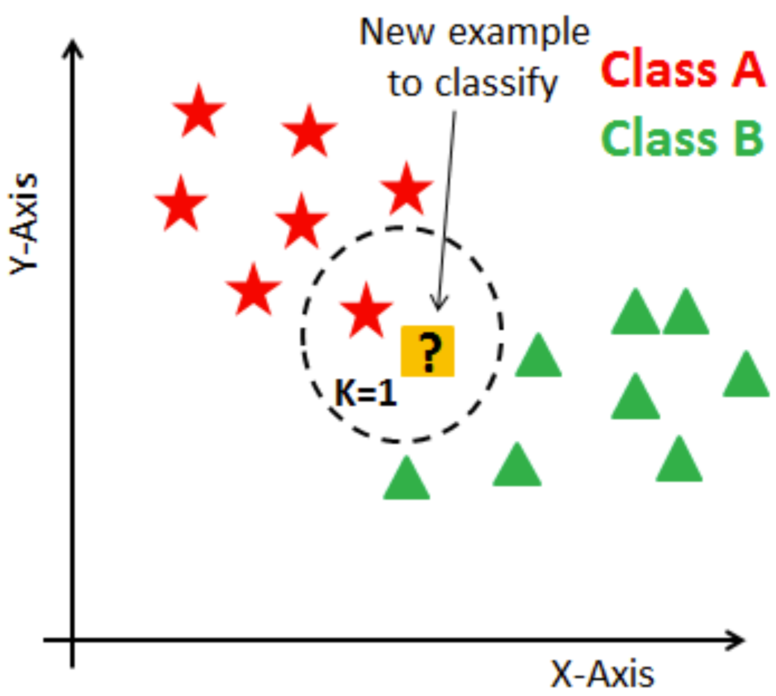
Chris Poskitt

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN



This week — *Data Analysis and Prediction*





NumPy Arrays

Refresher: how we *used* to do matrices

```
M = [ [0, 0, 0, 1, 0],
      [0, 0, 0, 0, 0],
      [0, 2, 0, 0, 0],
      [0, 0, 0, 0, 0],
      [0, 0, 0, 3, 0] ]
```

M =

$(0, 3) \longrightarrow 1$

$(2, 1) \longrightarrow 2$

$(4, 3) \longrightarrow 3$

*how would we slice **row 3**?*

*how would we slice **column 3**? (🤮)*

Matrices in NumPy

- matrices are represented as 2-dimensional array objects

  => ***M = np.array([[…], …])***

- equipped with several powerful and efficient methods

  => *M.sum(), M.T, np.sqrt(M), np.add(M_1, M_2), …*

- in general, an ndarray object can be N-dimensional

  => *M.shape returns the length of each dimension*

- 1D arrays are like lists: no rows/cols; just ordered elements

A realistic dataset for practice

- sklearn.datasets provides some datasets for practice

- we will use the Wisconsin breast cancer dataset

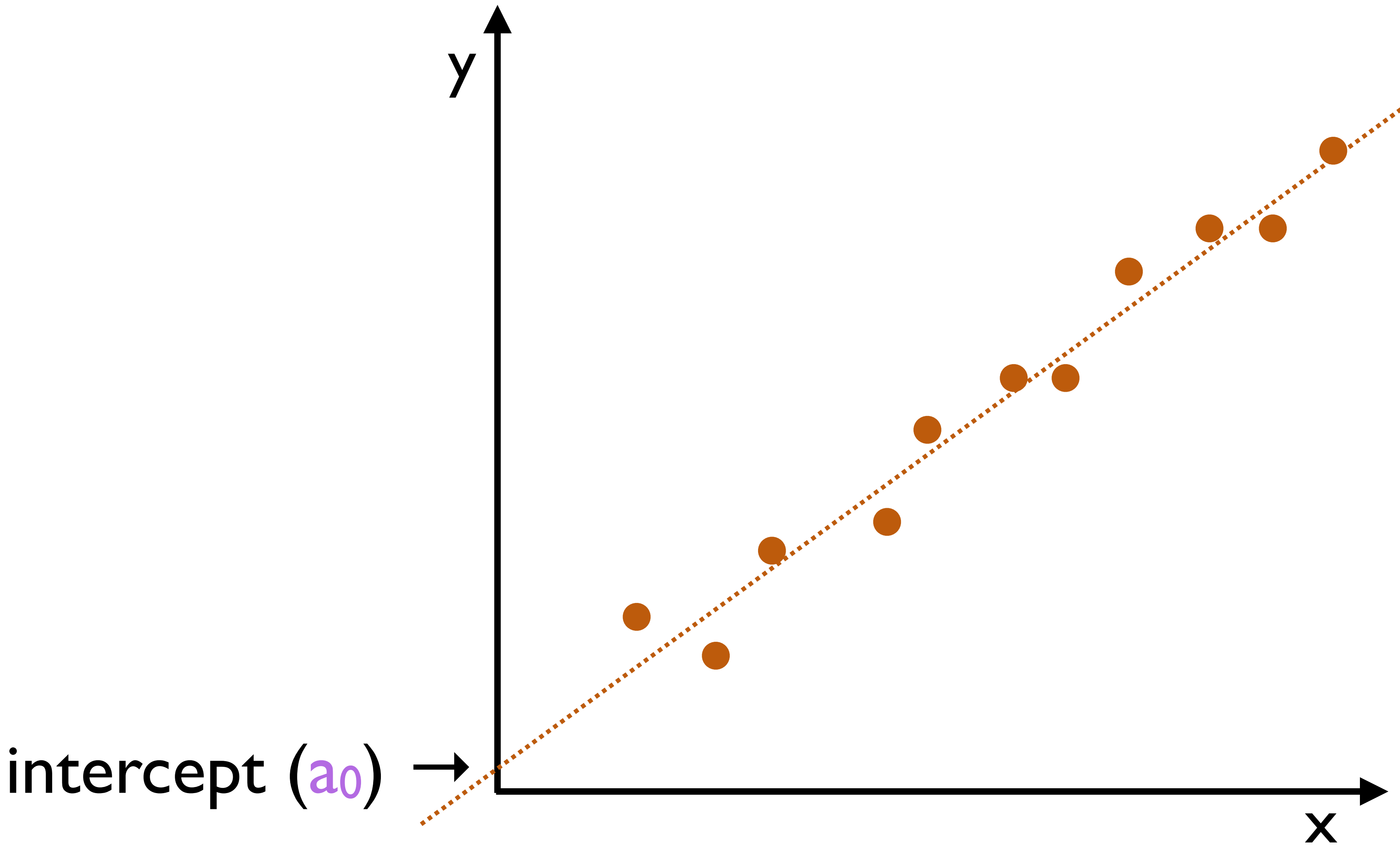
- several features of different malignant/benign masses

  => *radius, texture, perimeter, area, smoothness, …*

- we will load a dictionary-like "bunch object", with attributes such as data, feature_names, target, …

Linear Regression

Linear regression: *line of best fit*



$$y = a_0 + a_1 x$$

Implementing linear regression

- we can train a linear regression model using sklearn

    => *in particular, the **sklearn.linear_model** module*

- first, split our data into training and testing sets

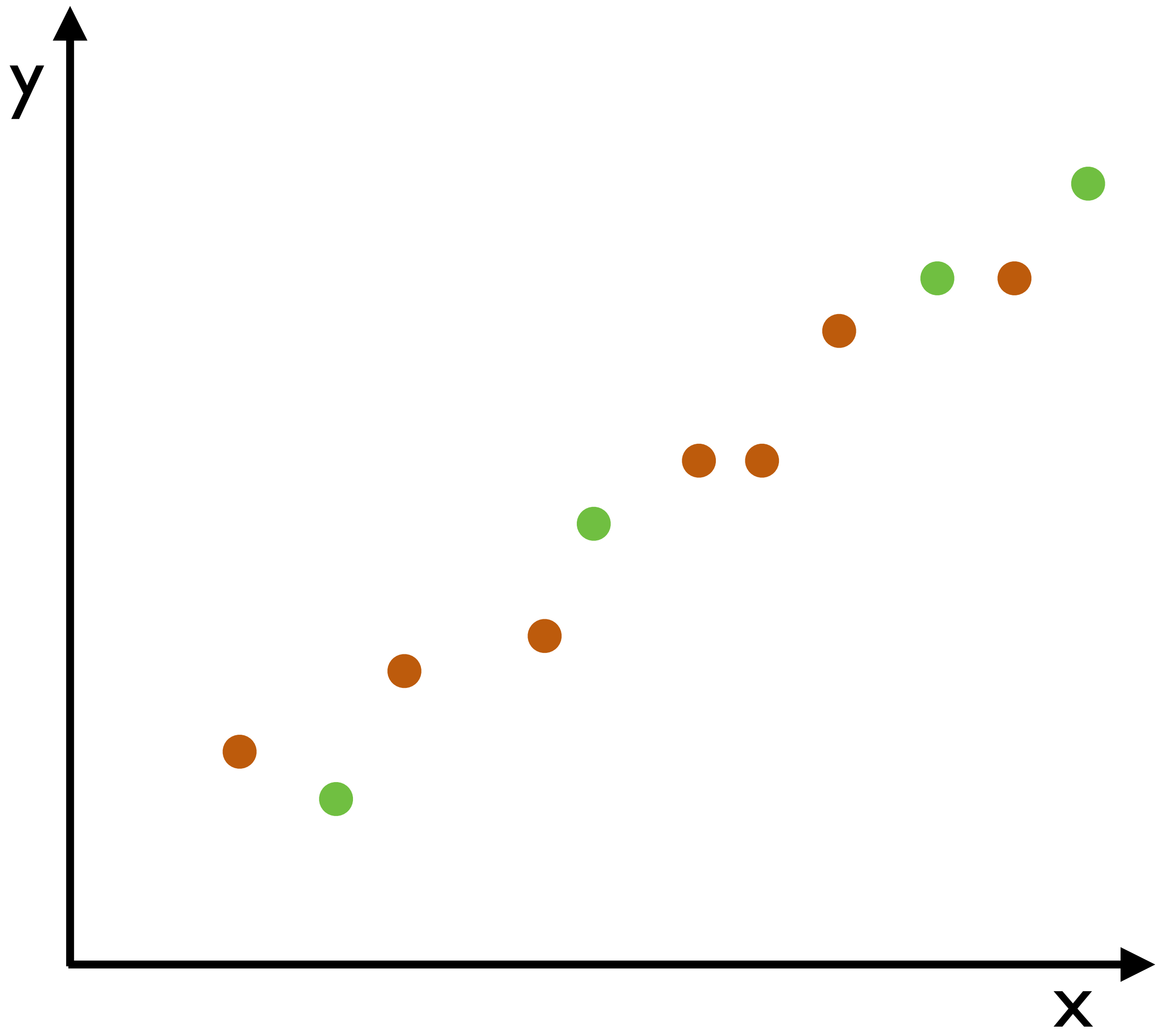
    => *typically a 60% : 40% split*
    => *use the **train_test_split()** function to do so*

- then, use the fit method on the training data

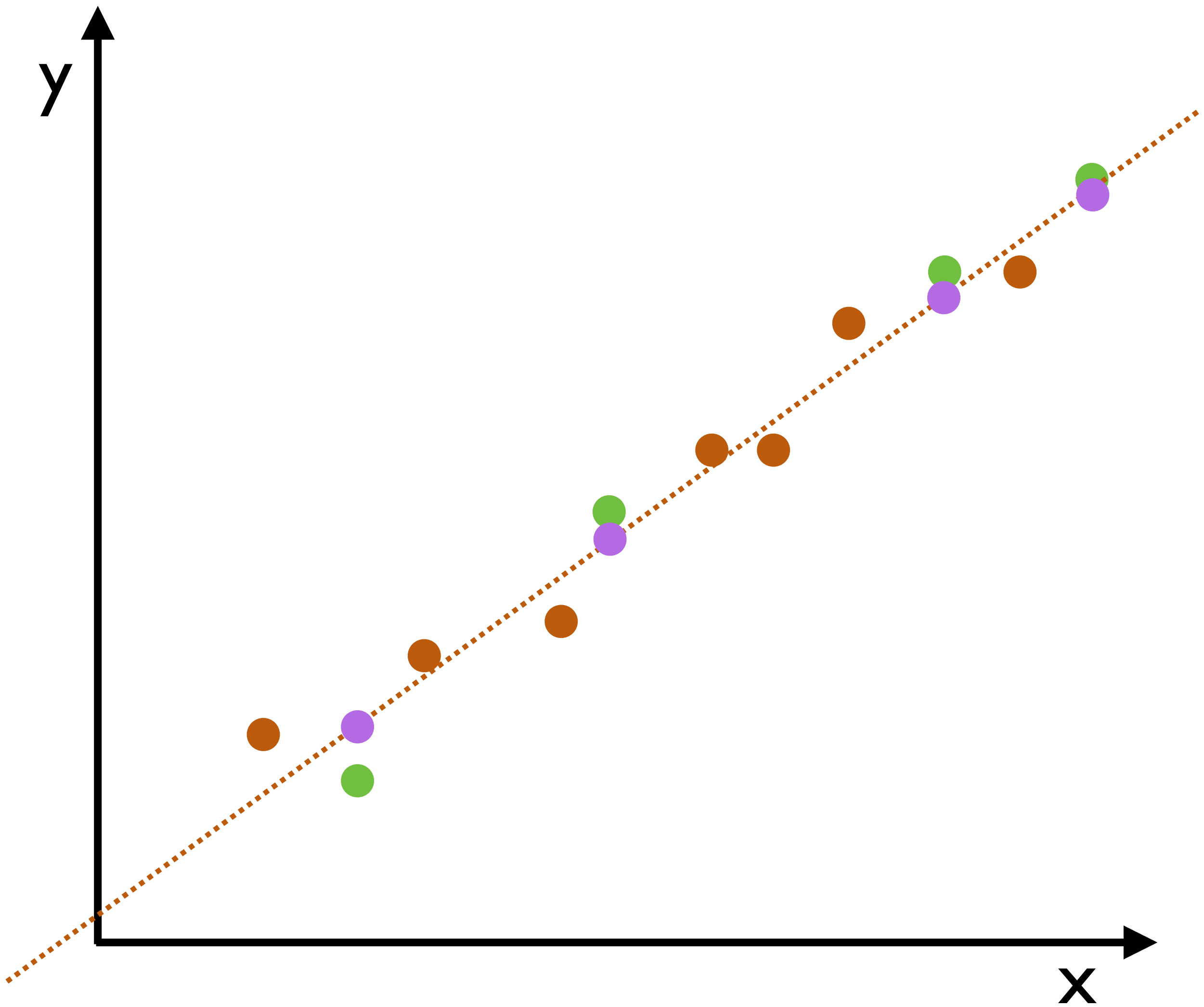
```
model = linear_model.LinearRegression()
model.fit(x_train, y_train)
```

- the model object stores the coefficient and intercept as attributes — *see the documentation!*

Training / testing split



Applying model.predict(x_test)



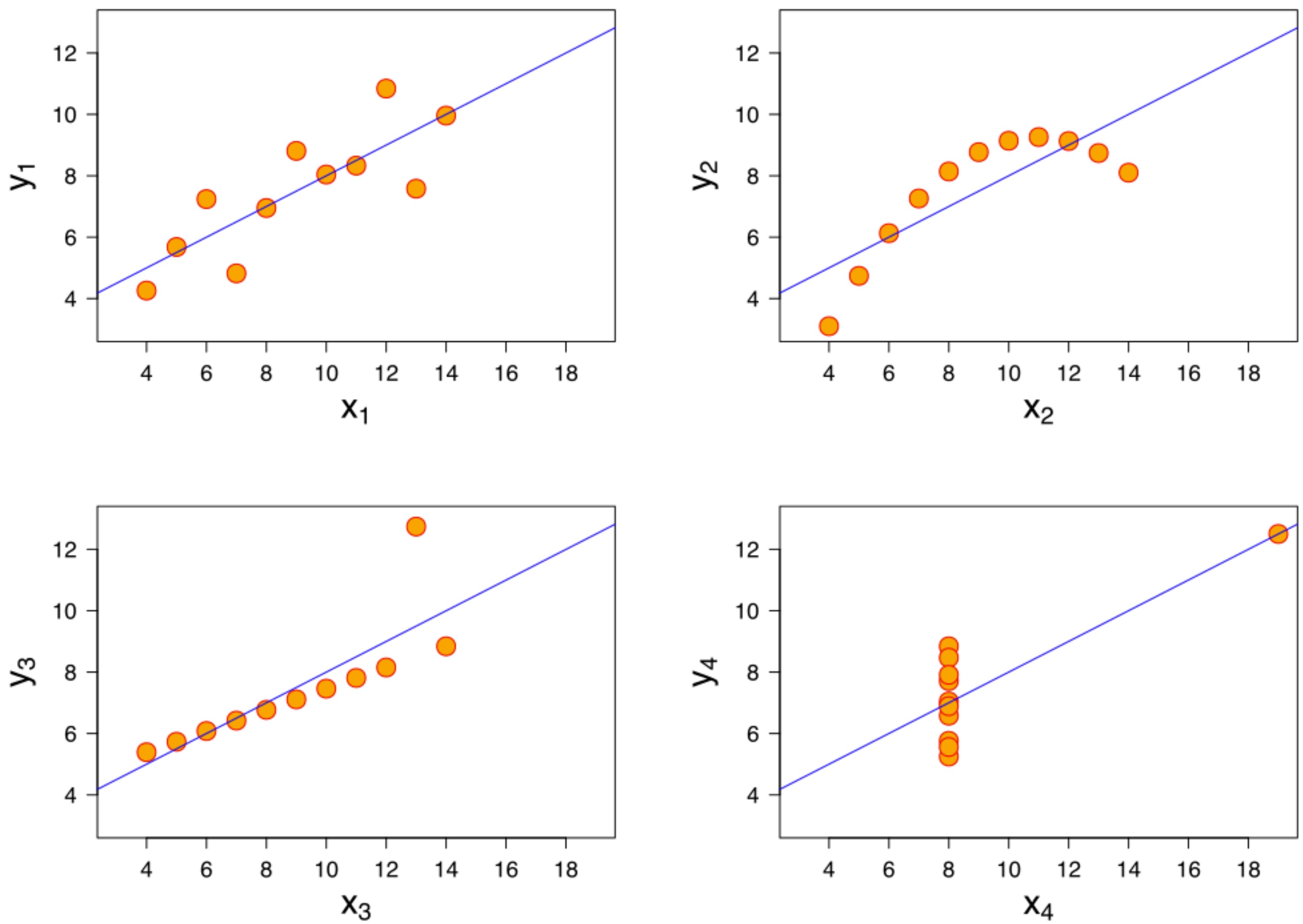
y = model.coef_ + model.intercept_x

Is your model any good?

- from the x test values predict the y values
- compare them against what the *actual* y values were
- there are different metrics we can then apply:

=> ***MSE:*** *mean of the squared errors (actual vs. predicted)*
=> ***R2:*** *measure of how well actual outcomes are predicted by the model ("% of variance accounted for")*

Anscombe's quartet



lesson: don’t rely *only* on your favourite metric!

Today: questions CS5 and CS6 only

(we'll do CS1-4, 7 on Tuesday and Thursday)

Polynomial regression

- often a simple linear model is not enough; we need:

$$y = a_0 + a_1x + a_2x^2 + a_3x^3 + \dots$$

- train a polynomial regression model

  => *special case of multiple linear regression*

- idea: take the original x data, and compute x^2, x^3, …

  => *take x^2, x^3, … as **new features** to train on*

- use PolynomialFeatures from sklearn.preprocessing

```
poly = PolynomialFeatures(order, include_bias=False)
poly_data = poly.fit_transform(x_data)
```

Summary

- NumPy provides powerful and efficient operations for analysing N-dimensional arrays

- sklearn can be used to train linear models

- metrics and plotting complement each other

  => *metrics alone can mislead — see Anscombe's quartet!*

- polynomial regression can take into account higher orders

  => *…but don't overfit!*