Probability and ML

PHY3287 - Computational Astronomy
Dr Andrea DeMarco
andrea.demarco@um.edu.mt

What is Probability?

- Similar to logic - but where uncertainty comes in.
- Logic: ‘If something flies, then it is a bird” + “If something is a bird, then it is delicious” = “if something flies, then it is delicious”
- But logic cannot help in situations where not all that flies is a bird, or where not all birds are delicious
- Probability: “if something flies, there is a 90% chance it is a bird” and “if it is a bird, there is an 80% chance it is delicious, otherwise disgusting”
- Therefore we can infer that there is a 72% chance that something that flies is delicious - something we cannot do with logic

Frequentist vs Bayesian

- There are two interpretations of probability - this will not be too important for us
- Frankly, most ML practitioners use both approaches in different situations
- But an interesting discussion on the subject can be found here: https://www.youtube.com/watch?v=KhAUfqhLakw

Probability: The Why

- Why do we need probability in Astronomy? Or science in general? A lot of modern computational astronomy revolves around the analysis of large amounts of data and Machine Learning

- Very few things are certain, so AI systems reason with probabilistic rules

Probability: The Why

- We often need to analyse ML algorithms probabilistically.
- E.g. when evaluating a classification model, we may refer to its accuracy i.e. the probability that the model will give the right answer given an example
- What is the probability that my algorithm will have accuracy X if I give it N examples to train on?

Probability: The Why

- A crucial reason: logic is discrete, and probability is continuous
- We may be certain about the answer but it is still important to be given output in terms of probability
- If our classifier was ‘certain’ and incorrect, we would not know what needs to be tweaked to make it output a correct answer

Sources of Uncertainty

- "Real" stochasticity - a fancy word for randomness. The world just flips a coin and we have to deal with that e.g. Guess the next word: "Malta is a …" - can you predict the next word?
- Inherent stochasticity vs Incomplete Observability: if somebody flips a coin is the answer random because of some quantum process, or only because we do not have information on speed, density of molecules etc.?
- For most applications - we will not care.

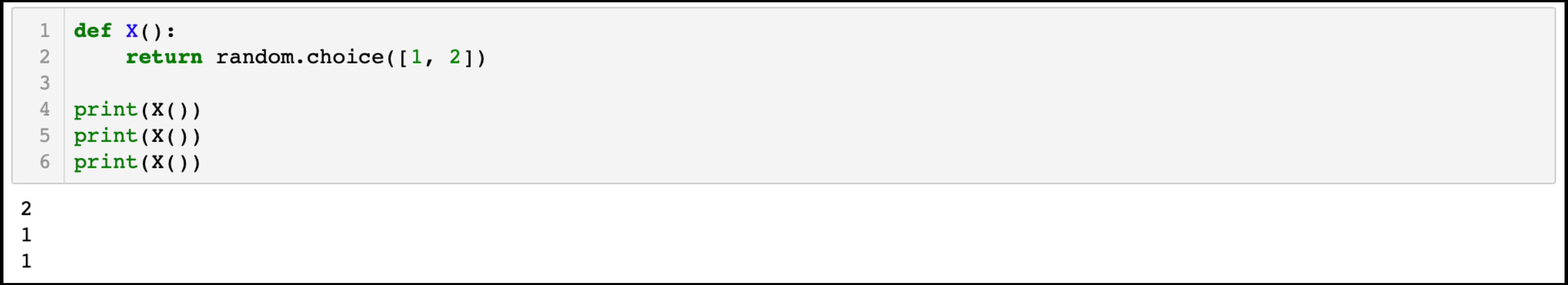
Sources of Uncertainty

- Incomplete modelling - sometimes you just cannot model everything
- Despite progress in ML, computers still can’t fully understand certain concepts (like interpretation of images)
- We can be 100% certain of many things where a machine is not
- So most of our algorithms will always rely on probabilistic output
- On the other hand, we often decide to not model all the complexity required to be able to achieve certainty - we are comfortable with being wrong some of the time
- Knowing that “most birds can fly” > “all birds can fly except sick, injured, young birds as well as kiwis, ostriches, emus, etc.”

Random Variables

- A random variable is a ‘variable’ whose value changes randomly

- Computationally, think more of this as a function which takes no argument, and returns a random value e.g. a random variable X:



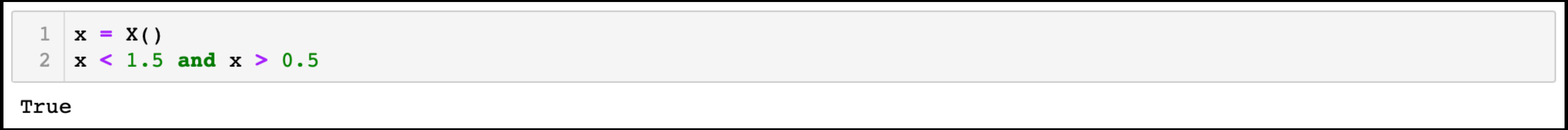
```python
def X():
    return random.choice([1, 2])

print(X())
print(X())
print(X())
```

```
2
1
1
```

Random Variables

- Caveat: if the random variable is used as an actual numerical value (derived from the function), it is not re-evaluated each time

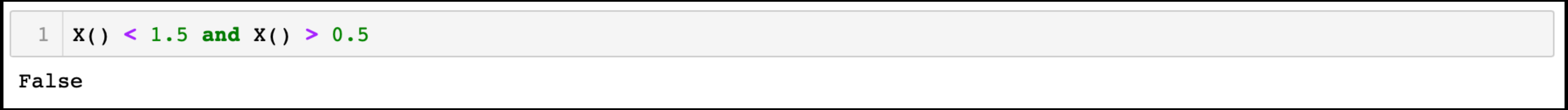
- Do this:



```
x = X()
x < 1.5 and x > 0.5
```

```
True
```

- Not this:



```
X() < 1.5 and X() > 0.5
```

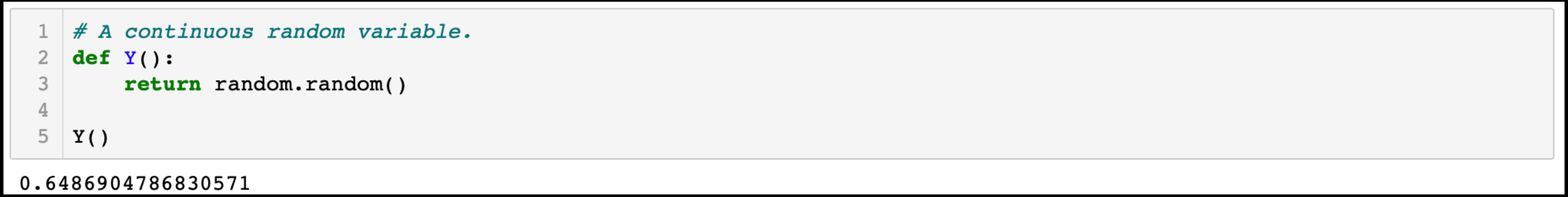
```
False
```

Random Variables

- RVs can be **continuous** or **discrete**
- Discrete: can take on only disconnected values. Between any two numbers that it can return, there are infinitely many more that it cannot return
    - e.g. [1,2] or [1,1.5,4,6,…]
- Continuous: can take on any number in one or many intervals
    - e.g. [0,1] can take on 0.03920, 0.23, etc.

Random Variables

- Discrete RVs can be deal with using regular arithmetic
- Continuous RVs require calculus
- All values taken on by a discrete RV have a probability > 0
- All values taken on by a continuous RV have a probability = 0



```python
# A continuous random variable.
def Y():
    return random.random()

Y()
```

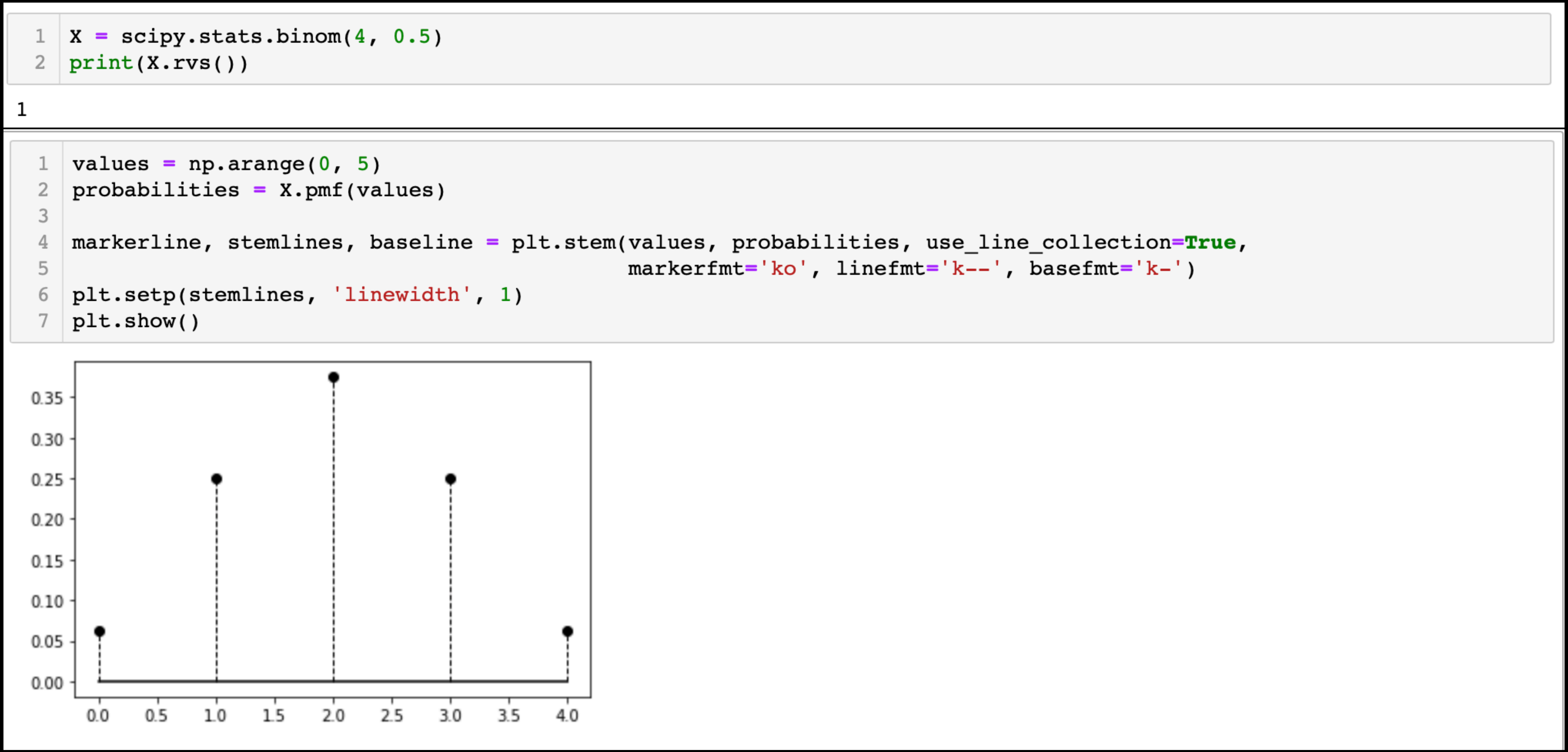
```
0.6486904786830571
```

Distributions, PMFs and PDFs

- Distributions describe the RV - “what is the probability that X will output a given value in a given interval?”
- Python supports many types: scipy.stats module
- A discrete distribution has a PMF (probability mass function). It takes in a value and outputs a probability
- If P is the PMF of our variable X, then P(1) = 0.5
- Sometimes written as P(X=1) so we are clearly talking about the RV ‘X’
- If you plot this function, the x-axis shows the values it can take on, and the y-axis will show the probability of that value
- The sum of all probabilities for a RV must be 1

Distributions, PMFs and PDFs

- We can create RVs using known distributions e.g.



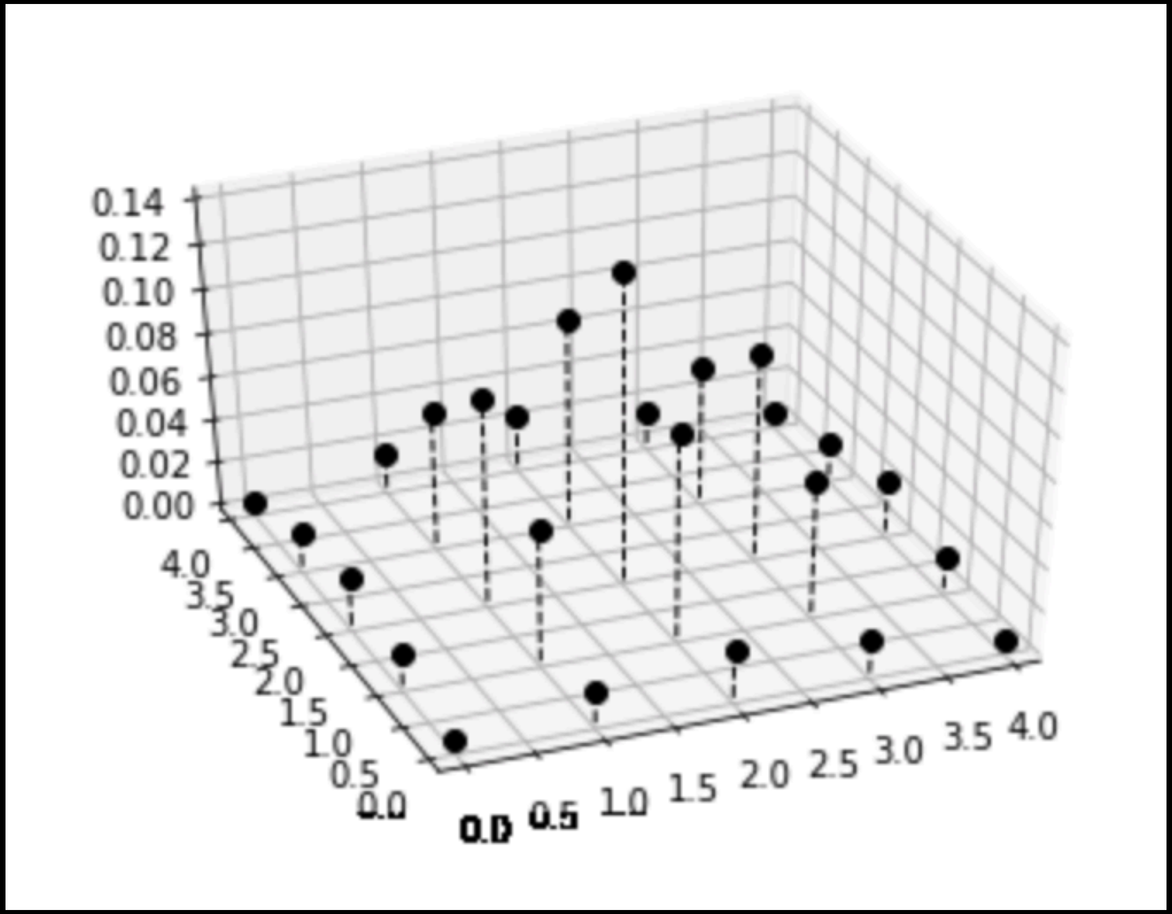
```
X = scipy.stats.binom(4, 0.5)
print(X.rvs())
```

```
1
```

```
values = np.arange(0, 5)
probabilities = X.pmf(values)

markerline, stemlines, baseline = plt.stem(values, probabilities, use_line_collection=True,
                                           markerfmt='ko', linefmt='k--', basefmt='k-')
plt.setp(stemlines, 'linewidth', 1)
plt.show()
```

Distributions, PMFs and PDFs

- We can have PMFs for multiple variables: P(X=x, Y=y)
- Sometimes, P(X=x, Y=y) = P(X=x) * P(Y=y) - but not always

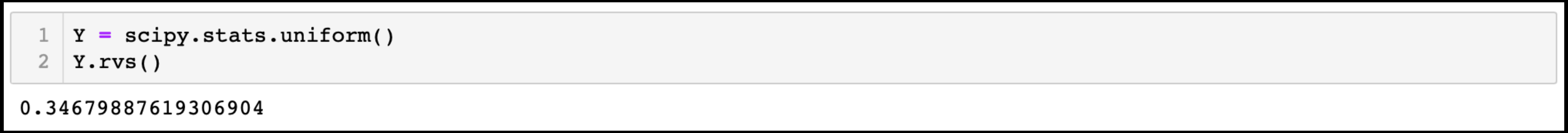


Distributions, PMFs and PDFs

- A continuous RV has a PDF (probability density function) - which does not return a probability for a given value (as this will always be 0)
- But it expresses the notion that values in a certain region are more likely than in another
- The value of a PDF ‘p’ at ‘x’ i.e. p(x) is proportional to the probability of getting a number within an interval that is close to ‘x’
- Mathematically defined as the area under the graph in the interval of interest (requires integration - which is the continuous equivalent of summation)
- Continuous distributions: https://docs.scipy.org/doc/scipy/reference/tutorial/stats/continuous.html

Distributions, PMFs and PDFs

- Let's define a RV Y, uniformly distributed from 0 to 1:

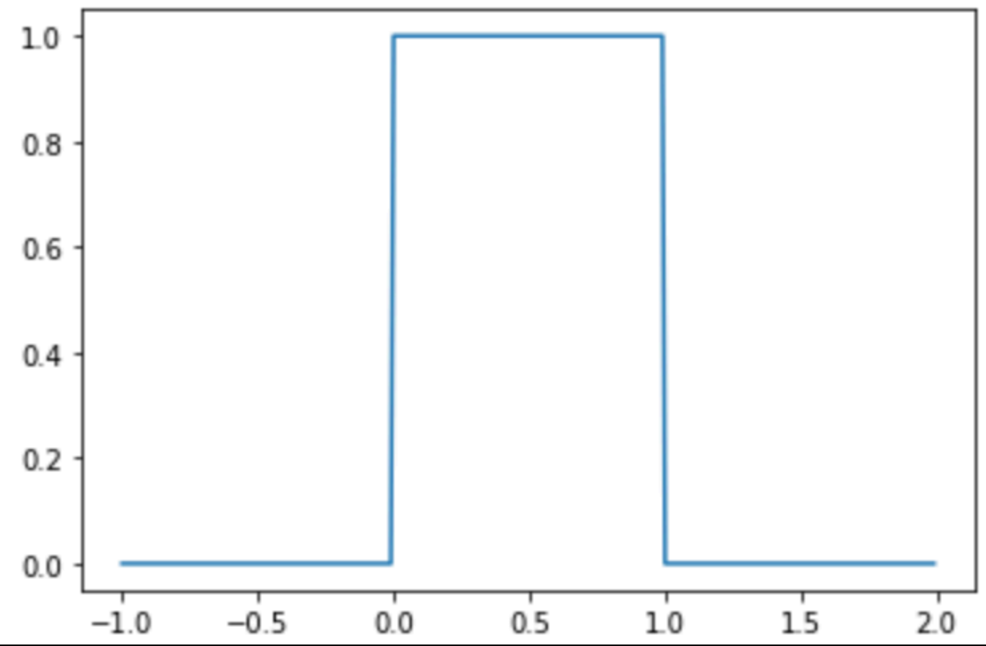


```
Y = scipy.stats.uniform()
Y.rvs()
```

```
0.34679887619306904
```

- All values in this interval should have an equal probability of being chosen:

```
x = np.arange(-1, 2, 0.01)
y = Y.pdf(x)
plt.plot(x, y)
plt.show()
```

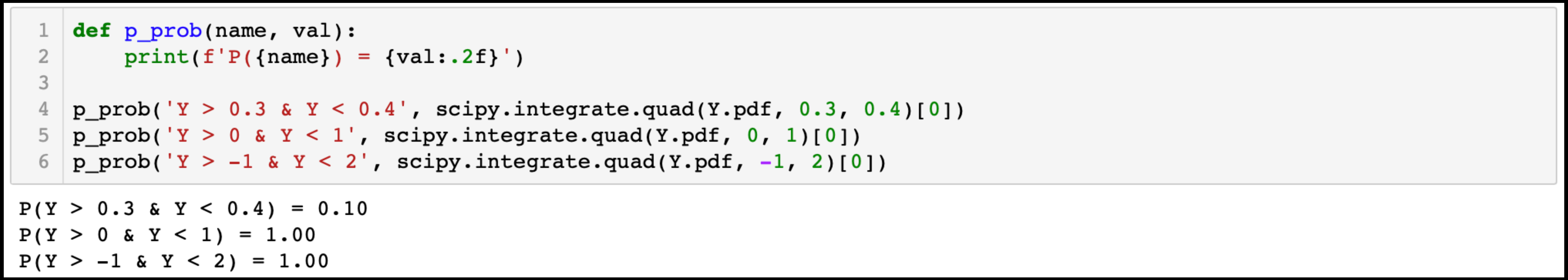


Distributions, PMFs and PDFs

- We can expect that Y returns a number between 0.3 and 0.4 to be 0.1 (as this is 10% of the range)

- Or that a value between 0 and 1 is 1.0, or also that a value between -1 and 2 is also 1.0

- This can be checked by integration:

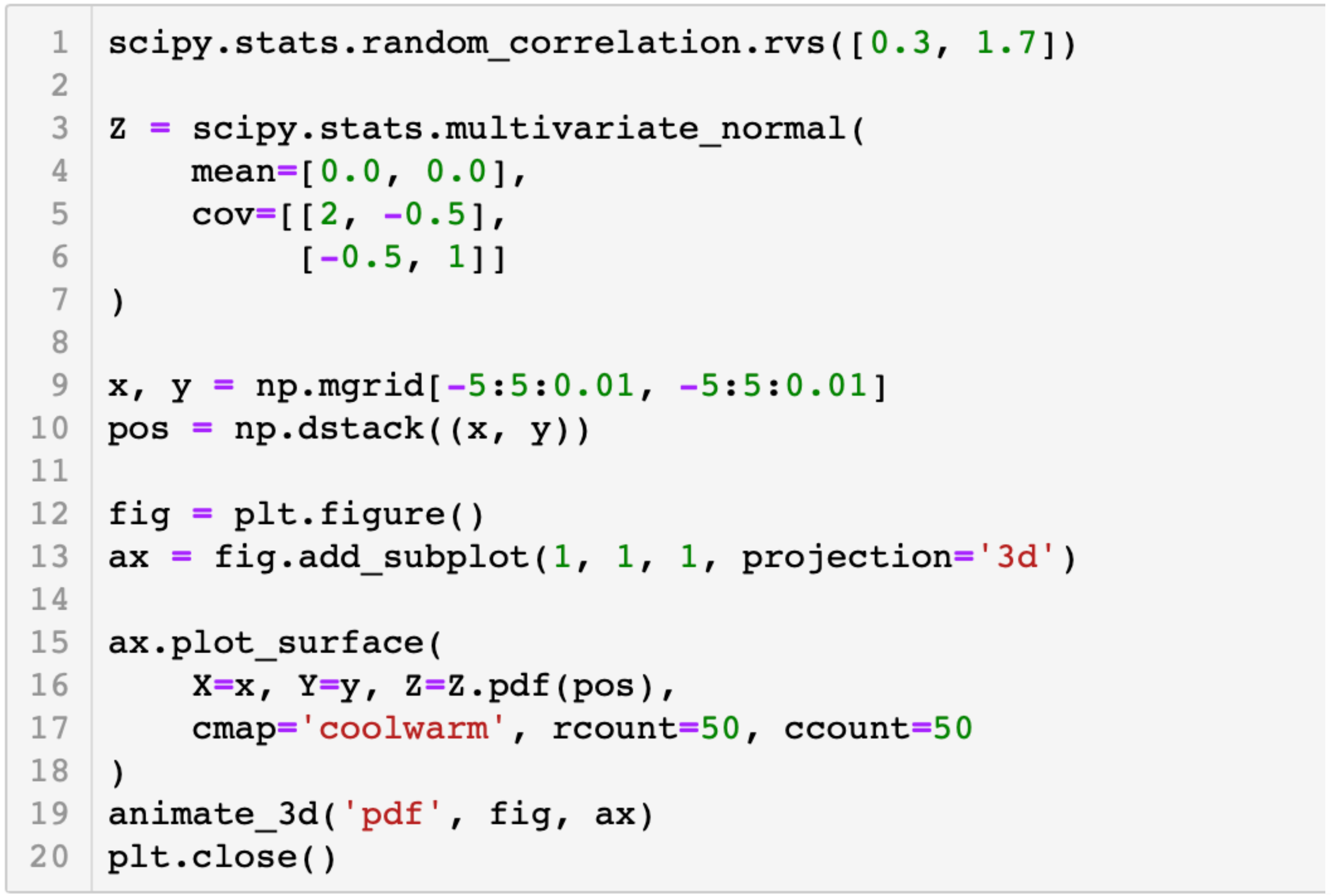


```python
def p_prob(name, val):
    print(f'P({name}) = {val:.2f}')

p_prob('Y > 0.3 & Y < 0.4', scipy.integrate.quad(Y.pdf, 0.3, 0.4)[0])
p_prob('Y > 0 & Y < 1', scipy.integrate.quad(Y.pdf, 0, 1)[0])
p_prob('Y > -1 & Y < 2', scipy.integrate.quad(Y.pdf, -1, 2)[0])
```

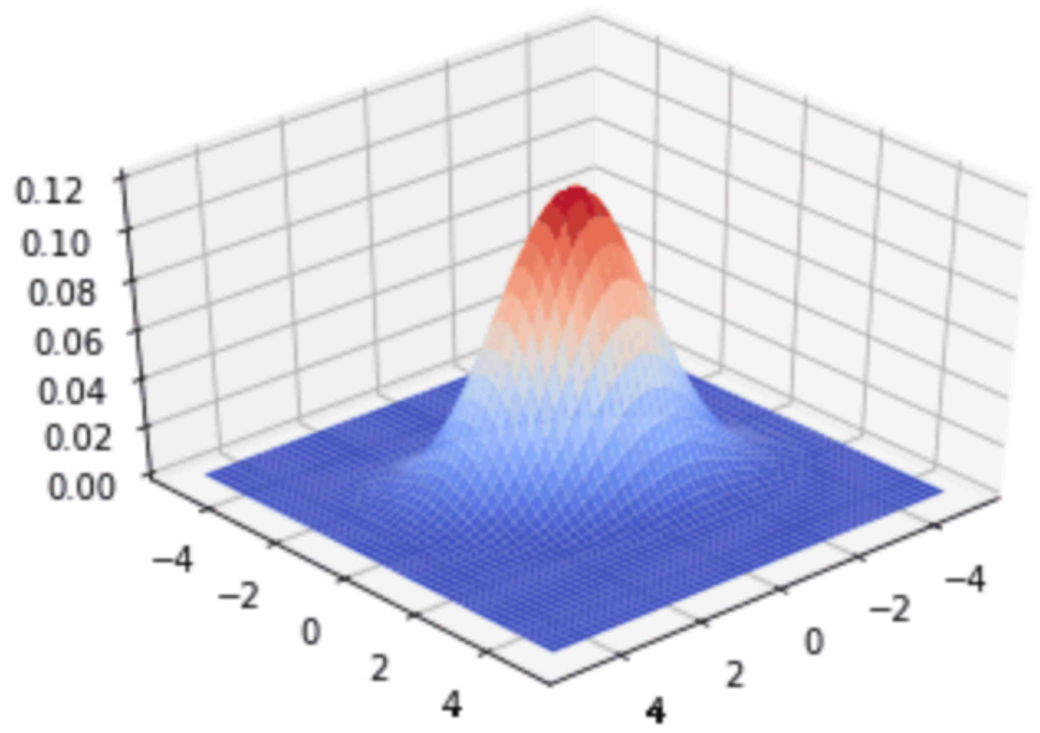
```
P(Y > 0.3 & Y < 0.4) = 0.10
P(Y > 0 & Y < 1) = 1.00
P(Y > -1 & Y < 2) = 1.00
```

Distributions, PMFs and PDFs

- Just like with PMFs, a PDF can also relate two different RVs - we call it a **joint**, or **multivariate** PDF:



```python
scipy.stats.random_correlation.rvs([0.3, 1.7])

Z = scipy.stats.multivariate_normal(
    mean=[0.0, 0.0],
    cov=[[2, -0.5],
         [-0.5, 1]]
)

x, y = np.mgrid[-5:5:0.01, -5:5:0.01]
pos = np.dstack((x, y))

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1, projection='3d')

ax.plot_surface(
    X=x, Y=y, Z=Z.pdf(pos),
    cmap='coolwarm', rcount=50, ccount=50
)
animate_3d('pdf', fig, ax)
plt.close()
```



Properties of Normal Distributions

- In the last example, we defined a multivariate normal distribution, via various properties - the **mean** and **covariance matrix**
- These are important descriptors, so we'll go back a few steps and define what they mean
- To do that we will consider some elementary definitions of mean and variance of a series of univariate values

Properties of Normal Distributions

- For a given list of values for a random variable: $X = x_1, x_2, \ldots$

- The expectation, or arithmetic mean (average), is defined by the sum of all values in the list, divide by the number of values in the list

$$E(X) = \mu = \frac{1}{N} \sum_{i=1}^{N} x_i$$

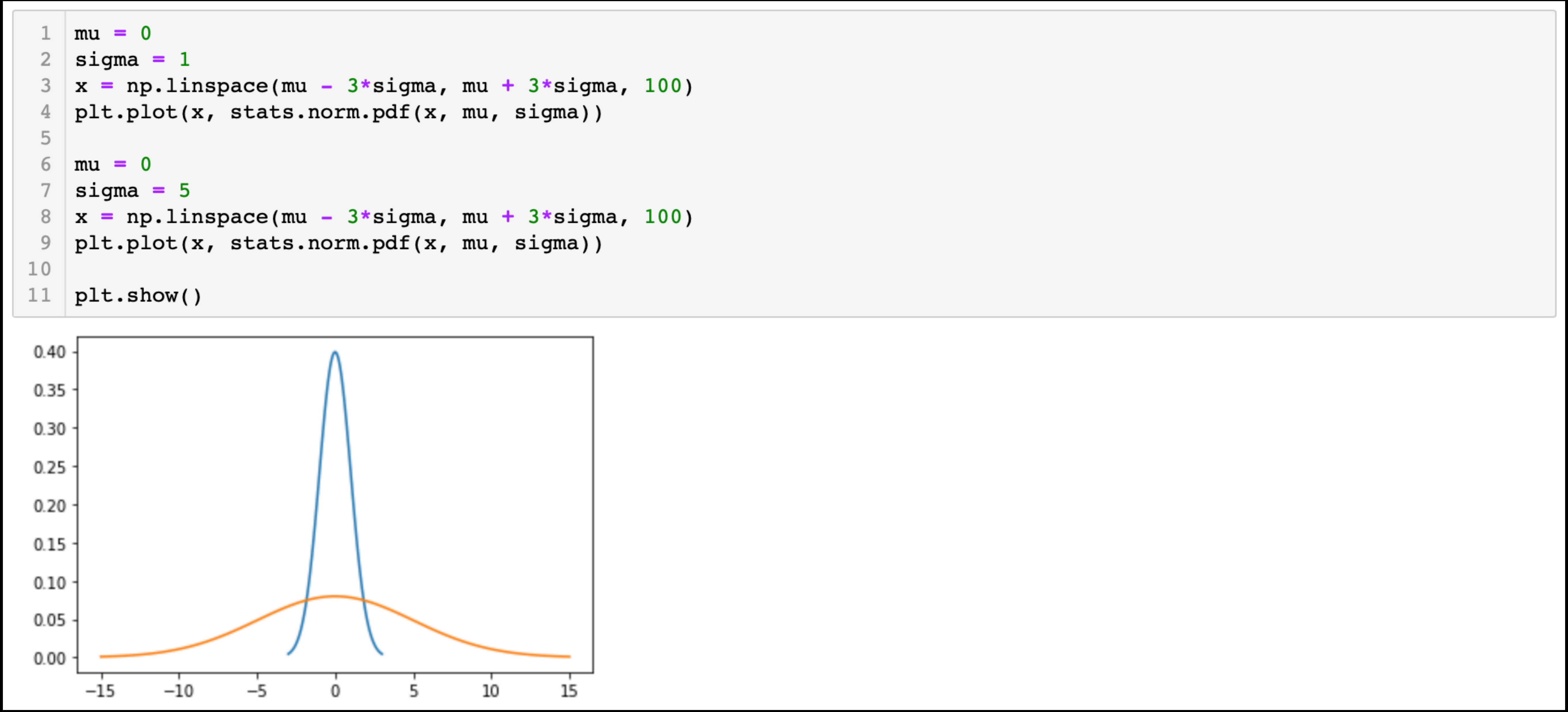
Properties of Normal Distributions

- If we had to sample from a distribution with a mean around a particular value e.g. 3, we can expect the actual mean of sampled values to be close to 3
- For the same list of values, we can calculate the variance, the expectation of the squared deviation of a random variable from its mean

$$Var(X) = \sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

Properties of Normal Distributions

- We can now define normal distributions (Gaussians) of various shapes



```
mu = 0
sigma = 1
x = np.linspace(mu - 3*sigma, mu + 3*sigma, 100)
plt.plot(x, stats.norm.pdf(x, mu, sigma))

mu = 0
sigma = 5
x = np.linspace(mu - 3*sigma, mu + 3*sigma, 100)
plt.plot(x, stats.norm.pdf(x, mu, sigma))

plt.show()
```

Properties of Normal Distributions

- Another characteristic of a Gaussian is derived from the variance, and is called the standard deviation: $\sigma = \sqrt{\sigma^2}$
- Standard deviation tells us what is ‘normal’ within a distribution of values, and what is ‘weird’. Any value that is one standard deviation (or one sigma) away from the mean is non-normal
- All these properties extend to multivariate distributions, we just extend the model for every random variable

Properties of Normal Distributions

- For K dimensions, the mean is defined as:

$$\mu = E(X) = [E(X_1, E(X_2), \ldots, E(X_K)]$$

- For variance, we define a variance matrix (or covariance matrix). For a univariate Gaussian:

$$Var(X) = Cov(X, X)$$

- For a multivariate Gaussian:

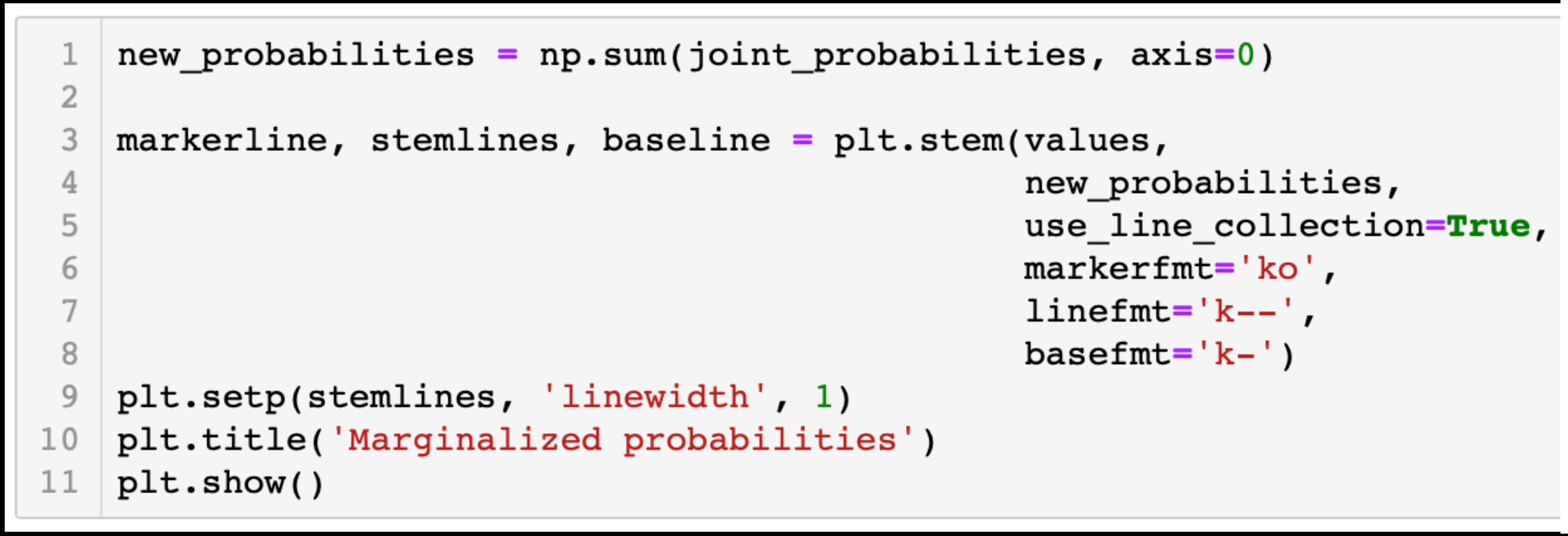
$$\begin{aligned} Cov(X, Y) &= E[(X - E[X])(Y - E[Y])] \\ &= \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu_x)(y_i - \mu_y) \end{aligned}$$

Marginalization

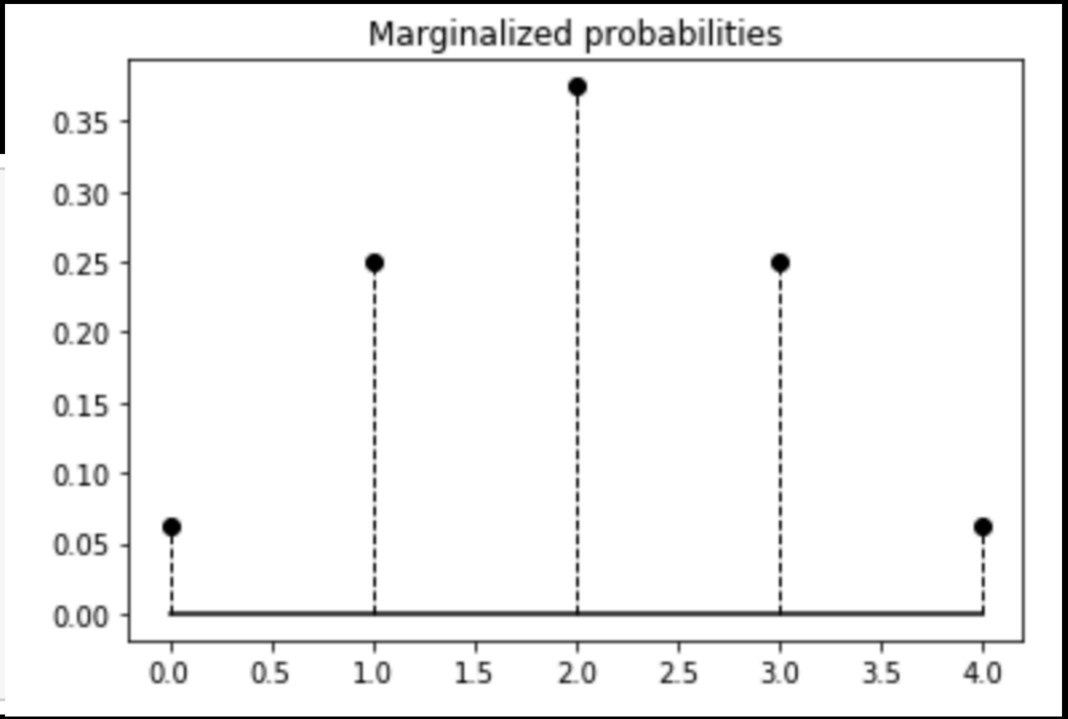
- What if you have a model for $P(X = x, Y = y)$ and want to know just $P(X = x)$?

- Logically, to ignore Y, we should sum for all possible values of Y i.e. $P(X = x, Y = 0) + P(X = x, Y = 1) + \ldots$

- This process is called marginalisation, and the resulting distribution of X is called the marginal distribution

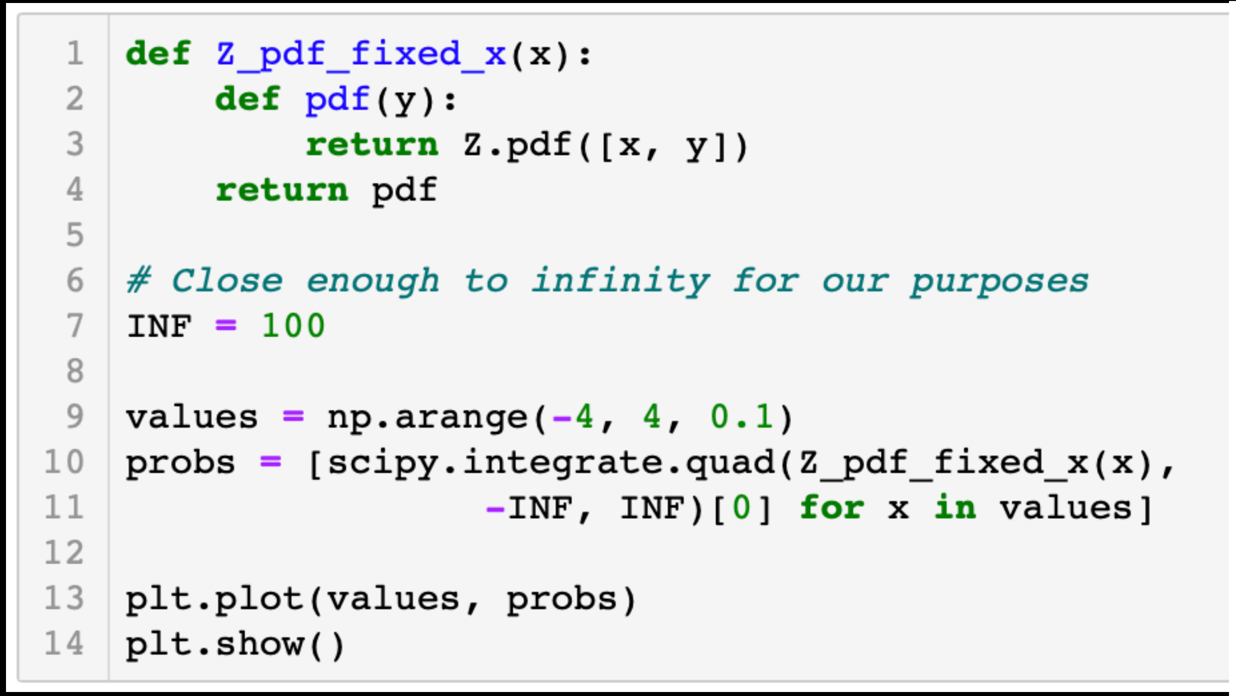


```python
new_probabilities = np.sum(joint_probabilities, axis=0)

markerline, stemlines, baseline = plt.stem(values,
                                           new_probabilities,
                                           use_line_collection=True,
                                           markerfmt='ko',
                                           linefmt='k--',
                                           basefmt='k-')
plt.setp(stemlines, 'linewidth', 1)
plt.title('Marginalized probabilities')
plt.show()
```

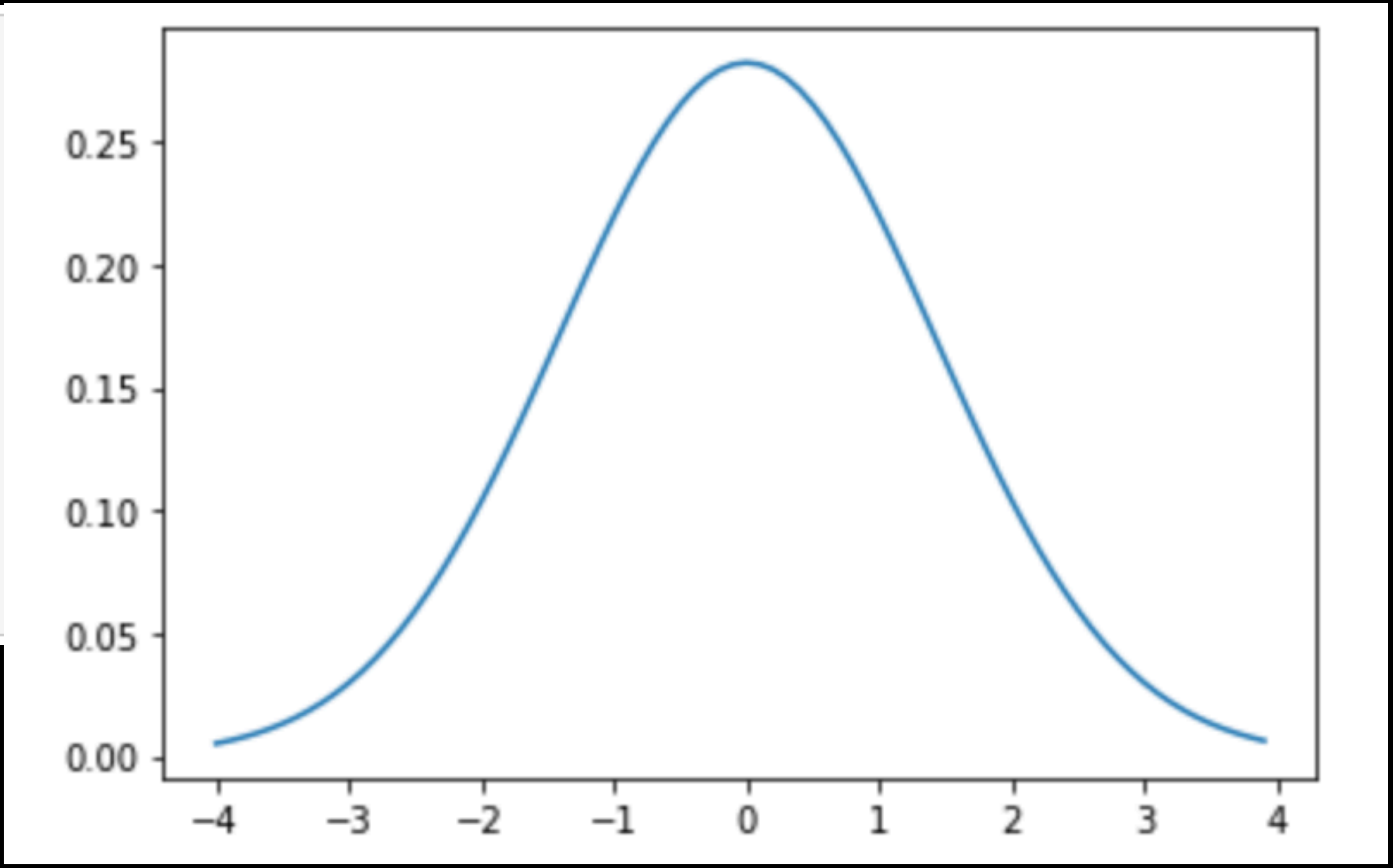


Marginalization

- Marginalization also works for PDFs - replacing summation with integration: $p(x) = \int_{-\infty}^{\infty} p(x, y) dy$



```python
def Z_pdf_fixed_x(x):
    def pdf(y):
        return Z.pdf([x, y])
    return pdf

# Close enough to infinity for our purposes
INF = 100

values = np.arange(-4, 4, 0.1)
probs = [scipy.integrate.quad(Z_pdf_fixed_x(x),
                -INF, INF)[0] for x in values]

plt.plot(values, probs)
plt.show()
```



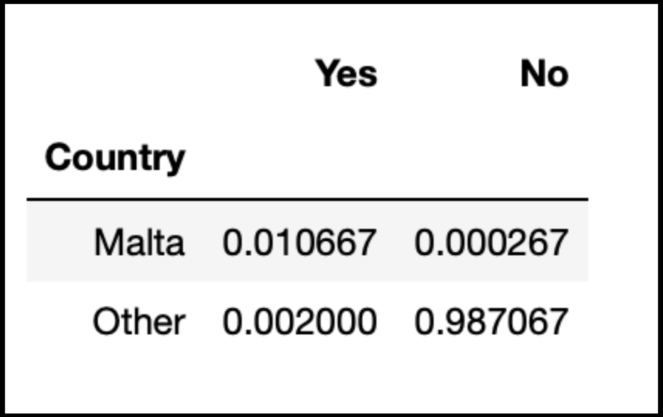
Conditional Probability

- Marginalization gives us the distribution of X, ignoring Y
- What if we want the distribution of X for a specific value of Y?
- This is called the conditional probability - we are setting a condition on Y, and is denoted like so: $P(X = x \mid Y = y)$
- The vertical bar is usually pronounced as “given"

Conditional Probability

- Suppose you want to know whether someone comes from Malta, and you know that they speak fluent Maltese.

- You are given the following joint PMF showing the proportion of the world that speaks Maltese and lives in Malta:



	Yes	No
Country		
Malta	0.010667	0.000267
Other	0.002000	0.987067

- We need to find:

$$P(\text{Country} = \text{Malta} \,|\, \text{Maltese} = \text{Yes}) = \frac{P(\text{Maltese} = \text{Yes}, \text{Country} = \text{Malta})}{P(\text{Maltese} = \text{Yes})}$$

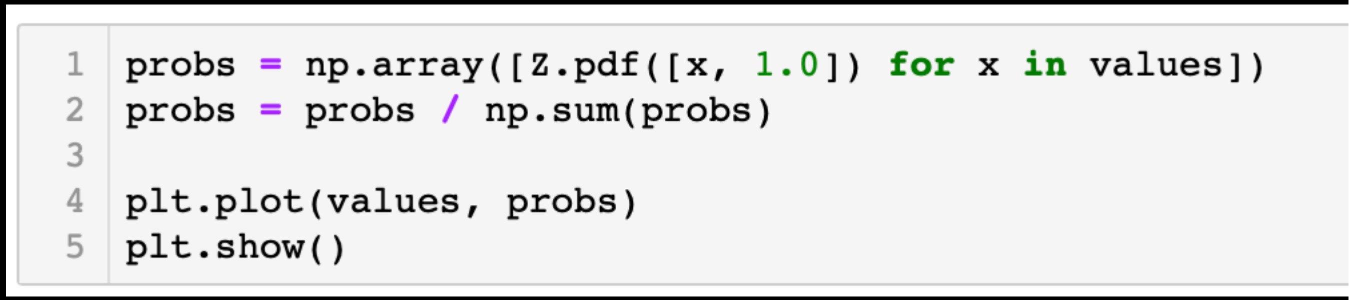
- The denominator can be extracted using marginalisation over the countries of everybody who speaks Maltese

Conditional Probability

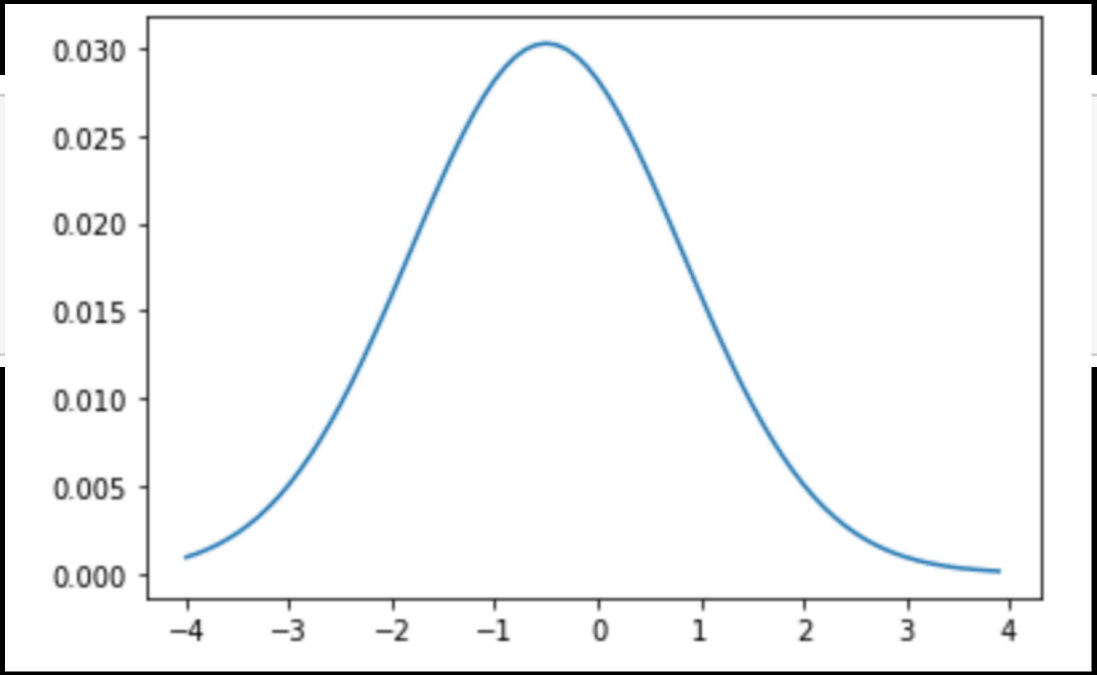
- This brings us to the general rule of conditional probability

$$P(X = x \,|\, Y = y) = \frac{P(X = x, Y = y)}{P(Y = y)}$$

- Applying this rule is like slicing a graph and renormalising so that the final distribution probabilities sum to 1. On our earlier example:

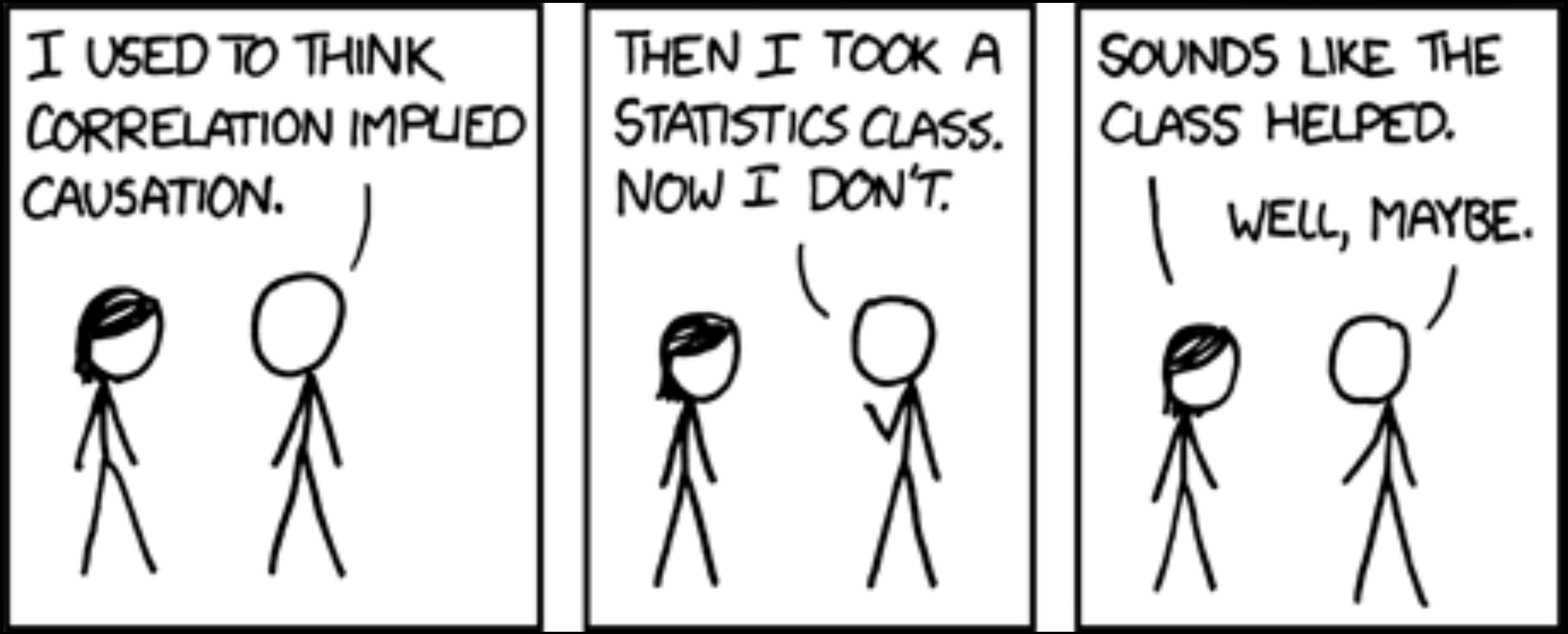


```
probs = np.array([Z.pdf([x, 1.0]) for x in values])
probs = probs / np.sum(probs)

plt.plot(values, probs)
plt.show()
```



Conditional Probability

- Do not confuse conditional probability with actionable fact. In our language example, it is clear that teaching someone Maltese will not make them more likely to be from Malta.


I USED TO THINK CORRELATION IMPLIED CAUSATION.
THEN I TOOK A STATISTICS CLASS. NOW I DON'T.
SOUNDS LIKE THE CLASS HELPED.
WELL, MAYBE.

Conditional Probability

- We now know how to go from a joint to a marginal distribution
- And from a joint or marginal distribution to a conditional distribution

Conditional Probability

- We'll now complete the cycle, going from a marginal and conditional distribution to a joint distribution

- Using our earlier example:

$$P(\text{Country} = \text{Malta}, \text{Maltese} = \text{Yes}) = P(\text{Country} = \text{Malta} \,|\, \text{Maltese} = \text{Yes}) P(\text{Maltese} = \text{Yes})$$

- This is known as the **chain rule** of conditional probability, and the genera form is:

$$P(x, y) = P(x \,|\, y) P(y)$$

Bayes' Rule

- Finally, suppose you do not want to know the probability that your friend is Maltese given that he speaks Maltese, but rather the probability that someone speaks Maltese given that they are from Malta.

- From the definition of conditional probability:

$$P(y|x) = \frac{P(x, y)}{P(x)}$$

- And applying the chain rule we can derive **Bayes' Rule**:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

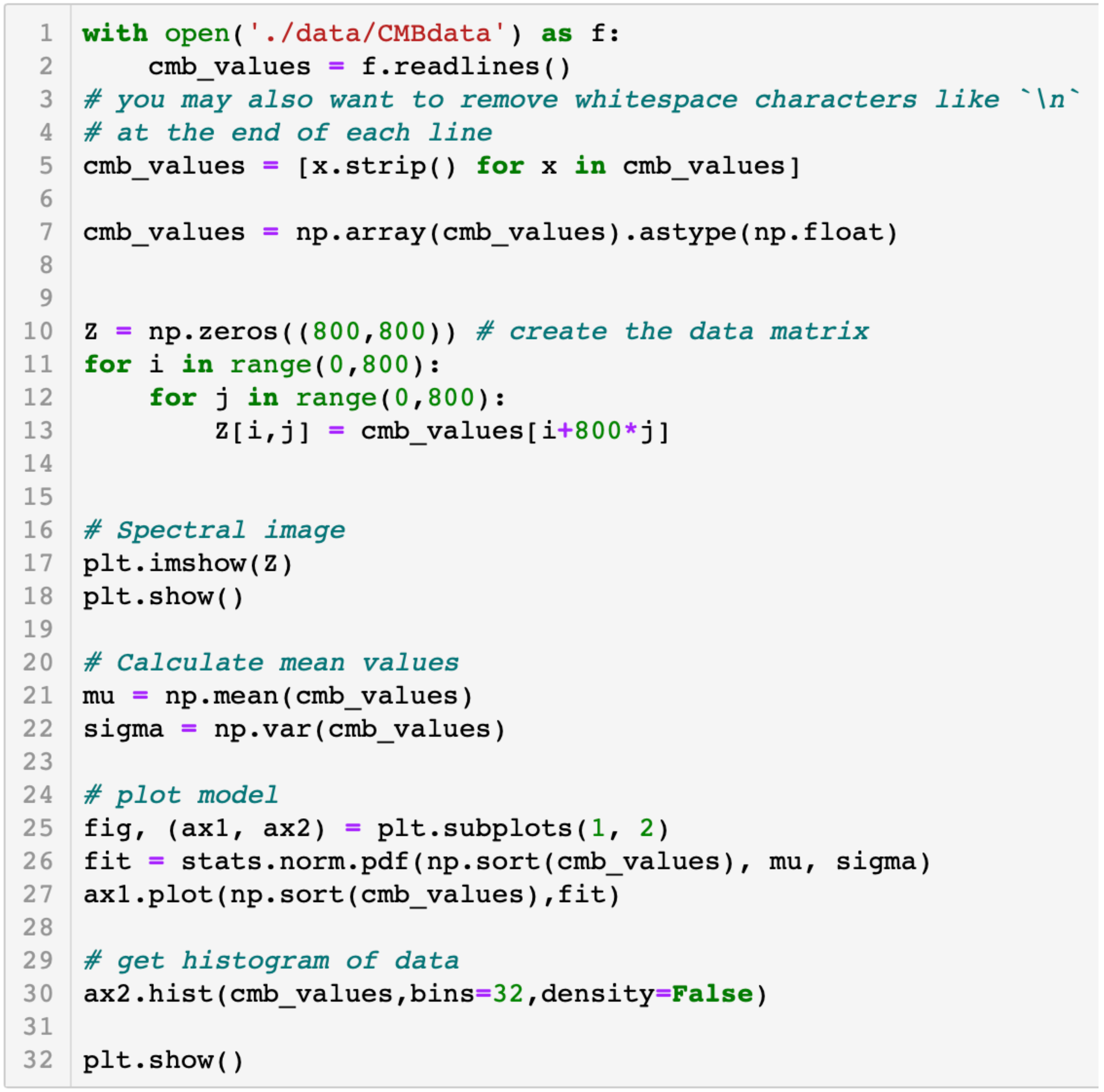
Case Study: CMB Model

- You are given some data representing the spectrum of the Cosmological Microwave Background i.e. electromagnetic radiation from photons back to 300,000 years after the Big Bang, expressed as a difference in apparent temperature from the mean temperature.

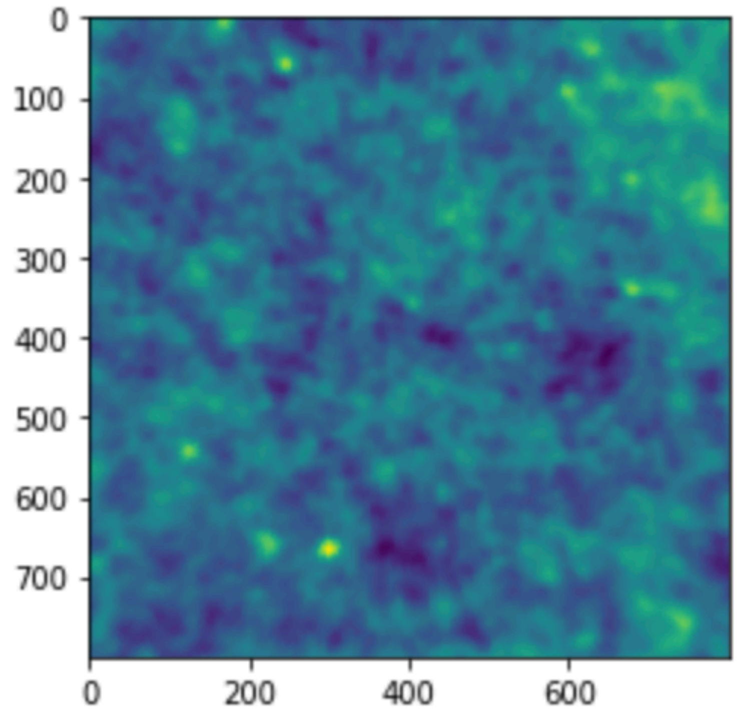
- It would be nice to plot this data, and also look at the distribution of the data.

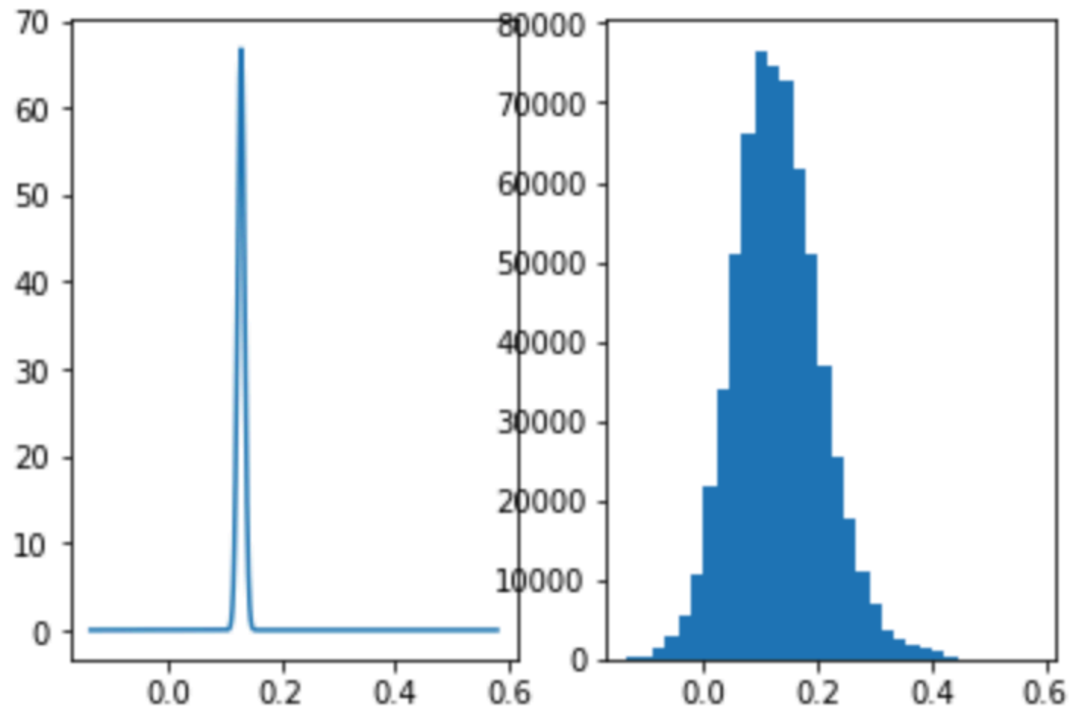
- The data is provided as a text file with values, one value per line. In total there are 640,000 values representing an 800x800 chunk of the spectrum.

Case Study: CMB Model



```python
with open('./data/CMBdata') as f:
    cmb_values = f.readlines()
# you may also want to remove whitespace characters like `\n`
# at the end of each line
cmb_values = [x.strip() for x in cmb_values]

cmb_values = np.array(cmb_values).astype(np.float)


Z = np.zeros((800,800)) # create the data matrix
for i in range(0,800):
    for j in range(0,800):
        Z[i,j] = cmb_values[i+800*j]


# Spectral image
plt.imshow(Z)
plt.show()

# Calculate mean values
mu = np.mean(cmb_values)
sigma = np.var(cmb_values)

# plot model
fig, (ax1, ax2) = plt.subplots(1, 2)
fit = stats.norm.pdf(np.sort(cmb_values), mu, sigma)
ax1.plot(np.sort(cmb_values),fit)

# get histogram of data
ax2.hist(cmb_values,bins=32,density=False)

plt.show()
```





Machine Learning

- You may have heard about ML - you probably used ML at some point in your life
- Idea: out of past experience (from data), you are about to get some form of model to learn over time - useful to predicting future events.

Linear Regression

- **Linear regression** is a basic predictive analytics technique that uses historical data to predict an output variable.

- For example: assume we have historical data that the price (P) of a house is linearly dependent upon its size (S) - in fact we find that a houses prices is exactly 90 times the size. The equation would look like:

$$P = 90 * S$$

Linear Regression

- Let’s define some terminology related to variables in a regression model - a lot of which is common across all of ML

- There are two kinds of variables in a regression model:

  - The input, or **predictor** variable: a variable to help us predict the value of the output variable, commonly X

  - The output, or **predictee**: the variable we want to predict, commonly Y

- The general equation for LR is: $Y_e = \alpha + \beta X$

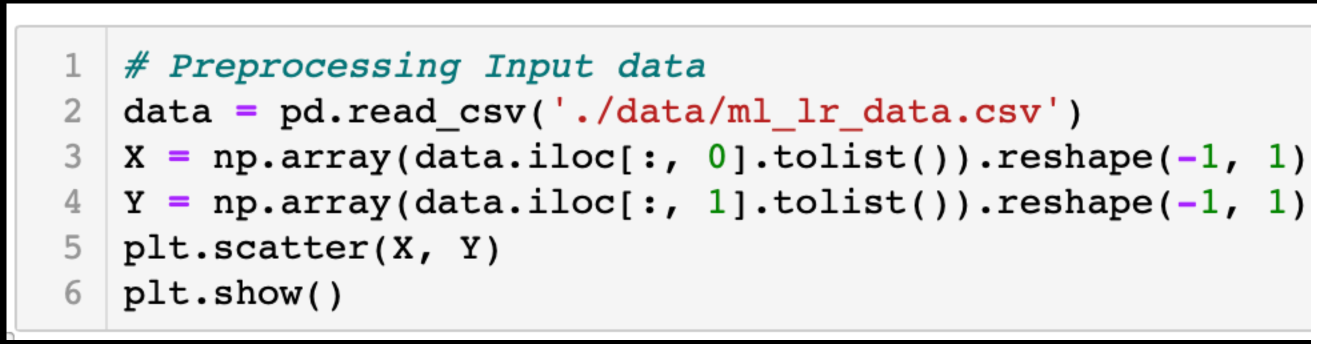
PAC Learning

- PAC (probably approximately correct) learning - a framework for the formal analysis of ML
- It’s the ‘scientific method’ for data: we receive some data, make a hypothesis about a model, with the goal of eventually:
- finding a model that maximises the **probability** for the observed data
- be good at **approximating** the output of future data
- learn the concept behind the data (not overfit), to be **correct**, within reason

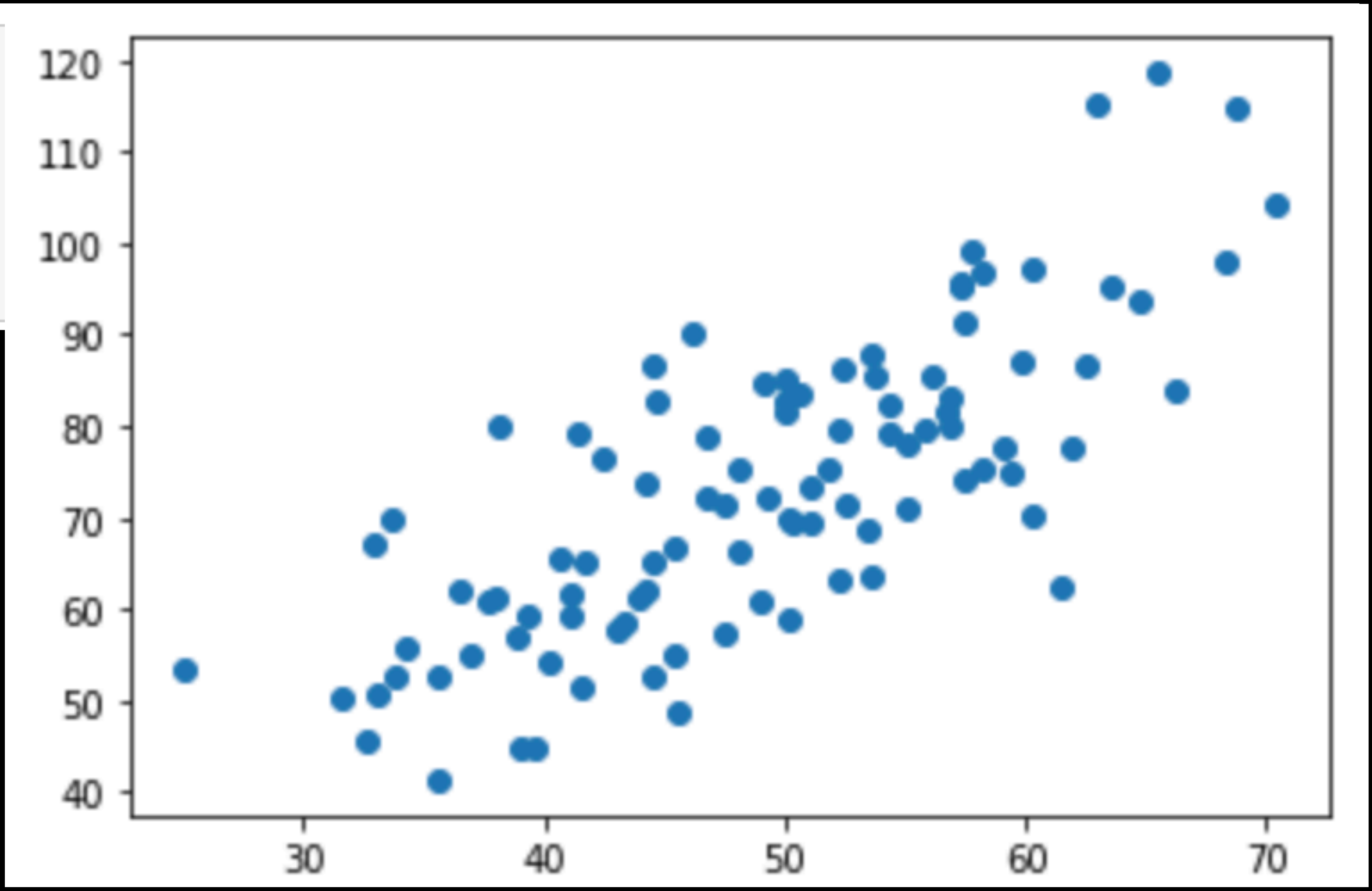
Ordinary Least Squares

- Back to our problem, we need to find statistically significant values for α and β that minimise the difference between Y and Y_e

- So, let's consider a dataset:



```python
# Preprocessing Input data
data = pd.read_csv('./data/ml_lr_data.csv')
X = np.array(data.iloc[:, 0].tolist()).reshape(-1, 1)
Y = np.array(data.iloc[:, 1].tolist()).reshape(-1, 1)
plt.scatter(X, Y)
plt.show()
```



Ordinary Least Squares

- We will not go through a derivation - but the unknown parameters are estimated via:

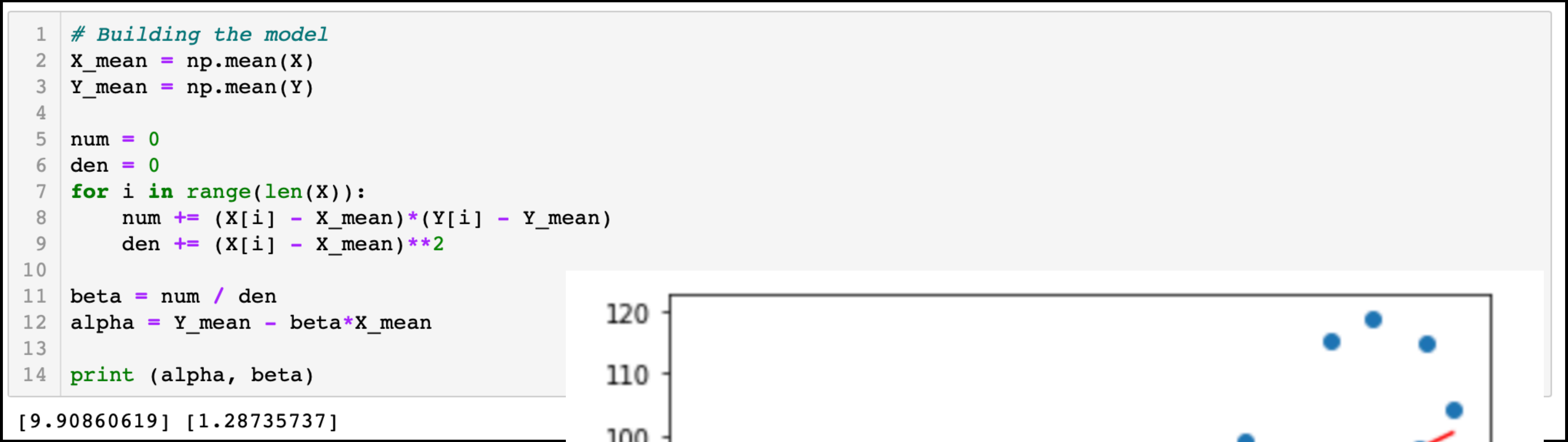
$$\beta = \frac{\sum_{i=1}^{n} (x_i - \mu_X)(y_i - \mu_Y)}{\sum_{i=1}^{n} (x_i - \mu_X)^2}$$

$$\alpha = \mu_Y - \beta * \mu_X$$

- Some terms should be familiar: β is simply:

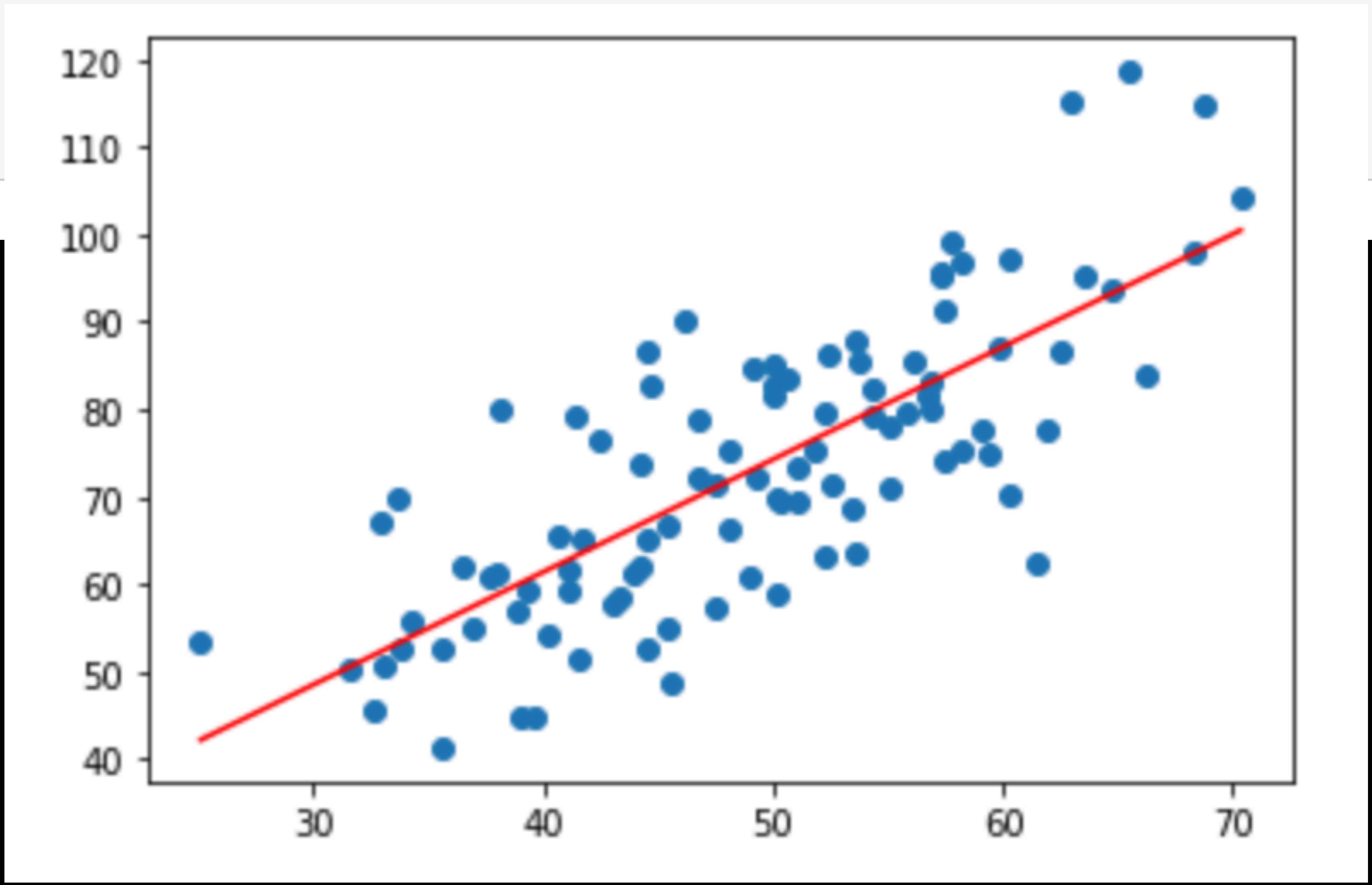
$$Cov(X, Y)/Var(X)$$

Ordinary Least Squares

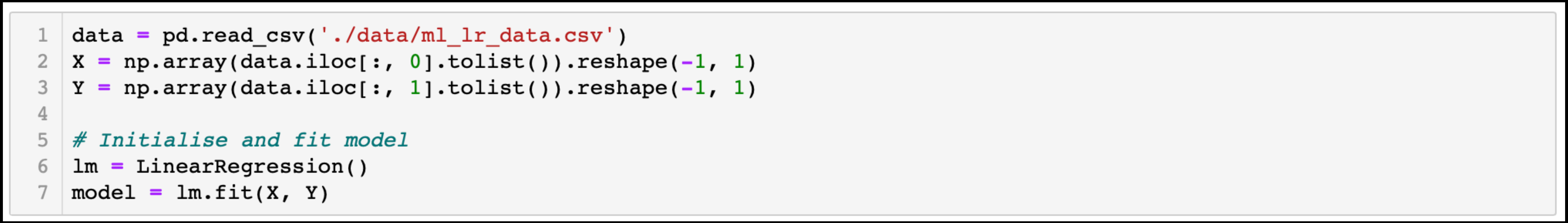


```python
# Building the model
X_mean = np.mean(X)
Y_mean = np.mean(Y)

num = 0
den = 0
for i in range(len(X)):
    num += (X[i] - X_mean)*(Y[i] - Y_mean)
    den += (X[i] - X_mean)**2

beta = num / den
alpha = Y_mean - beta*X_mean

print (alpha, beta)
```

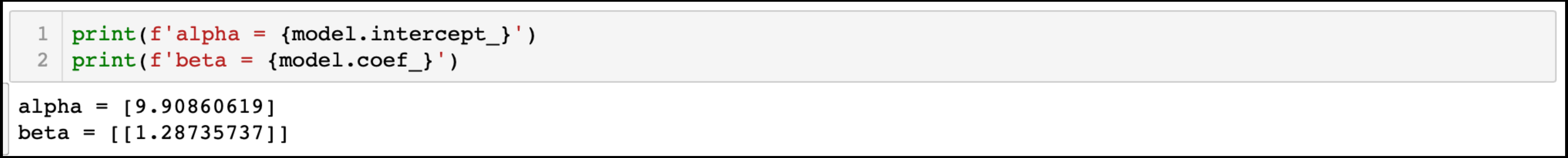
```
[9.90860619] [1.28735737]
```



Linear Regression with Scikit-Learn

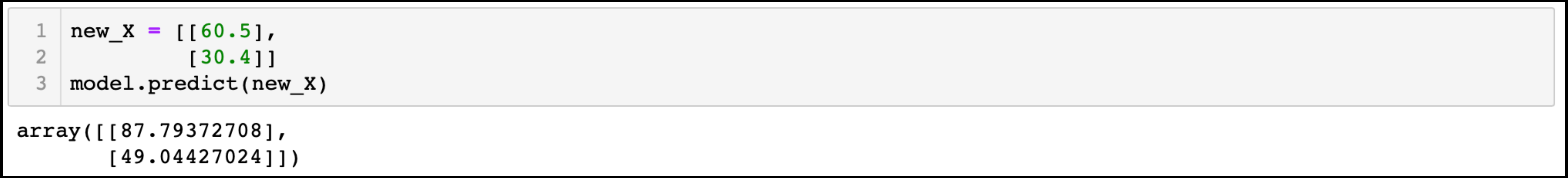


```python
data = pd.read_csv('./data/ml_lr_data.csv')
X = np.array(data.iloc[:, 0].tolist()).reshape(-1, 1)
Y = np.array(data.iloc[:, 1].tolist()).reshape(-1, 1)

# Initialise and fit model
lm = LinearRegression()
model = lm.fit(X, Y)
```



```python
print(f'alpha = {model.intercept_}')
print(f'beta = {model.coef_}')
```

```
alpha = [9.90860619]
beta = [[1.28735737]]
```



```python
new_X = [[60.5],
         [30.4]]
model.predict(new_X)
```

```
array([[87.79372708],
       [49.04427024]])
```

Multivariate Regression

- Scikit-Learn supports linear regression over multivariate data. The genera model structure is:

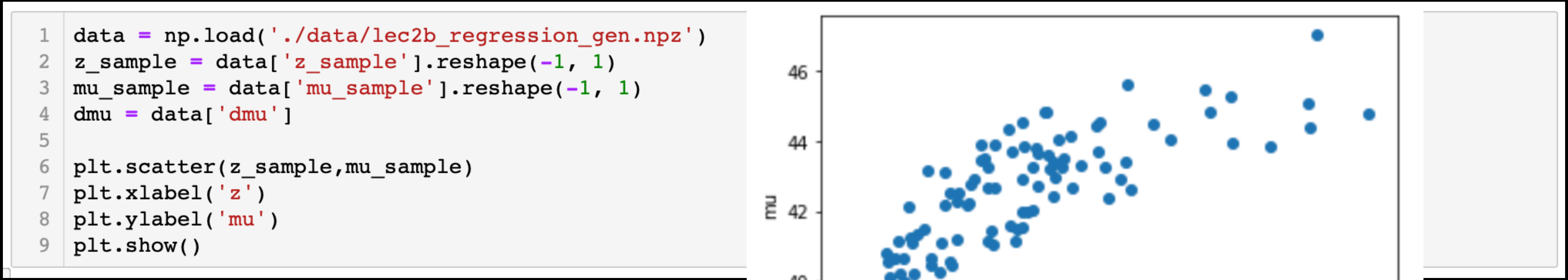
$$Y_e = \alpha + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$$

- You can work with as many predictors (p), or **features**, as needed

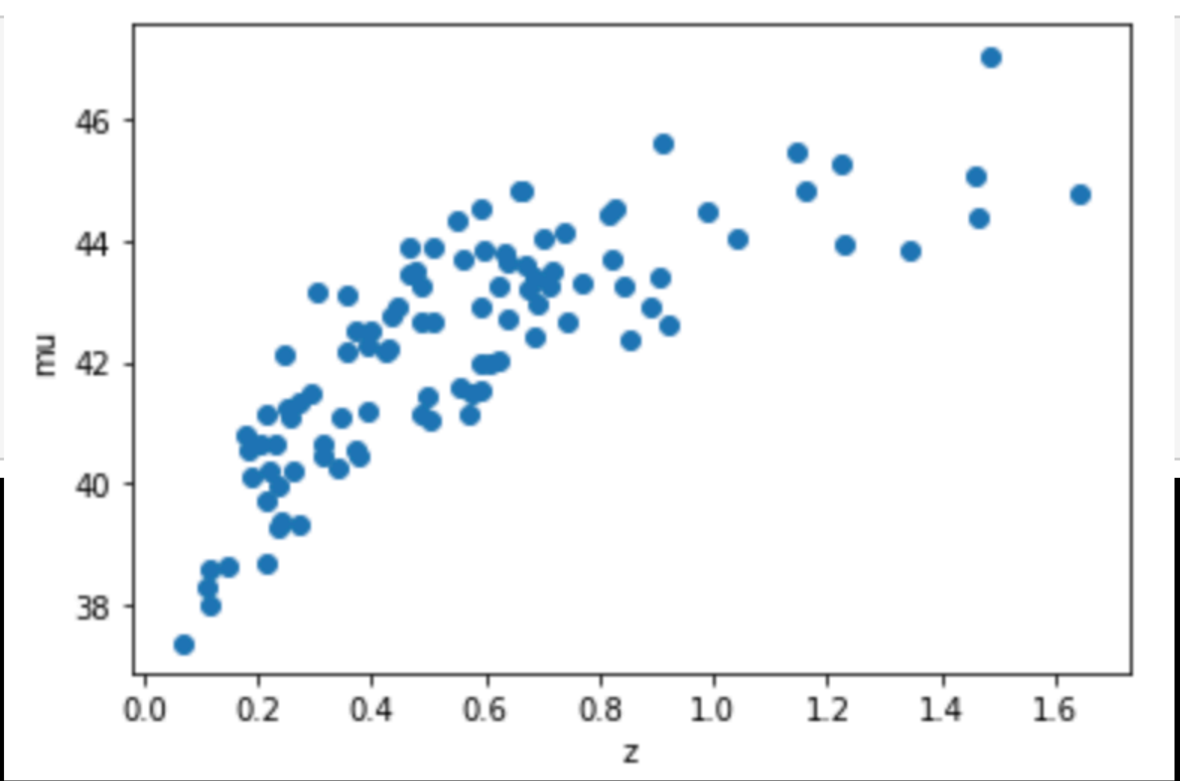
Case Study: Redshifts vs Luminosity Distance

Look at a parametrization of the expansion of the universe (redshifts of supernovas vs luminosity distance) - dataset generated by the model:

$$\mu(z) = -5\log_{10}\left((1+z)\frac{c}{H_0}\int\frac{dz}{\left(\Omega_m(1+z)^3+\Omega_\Lambda\right)^{1/2}}\right)$$

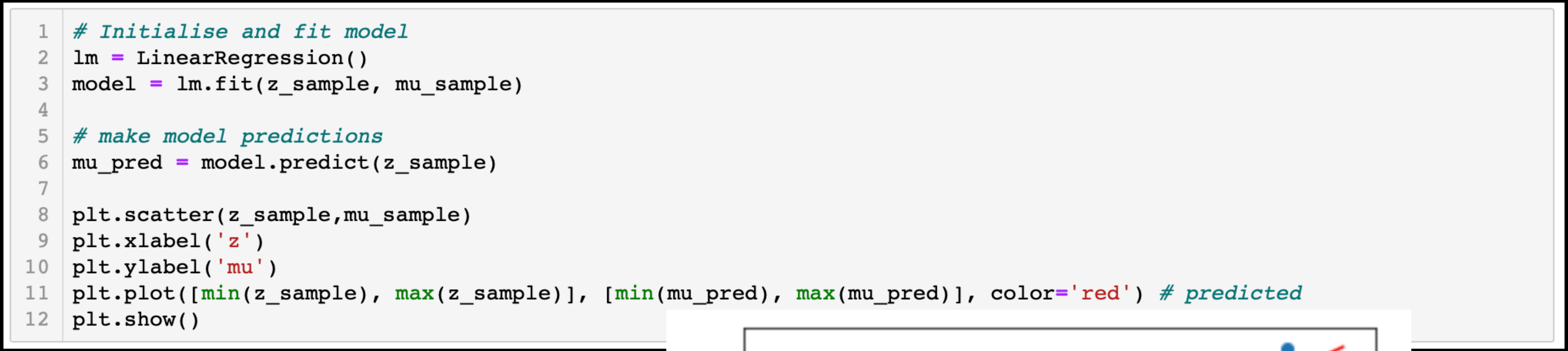


```
data = np.load('./data/lec2b_regression_gen.npz')
z_sample = data['z_sample'].reshape(-1, 1)
mu_sample = data['mu_sample'].reshape(-1, 1)
dmu = data['dmu']

plt.scatter(z_sample,mu_sample)
plt.xlabel('z')
plt.ylabel('mu')
plt.show()
```

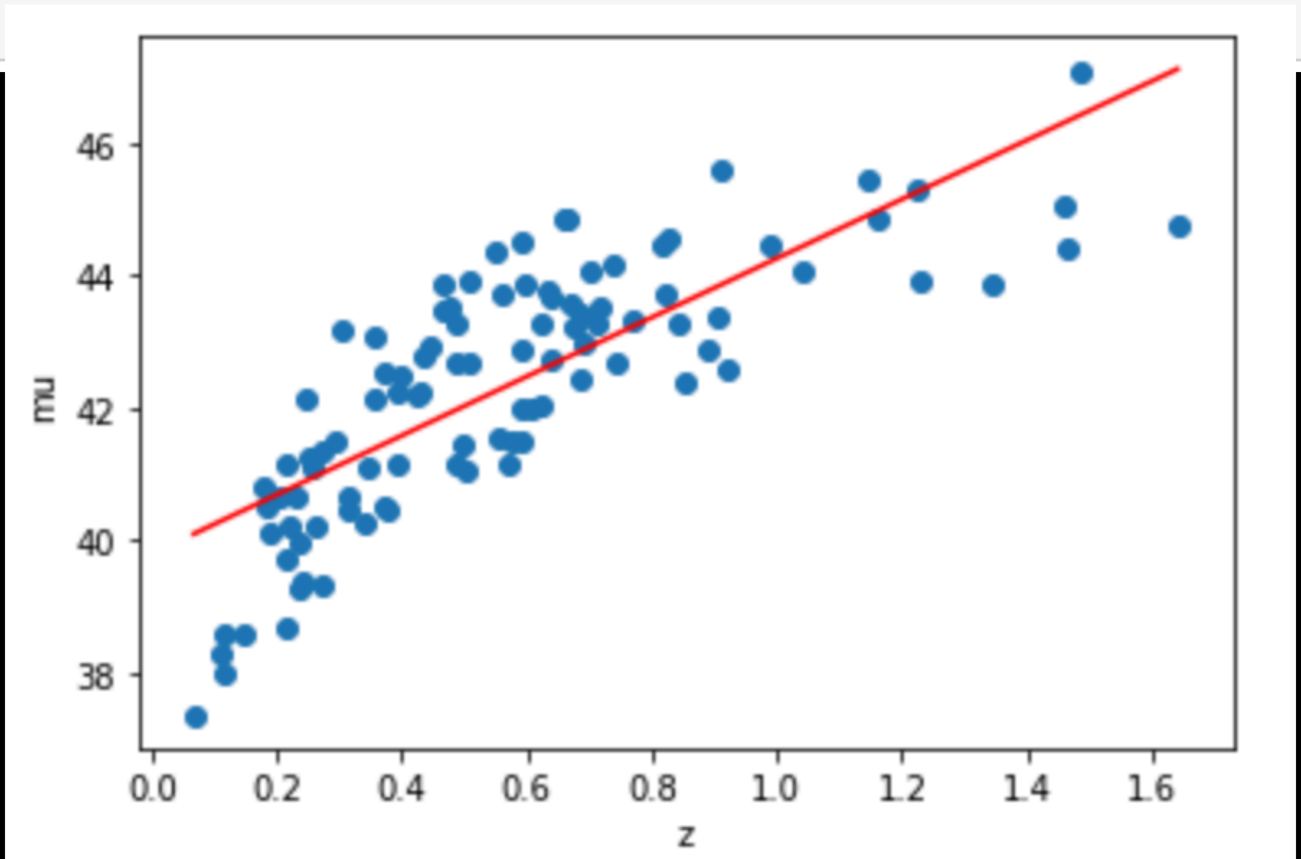


Case Study:
Redshifts vs Luminosity Distance

- Fit a linear regression model:



```python
# Initialise and fit model
lm = LinearRegression()
model = lm.fit(z_sample, mu_sample)

# make model predictions
mu_pred = model.predict(z_sample)

plt.scatter(z_sample,mu_sample)
plt.xlabel('z')
plt.ylabel('mu')
plt.plot([min(z_sample), max(z_sample)], [min(mu_pred), max(mu_pred)], color='red') # predicted
plt.show()
```



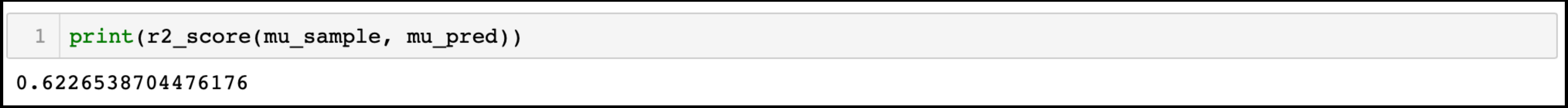
Case Study:
Redshifts vs Luminosity Distance

- Run an evaluation score, e.g. an R2 score.

- Sklearn model predictions can be compared to the truth values to return the coefficient of determination R^2 of the prediction.

- This is defined as $(1 - u/v)$, where u is the residual sum of squares ((y_true - y_pred) ** 2).sum() and v is the total sum of squares ((y_true - y_true.mean()) ** 2).sum()

- The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse).

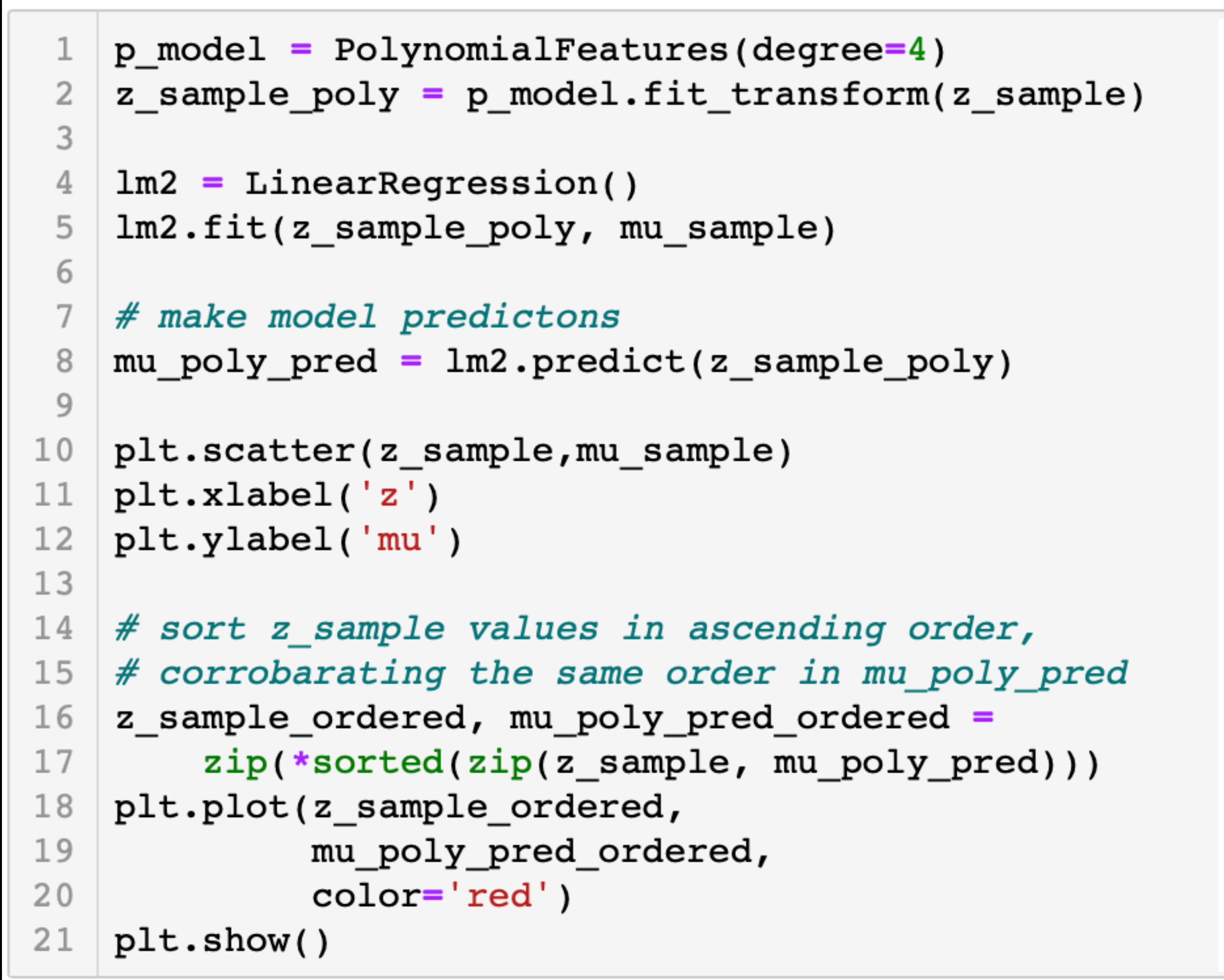


```
print(r2_score(mu_sample, mu_pred))
```

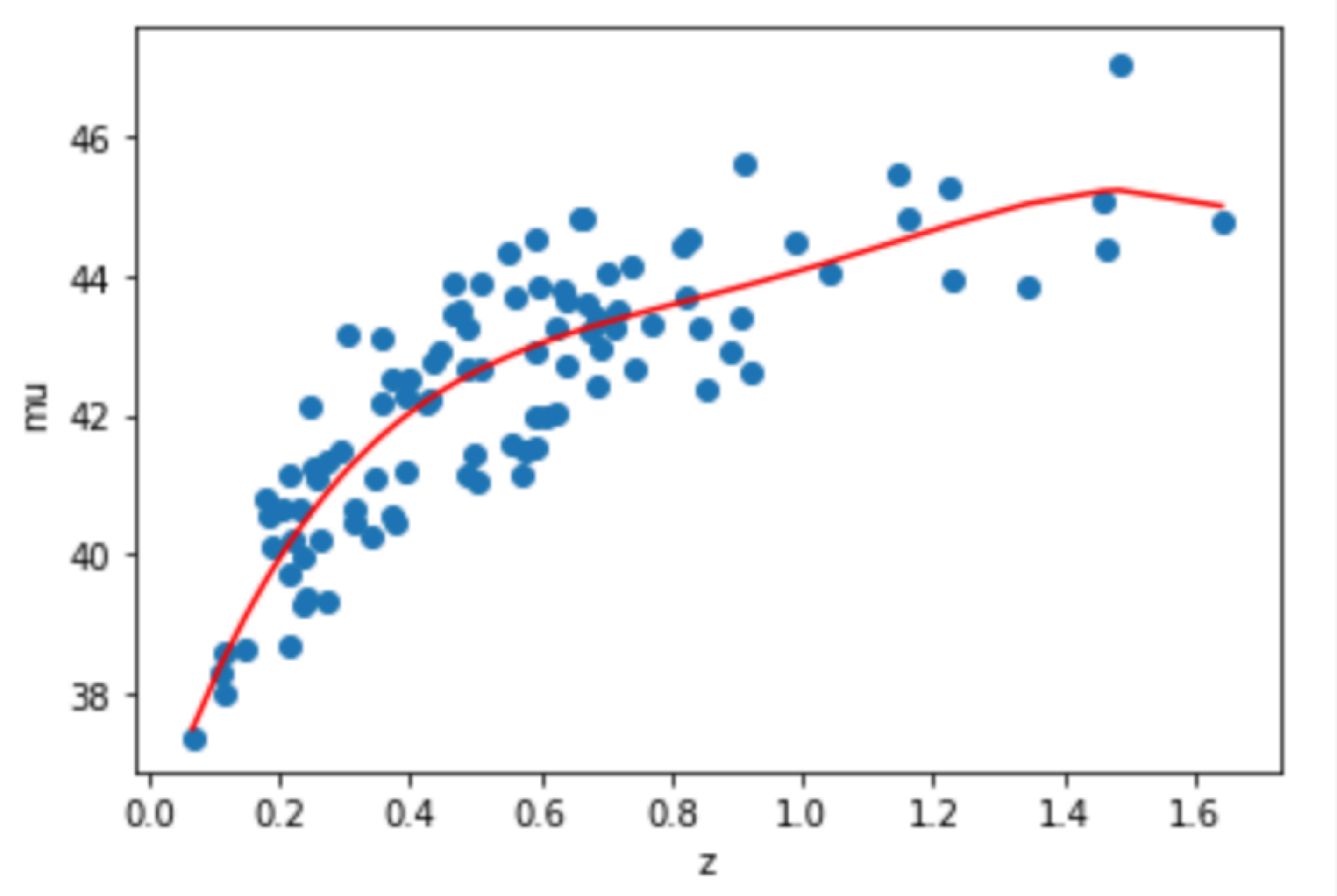
```
0.6226538704476176
```

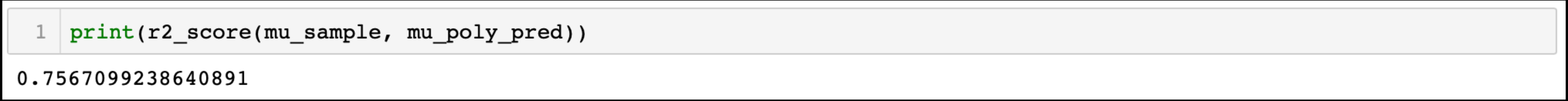
Case Study: Redshifts vs Luminosity Distance

- A better model can give us a better score. We'll introduce a non-linear, polynomial regression of the 4th degree:



```
p_model = PolynomialFeatures(degree=4)
z_sample_poly = p_model.fit_transform(z_sample)

lm2 = LinearRegression()
lm2.fit(z_sample_poly, mu_sample)

# make model predictons
mu_poly_pred = lm2.predict(z_sample_poly)

plt.scatter(z_sample,mu_sample)
plt.xlabel('z')
plt.ylabel('mu')

# sort z_sample values in ascending order,
# corrobarating the same order in mu_poly_pred
z_sample_ordered, mu_poly_pred_ordered =
    zip(*sorted(zip(z_sample, mu_poly_pred)))
plt.plot(z_sample_ordered,
         mu_poly_pred_ordered,
         color='red')
plt.show()
```





```
print(r2_score(mu_sample, mu_poly_pred))
```

```
0.7567099238640891
```

Conclusion

Machine Learning

Machine: The various models available to us e.g. linear regression, polynomial regression, etc.

Learning: Tuning of parameters of the model to fit a dataset.