**Exercise**

**Exercise**

**How many clusters?**

You are given an array points of size 300x2, where each row gives the (x, y) co-ordinates of a point on a map. Make a scatter plot of these points, and use the scatter plot to guess how many clusters there are.

matplotlib.pyplot has already been imported as plt. In the IPython Shell:

* Create an array called xs that contains the values of points[:,0] - that is, column 0 of points.
* Create an array called ys that contains the values of points[:,1] - that is, column 1 of points.
* Make a scatter plot by passing xs and ys to the plt.scatter() function.
* Call the plt.show() function to show your plot.

How many clusters do you see?

**Instructions**

**50 XP**

**Possible Answers**

* 

2

* 

3

* 

300

**Submit Answer**

[**Take Hint (-15 XP)**](javascript:void(0))

+50 XP

Correct! The scatter plot suggests that there are 3 distinct clusters.

In [1]: xs, ys = points[:,0], points[:,1]

In [2]: plt.scatter(xs, ys);\

ERROR! Session/line number was not unique in database. History logging moved to new session 6

In [3]: plt.scatter(xs, ys); plt.show()

ERROR! Session/line number was not unique in database. History logging moved to new session 7

In [4]: plt.scatter(xs, ys); plt.show()

ERROR! Session/line number was not unique in database. History logging moved to new session 8

In [5]: plt.scatter(xs, ys); plt.show()

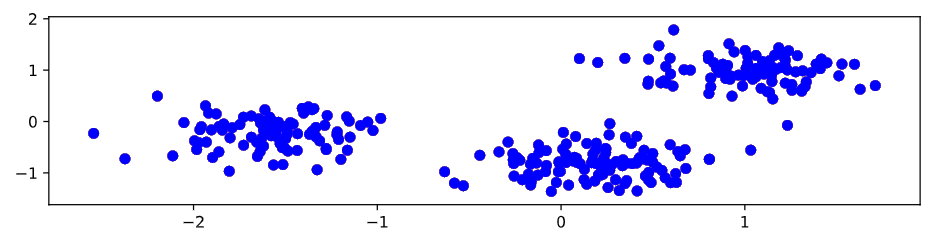
In [6]: plt.scatter(xs, ys); plt.show()

In [7]: plt.scatter(xs, ys); plt.show()

In [8]: plt.scatter(xs, ys); plt.show()

In [9]: plt.scatter(xs, ys, c='blue'); plt.show()

In [10]:



**Exercise**

**Exercise**

**Clustering 2D points**

From the scatter plot of the previous exercise, you saw that the points seem to separate into 3 clusters. You'll now create a KMeans model to find 3 clusters, and fit it to the data points from the previous exercise. After the model has been fit, you'll obtain the cluster labels for some new points using the .predict() method.

You are given the array points from the previous exercise, and also an array new\_points.

**Instructions**

**100 XP**

* Import KMeans from sklearn.cluster.
* Using KMeans(), create a KMeans instance called model to find 3 clusters. To specify the number of clusters, use the n\_clusters keyword argument.
* Use the .fit() method of model to fit the model to the array of points points.
* Use the .predict() method of model to predict the cluster labels of new\_points, assigning the result to labels.
* Hit 'Submit Answer' to see the cluster labels of new\_points.

[**Take Hint (-30 XP)**](javascript:void(0))

# Import KMeans

from sklearn.cluster import KMeans

# Create a KMeans instance with 3 clusters: model

model = KMeans(3)

# Fit model to points

model.fit(points)

# Determine the cluster labels of new\_points: labels

labels = model.predict(new\_points)

# Print cluster labels of new\_points

print(labels)

In [1]: \_\_main\_\_

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

\_\_main\_\_

NameError: name '\_\_main\_\_' is not defined

<script.py> output:

[2 1 0 2 1 2 1 1 1 0 2 1 1 0 0 1 0 0 1 1 0 1 2 1 2 0 1 0 0 2 2 1 1 1 0 2 1

1 2 1 0 2 2 0 2 1 0 0 1 1 1 1 0 0 2 2 0 0 0 2 2 1 1 1 2 1 0 1 2 0 2 2 2 1

2 0 0 2 1 0 2 0 2 1 0 1 0 2 1 1 1 2 1 1 2 0 0 0 0 2 1 2 0 0 2 2 1 2 0 0 2

0 0 0 1 1 1 1 0 0 1 2 1 0 1 2 0 1 0 0 1 0 1 0 2 1 2 2 1 0 2 1 2 2 0 1 1 2

0 2 0 1 2 0 0 2 0 1 1 0 1 0 0 1 1 2 1 1 0 2 0 2 2 1 2 1 1 2 2 0 2 2 2 0 1

1 2 0 2 0 0 1 1 1 2 1 1 1 0 0 2 1 2 2 2 0 1 1 1 1 1 1 0 0 1 0 0 0 0 1 0 0

1 1 2 0 2 2 0 2 0 2 0 1 1 0 1 1 1 0 2 2 0 1 1 0 1 0 0 1 0 0 2 0 2 2 2 1 0

0 0 2 1 2 0 2 0 0 1 2 2 2 0 1 1 1 2 1 0 0 1 2 2 0 2 2 0 2 1 2 0 0 0 0 1 0

0 1 1 2]

In [2]:

+100 XP

Great work! You've successfully performed k-Means clustering and predicted the labels of new points. But it is not easy to inspect the clustering by just looking at the printed labels. A visualization would be far more useful. In the next exercise, you'll inspect your clustering with a scatter plot!

**Exercise**

**Exercise**

**Inspect your clustering**

Let's now inspect the clustering you performed in the previous exercise!

A solution to the previous exercise has already run, so new\_points is an array of points and labels is the array of their cluster labels.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Import matplotlib.pyplot as plt.
* Assign column 0 of new\_points to xs, and column 1 of new\_points to ys.
* Make a scatter plot of xs and ys, specifying the c=labels keyword arguments to color the points by their cluster label. Also specify alpha=0.5.
* Compute the coordinates of the centroids using the .cluster\_centers\_ attribute of model.
* Assign column 0 of centroids to centroids\_x, and column 1 of centroids to centroids\_y.
* Make a scatter plot of centroids\_x and centroids\_y, using 'D' (a diamond) as a marker by specifying the marker parameter. Set the size of the markers to be 50 using s=50.

[**Take Hint (-30 XP)**](javascript:void(0))

# Import pyplot

import matplotlib.pyplot as plt

# Assign the columns of new\_points: xs and ys

xs = new\_points[:, 0]

ys = new\_points[:, 1]

# Make a scatter plot of xs and ys, using labels to define the colors

plt.scatter(xs, ys, alpha=0.5, c=labels)

# Assign the cluster centers: centroids

centroids = model.cluster\_centers\_

# Assign the columns of centroids: centroids\_x, centroids\_y

centroids\_x = centroids[:,0]

centroids\_y = centroids[:,1]

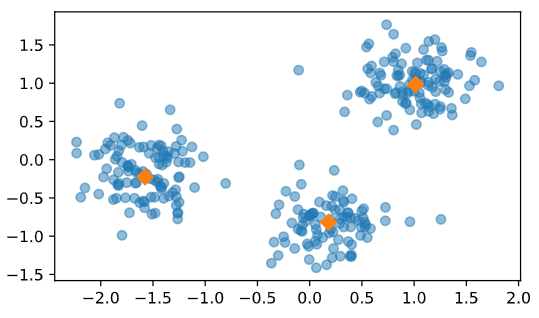
# Make a scatter plot oaf centroids\_x and centroids\_y

plt.scatter(centroids\_x, centroids\_y, marker='D', s=50)

plt.show()

**Incorrect Submission**

Check your call of plt.scatter(). Did you specify the argument c?



+100 XP

Fantastic! The clustering looks great! But how can you be sure that 3 clusters is the correct choice? In other words, how can you evaluate the quality of a clustering? Tune into the next video in which Ben will explain how to evaluate a clustering!

**Exercise**

**Exercise**

**How many clusters of grain?**

In the video, you learned how to choose a good number of clusters for a dataset using the k-means inertia graph. You are given an array samples containing the measurements (such as area, perimeter, length, and several others) of samples of grain. What's a good number of clusters in this case?

KMeans and PyPlot (plt) have already been imported for you.

This dataset was sourced from the [**UCI Machine Learning Repository**](https://archive.ics.uci.edu/ml/datasets/seeds).

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* For each of the given values of k, perform the following steps:
* Create a KMeans instance called model with k clusters.
* Fit the model to the grain data samples.
* Append the value of the inertia\_ attribute of model to the list inertias.
* The code to plot ks vs inertias has been written for you, so hit 'Submit Answer' to see the plot!

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Did you call KMeans() with the right argument? In each iteration of the for loop, you have to create a KMeans instance called model with k clusters.

Check the first for loop. Did you correctly specify the body? Did you call inertias.append()?

ks = range(1, 6)

inertias = []

for k in ks:

# Create a KMeans instance with k clusters: model

model = KMeans(n\_clusters=k)

# Fit model to samples

model.fit(samples)

# Append the inertia to the list of inertias

#inertias.append(model.inertia\_)

inertias += [model.inertia\_]

# Plot ks vs inertias

plt.plot(ks, inertias, '-o')

plt.xlabel('number of clusters, k')

plt.ylabel('inertia')

plt.xticks(ks)

plt.show()

In [1]: help(KMeans)

Help on class KMeans in module sklearn.cluster.k\_means\_:

class KMeans(sklearn.base.BaseEstimator, sklearn.base.ClusterMixin, sklearn.base.TransformerMixin)

| K-Means clustering

|

| Read more in the :ref:`User Guide <k\_means>`.

|

| Parameters

| ----------

|

| n\_clusters : int, optional, default: 8

| The number of clusters to form as well as the number of

| centroids to generate.

|

| init : {'k-means++', 'random' or an ndarray}

| Method for initialization, defaults to 'k-means++':

|

| 'k-means++' : selects initial cluster centers for k-mean

| clustering in a smart way to speed up convergence. See section

| Notes in k\_init for more details.

|

| 'random': choose k observations (rows) at random from data for

| the initial centroids.

|

| If an ndarray is passed, it should be of shape (n\_clusters, n\_features)

| and gives the initial centers.

|

| n\_init : int, default: 10

| Number of time the k-means algorithm will be run with different

| centroid seeds. The final results will be the best output of

| n\_init consecutive runs in terms of inertia.

|

| max\_iter : int, default: 300

| Maximum number of iterations of the k-means algorithm for a

| single run.

|

| tol : float, default: 1e-4

| Relative tolerance with regards to inertia to declare convergence

|

| precompute\_distances : {'auto', True, False}

| Precompute distances (faster but takes more memory).

|

| 'auto' : do not precompute distances if n\_samples \* n\_clusters > 12

| million. This corresponds to about 100MB overhead per job using

| double precision.

|

| True : always precompute distances

|

| False : never precompute distances

|

| verbose : int, default 0

| Verbosity mode.

|

| random\_state : int, RandomState instance or None (default)

| Determines random number generation for centroid initialization. Use

| an int to make the randomness deterministic.

| See :term:`Glossary <random\_state>`.

|

| copy\_x : boolean, optional

| When pre-computing distances it is more numerically accurate to center

| the data first. If copy\_x is True (default), then the original data is

| not modified, ensuring X is C-contiguous. If False, the original data

| is modified, and put back before the function returns, but small

| numerical differences may be introduced by subtracting and then adding

| the data mean, in this case it will also not ensure that data is

| C-contiguous which may cause a significant slowdown.

|

| n\_jobs : int or None, optional (default=None)

| The number of jobs to use for the computation. This works by computing

| each of the n\_init runs in parallel.

|

| ``None`` means 1 unless in a :obj:`joblib.parallel\_backend` context.

| ``-1`` means using all processors. See :term:`Glossary <n\_jobs>`

| for more details.

|

| algorithm : "auto", "full" or "elkan", default="auto"

| K-means algorithm to use. The classical EM-style algorithm is "full".

| The "elkan" variation is more efficient by using the triangle

| inequality, but currently doesn't support sparse data. "auto" chooses

| "elkan" for dense data and "full" for sparse data.

|

| Attributes

| ----------

| cluster\_centers\_ : array, [n\_clusters, n\_features]

| Coordinates of cluster centers. If the algorithm stops before fully

| converging (see ``tol`` and ``max\_iter``), these will not be

| consistent with ``labels\_``.

|

| labels\_ :

| Labels of each point

|

| inertia\_ : float

| Sum of squared distances of samples to their closest cluster center.

|

| n\_iter\_ : int

| Number of iterations run.

|

| Examples

| --------

|

| >>> from sklearn.cluster import KMeans

| >>> import numpy as np

| >>> X = np.array([[1, 2], [1, 4], [1, 0],

| ... [10, 2], [10, 4], [10, 0]])

| >>> kmeans = KMeans(n\_clusters=2, random\_state=0).fit(X)

| >>> kmeans.labels\_

| array([1, 1, 1, 0, 0, 0], dtype=int32)

| >>> kmeans.predict([[0, 0], [12, 3]])

| array([1, 0], dtype=int32)

| >>> kmeans.cluster\_centers\_

| array([[10., 2.],

| [ 1., 2.]])

|

| See also

| --------

|

| MiniBatchKMeans

| Alternative online implementation that does incremental updates

| of the centers positions using mini-batches.

| For large scale learning (say n\_samples > 10k) MiniBatchKMeans is

| probably much faster than the default batch implementation.

|

| Notes

| -----

| The k-means problem is solved using either Lloyd's or Elkan's algorithm.

|

| The average complexity is given by O(k n T), were n is the number of

| samples and T is the number of iteration.

|

| The worst case complexity is given by O(n^(k+2/p)) with

| n = n\_samples, p = n\_features. (D. Arthur and S. Vassilvitskii,

| 'How slow is the k-means method?' SoCG2006)

|

| In practice, the k-means algorithm is very fast (one of the fastest

| clustering algorithms available), but it falls in local minima. That's why

| it can be useful to restart it several times.

|

| If the algorithm stops before fully converging (because of ``tol`` or

| ``max\_iter``), ``labels\_`` and ``cluster\_centers\_`` will not be consistent,

| i.e. the ``cluster\_centers\_`` will not be the means of the points in each

| cluster. Also, the estimator will reassign ``labels\_`` after the last

| iteration to make ``labels\_`` consistent with ``predict`` on the training

| set.

|

| Method resolution order:

| KMeans

| sklearn.base.BaseEstimator

| sklearn.base.ClusterMixin

| sklearn.base.TransformerMixin

| builtins.object

|

| Methods defined here:

|

| \_\_init\_\_(self, n\_clusters=8, init='k-means++', n\_init=10, max\_iter=300, tol=0.0001, precompute\_distances='auto', verbose=0, random\_state=None, copy\_x=True, n\_jobs=None, algorithm='auto')

| Initialize self. See help(type(self)) for accurate signature.

|

| fit(self, X, y=None, sample\_weight=None)

| Compute k-means clustering.

|

| Parameters

| ----------

| X : array-like or sparse matrix, shape=(n\_samples, n\_features)

| Training instances to cluster. It must be noted that the data

| will be converted to C ordering, which will cause a memory

| copy if the given data is not C-contiguous.

|

| y : Ignored

| not used, present here for API consistency by convention.

|

| sample\_weight : array-like, shape (n\_samples,), optional

| The weights for each observation in X. If None, all observations

| are assigned equal weight (default: None)

|

| fit\_predict(self, X, y=None, sample\_weight=None)

| Compute cluster centers and predict cluster index for each sample.

|

| Convenience method; equivalent to calling fit(X) followed by

| predict(X).

|

| Parameters

| ----------

| X : {array-like, sparse matrix}, shape = [n\_samples, n\_features]

| New data to transform.

|

| y : Ignored

| not used, present here for API consistency by convention.

|

| sample\_weight : array-like, shape (n\_samples,), optional

| The weights for each observation in X. If None, all observations

| are assigned equal weight (default: None)

|

| Returns

| -------

| labels : array, shape [n\_samples,]

| Index of the cluster each sample belongs to.

|

| fit\_transform(self, X, y=None, sample\_weight=None)

| Compute clustering and transform X to cluster-distance space.

|

| Equivalent to fit(X).transform(X), but more efficiently implemented.

|

| Parameters

| ----------

| X : {array-like, sparse matrix}, shape = [n\_samples, n\_features]

| New data to transform.

|

| y : Ignored

| not used, present here for API consistency by convention.

|

| sample\_weight : array-like, shape (n\_samples,), optional

| The weights for each observation in X. If None, all observations

| are assigned equal weight (default: None)

|

| Returns

| -------

| X\_new : array, shape [n\_samples, k]

| X transformed in the new space.

|

| predict(self, X, sample\_weight=None)

| Predict the closest cluster each sample in X belongs to.

|

| In the vector quantization literature, `cluster\_centers\_` is called

| the code book and each value returned by `predict` is the index of

| the closest code in the code book.

|

| Parameters

| ----------

| X : {array-like, sparse matrix}, shape = [n\_samples, n\_features]

| New data to predict.

|

| sample\_weight : array-like, shape (n\_samples,), optional

| The weights for each observation in X. If None, all observations

| are assigned equal weight (default: None)

|

| Returns

| -------

| labels : array, shape [n\_samples,]

| Index of the cluster each sample belongs to.

|

| score(self, X, y=None, sample\_weight=None)

| Opposite of the value of X on the K-means objective.

|

| Parameters

| ----------

| X : {array-like, sparse matrix}, shape = [n\_samples, n\_features]

| New data.

|

| y : Ignored

| not used, present here for API consistency by convention.

|

| sample\_weight : array-like, shape (n\_samples,), optional

| The weights for each observation in X. If None, all observations

| are assigned equal weight (default: None)

|

| Returns

| -------

| score : float

| Opposite of the value of X on the K-means objective.

|

| transform(self, X)

| Transform X to a cluster-distance space.

|

| In the new space, each dimension is the distance to the cluster

| centers. Note that even if X is sparse, the array returned by

| `transform` will typically be dense.

|

| Parameters

| ----------

| X : {array-like, sparse matrix}, shape = [n\_samples, n\_features]

| New data to transform.

|

| Returns

| -------

| X\_new : array, shape [n\_samples, k]

| X transformed in the new space.

|

| ----------------------------------------------------------------------

| Methods inherited from sklearn.base.BaseEstimator:

|

| \_\_getstate\_\_(self)

|

| \_\_repr\_\_(self, N\_CHAR\_MAX=700)

| Return repr(self).

|

| \_\_setstate\_\_(self, state)

|

| get\_params(self, deep=True)

| Get parameters for this estimator.

|

| Parameters

| ----------

| deep : boolean, optional

| If True, will return the parameters for this estimator and

| contained subobjects that are estimators.

|

| Returns

| -------

| params : mapping of string to any

| Parameter names mapped to their values.

|

| set\_params(self, \*\*params)

| Set the parameters of this estimator.

|

| The method works on simple estimators as well as on nested objects

| (such as pipelines). The latter have parameters of the form

| ``<component>\_\_<parameter>`` so that it's possible to update each

| component of a nested object.

|

| Returns

| -------

| self

|

| ----------------------------------------------------------------------

| Data descriptors inherited from sklearn.base.BaseEstimator:

|

| \_\_dict\_\_

| dictionary for instance variables (if defined)

|

| \_\_weakref\_\_

| list of weak references to the object (if defined)

In [2]:

+100 XP

Excellent job! The inertia decreases very slowly from 3 clusters to 4, so it looks like 3 clusters would be a good choice for this data.

**Exercise**

**Exercise**

**Evaluating the grain clustering**

In the previous exercise, you observed from the inertia plot that 3 is a good number of clusters for the grain data. In fact, the grain samples come from a mix of 3 different grain varieties: "Kama", "Rosa" and "Canadian". In this exercise, cluster the grain samples into three clusters, and compare the clusters to the grain varieties using a cross-tabulation.

You have the array samples of grain samples, and a list varieties giving the grain variety for each sample. Pandas (pd) and KMeans have already been imported for you.

**Instructions**

**100 XP**

* Create a KMeans model called model with 3 clusters.
* Use the .fit\_predict() method of model to fit it to samples and derive the cluster labels. Using .fit\_predict() is the same as using .fit() followed by .predict().
* Create a DataFrame df with two columns named 'labels' and 'varieties', using labels and varieties, respectively, for the column values. This has been done for you.
* Use the pd.crosstab() function on df['labels'] and df['varieties'] to count the number of times each grain variety coincides with each cluster label. Assign the result to ct.
* Hit 'Submit Answer' to see the cross-tabulation!

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Check your call of model.fit\_predict(). Did you correctly specify the first argument? Expected something different.

# Create a KMeans model with 3 clusters: model

model = KMeans(n\_clusters=3)

# Use fit\_predict to fit model and obtain cluster labels: labels

labels = model.fit\_predict(samples)

# Create a DataFrame with labels and varieties as columns: df

df = pd.DataFrame({'labels': labels, 'varieties': varieties})

# Create crosstab: ct

ct = pd.crosstab(df['labels'], df['varieties'])

# Display ct

print(ct)

Traceback (most recent call last):

File "script.py", line 5, in <module>

labels = model.fit\_predict(model)

File "script.py", line 998, in fit\_predict

return self.fit(X, sample\_weight=sample\_weight).labels\_

File "script.py", line 972, in fit

return\_n\_iter=True)

File "script.py", line 312, in k\_means

order=order, copy=copy\_x)

File "script.py", line 496, in check\_array

array = np.asarray(array, dtype=dtype, order=order)

File "script.py", line 85, in asarray

return array(a, dtype, copy=False, order=order)

TypeError: float() argument must be a string or a number, not 'KMeans'

<script.py> output:

varieties Canadian wheat Kama wheat Rosa wheat

labels

0 0 1 60

1 68 9 0

2 2 60 10

In [1]:

+100 XP

Great work! The cross-tabulation shows that the 3 varieties of grain separate really well into 3 clusters. But depending on the type of data you are working with, the clustering may not always be this good. Is there anything you can do in such situations to improve your clustering? You'll find out in the next video!

**Exercise**

**Exercise**

**Scaling fish data for clustering**

You are given an array samples giving measurements of fish. Each row represents an individual fish. The measurements, such as weight in grams, length in centimeters, and the percentage ratio of height to length, have very different scales. In order to cluster this data effectively, you'll need to standardize these features first. In this exercise, you'll build a pipeline to standardize and cluster the data.

These fish measurement data were sourced from the [**Journal of Statistics Education**](http://ww2.amstat.org/publications/jse/jse_data_archive.htm).

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Import:
  + make\_pipeline from sklearn.pipeline.
  + StandardScaler from sklearn.preprocessing.
  + KMeans from sklearn.cluster.
* Create an instance of StandardScaler called scaler.
* Create an instance of KMeans with 4 clusters called kmeans.
* Create a pipeline called pipeline that chains scaler and kmeans. To do this, you just need to pass them in as arguments to make\_pipeline().

[**Take Hint (-30 XP)**](javascript:void(0))

# Perform the necessary imports

from sklearn.pipeline import make\_pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

# Create scaler: scaler

scaler = StandardScaler()

# Create KMeans instance: kmeans

kmeans = KMeans(n\_clusters=4)

# Create pipeline: pipeline

pipeline = make\_pipeline(scaler, kmeans)

+100 XP

Great work! Now that you've built the pipeline, you'll use it in the next exercise to cluster the fish by their measurements.

**Exercise**

**Exercise**

**Clustering the fish data**

You'll now use your standardization and clustering pipeline from the previous exercise to cluster the fish by their measurements, and then create a cross-tabulation to compare the cluster labels with the fish species.

As before, samples is the 2D array of fish measurements. Your pipeline is available as pipeline, and the species of every fish sample is given by the list species.

**Instructions**

**70 XP**

**Instructions**

**70 XP**

* Import pandas as pd.
* Fit the pipeline to the fish measurements samples.
* Obtain the cluster labels for samples by using the .predict() method of pipeline.
* Using pd.DataFrame(), create a DataFrame df with two columns named 'labels' and 'species', using labels and species, respectively, for the column values.
* Using pd.crosstab(), create a cross-tabulation ct of df['labels'] and df['species'].

[**Show Answer (-70 XP)**](javascript:void(0))

**Hint**

* The command import x as y imports x as y.
* Use the .fit() method of pipeline with samples as the argument to fit the pipeline to the fish measurement data.
* To obtain the cluster labels, use pipeline.predict() with samples as the argument.
* To create df, you need to first create a dictionary with 'labels' and 'species' as keys and labels and species as the values. This dictionary should then be passed in as an argument to pd.DataFrame().
* Use df['labels'] and df['species'] as arguments to pd.crosstab() to create the cross-tabulation.

 Awesome, thanks for your feedback!

**Incorrect Submission**

Check your call of pipeline.fit(). Did you correctly specify the first argument? Expected something different.

Did you create the DataFrame df from a dictionary with the correct keys and values? 'labels' and 'species' should be the keys with labels and species as the corresponding values.

# Import pandas

import pandas as pd

# Fit the pipeline to samples

pipeline.fit(samples)

# Calculate the cluster labels: labels

labels = pipeline.predict(samples)

# Create a DataFrame with labels and species as columns: df

df = pd.DataFrame({'labels': labels, 'species': species})

# Create crosstab: ct

ct = pd.crosstab(df.labels, df.species)

# Display ct

print(ct)

In [1]: pipeline.predict

Out[1]: <function sklearn.pipeline.Pipeline.predict>

In [2]: kmeans.fit(pipeline)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

kmeans.fit(pipeline)

File "<stdin>", line 972, in fit

return\_n\_iter=True)

File "<stdin>", line 312, in k\_means

order=order, copy=copy\_x)

File "<stdin>", line 496, in check\_array

array = np.asarray(array, dtype=dtype, order=order)

File "<stdin>", line 85, in asarray

return array(a, dtype, copy=False, order=order)

TypeError: float() argument must be a string or a number, not 'StandardScaler'

In [3]: # Import pandas

import pandas as pd

# Fit the pipeline to samples

samples = pipeline.fit(pipeline)

# Calculate the cluster labels: labels

labels = pipeline.predict(samples)

# Create a DataFrame with labels and species as columns: df

df = \_\_\_\_

# Create crosstab: ct

ct = \_\_\_\_

# Display ct

print(ct)

Traceback (most recent call last):

File "<stdin>", line 5, in <module>

samples = pipeline.fit(pipeline)

File "<stdin>", line 352, in fit

Xt, fit\_params = self.\_fit(X, y, \*\*fit\_params)

File "<stdin>", line 317, in \_fit

\*\*fit\_params\_steps[name])

File "<stdin>", line 352, in \_\_call\_\_

return self.func(\*args, \*\*kwargs)

File "<stdin>", line 716, in \_fit\_transform\_one

res = transformer.fit\_transform(X, y, \*\*fit\_params)

File "<stdin>", line 553, in fit\_transform

return self.fit(X, \*\*fit\_params).transform(X)

File "<stdin>", line 639, in fit

return self.partial\_fit(X, y)

File "<stdin>", line 663, in partial\_fit

force\_all\_finite='allow-nan')

File "<stdin>", line 496, in check\_array

array = np.asarray(array, dtype=dtype, order=order)

File "<stdin>", line 85, in asarray

return array(a, dtype, copy=False, order=order)

TypeError: float() argument must be a string or a number, not 'StandardScaler'

In [4]: # Import pandas

import pandas as pd

# Fit the pipeline to samples

samples = pipeline.fit()

# Calculate the cluster labels: labels

labels = pipeline.predict(samples)

# Create a DataFrame with labels and species as columns: df

df = \_\_\_\_

# Create crosstab: ct

ct = \_\_\_\_

# Display ct

print(ct)

Traceback (most recent call last):

File "<stdin>", line 5, in <module>

samples = pipeline.fit()

TypeError: fit() missing 1 required positional argument: 'X'

Traceback (most recent call last):

File "script.py", line 5, in <module>

samples = pipeline.fit(species)

File "script.py", line 352, in fit

Xt, fit\_params = self.\_fit(X, y, \*\*fit\_params)

File "script.py", line 317, in \_fit

\*\*fit\_params\_steps[name])

File "script.py", line 352, in \_\_call\_\_

return self.func(\*args, \*\*kwargs)

File "script.py", line 716, in \_fit\_transform\_one

res = transformer.fit\_transform(X, y, \*\*fit\_params)

File "script.py", line 553, in fit\_transform

return self.fit(X, \*\*fit\_params).transform(X)

File "script.py", line 639, in fit

return self.partial\_fit(X, y)

File "script.py", line 663, in partial\_fit

force\_all\_finite='allow-nan')

File "script.py", line 496, in check\_array

array = np.asarray(array, dtype=dtype, order=order)

File "script.py", line 85, in asarray

return array(a, dtype, copy=False, order=order)

ValueError: could not convert string to float: 'Bream'

Traceback (most recent call last):

File "script.py", line 8, in <module>

labels = pipeline.predict(samples)

File "script.py", line 116, in <lambda>

out = lambda \*args, \*\*kwargs: self.fn(obj, \*args, \*\*kwargs)

File "script.py", line 421, in predict

Xt = transform.transform(Xt)

File "script.py", line 758, in transform

force\_all\_finite='allow-nan')

File "script.py", line 496, in check\_array

array = np.asarray(array, dtype=dtype, order=order)

File "script.py", line 85, in asarray

return array(a, dtype, copy=False, order=order)

TypeError: float() argument must be a string or a number, not 'StandardScaler'

In [5]: samples

Out[5]:

array([[ 242. , 23.2, 25.4, 30. , 38.4, 13.4],

[ 290. , 24. , 26.3, 31.2, 40. , 13.8],

[ 340. , 23.9, 26.5, 31.1, 39.8, 15.1],

[ 363. , 26.3, 29. , 33.5, 38. , 13.3],

[ 430. , 26.5, 29. , 34. , 36.6, 15.1],

[ 450. , 26.8, 29.7, 34.7, 39.2, 14.2],

[ 500. , 26.8, 29.7, 34.5, 41.1, 15.3],

[ 390. , 27.6, 30. , 35. , 36.2, 13.4],

[ 450. , 27.6, 30. , 35.1, 39.9, 13.8],

[ 500. , 28.5, 30.7, 36.2, 39.3, 13.7],

[ 475. , 28.4, 31. , 36.2, 39.4, 14.1],

[ 500. , 28.7, 31. , 36.2, 39.7, 13.3],

[ 500. , 29.1, 31.5, 36.4, 37.8, 12. ],

[ 600. , 29.4, 32. , 37.2, 40.2, 13.9],

[ 600. , 29.4, 32. , 37.2, 41.5, 15. ],

[ 700. , 30.4, 33. , 38.3, 38.8, 13.8],

[ 700. , 30.4, 33. , 38.5, 38.8, 13.5],

[ 610. , 30.9, 33.5, 38.6, 40.5, 13.3],

[ 650. , 31. , 33.5, 38.7, 37.4, 14.8],

[ 575. , 31.3, 34. , 39.5, 38.3, 14.1],

[ 685. , 31.4, 34. , 39.2, 40.8, 13.7],

[ 620. , 31.5, 34.5, 39.7, 39.1, 13.3],

[ 680. , 31.8, 35. , 40.6, 38.1, 15.1],

[ 700. , 31.9, 35. , 40.5, 40.1, 13.8],

[ 725. , 31.8, 35. , 40.9, 40. , 14.8],

[ 720. , 32. , 35. , 40.6, 40.3, 15. ],

[ 714. , 32.7, 36. , 41.5, 39.8, 14.1],

[ 850. , 32.8, 36. , 41.6, 40.6, 14.9],

[1000. , 33.5, 37. , 42.6, 44.5, 15.5],

[ 920. , 35. , 38.5, 44.1, 40.9, 14.3],

[ 955. , 35. , 38.5, 44. , 41.1, 14.3],

[ 925. , 36.2, 39.5, 45.3, 41.4, 14.9],

[ 975. , 37.4, 41. , 45.9, 40.6, 14.7],

[ 950. , 38. , 41. , 46.5, 37.9, 13.7],

[ 40. , 12.9, 14.1, 16.2, 25.6, 14. ],

[ 69. , 16.5, 18.2, 20.3, 26.1, 13.9],

[ 78. , 17.5, 18.8, 21.2, 26.3, 13.7],

[ 87. , 18.2, 19.8, 22.2, 25.3, 14.3],

[ 120. , 18.6, 20. , 22.2, 28. , 16.1],

[ 0. , 19. , 20.5, 22.8, 28.4, 14.7],

[ 110. , 19.1, 20.8, 23.1, 26.7, 14.7],

[ 120. , 19.4, 21. , 23.7, 25.8, 13.9],

[ 150. , 20.4, 22. , 24.7, 23.5, 15.2],

[ 145. , 20.5, 22. , 24.3, 27.3, 14.6],

[ 160. , 20.5, 22.5, 25.3, 27.8, 15.1],

[ 140. , 21. , 22.5, 25. , 26.2, 13.3],

[ 160. , 21.1, 22.5, 25. , 25.6, 15.2],

[ 169. , 22. , 24. , 27.2, 27.7, 14.1],

[ 161. , 22. , 23.4, 26.7, 25.9, 13.6],

[ 200. , 22.1, 23.5, 26.8, 27.6, 15.4],

[ 180. , 23.6, 25.2, 27.9, 25.4, 14. ],

[ 290. , 24. , 26. , 29.2, 30.4, 15.4],

[ 272. , 25. , 27. , 30.6, 28. , 15.6],

[ 390. , 29.5, 31.7, 35. , 27.1, 15.3],

[ 6.7, 9.3, 9.8, 10.8, 16.1, 9.7],

[ 7.5, 10. , 10.5, 11.6, 17. , 10. ],

[ 7. , 10.1, 10.6, 11.6, 14.9, 9.9],

[ 9.7, 10.4, 11. , 12. , 18.3, 11.5],

[ 9.8, 10.7, 11.2, 12.4, 16.8, 10.3],

[ 8.7, 10.8, 11.3, 12.6, 15.7, 10.2],

[ 10. , 11.3, 11.8, 13.1, 16.9, 9.8],

[ 9.9, 11.3, 11.8, 13.1, 16.9, 8.9],

[ 9.8, 11.4, 12. , 13.2, 16.7, 8.7],

[ 12.2, 11.5, 12.2, 13.4, 15.6, 10.4],

[ 13.4, 11.7, 12.4, 13.5, 18. , 9.4],

[ 12.2, 12.1, 13. , 13.8, 16.5, 9.1],

[ 19.7, 13.2, 14.3, 15.2, 18.9, 13.6],

[ 19.9, 13.8, 15. , 16.2, 18.1, 11.6],

[ 200. , 30. , 32.3, 34.8, 16. , 9.7],

[ 300. , 31.7, 34. , 37.8, 15.1, 11. ],

[ 300. , 32.7, 35. , 38.8, 15.3, 11.3],

[ 300. , 34.8, 37.3, 39.8, 15.8, 10.1],

[ 430. , 35.5, 38. , 40.5, 18. , 11.3],

[ 345. , 36. , 38.5, 41. , 15.6, 9.7],

[ 456. , 40. , 42.5, 45.5, 16. , 9.5],

[ 510. , 40. , 42.5, 45.5, 15. , 9.8],

[ 540. , 40.1, 43. , 45.8, 17. , 11.2],

[ 500. , 42. , 45. , 48. , 14.5, 10.2],

[ 567. , 43.2, 46. , 48.7, 16. , 10. ],

[ 770. , 44.8, 48. , 51.2, 15. , 10.5],

[ 950. , 48.3, 51.7, 55.1, 16.2, 11.2],

[1250. , 52. , 56. , 59.7, 17.9, 11.7],

[1600. , 56. , 60. , 64. , 15. , 9.6],

[1550. , 56. , 60. , 64. , 15. , 9.6],

[1650. , 59. , 63.4, 68. , 15.9, 11. ]])

Traceback (most recent call last):

File "script.py", line 5, in <module>

samples = pipeline.fit(species)

File "script.py", line 352, in fit

Xt, fit\_params = self.\_fit(X, y, \*\*fit\_params)

File "script.py", line 317, in \_fit

\*\*fit\_params\_steps[name])

File "script.py", line 352, in \_\_call\_\_

return self.func(\*args, \*\*kwargs)

File "script.py", line 716, in \_fit\_transform\_one

res = transformer.fit\_transform(X, y, \*\*fit\_params)

File "script.py", line 553, in fit\_transform

return self.fit(X, \*\*fit\_params).transform(X)

File "script.py", line 639, in fit

return self.partial\_fit(X, y)

File "script.py", line 663, in partial\_fit

force\_all\_finite='allow-nan')

File "script.py", line 496, in check\_array

array = np.asarray(array, dtype=dtype, order=order)

File "script.py", line 85, in asarray

return array(a, dtype, copy=False, order=order)

ValueError: could not convert string to float: 'Bream'

In [6]: species

Out[6]:

['Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Bream',

'Roach',

'Roach',

'Roach',

'Roach',

'Roach',

'Roach',

'Roach',

'Roach',

'Roach',

'Roach',

'Roach',

'Roach',

'Roach',

'Roach',

'Roach',

'Roach',

'Roach',

'Roach',

'Roach',

'Roach',

'Smelt',

'Smelt',

'Smelt',

'Smelt',

'Smelt',

'Smelt',

'Smelt',

'Smelt',

'Smelt',

'Smelt',

'Smelt',

'Smelt',

'Smelt',

'Smelt',

'Pike',

'Pike',

'Pike',

'Pike',

'Pike',

'Pike',

'Pike',

'Pike',

'Pike',

'Pike',

'Pike',

'Pike',

'Pike',

'Pike',

'Pike',

'Pike',

'Pike']

Traceback (most recent call last):

File "script.py", line 8, in <module>

labels = pipeline.predict(samples)

File "script.py", line 116, in <lambda>

out = lambda \*args, \*\*kwargs: self.fn(obj, \*args, \*\*kwargs)

File "script.py", line 421, in predict

Xt = transform.transform(Xt)

File "script.py", line 758, in transform

force\_all\_finite='allow-nan')

File "script.py", line 496, in check\_array

array = np.asarray(array, dtype=dtype, order=order)

File "script.py", line 85, in asarray

return array(a, dtype, copy=False, order=order)

TypeError: float() argument must be a string or a number, not 'StandardScaler'

Traceback (most recent call last):

File "script.py", line 11, in <module>

df = pd.DataFrame({'labels': labels, 'varieties': varieties})

NameError: name 'varieties' is not defined

Traceback (most recent call last):

File "script.py", line 11, in <module>

df = pd.DataFrame({'labels': labels, 'varieties': varieties})

NameError: name 'varieties' is not defined

<script.py> output:

species Bream Pike Roach Smelt

labels

0 0 0 0 13

1 33 0 1 0

2 0 17 0 0

3 1 0 19 1

In [7]:

+0 XP

Excellent! It looks like the fish data separates really well into 4 clusters!

**Exercise**

**Exercise**

**Clustering stocks using KMeans**

In this exercise, you'll cluster companies using their daily stock price movements (i.e. the dollar difference between the closing and opening prices for each trading day). You are given a NumPy array movements of daily price movements from 2010 to 2015 (obtained from Yahoo! Finance), where each row corresponds to a company, and each column corresponds to a trading day.

Some stocks are more expensive than others. To account for this, include a Normalizer at the beginning of your pipeline. The Normalizer will separately transform each company's stock price to a relative scale before the clustering begins.

Note that Normalizer() is different to StandardScaler(), which you used in the previous exercise. While StandardScaler() standardizes **features** (such as the features of the fish data from the previous exercise) by removing the mean and scaling to unit variance, Normalizer() rescales **each sample** - here, each company's stock price - independently of the other.

KMeans and make\_pipeline have already been imported for you.

**Instructions**

**100 XP**

* Import Normalizer from sklearn.preprocessing.
* Create an instance of Normalizer called normalizer.
* Create an instance of KMeans called kmeans with 10 clusters.
* Using make\_pipeline(), create a pipeline called pipeline that chains normalizer and kmeans.
* Fit the pipeline to the movements array.

[**Take Hint (-30 XP)**](javascript:void(0))

# Import Normalizer

from sklearn.preprocessing import Normalizer

# Create a normalizer: normalizer

normalizer = Normalizer()

# Create a KMeans model with 10 clusters: kmeans

kmeans = KMeans(n\_clusters=10)

# Make a pipeline chaining normalizer and kmeans: pipeline

pipeline = make\_pipeline(normalizer, kmeans)

# Fit pipeline to the daily price movements

pipeline.fit(movements)

+100 XP

Great work - you're really getting the hang of this. Now that your pipeline has been set up, you can find out which stocks move together in the next exercise!

**Exercise**

**Exercise**

**Which stocks move together?**

In the previous exercise, you clustered companies by their daily stock price movements. So which company have stock prices that tend to change in the same way? You'll now inspect the cluster labels from your clustering to find out.

Your solution to the previous exercise has already been run. Recall that you constructed a Pipeline pipeline containing a KMeans model and fit it to the NumPy array movements of daily stock movements. In addition, a list companies of the company names is available.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Import pandas as pd.
* Use the .predict() method of the pipeline to predict the labels for movements.
* Align the cluster labels with the list of company names companies by creating a DataFrame df with labels and companies as columns. This has been done for you.
* Use the .sort\_values() method of df to sort the DataFrame by the 'labels' column, and print the result.
* Hit 'Submit Answer' and take a moment to see which companies are together in each cluster!

[**Take Hint (-30 XP)**](javascript:void(0))

# Import pandas

import pandas as pd

# Predict the cluster labels: labels

labels = pipeline.predict(movements)

# Create a DataFrame aligning labels and companies: df

df = pd.DataFrame({'labels': labels, 'companies': companies})

# Display df sorted by cluster label

#print(df.sort\_values('labels'))

print(df.sort\_values(['labels']))

<script.py> output:

labels companies

59 0 Yahoo

15 0 Ford

35 0 Navistar

26 1 JPMorgan Chase

16 1 General Electrics

58 1 Xerox

11 1 Cisco

18 1 Goldman Sachs

20 1 Home Depot

5 1 Bank of America

3 1 American express

55 1 Wells Fargo

1 1 AIG

38 2 Pepsi

40 2 Procter Gamble

28 2 Coca Cola

27 2 Kimberly-Clark

9 2 Colgate-Palmolive

54 3 Walgreen

36 3 Northrop Grumman

29 3 Lookheed Martin

4 3 Boeing

0 4 Apple

47 4 Symantec

33 4 Microsoft

32 4 3M

31 4 McDonalds

30 4 MasterCard

50 4 Taiwan Semiconductor Manufacturing

14 4 Dell

17 4 Google/Alphabet

24 4 Intel

23 4 IBM

2 4 Amazon

51 4 Texas instruments

43 4 SAP

45 5 Sony

48 5 Toyota

21 5 Honda

22 5 HP

34 5 Mitsubishi

7 5 Canon

56 6 Wal-Mart

57 7 Exxon

44 7 Schlumberger

8 7 Caterpillar

10 7 ConocoPhillips

12 7 Chevron

13 7 DuPont de Nemours

53 7 Valero Energy

39 8 Pfizer

41 8 Philip Morris

25 8 Johnson & Johnson

49 9 Total

46 9 Sanofi-Aventis

37 9 Novartis

42 9 Royal Dutch Shell

19 9 GlaxoSmithKline

52 9 Unilever

6 9 British American Tobacco

<script.py> output:

labels companies

59 0 Yahoo

15 0 Ford

35 0 Navistar

26 1 JPMorgan Chase

16 1 General Electrics

58 1 Xerox

11 1 Cisco

18 1 Goldman Sachs

20 1 Home Depot

5 1 Bank of America

3 1 American express

55 1 Wells Fargo

1 1 AIG

38 2 Pepsi

40 2 Procter Gamble

28 2 Coca Cola

27 2 Kimberly-Clark

9 2 Colgate-Palmolive

54 3 Walgreen

36 3 Northrop Grumman

29 3 Lookheed Martin

4 3 Boeing

0 4 Apple

47 4 Symantec

33 4 Microsoft

32 4 3M

31 4 McDonalds

30 4 MasterCard

50 4 Taiwan Semiconductor Manufacturing

14 4 Dell

17 4 Google/Alphabet

24 4 Intel

23 4 IBM

2 4 Amazon

51 4 Texas instruments

43 4 SAP

45 5 Sony

48 5 Toyota

21 5 Honda

22 5 HP

34 5 Mitsubishi

7 5 Canon

56 6 Wal-Mart

57 7 Exxon

44 7 Schlumberger

8 7 Caterpillar

10 7 ConocoPhillips

12 7 Chevron

13 7 DuPont de Nemours

53 7 Valero Energy

39 8 Pfizer

41 8 Philip Morris

25 8 Johnson & Johnson

49 9 Total

46 9 Sanofi-Aventis

37 9 Novartis

42 9 Royal Dutch Shell

19 9 GlaxoSmithKline

52 9 Unilever

6 9 British American Tobacco

In [1]:

+100 XP

Fantastic job - you have completed Chapter 1! Take a look at the clusters. Are you surprised by any of the results? In the next chapter, you'll learn about how to communicate results such as this through visualizations.

**How many merges?**

If there are 5 data samples, how many merge operations will occur in a hierarchical clustering? To help answer this question, think back to the video, in which Ben walked through an example of hierarchical clustering using 6 countries. How many merge operations did that example have?

**Answer the question**

**50 XP**

**Possible Answers**

4 merges.

press

1

5 merges.

press

2

This can't be known in advance.

press

3

+50 XP

Well done! With 5 data samples, there would be 4 merge operations, and with 6 data samples, there would be 5 merges, and so on.

**Exercise**

**Exercise**

**Hierarchical clustering of the grain data**

In the video, you learned that the SciPy linkage() function performs hierarchical clustering on an array of samples. Use the linkage() function to obtain a hierarchical clustering of the grain samples, and use dendrogram() to visualize the result. A sample of the grain measurements is provided in the array samples, while the variety of each grain sample is given by the list varieties.

**Instructions**

**100 XP**

* Import:
  + linkage and dendrogram from scipy.cluster.hierarchy.
  + matplotlib.pyplot as plt.
* Perform hierarchical clustering on samples using the linkage() function with the method='complete' keyword argument. Assign the result to mergings.
* Plot a dendrogram using the dendrogram() function on mergings. Specify the keyword arguments labels=varieties, leaf\_rotation=90, and leaf\_font\_size=6.

[**Take Hint (-30 XP)**](javascript:void(0))

# Perform the necessary imports

from scipy.cluster.hierarchy import linkage, dendrogram

import matplotlib.pyplot as plt

# Calculate the linkage: mergings

mergings = linkage(samples, method='complete')

# Plot the dendrogram, using varieties as labels

dendrogram(mergings,

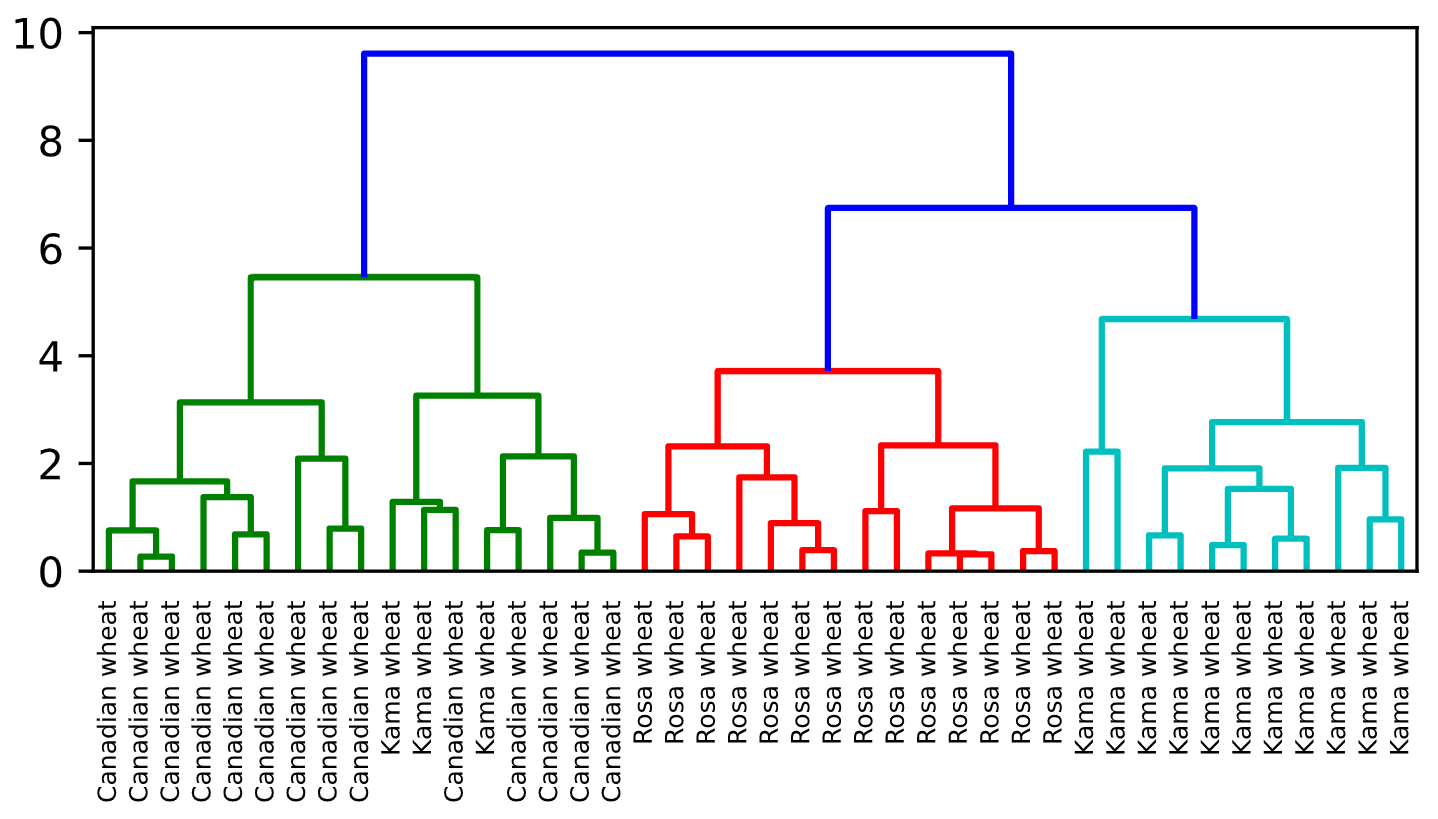
labels=varieties,

leaf\_rotation=90,

leaf\_font\_size=6,

)

plt.show()



In [1]: samples

Out[1]:

array([[14.88 , 14.57 , 0.8811, 5.554 , 3.333 , 1.018 , 4.956 ],

[14.69 , 14.49 , 0.8799, 5.563 , 3.259 , 3.586 , 5.219 ],

[14.03 , 14.16 , 0.8796, 5.438 , 3.201 , 1.717 , 5.001 ],

[13.99 , 13.83 , 0.9183, 5.119 , 3.383 , 5.234 , 4.781 ],

[14.11 , 14.26 , 0.8722, 5.52 , 3.168 , 2.688 , 5.219 ],

[13.02 , 13.76 , 0.8641, 5.395 , 3.026 , 3.373 , 4.825 ],

[15.49 , 14.94 , 0.8724, 5.757 , 3.371 , 3.412 , 5.228 ],

[16.2 , 15.27 , 0.8734, 5.826 , 3.464 , 2.823 , 5.527 ],

[13.5 , 13.85 , 0.8852, 5.351 , 3.158 , 2.249 , 5.176 ],

[15.36 , 14.76 , 0.8861, 5.701 , 3.393 , 1.367 , 5.132 ],

[15.78 , 14.91 , 0.8923, 5.674 , 3.434 , 5.593 , 5.136 ],

[14.46 , 14.35 , 0.8818, 5.388 , 3.377 , 2.802 , 5.044 ],

[11.23 , 12.63 , 0.884 , 4.902 , 2.879 , 2.269 , 4.703 ],

[14.34 , 14.37 , 0.8726, 5.63 , 3.19 , 1.313 , 5.15 ],

[16.84 , 15.67 , 0.8623, 5.998 , 3.484 , 4.675 , 5.877 ],

[17.32 , 15.91 , 0.8599, 6.064 , 3.403 , 3.824 , 5.922 ],

[18.72 , 16.19 , 0.8977, 6.006 , 3.857 , 5.324 , 5.879 ],

[18.88 , 16.26 , 0.8969, 6.084 , 3.764 , 1.649 , 6.109 ],

[18.76 , 16.2 , 0.8984, 6.172 , 3.796 , 3.12 , 6.053 ],

[19.31 , 16.59 , 0.8815, 6.341 , 3.81 , 3.477 , 6.238 ],

[17.99 , 15.86 , 0.8992, 5.89 , 3.694 , 2.068 , 5.837 ],

[18.85 , 16.17 , 0.9056, 6.152 , 3.806 , 2.843 , 6.2 ],

[19.38 , 16.72 , 0.8716, 6.303 , 3.791 , 3.678 , 5.965 ],

[18.96 , 16.2 , 0.9077, 6.051 , 3.897 , 4.334 , 5.75 ],

[18.14 , 16.12 , 0.8772, 6.059 , 3.563 , 3.619 , 6.011 ],

[18.65 , 16.41 , 0.8698, 6.285 , 3.594 , 4.391 , 6.102 ],

[18.94 , 16.32 , 0.8942, 6.144 , 3.825 , 2.908 , 5.949 ],

[17.36 , 15.76 , 0.8785, 6.145 , 3.574 , 3.526 , 5.971 ],

[13.32 , 13.94 , 0.8613, 5.541 , 3.073 , 7.035 , 5.44 ],

[11.43 , 13.13 , 0.8335, 5.176 , 2.719 , 2.221 , 5.132 ],

[12.01 , 13.52 , 0.8249, 5.405 , 2.776 , 6.992 , 5.27 ],

[11.34 , 12.87 , 0.8596, 5.053 , 2.849 , 3.347 , 5.003 ],

[12.02 , 13.33 , 0.8503, 5.35 , 2.81 , 4.271 , 5.308 ],

[12.44 , 13.59 , 0.8462, 5.319 , 2.897 , 4.924 , 5.27 ],

[11.55 , 13.1 , 0.8455, 5.167 , 2.845 , 6.715 , 4.956 ],

[11.26 , 13.01 , 0.8355, 5.186 , 2.71 , 5.335 , 5.092 ],

[12.46 , 13.41 , 0.8706, 5.236 , 3.017 , 4.987 , 5.147 ],

[11.81 , 13.45 , 0.8198, 5.413 , 2.716 , 4.898 , 5.352 ],

[11.27 , 12.86 , 0.8563, 5.091 , 2.804 , 3.985 , 5.001 ],

[12.79 , 13.53 , 0.8786, 5.224 , 3.054 , 5.483 , 4.958 ],

[12.67 , 13.32 , 0.8977, 4.984 , 3.135 , 2.3 , 4.745 ],

[11.23 , 12.88 , 0.8511, 5.14 , 2.795 , 4.325 , 5.003 ]])

In [2]: samples.linkage

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

samples.linkage

AttributeError: 'numpy.ndarray' object has no attribute 'linkage'

In [3]:

+100 XP

Superb! Dendrograms are a great way to illustrate the arrangement of the clusters produced by hierarchical clustering.

**Exercise**

**Exercise**

**Hierarchies of stocks**

In chapter 1, you used k-means clustering to cluster companies according to their stock price movements. Now, you'll perform hierarchical clustering of the companies. You are given a NumPy array of price movements movements, where the rows correspond to companies, and a list of the company names companies. SciPy hierarchical clustering doesn't fit into a sklearn pipeline, so you'll need to use the normalize() function from sklearn.preprocessing instead of Normalizer.

linkage and dendrogram have already been imported from scipy.cluster.hierarchy, and PyPlot has been imported as plt.

**Instructions**

**100 XP**

* Import normalize from sklearn.preprocessing.
* Rescale the price movements for each stock by using the normalize() function on movements.
* Apply the linkage() function to normalized\_movements, using 'complete' linkage, to calculate the hierarchical clustering. Assign the result to mergings.
* Plot a dendrogram of the hierarchical clustering, using the list companies of company names as the labels. In addition, specify the leaf\_rotation=90, and leaf\_font\_size=6 keyword arguments as you did in the previous exercise.

[**Take Hint (-30 XP)**](javascript:void(0))

# Import normalize

from sklearn.preprocessing import normalize

# Normalize the movements: normalized\_movements

normalized\_movements = normalize(movements)

# Calculate the linkage: mergings

mergings = linkage(normalized\_movements, method='complete')

# Plot the dendrogram

dendrogram(mergings,

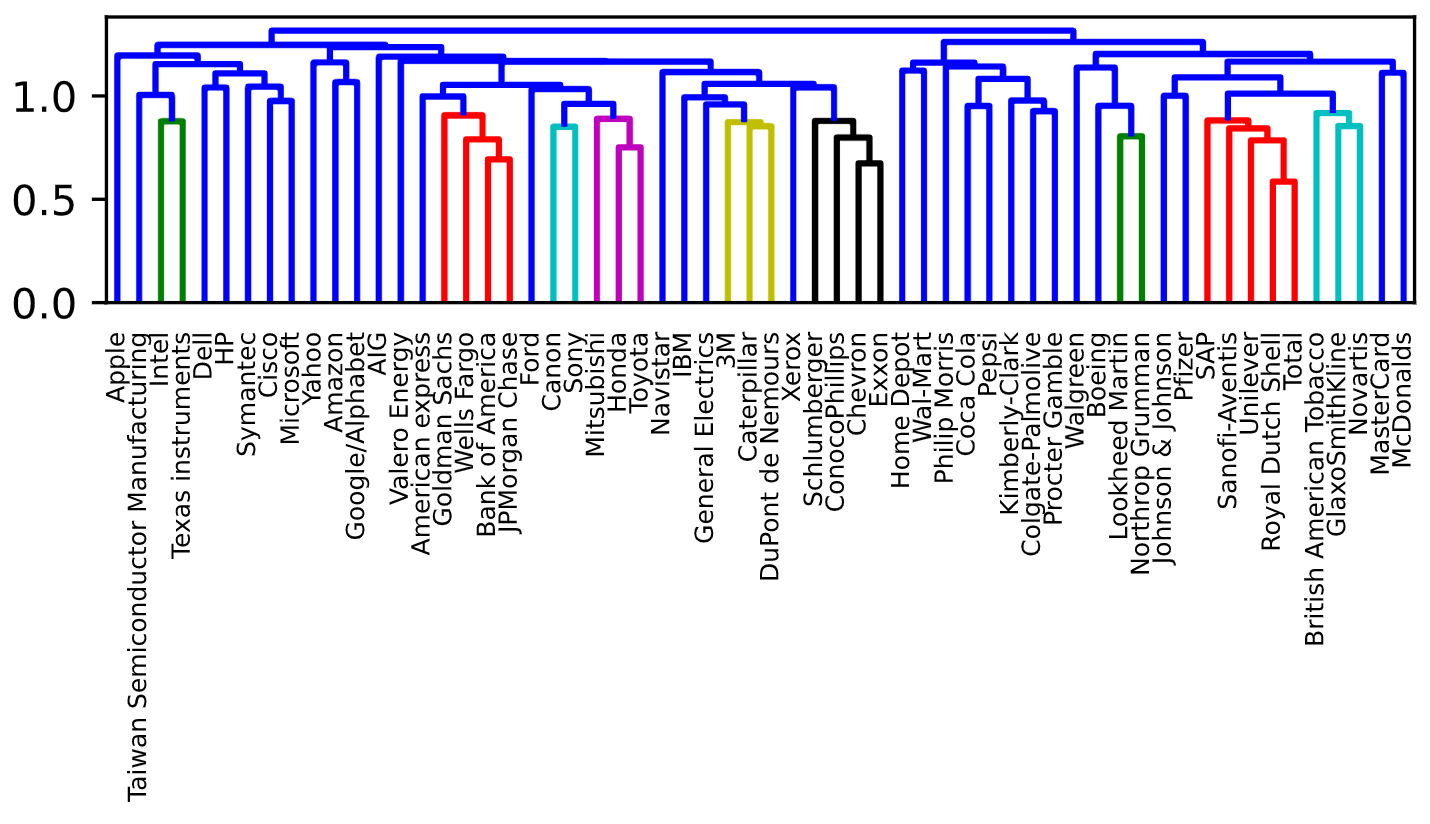
labels=companies,

leaf\_rotation=90,

leaf\_font\_size=6,

)

plt.show()



In [1]: dendrogram

Out[1]: <function scipy.cluster.hierarchy.dendrogram>

In [2]:

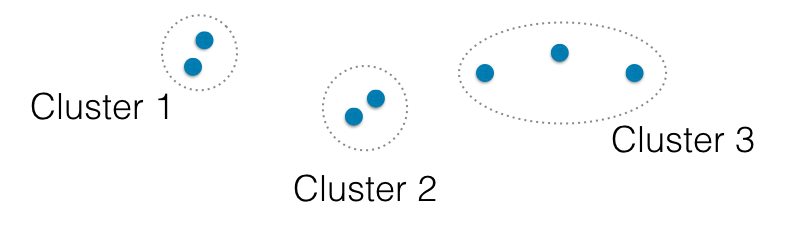
+100 XP

Great work! You can produce great visualizations such as this with hierarchical clustering, but it can be used for more than just visualizations. You'll find out more about this in the next video!

**Which clusters are closest?**

In the video, you learned that the linkage method defines how the distance between clusters is measured. In *complete* linkage, the distance between clusters is the distance between the *furthest* points of the clusters. In *single* linkage, the distance between clusters is the distance between the *closest* points of the clusters.

Consider the three clusters in the diagram. Which of the following statements are true?



**A.** In single linkage, Cluster 3 is the closest to Cluster 2.

**B.** In complete linkage, Cluster 1 is the closest to Cluster 2.

**Answer the question**

**50 XP**

**Possible Answers**

Neither A nor B.

press

1

A only.

press

2

Both A and B.

press

3

+50 XP

Well done!

**Exercise**

**Exercise**

**Different linkage, different hierarchical clustering!**

In the video, you saw a hierarchical clustering of the voting countries at the Eurovision song contest using 'complete' linkage. Now, perform a hierarchical clustering of the voting countries with 'single' linkage, and compare the resulting dendrogram with the one in the video. Different linkage, different hierarchical clustering!

You are given an array samples. Each row corresponds to a voting country, and each column corresponds to a performance that was voted for. The list country\_names gives the name of each voting country. This dataset was obtained from [**Eurovision**](http://www.eurovision.tv/page/results).

**Instructions**

**100 XP**

* Import linkage and dendrogram from scipy.cluster.hierarchy.
* Perform hierarchical clustering on samples using the linkage() function with the method='single' keyword argument. Assign the result to mergings.
* Plot a dendrogram of the hierarchical clustering, using the list country\_names as the labels. In addition, specify the leaf\_rotation=90, and leaf\_font\_size=6 keyword arguments as you have done earlier.

Ctrl+H

[**Take Hint (-30 XP)**](javascript:void(0))

# Perform the necessary imports

import matplotlib.pyplot as plt

from scipy.cluster.hierarchy import linkage, dendrogram

# Calculate the linkage: mergings

mergings = linkage(samples, method='single')

# Plot the dendrogram

dendrogram(mergings,

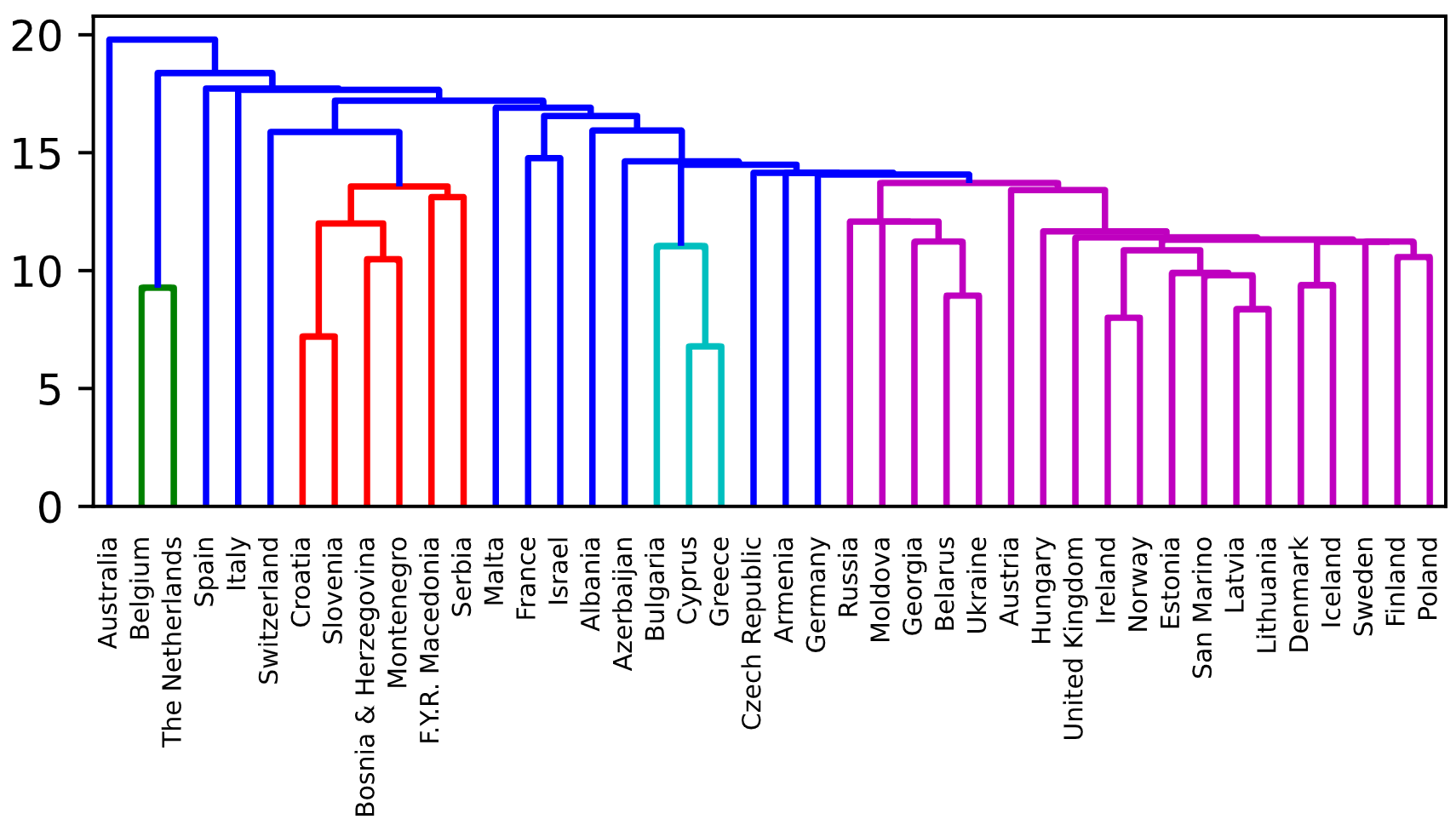
labels=country\_names,

leaf\_rotation=90,

leaf\_font\_size=6,

)

plt.show()



+100 XP

Great work! As you can see, performing single linkage hierarchical clustering produces a different dendrogram!

**Exercise**

**Exercise**

**Intermediate clusterings**

Displayed on the right is the dendrogram for the hierarchical clustering of the grain samples that you computed earlier. If the hierarchical clustering were stopped at height 6 on the dendrogram, how many clusters would there be?

**Instructions**

**50 XP**

**Possible Answers**

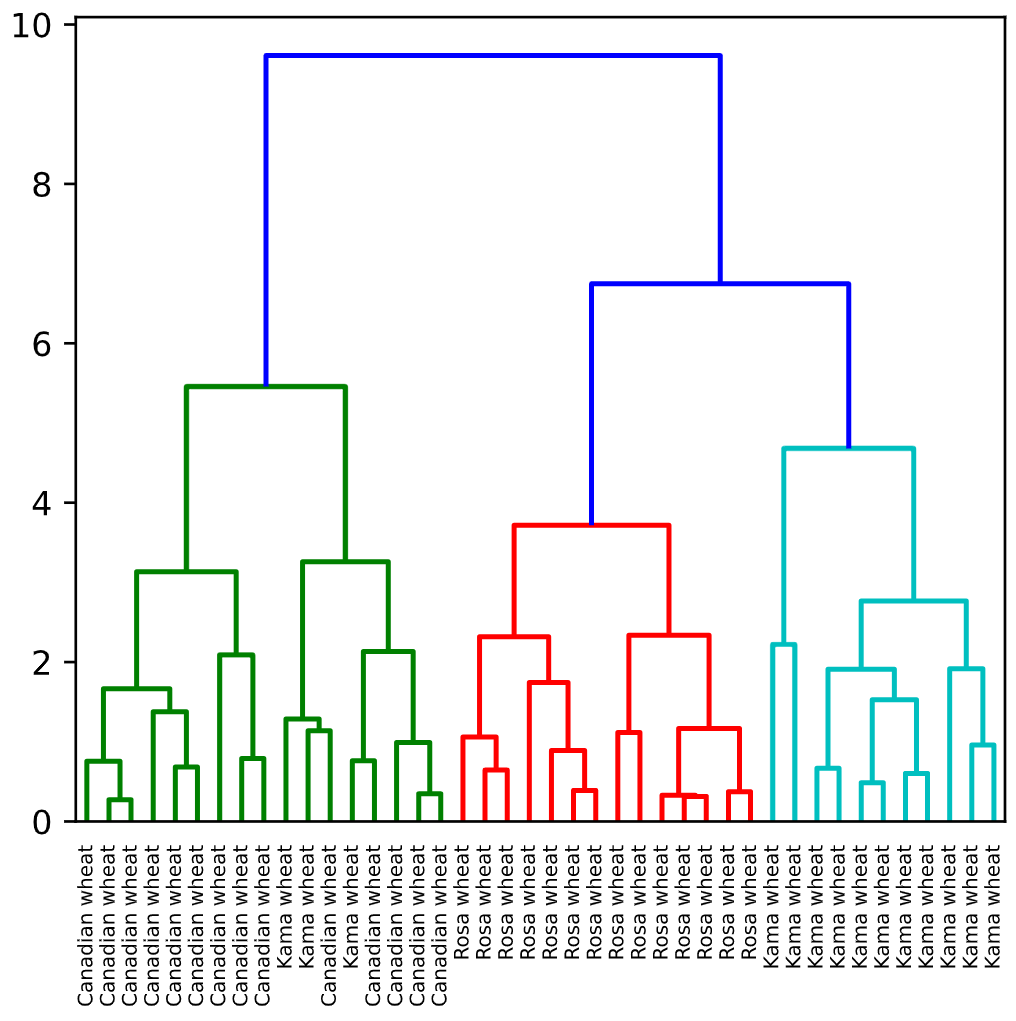
1.

3.

As many as there were at the beginning.

Submit Answer

Take Hint (-15 XP)



**Incorrect Submission**

No, some clusters will have been merged!

+50 XP

Exactly - great work!

**Exercise**

**Exercise**

**Extracting the cluster labels**

In the previous exercise, you saw that the intermediate clustering of the grain samples at height 6 has 3 clusters. Now, use the fcluster() function to extract the cluster labels for this intermediate clustering, and compare the labels with the grain varieties using a cross-tabulation.

The hierarchical clustering has already been performed and mergings is the result of the linkage() function. The list varieties gives the variety of each grain sample.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Import:
  + pandas as pd.
  + fcluster from scipy.cluster.hierarchy.
* Perform a flat hierarchical clustering by using the fcluster() function on mergings. Specify a maximum height of 6 and the keyword argument criterion='distance'.
* Create a DataFrame df with two columns named 'labels' and 'varieties', using labels and varieties, respectively, for the column values. This has been done for you.
* Create a cross-tabulation ct between df['labels'] and df['varieties'] to count the number of times each grain variety coincides with each cluster label.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Check your call of pd.crosstab(). Did you correctly specify the second argument? Running it generated an error: 'DataFrame' object has no attribute 'species'.

 Awesome, thanks for your feedback!

# Perform the necessary imports

import pandas as pd

from scipy.cluster.hierarchy import fcluster

# Use fcluster to extract labels: labels

labels = fcluster(mergings, 6, criterion='distance')

# Create a DataFrame with labels and varieties as columns: df

df = pd.DataFrame({'labels': labels, 'varieties': varieties})

# Create crosstab: ct

ct = pd.crosstab(df.labels, df.varieties)

# Display ct

print(ct)

In [1]: mergings

Out[1]:

array([[33. , 36. , 0.27162909, 2. ],

[21. , 26. , 0.31365739, 2. ],

[18. , 43. , 0.32846589, 3. ],

[38. , 41. , 0.34657328, 2. ],

[19. , 22. , 0.37233454, 2. ],

[15. , 27. , 0.38916958, 2. ],

[ 4. , 11. , 0.48519909, 2. ],

[ 2. , 13. , 0.60220511, 2. ],

[23. , 25. , 0.64447995, 2. ],

[ 0. , 9. , 0.66671658, 2. ],

[32. , 37. , 0.68359363, 2. ],

[39. , 42. , 0.75541297, 3. ],

[12. , 29. , 0.76129577, 2. ],

[30. , 34. , 0.79066703, 2. ],

[24. , 47. , 0.89015184, 3. ],

[ 1. , 6. , 0.96077742, 2. ],

[31. , 45. , 0.98956619, 3. ],

[16. , 50. , 1.05891757, 3. ],

[17. , 20. , 1.11543099, 2. ],

[ 8. , 40. , 1.13733735, 2. ],

[44. , 46. , 1.1662041 , 5. ],

[ 5. , 61. , 1.28676337, 3. ],

[35. , 52. , 1.37690488, 3. ],

[48. , 49. , 1.52865125, 4. ],

[53. , 64. , 1.66517195, 6. ],

[14. , 56. , 1.74234784, 4. ],

[51. , 65. , 1.91015424, 6. ],

[ 7. , 57. , 1.91749035, 3. ],

[28. , 55. , 2.08980038, 3. ],

[54. , 58. , 2.13385537, 5. ],

[ 3. , 10. , 2.22187038, 2. ],

[59. , 67. , 2.31852251, 7. ],

[60. , 62. , 2.33686195, 7. ],

[68. , 69. , 2.76779035, 9. ],

[66. , 70. , 3.13448417, 9. ],

[63. , 71. , 3.25744652, 8. ],

[73. , 74. , 3.71580316, 14. ],

[72. , 75. , 4.68116988, 11. ],

[76. , 77. , 5.45789312, 17. ],

[78. , 79. , 6.74608427, 25. ],

[80. , 81. , 9.61230238, 42. ]])

Traceback (most recent call last):

File "script.py", line 12, in <module>

ct = pd.crosstab(df.labels, df.species)

File "script.py", line 5179, in \_\_getattr\_\_

return object.\_\_getattribute\_\_(self, name)

AttributeError: 'DataFrame' object has no attribute 'species'

<script.py> output:

varieties Canadian wheat Kama wheat Rosa wheat

labels

1 14 3 0

2 0 0 14

3 0 11 0

In [2]:

+100 XP

Fantastic - you've now mastered the fundamentals of k-Means and agglomerative hierarchical clustering. Next, you'll learn about t-SNE, which is a powerful tool for visualizing high dimensional data.

**Exercise**

**Exercise**

**t-SNE visualization of grain dataset**

In the video, you saw t-SNE applied to the iris dataset. In this exercise, you'll apply t-SNE to the grain samples data and inspect the resulting t-SNE features using a scatter plot. You are given an array samples of grain samples and a list variety\_numbers giving the variety number of each grain sample.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Import TSNE from sklearn.manifold.
* Create a TSNE instance called model with learning\_rate=200.
* Apply the .fit\_transform() method of model to samples. Assign the result to tsne\_features.
* Select the column 0 of tsne\_features. Assign the result to xs.
* Select the column 1 of tsne\_features. Assign the result to ys.
* Make a scatter plot of the t-SNE features xs and ys. To color the points by the grain variety, specify the additional keyword argument c=variety\_numbers.

[**Take Hint (-30 XP)**](javascript:void(0))

# Import TSNE

from sklearn.manifold import TSNE

# Create a TSNE instance: model

model = TSNE(learning\_rate=200)

# Apply fit\_transform to samples: tsne\_features

tsne\_features = model.fit\_transform(samples)

# Select the 0th feature: xs

xs = tsne\_features[:,0]

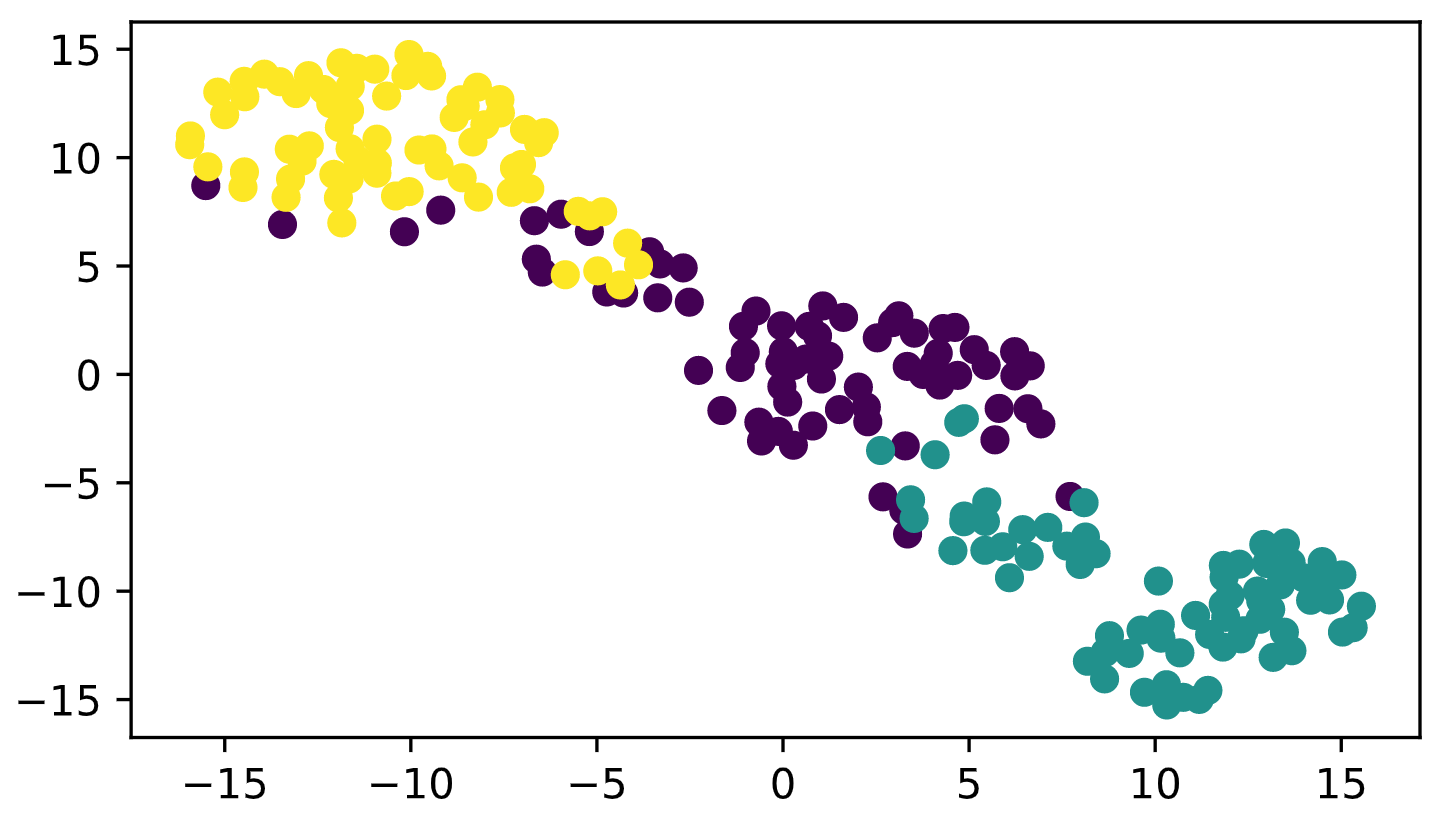
# Select the 1st feature: ys

ys = tsne\_features[:,1]

# Scatter plot, coloring by variety\_numbers

plt.scatter(xs, ys, c=variety\_numbers)

plt.show()



+100 XP

Excellent! As you can see, the t-SNE visualization manages to separate the 3 varieties of grain samples. But how will it perform on the stock data? You'll find out in the next exercise!

**Exercise**

**Exercise**

**A t-SNE map of the stock market**

t-SNE provides great visualizations when the individual samples can be labeled. In this exercise, you'll apply t-SNE to the company stock price data. A scatter plot of the resulting t-SNE features, labeled by the company names, gives you a map of the stock market! The stock price movements for each company are available as the array normalized\_movements (these have already been normalized for you). The list companies gives the name of each company. PyPlot (plt) has been imported for you.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Import TSNE from sklearn.manifold.
* Create a TSNE instance called model with learning\_rate=50.
* Apply the .fit\_transform() method of model to normalized\_movements. Assign the result to tsne\_features.
* Select column 0 and column 1 of tsne\_features.
* Make a scatter plot of the t-SNE features xs and ys. Specify the additional keyword argument alpha=0.5.
* Code to label each point with its company name has been written for you using plt.annotate(), so just hit 'Submit Answer' to see the visualization!

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Did you correctly define the variable xs? Expected something different.

# Import TSNE

from sklearn.manifold import TSNE

# Create a TSNE instance: model

model = TSNE(learning\_rate=50)

# Apply fit\_transform to normalized\_movements: tsne\_features

tsne\_features = model.fit\_transform(normalized\_movements)

# Select the 0th feature: xs

xs = tsne\_features[:,0]

# Select the 1th feature: ys

ys = tsne\_features[:,1]

# Scatter plot

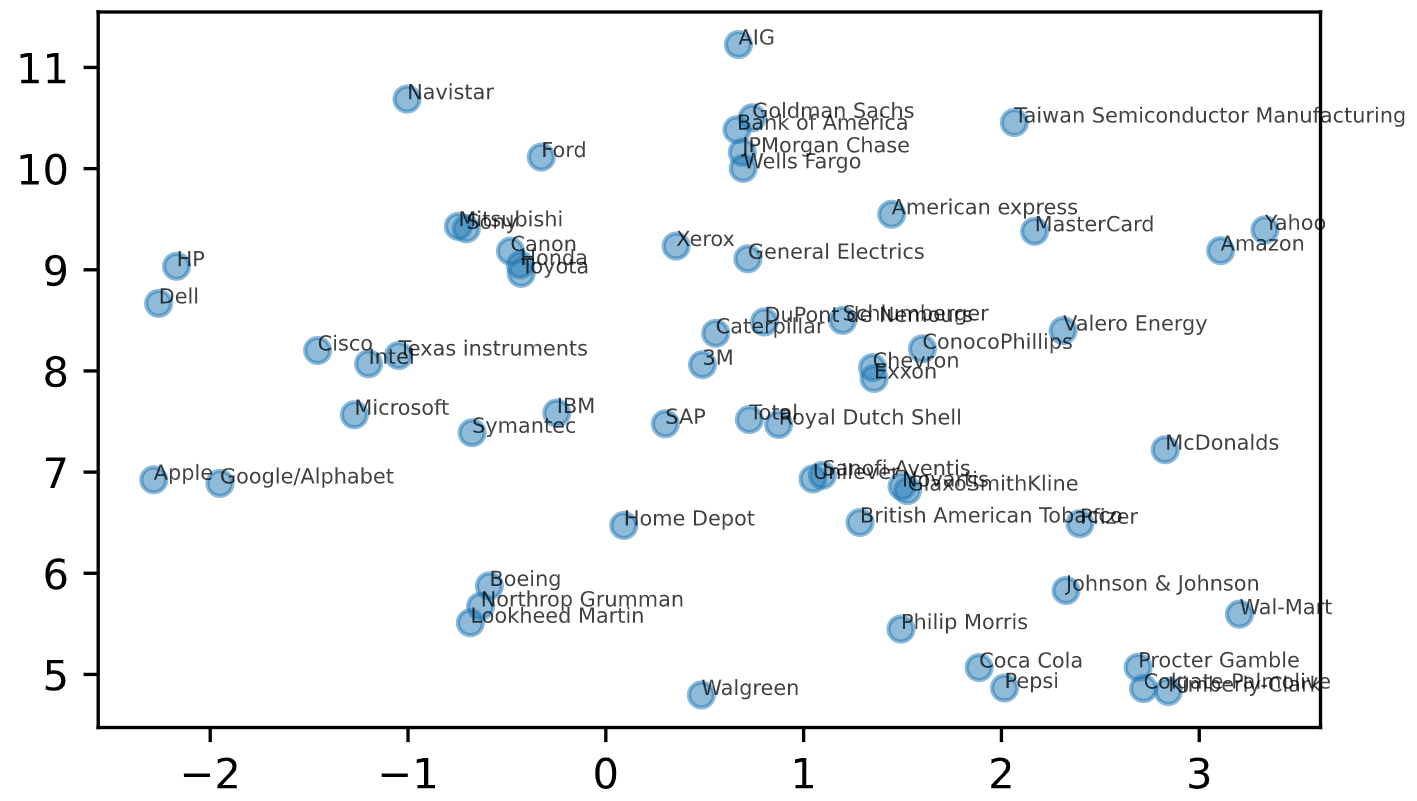
plt.scatter(xs, ys, alpha=0.5)

# Annotate the points

for x, y, company in zip(xs, ys, companies):

plt.annotate(company, (x, y), fontsize=5, alpha=0.75)

plt.show()



In [1]: # Import TSNE

from sklearn.manifold import TSNE

# Create a TSNE instance: model

model = TSNE(learning\_rate=50)

# Apply fit\_transform to normalized\_movements: tsne\_features

tsne\_features = model.fit\_transform(normalized\_movements)

# Select the 0th feature: xs

xs = tsne\_features[:1]

# Select the 1th feature: ys

ys = tsne\_features[:,1]

# Scatter plot

\_\_\_\_

# Annotate the points

for x, y, company in zip(xs, ys, companies):

plt.annotate(company, (x, y), fontsize=5, alpha=0.75)

plt.show()

Traceback (most recent call last):

File "<stdin>", line 17, in <module>

\_\_\_\_

NameError: name '\_\_\_\_' is not defined

Traceback (most recent call last):

File "script.py", line 17, in <module>

plt.scatter(xs, ys, alpha=0.5)

File "script.py", line 2816, in scatter

None else {}), \*\*kwargs)

File "script.py", line 1565, in inner

return func(ax, \*map(sanitize\_sequence, args), \*\*kwargs)

File "script.py", line 358, in wrapper

return func(\*args, \*\*kwargs)

File "script.py", line 4380, in scatter

raise ValueError("x and y must be the same size")

ValueError: x and y must be the same size

Traceback (most recent call last):

File "script.py", line 17, in <module>

plt.scatter(xs, ys, alpha=0.5)

File "script.py", line 2816, in scatter

None else {}), \*\*kwargs)

File "script.py", line 1565, in inner

return func(ax, \*map(sanitize\_sequence, args), \*\*kwargs)

File "script.py", line 358, in wrapper

return func(\*args, \*\*kwargs)

File "script.py", line 4380, in scatter

raise ValueError("x and y must be the same size")

ValueError: x and y must be the same size

Traceback (most recent call last):

File "script.py", line 17, in <module>

plt.scatter(xs, ys, alpha=0.5)

File "script.py", line 2816, in scatter

None else {}), \*\*kwargs)

File "script.py", line 1565, in inner

return func(ax, \*map(sanitize\_sequence, args), \*\*kwargs)

File "script.py", line 358, in wrapper

return func(\*args, \*\*kwargs)

File "script.py", line 4380, in scatter

raise ValueError("x and y must be the same size")

ValueError: x and y must be the same size

Traceback (most recent call last):

File "script.py", line 17, in <module>

plt.scatter(xs, ys, alpha=0.5)

File "script.py", line 2816, in scatter

None else {}), \*\*kwargs)

File "script.py", line 1565, in inner

return func(ax, \*map(sanitize\_sequence, args), \*\*kwargs)

File "script.py", line 358, in wrapper

return func(\*args, \*\*kwargs)

File "script.py", line 4380, in scatter

raise ValueError("x and y must be the same size")

ValueError: x and y must be the same size

Traceback (most recent call last):

File "script.py", line 17, in <module>

plt.scatter(xs, ys, alpha=0.5)

File "script.py", line 2816, in scatter

None else {}), \*\*kwargs)

File "script.py", line 1565, in inner

return func(ax, \*map(sanitize\_sequence, args), \*\*kwargs)

File "script.py", line 358, in wrapper

return func(\*args, \*\*kwargs)

File "script.py", line 4380, in scatter

raise ValueError("x and y must be the same size")

ValueError: x and y must be the same size

Traceback (most recent call last):

File "script.py", line 17, in <module>

plt.scatter(xs, ys, alpha=0.5)

File "script.py", line 2816, in scatter

None else {}), \*\*kwargs)

File "script.py", line 1565, in inner

return func(ax, \*map(sanitize\_sequence, args), \*\*kwargs)

File "script.py", line 358, in wrapper

return func(\*args, \*\*kwargs)

File "script.py", line 4380, in scatter

raise ValueError("x and y must be the same size")

ValueError: x and y must be the same size

In [2]:

+100 XP

Fantastic! It's visualizations such as this that make t-SNE such a powerful tool for extracting quick insights from high dimensional data.

**Exercise**

**Exercise**

**Correlated data in nature**

You are given an array grains giving the width and length of samples of grain. You suspect that width and length will be correlated. To confirm this, make a scatter plot of width vs length and measure their Pearson correlation.

**Instructions**

**100 XP**

* Import:
  + matplotlib.pyplot as plt.
  + pearsonr from scipy.stats.
* Assign column 0 of grains to width and column 1 of grains to length.
* Make a scatter plot with width on the x-axis and length on the y-axis.
* Use the pearsonr() function to calculate the Pearson correlation of width and length.

[**Take Hint (-30 XP)**](javascript:void(0))

# Perform the necessary imports

import matplotlib.pyplot as plt

from scipy.stats import pearsonr

# Assign the 0th column of grains: width

width = grains[:,0]

# Assign the 1st column of grains: length

length = grains[:,1]

# Scatter plot width vs length

plt.scatter(width, length)

plt.axis('equal')

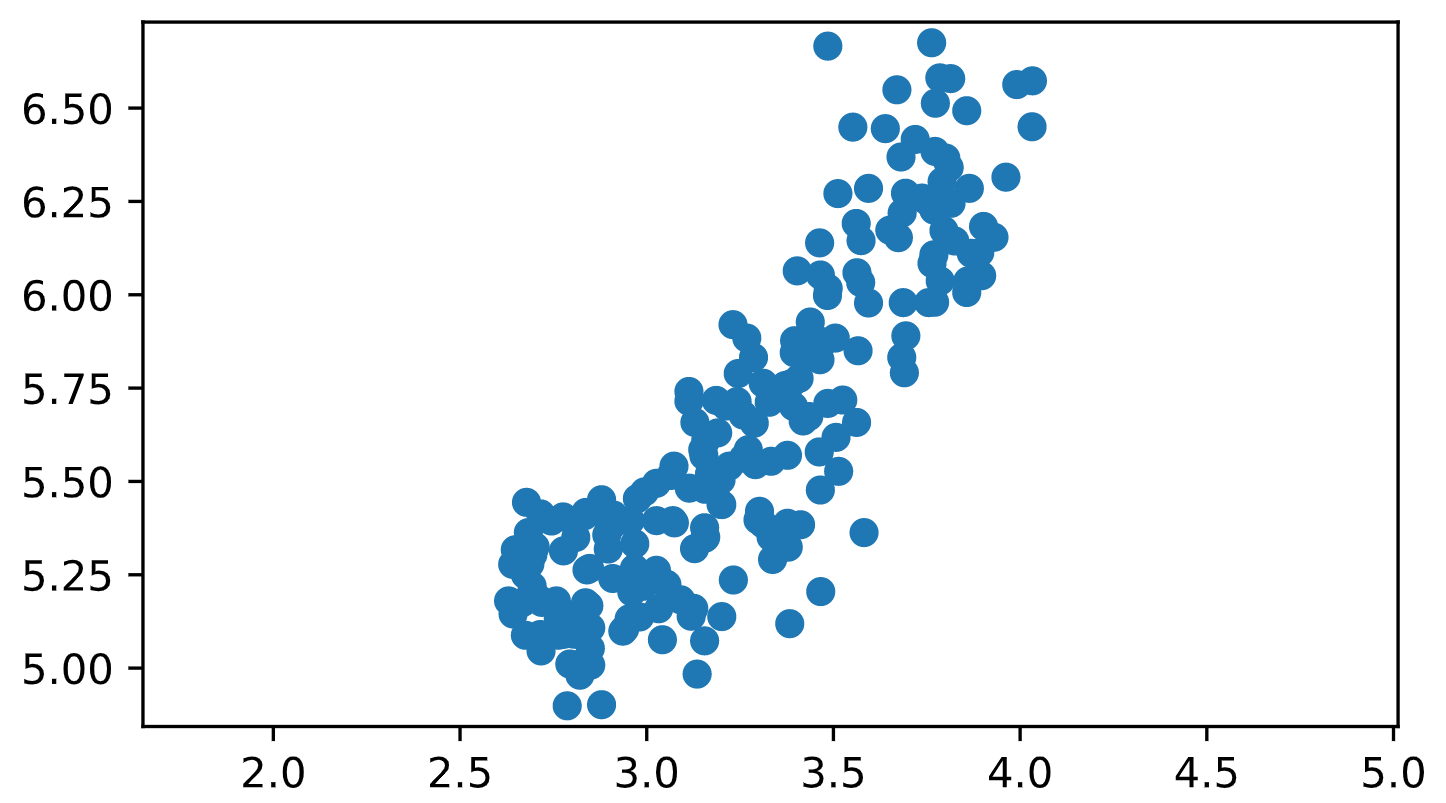
plt.show()

# Calculate the Pearson correlation

correlation, pvalue = pearsonr(width, length)

# Display the correlation

print(correlation)



In [1]: from scipy.stats import(pearsonr)

In [2]: from scipy.stats importpearsonr

File "<stdin>", line 1

from scipy.stats importpearsonr

^

SyntaxError: invalid syntax

<script.py> output:

0.8604149377143466

In [3]:

+100 XP

Great work! As you would expect, the width and length of the grain samples are highly correlated.

**Exercise**

**Exercise**

**Decorrelating the grain measurements with PCA**

You observed in the previous exercise that the width and length measurements of the grain are correlated. Now, you'll use PCA to decorrelate these measurements, then plot the decorrelated points and measure their Pearson correlation.

**Instructions**

**100 XP**

* Import PCA from sklearn.decomposition.
* Create an instance of PCA called model.
* Use the .fit\_transform() method of model to apply the PCA transformation to grains. Assign the result to pca\_features.
* The subsequent code to extract, plot, and compute the Pearson correlation of the first two columns pca\_features has been written for you, so hit 'Submit Answer' to see the result!

[**Take Hint (-30 XP)**](javascript:void(0))

# Import PCA

from sklearn.decomposition import PCA

# Create PCA instance: model

model = PCA()

# Apply the fit\_transform method of model to grains: pca\_features

pca\_features = model.fit\_transform(grains)

# Assign 0th column of pca\_features: xs

xs = pca\_features[:,0]

# Assign 1st column of pca\_features: ys

ys = pca\_features[:,1]

# Scatter plot xs vs ys

plt.scatter(xs, ys)

plt.axis('equal')

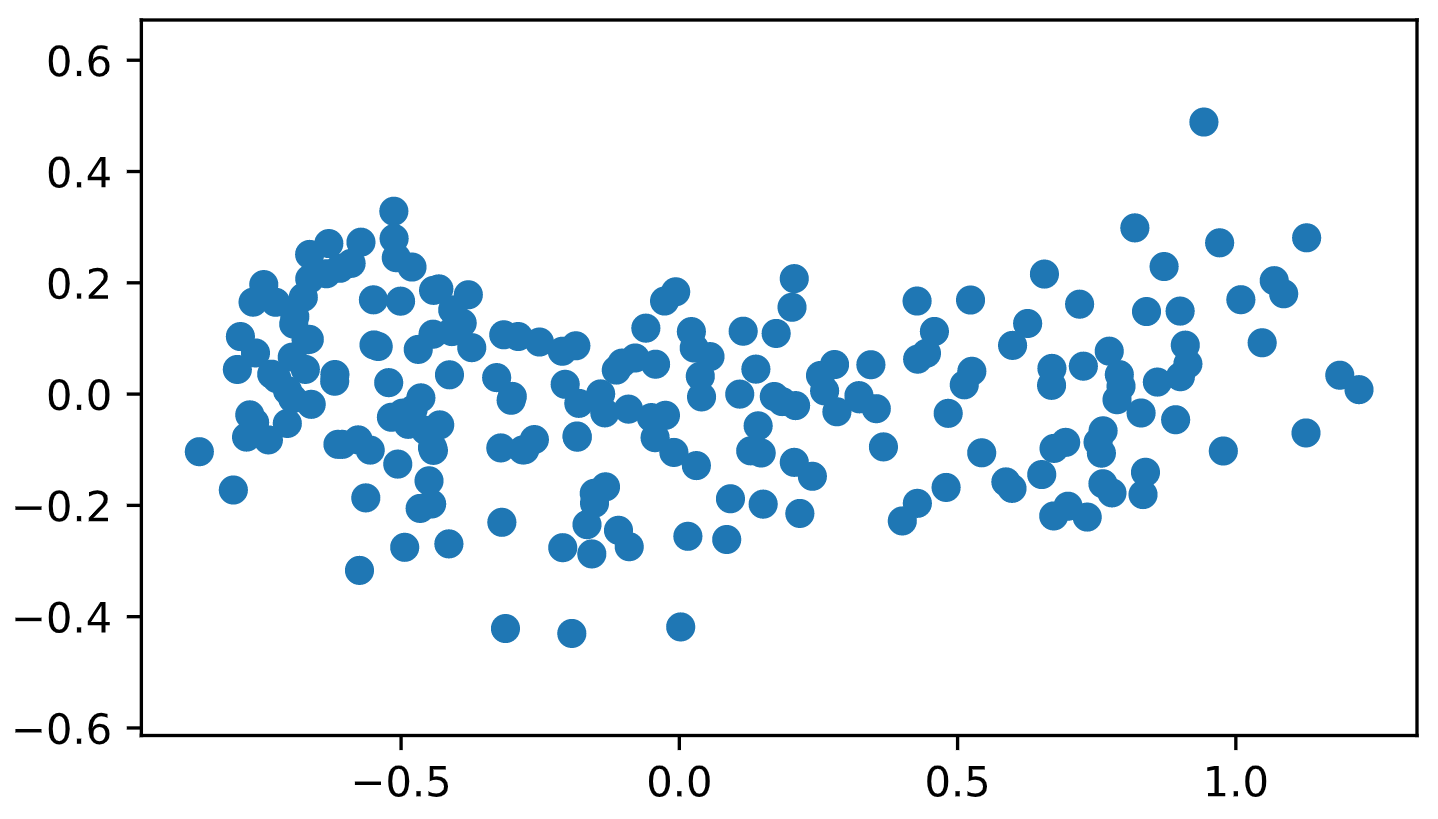
plt.show()

# Calculate the Pearson correlation of xs and ys

correlation, pvalue = pearsonr(xs, ys)

# Display the correlation

print(correlation)



In [1]: grains

Out[1]:

array([[3.312, 5.763],

[3.333, 5.554],

[3.337, 5.291],

[3.379, 5.324],

[3.562, 5.658],

[3.312, 5.386],

[3.259, 5.563],

[3.302, 5.42 ],

[3.465, 6.053],

[3.505, 5.884],

[3.242, 5.714],

[3.201, 5.438],

[3.199, 5.439],

[3.156, 5.479],

[3.114, 5.482],

[3.333, 5.351],

[3.383, 5.119],

[3.514, 5.527],

[3.466, 5.205],

[3.049, 5.226],

[3.129, 5.658],

[3.168, 5.52 ],

[3.507, 5.618],

[2.936, 5.099],

[3.245, 5.789],

[3.421, 5.833],

[3.026, 5.395],

[2.956, 5.395],

[3.221, 5.541],

[3.065, 5.516],

[2.975, 5.454],

[3.371, 5.757],

[3.186, 5.717],

[3.15 , 5.585],

[3.328, 5.712],

[3.485, 5.709],

[3.464, 5.826],

[3.683, 5.832],

[3.288, 5.656],

[3.298, 5.397],

[3.156, 5.348],

[3.158, 5.351],

[3.201, 5.138],

[3.396, 5.877],

[3.462, 5.579],

[3.155, 5.376],

[3.393, 5.701],

[3.377, 5.57 ],

[3.291, 5.545],

[3.258, 5.678],

[3.272, 5.585],

[3.434, 5.674],

[3.113, 5.715],

[3.199, 5.504],

[3.113, 5.741],

[3.212, 5.702],

[3.377, 5.388],

[3.412, 5.384],

[3.419, 5.662],

[3.032, 5.159],

[2.85 , 5.008],

[2.879, 4.902],

[3.042, 5.076],

[3.07 , 5.395],

[3.026, 5.262],

[3.119, 5.139],

[3.19 , 5.63 ],

[3.158, 5.609],

[3.153, 5.569],

[2.882, 5.412],

[3.561, 6.191],

[3.484, 5.998],

[3.594, 5.978],

[3.93 , 6.154],

[3.486, 6.017],

[3.438, 5.927],

[3.403, 6.064],

[3.814, 6.579],

[3.639, 6.445],

[3.566, 5.85 ],

[3.467, 5.875],

[3.857, 6.006],

[3.864, 6.285],

[3.772, 6.384],

[3.801, 6.366],

[3.651, 6.173],

[3.764, 6.084],

[3.67 , 6.549],

[4.033, 6.573],

[4.032, 6.45 ],

[3.785, 6.581],

[3.796, 6.172],

[3.693, 6.272],

[3.86 , 6.037],

[3.485, 6.666],

[3.463, 6.139],

[3.81 , 6.341],

[3.552, 6.449],

[3.512, 6.271],

[3.684, 6.219],

[3.525, 5.718],

[3.694, 5.89 ],

[3.892, 6.113],

[3.681, 6.369],

[3.755, 6.248],

[3.786, 6.037],

[3.806, 6.152],

[3.573, 6.033],

[3.763, 6.675],

[3.674, 6.153],

[3.769, 6.107],

[3.791, 6.303],

[3.902, 6.183],

[3.737, 6.259],

[3.991, 6.563],

[3.719, 6.416],

[3.897, 6.051],

[3.815, 6.245],

[3.769, 6.227],

[3.857, 6.493],

[3.962, 6.315],

[3.563, 6.059],

[3.387, 5.762],

[3.771, 5.98 ],

[3.582, 5.363],

[3.869, 6.111],

[3.594, 6.285],

[3.687, 5.979],

[3.773, 6.513],

[3.69 , 5.791],

[3.755, 5.979],

[3.825, 6.144],

[3.268, 5.884],

[3.395, 5.845],

[3.408, 5.776],

[3.465, 5.477],

[3.574, 6.145],

[3.231, 5.92 ],

[3.286, 5.832],

[3.472, 5.872],

[2.994, 5.472],

[3.073, 5.541],

[3.074, 5.389],

[2.967, 5.224],

[2.777, 5.314],

[2.687, 5.279],

[2.719, 5.176],

[2.967, 5.267],

[2.911, 5.386],

[2.648, 5.317],

[2.84 , 5.263],

[2.776, 5.405],

[2.833, 5.408],

[2.693, 5.22 ],

[2.755, 5.175],

[2.675, 5.25 ],

[2.849, 5.053],

[2.745, 5.394],

[2.678, 5.444],

[2.695, 5.304],

[2.879, 5.451],

[2.81 , 5.35 ],

[2.847, 5.267],

[2.968, 5.333],

[2.794, 5.011],

[2.941, 5.105],

[2.897, 5.319],

[2.837, 5.417],

[2.668, 5.176],

[2.715, 5.09 ],

[2.701, 5.325],

[2.845, 5.167],

[2.763, 5.088],

[2.763, 5.136],

[2.641, 5.278],

[2.821, 4.981],

[2.71 , 5.186],

[2.642, 5.145],

[2.758, 5.18 ],

[2.893, 5.357],

[2.775, 5.09 ],

[3.017, 5.236],

[2.909, 5.24 ],

[2.85 , 5.108],

[3.026, 5.495],

[2.683, 5.363],

[2.716, 5.413],

[2.675, 5.088],

[2.821, 5.089],

[2.787, 4.899],

[2.717, 5.046],

[2.804, 5.091],

[2.953, 5.132],

[2.63 , 5.18 ],

[2.975, 5.236],

[3.126, 5.16 ],

[3.054, 5.224],

[3.128, 5.32 ],

[2.911, 5.41 ],

[3.155, 5.073],

[2.989, 5.219],

[3.135, 4.984],

[2.81 , 5.009],

[3.091, 5.183],

[2.96 , 5.204],

[2.981, 5.137],

[2.795, 5.14 ],

[3.232, 5.236],

[2.836, 5.175],

[2.974, 5.243]])

In [2]: fit\_transform

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

fit\_transform

NameError: name 'fit\_transform' is not defined

<script.py> output:

2.5478751053409354e-17

In [3]:

+100 XP

Excellent! You've successfully decorrelated the grain measurements with PCA!

**Exercise**

**Exercise**

**Principal components**

On the right are three scatter plots of the same point cloud. Each scatter plot shows a different set of axes (in red). In which of the plots could the axes represent the principal components of the point cloud?

Recall that the principal components are the directions along which the the data varies.

**Instructions**

**50 XP**

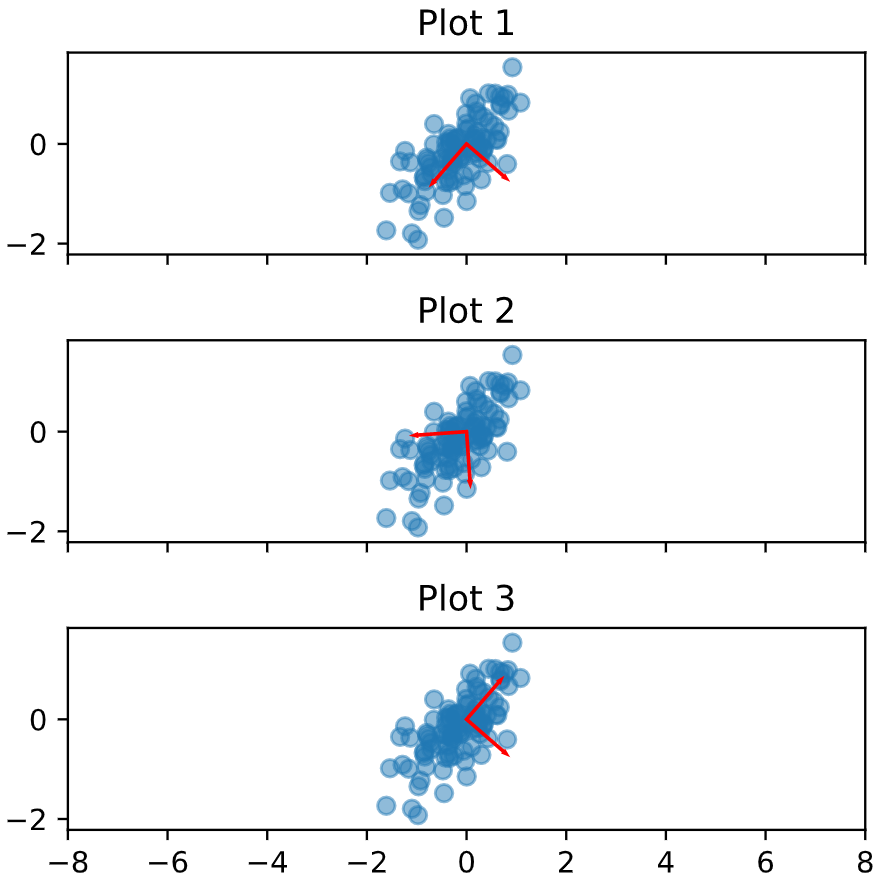
**Possible Answers**

None of them.

Both plot 1 and plot 3.

Plot 2.

Submit Answer



+50 XP

Well done! You've correctly inferred that the principal components have to align with the axes of the point cloud. This happens in both plot 1 and plot 3.

**Exercise**

**Exercise**

**The first principal component**

The first principal component of the data is the direction in which the data varies the most. In this exercise, your job is to use PCA to find the first principal component of the length and width measurements of the grain samples, and represent it as an arrow on the scatter plot.

The array grains gives the length and width of the grain samples. PyPlot (plt) and PCA have already been imported for you.

**Instructions**

**100 XP**

* Make a scatter plot of the grain measurements. This has been done for you.
* Create a PCA instance called model.
* Fit the model to the grains data.
* Extract the coordinates of the mean of the data using the .mean\_ attribute of model.
* Get the first principal component of model using the .components\_[0,:] attribute.
* Plot the first principal component as an arrow on the scatter plot, using the plt.arrow() function. You have to specify the first two arguments - mean[0] and mean[1].

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Did you call model.fit()?

Did you define mean correctly, using the .mean\_ attribute of model?

# Make a scatter plot of the untransformed points

plt.scatter(grains[:,0], grains[:,1])

# Create a PCA instance: model

model = PCA()

# Fit model to points

pca\_features = model.fit(grains)

# Get the mean of the grain samples: mean

mean = model.mean\_

# Get the first principal component: first\_pc

first\_pc = model.components\_[0,:]

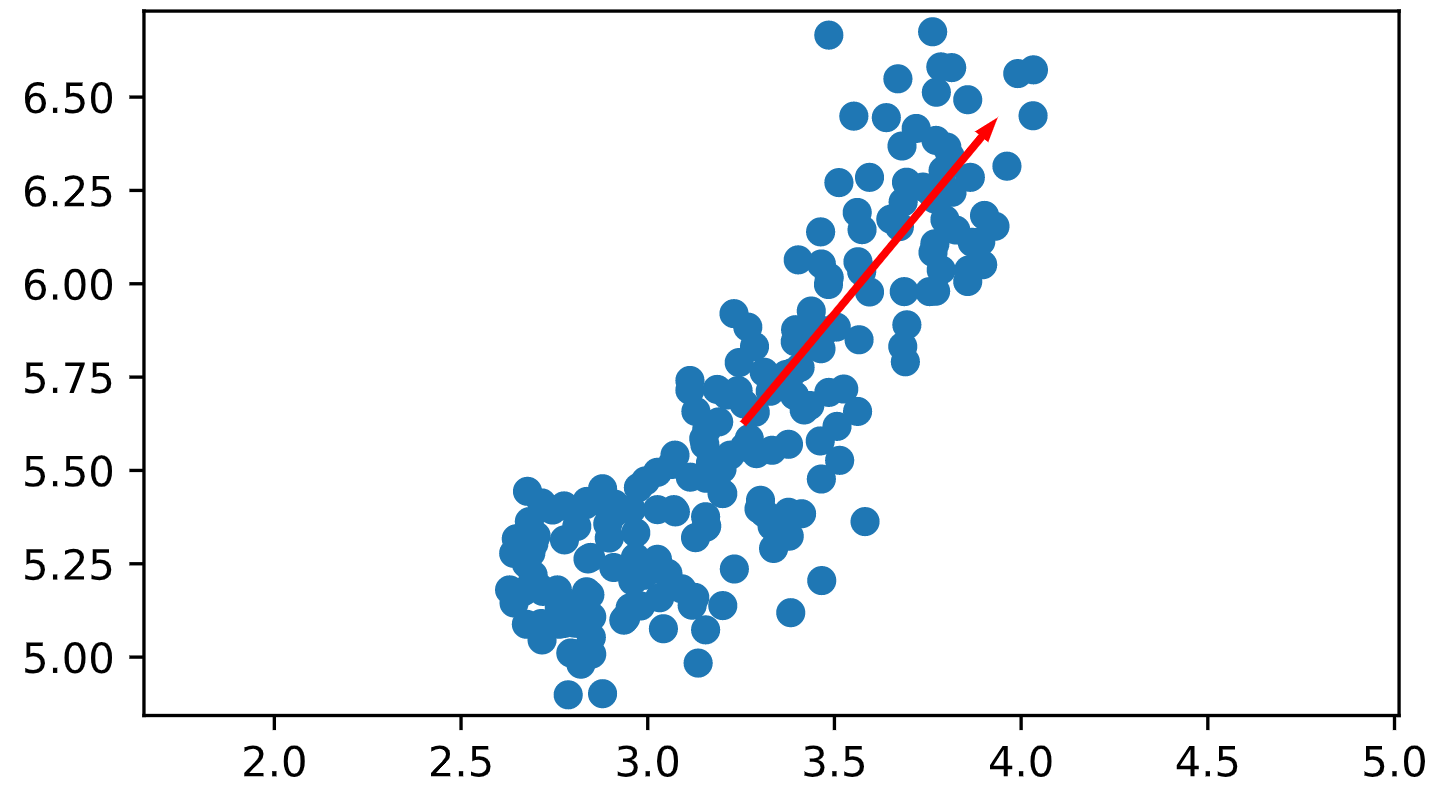
# Plot first\_pc as an arrow, starting at mean

plt.arrow(mean[0], mean[1], first\_pc[0], first\_pc[1], color='red', width=0.01)

# Keep axes on same scale

plt.axis('equal')

plt.show()



Traceback (most recent call last):

File "script.py", line 11, in <module>

mean = model.mean\_()

TypeError: 'numpy.ndarray' object is not callable

Traceback (most recent call last):

File "script.py", line 11, in <module>

mean = model.mean\_()

TypeError: 'numpy.ndarray' object is not callable

In [1]:

+100 XP

Excellent job! This is the direction in which the grain data varies the most.

**Exercise**

**Exercise**

**Variance of the PCA features**

The fish dataset is 6-dimensional. But what is its *intrinsic* dimension? Make a plot of the variances of the PCA features to find out. As before, samples is a 2D array, where each row represents a fish. You'll need to standardize the features first.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Create an instance of StandardScaler called scaler.
* Create a PCA instance called pca.
* Use the make\_pipeline() function to create a pipeline chaining scaler and pca.
* Use the .fit() method of pipeline to fit it to the fish samples samples.
* Extract the number of components used using the .n\_components\_ attribute of pca. Place this inside a range() function and store the result as features.
* Use the plt.bar() function to plot the explained variances, with features on the x-axis and pca.explained\_variance\_ on the y-axis.

[**Take Hint (-30 XP)**](javascript:void(0))

# Perform the necessary imports

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import make\_pipeline

import matplotlib.pyplot as plt

# Create scaler: scaler

scaler = StandardScaler()

# Create a PCA instance: pca

pca = PCA()

# Create pipeline: pipeline

pipeline = make\_pipeline(scaler, pca)

# Fit the pipeline to 'samples'

pipeline.fit(samples)

# Plot the explained variances

features = range(pca.n\_components\_)

plt.bar(features, pca.explained\_variance\_)

plt.xlabel('PCA feature')

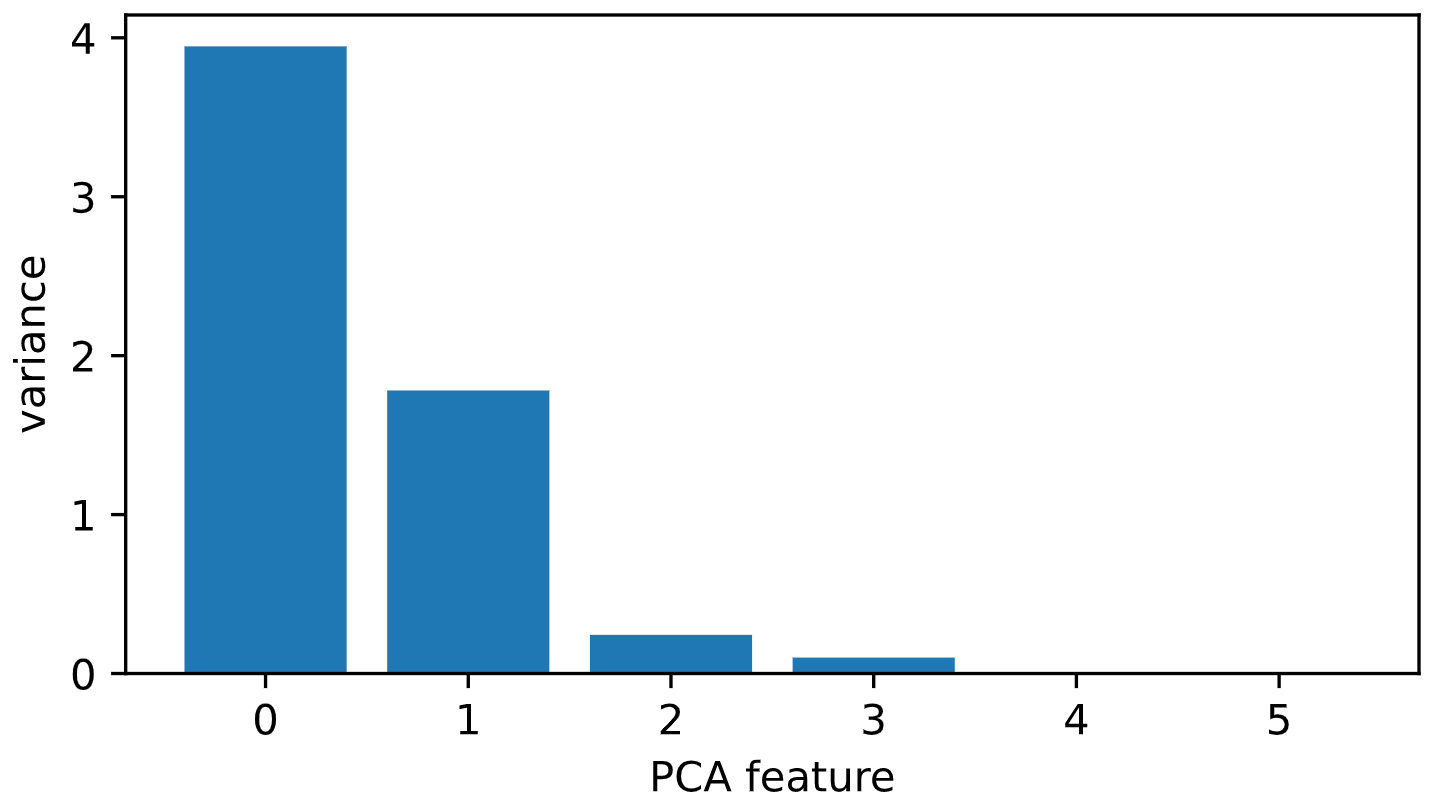
plt.ylabel('variance')

plt.xticks(features)

plt.show()

+50 XP

Great job! Since PCA features 0 and 1 have significant variance, the intrinsic dimension of this dataset appears to be 2.



+100 XP

Great work! It looks like PCA features 0 and 1 have significant variance.

**Exercise**

**Exercise**

**Intrinsic dimension of the fish data**

In the previous exercise, you plotted the variance of the PCA features of the fish measurements. Looking again at your plot, what do you think would be a reasonable choice for the "intrinsic dimension" of the the fish measurements? Recall that the intrinsic dimension is the number of PCA features with significant variance.

**Instructions**

**50 XP**

**Possible Answers**

1

2

5

**Submit Answer**

[**Take Hint (-15 XP)**](javascript:void(0))

+50 XP

Great job! Since PCA features 0 and 1 have significant variance, the intrinsic dimension of this dataset appears to be 2.

**Exercise**

**Exercise**

**Dimension reduction of the fish measurements**

In a previous exercise, you saw that 2 was a reasonable choice for the "intrinsic dimension" of the fish measurements. Now use PCA for dimensionality reduction of the fish measurements, retaining only the 2 most important components.

The fish measurements have already been scaled for you, and are available as scaled\_samples.

**Instructions**

**100 XP**

* Import PCA from sklearn.decomposition.
* Create a PCA instance called pca with n\_components=2.
* Use the .fit() method of pca to fit it to the scaled fish measurements scaled\_samples.
* Use the .transform() method of pca to transform the scaled\_samples. Assign the result to pca\_features.

[**Take Hint (-30 XP)**](javascript:void(0))

# Import PCA

from sklearn.decomposition import PCA

# Create a PCA model with 2 components: pca

pca = PCA(n\_components=2)

# Fit the PCA instance to the scaled samples

pca.fit(scaled\_samples)

# Transform the scaled samples: pca\_features

pca\_features = pca.transform(scaled\_samples)

# Print the shape of pca\_features

print(pca\_features.shape)

<script.py> output:

(85, 2)

In [1]:

+100 XP

Superb! You've successfully reduced the dimensionality from 6 to 2.

**Exercise**

**Exercise**

**A tf-idf word-frequency array**

In this exercise, you'll create a tf-idf word frequency array for a toy collection of documents. For this, use the TfidfVectorizer from sklearn. It transforms a list of documents into a word frequency array, which it outputs as a csr\_matrix. It has fit() and transform() methods like other sklearn objects.

You are given a list documents of toy documents about pets. Its contents have been printed in the IPython Shell.

**Instructions**

**100 XP**

* Import TfidfVectorizer from sklearn.feature\_extraction.text.
* Create a TfidfVectorizer instance called tfidf.
* Apply .fit\_transform() method of tfidf to documents and assign the result to csr\_mat. This is a word-frequency array in csr\_matrix format.
* Inspect csr\_mat by calling its .toarray() method and printing the result. This has been done for you.
* The columns of the array correspond to words. Get the list of words by calling the .get\_feature\_names() method of tfidf, and assign the result to words.

[**Take Hint (-30 XP)**](javascript:void(0))

# Import TfidfVectorizer

from sklearn.feature\_extraction.text import TfidfVectorizer

# Create a TfidfVectorizer: tfidf

tfidf = TfidfVectorizer()

# Apply fit\_transform to document: csr\_mat

csr\_mat = tfidf.fit\_transform(documents)

# Print result of toarray() method

print(csr\_mat.toarray())

# Get the words: words

words = tfidf.get\_feature\_names()

# Print words

print(words)

['cats say meow', 'dogs say woof', 'dogs chase cats']

<script.py> output:

[[0.51785612 0. 0. 0.68091856 0.51785612 0. ]

[0. 0. 0.51785612 0. 0.51785612 0.68091856]

[0.51785612 0.68091856 0.51785612 0. 0. 0. ]]

['cats', 'chase', 'dogs', 'meow', 'say', 'woof']

In [1]:

+100 XP

Great work! You'll now move to clustering Wikipedia articles!

**Exercise**

**Exercise**

**Clustering Wikipedia part I**

You saw in the video that TruncatedSVD is able to perform PCA on sparse arrays in csr\_matrix format, such as word-frequency arrays. Combine your knowledge of TruncatedSVD and k-means to cluster some popular pages from Wikipedia. In this exercise, build the pipeline. In the next exercise, you'll apply it to the word-frequency array of some Wikipedia articles.

Create a Pipeline object consisting of a TruncatedSVD followed by KMeans. (This time, we've precomputed the word-frequency matrix for you, so there's no need for a TfidfVectorizer).

The Wikipedia dataset you will be working with was obtained from [**here**](https://blog.lateral.io/2015/06/the-unknown-perils-of-mining-wikipedia/).

**Instructions**

**100 XP**

* Import:
  + TruncatedSVD from sklearn.decomposition.
  + KMeans from sklearn.cluster.
  + make\_pipeline from sklearn.pipeline.
* Create a TruncatedSVD instance called svd with n\_components=50.
* Create a KMeans instance called kmeans with n\_clusters=6.
* Create a pipeline called pipeline consisting of svd and kmeans.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Did you call KMeans()?

# Perform the necessary imports

from sklearn.decomposition import TruncatedSVD

from sklearn.cluster import KMeans

from sklearn.pipeline import make\_pipeline

# Create a TruncatedSVD instance: svd

svd = TruncatedSVD(n\_components=50)

# Create a KMeans instance: kmeans

kmeans = KMeans(n\_clusters=6)

# Create a pipeline: pipeline

pipeline = make\_pipeline(svd, kmeans)

Traceback (most recent call last):

File "script.py", line 10, in <module>

kmeans = kmeans(n\_clusters=6)

NameError: name 'kmeans' is not defined

In [1]:

+100 XP

Excellent! Now that you have set up your pipeline, you will use it in the next exercise to cluster the articles.

**Exercise**

**Exercise**

**Clustering Wikipedia part II**

It is now time to put your pipeline from the previous exercise to work! You are given an array articles of tf-idf word-frequencies of some popular Wikipedia articles, and a list titles of their titles. Use your pipeline to cluster the Wikipedia articles.

A solution to the previous exercise has been pre-loaded for you, so a Pipeline pipeline chaining TruncatedSVD with KMeans is available.

**Instructions**

**100 XP**

* Import pandas as pd.
* Fit the pipeline to the word-frequency array articles.
* Predict the cluster labels.
* Align the cluster labels with the list titles of article titles by creating a DataFrame df with labels and titles as columns. This has been done for you.
* Use the .sort\_values() method of df to sort the DataFrame by the 'label' column, and print the result.
* Hit 'Submit Answer' and take a moment to investigate your amazing clustering of Wikipedia pages!

Ctrl+H

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Did you call df.sort\_values()?

Have you specified the arguments for df.sort\_values() using the right syntax?

# Import pandas

import pandas as pd

# Fit the pipeline to articles

pipeline.fit(articles)

# Calculate the cluster labels: labels

labels = pipeline.predict(articles)

# Create a DataFrame aligning labels and titles: df

df = pd.DataFrame({'label': labels, 'article': titles})

# Display df sorted by cluster label

print(df.sort\_values('label'))

In [1]: dir()

Out[1]:

['In',

'KMeans',

'Out',

'TruncatedSVD',

'\_',

'\_\_',

'\_\_\_',

'\_\_builtin\_\_',

'\_\_builtins\_\_',

'\_\_name\_\_',

'\_dh',

'\_i',

'\_i1',

'\_ih',

'\_ii',

'\_iii',

'\_oh',

'\_sh',

'articles',

'csr\_matrix',

'df',

'exit',

'get\_ipython',

'kmeans',

'make\_pipeline',

'pipeline',

'quit',

'secret\_pandas',

'svd',

'titles']

In [2]: del In

In [3]: In

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

In

NameError: name 'In' is not defined

In [4]: In

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

In

NameError: name 'In' is not defined

In [5]: del Out

In [6]: 'a'

Out[6]: 'a'

In [7]: Out

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

Out

NameError: name 'Out' is not defined

In [8]: df

Out[8]:

HTTP 404 Alexa Internet ... Sepsis Adam Levine

0 0.0 0.000000 ... 0.000000 0.0

1 0.0 0.000000 ... 0.006110 0.0

2 0.0 0.029607 ... 0.000000 0.0

3 0.0 0.000000 ... 0.000000 0.0

4 0.0 0.000000 ... 0.000000 0.0

... ... ... ... ... ...

13120 0.0 0.000000 ... 0.000000 0.0

13121 0.0 0.000000 ... 0.000000 0.0

13122 0.0 0.000000 ... 0.000000 0.0

13123 0.0 0.000000 ... 0.005476 0.0

13124 0.0 0.000000 ... 0.000000 0.0

[13125 rows x 60 columns]

<script.py> output:

label article

0 4 HTTP 404

1 4 Alexa Internet

2 4 Internet Explorer

3 4 HTTP cookie

4 4 Google Search

5 4 Tumblr

6 4 Hypertext Transfer Protocol

7 4 Social search

8 4 Firefox

9 4 LinkedIn

10 3 Global warming

11 3 Nationally Appropriate Mitigation Action

12 3 Nigel Lawson

13 3 Connie Hedegaard

14 3 Climate change

15 3 Kyoto Protocol

16 3 350.org

17 3 Greenhouse gas emissions by the United States

18 3 2010 United Nations Climate Change Conference

19 3 2007 United Nations Climate Change Conference

20 2 Angelina Jolie

21 2 Michael Fassbender

22 2 Denzel Washington

23 2 Catherine Zeta-Jones

24 2 Jessica Biel

25 2 Russell Crowe

26 2 Mila Kunis

27 2 Dakota Fanning

28 2 Anne Hathaway

29 2 Jennifer Aniston

30 1 France national football team

31 1 Cristiano Ronaldo

32 1 Arsenal F.C.

33 1 Radamel Falcao

34 1 Zlatan Ibrahimović

35 1 Colombia national football team

36 1 2014 FIFA World Cup qualification

37 1 Football

38 1 Neymar

39 1 Franck Ribéry

40 5 Tonsillitis

41 5 Hepatitis B

42 4 Doxycycline

43 4 Leukemia

44 4 Gout

45 5 Hepatitis C

46 4 Prednisone

47 4 Fever

48 4 Gabapentin

49 4 Lymphoma

50 0 Chad Kroeger

51 0 Nate Ruess

52 0 The Wanted

53 0 Stevie Nicks

54 0 Arctic Monkeys

55 0 Black Sabbath

56 0 Skrillex

57 0 Red Hot Chili Peppers

58 0 Sepsis

59 0 Adam Levine

Traceback (most recent call last):

File "script.py", line 14, in <module>

print(dfsort\_values())

NameError: name 'dfsort\_values' is not defined

Traceback (most recent call last):

File "script.py", line 14, in <module>

print(df.sort\_values())

TypeError: sort\_values() missing 1 required positional argument: 'by'

<script.py> output:

label article

59 0 Adam Levine

57 0 Red Hot Chili Peppers

56 0 Skrillex

55 0 Black Sabbath

54 0 Arctic Monkeys

53 0 Stevie Nicks

52 0 The Wanted

51 0 Nate Ruess

50 0 Chad Kroeger

58 0 Sepsis

30 1 France national football team

31 1 Cristiano Ronaldo

32 1 Arsenal F.C.

33 1 Radamel Falcao

37 1 Football

35 1 Colombia national football team

36 1 2014 FIFA World Cup qualification

38 1 Neymar

39 1 Franck Ribéry

34 1 Zlatan Ibrahimović

26 2 Mila Kunis

28 2 Anne Hathaway

27 2 Dakota Fanning

25 2 Russell Crowe

29 2 Jennifer Aniston

23 2 Catherine Zeta-Jones

22 2 Denzel Washington

21 2 Michael Fassbender

20 2 Angelina Jolie

24 2 Jessica Biel

10 3 Global warming

11 3 Nationally Appropriate Mitigation Action

13 3 Connie Hedegaard

14 3 Climate change

12 3 Nigel Lawson

16 3 350.org

17 3 Greenhouse gas emissions by the United States

18 3 2010 United Nations Climate Change Conference

19 3 2007 United Nations Climate Change Conference

15 3 Kyoto Protocol

8 4 Firefox

1 4 Alexa Internet

2 4 Internet Explorer

3 4 HTTP cookie

4 4 Google Search

5 4 Tumblr

6 4 Hypertext Transfer Protocol

7 4 Social search

49 4 Lymphoma

42 4 Doxycycline

47 4 Fever

46 4 Prednisone

44 4 Gout

43 4 Leukemia

9 4 LinkedIn

48 4 Gabapentin

0 4 HTTP 404

45 5 Hepatitis C

41 5 Hepatitis B

40 5 Tonsillitis

In [9]:

+100 XP

Fantastic! Take a look at the cluster labels and see if you can identify any patterns!

**Non-negative data**

Which of the following 2-dimensional arrays are examples of non-negative data?

1. A tf-idf word-frequency array.
2. An array daily stock market price movements (up and down), where each row represents a company.
3. An array where rows are customers, columns are products and entries are 0 or 1, indicating whether a customer has purchased a product.

**Answer the question**

**50 XP**

**Possible Answers**

1 only

press

1

2 and 3

press

2

1 and 3

press

3

+50 XP

Well done! Stock prices can go down as well as up, so an array of daily stock market price movements is not an example of non-negative data.

**Exercise**

**Exercise**

**NMF applied to Wikipedia articles**

In the video, you saw NMF applied to transform a toy word-frequency array. Now it's your turn to apply NMF, this time using the tf-idf word-frequency array of Wikipedia articles, given as a csr matrix articles. Here, fit the model and transform the articles. In the next exercise, you'll explore the result.

**Instructions**

**100 XP**

* Import NMF from sklearn.decomposition.
* Create an NMF instance called model with 6 components.
* Fit the model to the word count data articles.
* Use the .transform() method of model to transform articles, and assign the result to nmf\_features.
* Print nmf\_features to get a first idea what it looks like.

[**Take Hint (-30 XP)**](javascript:void(0))

# Import NMF

from sklearn.decomposition import NMF

# Create an NMF instance: model

model = NMF(6)

# Fit the model to articles

model.fit(articles)

# Transform the articles: nmf\_features

nmf\_features = model.transform(articles)

# Print the NMF features

print(nmf\_features)

<script.py> output:

[[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00

0.00000000e+00 4.40623130e-01]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00

0.00000000e+00 5.66807830e-01]

[3.82006369e-03 0.00000000e+00 0.00000000e+00 0.00000000e+00

0.00000000e+00 3.98789405e-01]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00

0.00000000e+00 3.81876582e-01]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00

0.00000000e+00 4.85690952e-01]

[1.29273684e-02 1.37882014e-02 7.76395971e-03 3.34470747e-02

0.00000000e+00 3.34641946e-01]

[0.00000000e+00 0.00000000e+00 2.06761507e-02 0.00000000e+00

6.04526007e-03 3.59189684e-01]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00

0.00000000e+00 4.91152512e-01]

[1.54253813e-02 1.42809448e-02 3.76669015e-03 2.37100833e-02

2.62637014e-02 4.80946740e-01]

[1.11723843e-02 3.13660366e-02 3.09514446e-02 6.56969239e-02

1.96689871e-02 3.38410141e-01]

[0.00000000e+00 0.00000000e+00 5.30768937e-01 0.00000000e+00

2.83698153e-02 0.00000000e+00]

[0.00000000e+00 0.00000000e+00 3.56542551e-01 0.00000000e+00

0.00000000e+00 0.00000000e+00]

[1.20111573e-02 6.49999720e-03 3.12274374e-01 6.09743127e-02

1.13868844e-02 1.92671584e-02]

[3.93434173e-03 6.24399341e-03 3.42405187e-01 1.10763956e-02

0.00000000e+00 0.00000000e+00]

[4.63760690e-03 0.00000000e+00 4.34955622e-01 0.00000000e+00

3.84300364e-02 3.08244428e-03]

[0.00000000e+00 0.00000000e+00 4.83334170e-01 0.00000000e+00

0.00000000e+00 0.00000000e+00]

[5.64942921e-03 1.83522788e-02 3.76568114e-01 3.25446119e-02

0.00000000e+00 1.13375392e-02]

[0.00000000e+00 0.00000000e+00 4.80958620e-01 0.00000000e+00

0.00000000e+00 0.00000000e+00]

[0.00000000e+00 9.01801798e-03 5.51059312e-01 0.00000000e+00

0.00000000e+00 0.00000000e+00]

[0.00000000e+00 0.00000000e+00 4.66013077e-01 0.00000000e+00

0.00000000e+00 0.00000000e+00]

[0.00000000e+00 1.14073022e-02 2.08675156e-02 5.17743929e-01

5.81446974e-02 1.37903957e-02]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 5.10452242e-01

0.00000000e+00 0.00000000e+00]

[0.00000000e+00 5.60066077e-03 0.00000000e+00 4.22360790e-01

0.00000000e+00 0.00000000e+00]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 4.36731545e-01

0.00000000e+00 0.00000000e+00]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 4.98069560e-01

0.00000000e+00 0.00000000e+00]

[9.88264848e-02 8.59984085e-02 3.91072496e-03 3.81000310e-01

4.39271788e-04 5.22342392e-03]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 5.72144050e-01

0.00000000e+00 7.13804247e-03]

[1.31451598e-02 1.04846119e-02 0.00000000e+00 4.68884875e-01

0.00000000e+00 1.16352017e-02]

[3.84499664e-03 0.00000000e+00 0.00000000e+00 5.75684563e-01

0.00000000e+00 0.00000000e+00]

[2.25215918e-03 1.38727788e-03 0.00000000e+00 5.27921916e-01

1.20272657e-02 1.49538235e-02]

[0.00000000e+00 4.07519476e-01 1.85731905e-03 0.00000000e+00

2.96630327e-03 4.52500939e-04]

[1.53401124e-03 6.08130207e-01 5.22326440e-04 6.24824822e-03

1.18452509e-03 4.40230477e-04]

[5.38749019e-03 2.64998396e-01 5.38561333e-04 1.86917366e-02

6.38693589e-03 2.90209138e-03]

[0.00000000e+00 6.44870483e-01 0.00000000e+00 0.00000000e+00

0.00000000e+00 0.00000000e+00]

[0.00000000e+00 6.08864091e-01 0.00000000e+00 0.00000000e+00

0.00000000e+00 0.00000000e+00]

[0.00000000e+00 3.43661046e-01 0.00000000e+00 0.00000000e+00

3.97821375e-03 0.00000000e+00]

[6.10428893e-03 3.15290611e-01 1.54894430e-02 0.00000000e+00

5.06278213e-03 4.74505398e-03]

[6.47289002e-03 2.13313539e-01 9.49584427e-03 4.56960436e-02

1.71925896e-02 9.52405364e-03]

[7.99042762e-03 4.67562239e-01 0.00000000e+00 2.43414353e-02

0.00000000e+00 0.00000000e+00]

[0.00000000e+00 6.42774843e-01 0.00000000e+00 2.35844148e-03

0.00000000e+00 0.00000000e+00]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00

4.77111613e-01 0.00000000e+00]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00

4.94285809e-01 0.00000000e+00]

[0.00000000e+00 2.99043288e-04 2.14507572e-03 0.00000000e+00

3.81801632e-01 5.83991133e-03]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 5.64666945e-03

5.42274085e-01 0.00000000e+00]

[1.78036081e-03 7.84356703e-04 1.41641210e-02 4.59792631e-04

4.24327940e-01 0.00000000e+00]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00

5.11422562e-01 0.00000000e+00]

[0.00000000e+00 0.00000000e+00 3.28416355e-03 0.00000000e+00

3.72909283e-01 0.00000000e+00]

[0.00000000e+00 2.62064782e-04 3.61138437e-02 2.32325695e-04

2.30524575e-01 0.00000000e+00]

[1.12502927e-02 2.12312379e-03 1.60987509e-02 1.02480072e-02

3.25481184e-01 3.76015664e-02]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00

4.18983473e-01 3.57828039e-04]

[3.08333122e-01 0.00000000e+00 0.00000000e+00 0.00000000e+00

0.00000000e+00 0.00000000e+00]

[3.68133416e-01 0.00000000e+00 0.00000000e+00 0.00000000e+00

0.00000000e+00 0.00000000e+00]

[3.97901156e-01 2.81683253e-02 3.67046415e-03 1.70058961e-02

1.95979270e-03 2.11720443e-02]

[3.75753335e-01 2.07506032e-03 0.00000000e+00 3.72137458e-02

0.00000000e+00 5.86138262e-03]

[4.37980096e-01 0.00000000e+00 0.00000000e+00 0.00000000e+00

0.00000000e+00 0.00000000e+00]

[4.57830730e-01 0.00000000e+00 0.00000000e+00 0.00000000e+00

0.00000000e+00 0.00000000e+00]

[2.75446975e-01 4.46925230e-03 0.00000000e+00 5.29631428e-02

0.00000000e+00 1.91066265e-02]

[4.45145032e-01 0.00000000e+00 0.00000000e+00 0.00000000e+00

5.48731978e-03 0.00000000e+00]

[2.92708241e-01 1.33655378e-02 1.14274148e-02 1.05195225e-02

1.87707782e-01 9.24296799e-03]

[3.78224952e-01 1.43960157e-02 0.00000000e+00 9.85194849e-02

1.35908675e-02 0.00000000e+00]]

In [1]:

+100 XP

Fantastic - these NMF features don't make much sense at this point, but you will explore them in the next exercise!

**Exercise**

**Exercise**

**NMF features of the Wikipedia articles**

Now you will explore the NMF features you created in the previous exercise. A solution to the previous exercise has been pre-loaded, so the array nmf\_features is available. Also available is a list titles giving the title of each Wikipedia article.

When investigating the features, notice that for both actors, the NMF feature 3 has by far the highest value. This means that both articles are reconstructed using mainly the 3rd NMF component. In the next video, you'll see why: NMF components represent topics (for instance, acting!).

**Instructions**

**100 XP**

* Import pandas as pd.
* Create a DataFrame df from nmf\_features using pd.DataFrame(). Set the index to titles using index=titles.
* Use the .loc[] accessor of df to select the row with title 'Anne Hathaway', and print the result. These are the NMF features for the article about the actress Anne Hathaway.
* Repeat the last step for 'Denzel Washington' (another actor).

[**Take Hint (-30 XP)**](javascript:void(0))

# Import pandas

import pandas as pd

# Create a pandas DataFrame: df

df = pd.DataFrame(nmf\_features, index=titles)

# Print the row for 'Anne Hathaway'

print(df.loc['Anne Hathaway'])

# Print the row for 'Denzel Washington'

print(df.loc['Denzel Washington'])

In [1]: nmf\_features.

File "<stdin>", line 1

nmf\_features.

^

SyntaxError: invalid syntax

In [2]: nmf\_features

Out[2]:

array([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

0.00000000e+00, 4.40465322e-01],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

0.00000000e+00, 5.66604756e-01],

[3.82058457e-03, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

0.00000000e+00, 3.98646427e-01],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

0.00000000e+00, 3.81739753e-01],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

0.00000000e+00, 4.85517119e-01],

[1.29290459e-02, 1.37889184e-02, 7.76313733e-03, 3.34487322e-02,

0.00000000e+00, 3.34521818e-01],

[0.00000000e+00, 0.00000000e+00, 2.06738655e-02, 0.00000000e+00,

6.04486283e-03, 3.59061038e-01],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

0.00000000e+00, 4.90976703e-01],

[1.54274156e-02, 1.42817184e-02, 3.76628748e-03, 2.37111817e-02,

2.62619893e-02, 4.80774318e-01],

[1.11738298e-02, 3.13676816e-02, 3.09480486e-02, 6.57000345e-02,

1.96677390e-02, 3.38288648e-01],

[0.00000000e+00, 0.00000000e+00, 5.30710112e-01, 0.00000000e+00,

2.83679347e-02, 0.00000000e+00],

[0.00000000e+00, 0.00000000e+00, 3.56503042e-01, 0.00000000e+00,

0.00000000e+00, 0.00000000e+00],

[1.20127255e-02, 6.50033756e-03, 3.12239769e-01, 6.09770818e-02,

1.13861318e-02, 1.92602102e-02],

[3.93485615e-03, 6.24432234e-03, 3.42367241e-01, 1.10768934e-02,

0.00000000e+00, 0.00000000e+00],

[4.63821398e-03, 0.00000000e+00, 4.34907414e-01, 0.00000000e+00,

3.84274924e-02, 3.08133140e-03],

[0.00000000e+00, 0.00000000e+00, 4.83280609e-01, 0.00000000e+00,

0.00000000e+00, 0.00000000e+00],

[5.65016594e-03, 1.83532464e-02, 3.76526384e-01, 3.25460935e-02,

0.00000000e+00, 1.13334438e-02],

[0.00000000e+00, 0.00000000e+00, 4.80905320e-01, 0.00000000e+00,

0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 9.01849274e-03, 5.50998244e-01, 0.00000000e+00,

0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 0.00000000e+00, 4.65961434e-01, 0.00000000e+00,

0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 1.14078934e-02, 2.08651998e-02, 5.17767327e-01,

5.81408435e-02, 1.37853805e-02],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 5.10475273e-01,

0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 5.60094760e-03, 0.00000000e+00, 4.22379854e-01,

0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.36751253e-01,

0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.98092036e-01,

0.00000000e+00, 0.00000000e+00],

[9.88394018e-02, 8.60029454e-02, 3.91028856e-03, 3.81017532e-01,

4.39239942e-04, 5.22151481e-03],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 5.72169888e-01,

0.00000000e+00, 7.13541985e-03],

[1.31468715e-02, 1.04851581e-02, 0.00000000e+00, 4.68906049e-01,

0.00000000e+00, 1.16309971e-02],

[3.84548688e-03, 0.00000000e+00, 0.00000000e+00, 5.75710557e-01,

0.00000000e+00, 0.00000000e+00],

[2.25244164e-03, 1.38734017e-03, 0.00000000e+00, 5.27945778e-01,

1.20264671e-02, 1.49483895e-02],

[0.00000000e+00, 4.07541038e-01, 1.85711419e-03, 0.00000000e+00,

2.96610497e-03, 4.52342305e-04],

[1.53421227e-03, 6.08162380e-01, 5.22269328e-04, 6.24853541e-03,

1.18444733e-03, 4.40074955e-04],

[5.38819366e-03, 2.65012410e-01, 5.38501825e-04, 1.86925929e-02,

6.38651189e-03, 2.90104872e-03],

[0.00000000e+00, 6.44904602e-01, 0.00000000e+00, 0.00000000e+00,

0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 6.08896305e-01, 0.00000000e+00, 0.00000000e+00,

0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 3.43679231e-01, 0.00000000e+00, 0.00000000e+00,

3.97794829e-03, 0.00000000e+00],

[6.10508849e-03, 3.15307295e-01, 1.54877295e-02, 0.00000000e+00,

5.06244563e-03, 4.74335450e-03],

[6.47373328e-03, 2.13324818e-01, 9.49479174e-03, 4.56981207e-02,

1.71914481e-02, 9.52063620e-03],

[7.99147234e-03, 4.67586974e-01, 0.00000000e+00, 2.43425413e-02,

0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 6.42808850e-01, 0.00000000e+00, 2.35855413e-03,

0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

4.77080113e-01, 0.00000000e+00],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

4.94253212e-01, 0.00000000e+00],

[0.00000000e+00, 2.99063787e-04, 2.14484813e-03, 0.00000000e+00,

3.81776318e-01, 5.83780479e-03],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 5.64692771e-03,

5.42238170e-01, 0.00000000e+00],

[1.78059654e-03, 7.84399946e-04, 1.41625538e-02, 4.59813501e-04,

4.24299848e-01, 0.00000000e+00],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

5.11388821e-01, 0.00000000e+00],

[0.00000000e+00, 0.00000000e+00, 3.28381076e-03, 0.00000000e+00,

3.72884543e-01, 0.00000000e+00],

[0.00000000e+00, 2.62079735e-04, 3.61098295e-02, 2.32336069e-04,

2.30509332e-01, 0.00000000e+00],

[1.12517699e-02, 2.12323850e-03, 1.60969841e-02, 1.02484866e-02,

3.25459591e-01, 3.75880596e-02],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

4.18955707e-01, 3.57698689e-04],

[3.08373460e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

0.00000000e+00, 0.00000000e+00],

[3.68181576e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

0.00000000e+00, 0.00000000e+00],

[3.97953206e-01, 2.81698167e-02, 3.67005937e-03, 1.70066802e-02,

1.95966668e-03, 2.11644501e-02],

[3.75802488e-01, 2.07517079e-03, 0.00000000e+00, 3.72154373e-02,

0.00000000e+00, 5.85927894e-03],

[4.38037394e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

0.00000000e+00, 0.00000000e+00],

[4.57890626e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

0.00000000e+00, 0.00000000e+00],

[2.75483006e-01, 4.46948936e-03, 0.00000000e+00, 5.29655484e-02,

0.00000000e+00, 1.90997682e-02],

[4.45203267e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

5.48695813e-03, 0.00000000e+00],

[2.92746533e-01, 1.33662475e-02, 1.14261498e-02, 1.05200030e-02,

1.87695369e-01, 9.23965223e-03],

[3.78274425e-01, 1.43967752e-02, 0.00000000e+00, 9.85239494e-02,

1.35899666e-02, 0.00000000e+00]])

In [3]: titles

Out[3]:

['HTTP 404',

'Alexa Internet',

'Internet Explorer',

'HTTP cookie',

'Google Search',

'Tumblr',

'Hypertext Transfer Protocol',

'Social search',

'Firefox',

'LinkedIn',

'Global warming',

'Nationally Appropriate Mitigation Action',

'Nigel Lawson',

'Connie Hedegaard',

'Climate change',

'Kyoto Protocol',

'350.org',

'Greenhouse gas emissions by the United States',

'2010 United Nations Climate Change Conference',

'2007 United Nations Climate Change Conference',

'Angelina Jolie',

'Michael Fassbender',

'Denzel Washington',

'Catherine Zeta-Jones',

'Jessica Biel',

'Russell Crowe',

'Mila Kunis',

'Dakota Fanning',

'Anne Hathaway',

'Jennifer Aniston',

'France national football team',

'Cristiano Ronaldo',

'Arsenal F.C.',

'Radamel Falcao',

'Zlatan Ibrahimović',

'Colombia national football team',

'2014 FIFA World Cup qualification',

'Football',

'Neymar',

'Franck Ribéry',

'Tonsillitis',

'Hepatitis B',

'Doxycycline',

'Leukemia',

'Gout',

'Hepatitis C',

'Prednisone',

'Fever',

'Gabapentin',

'Lymphoma',

'Chad Kroeger',

'Nate Ruess',

'The Wanted',

'Stevie Nicks',

'Arctic Monkeys',

'Black Sabbath',

'Skrillex',

'Red Hot Chili Peppers',

'Sepsis',

'Adam Levine']

In [4]: pd.DataFrame(nmf\_features, index=titles)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

pd.DataFrame(nmf\_features, index=titles)

NameError: name 'pd' is not defined

In [5]: import pandas as pd

In [6]: pd.DataFrame(nmf\_features, index=titles)

Out[6]:

0 ... 5

HTTP 404 0.000000 ... 0.440465

Alexa Internet 0.000000 ... 0.566605

Internet Explorer 0.003821 ... 0.398646

HTTP cookie 0.000000 ... 0.381740

Google Search 0.000000 ... 0.485517

Tumblr 0.012929 ... 0.334522

Hypertext Transfer Protocol 0.000000 ... 0.359061

Social search 0.000000 ... 0.490977

Firefox 0.015427 ... 0.480774

LinkedIn 0.011174 ... 0.338289

Global warming 0.000000 ... 0.000000

Nationally Appropriate Mitigation Action 0.000000 ... 0.000000

Nigel Lawson 0.012013 ... 0.019260

Connie Hedegaard 0.003935 ... 0.000000

Climate change 0.004638 ... 0.003081

Kyoto Protocol 0.000000 ... 0.000000

350.org 0.005650 ... 0.011333

Greenhouse gas emissions by the United States 0.000000 ... 0.000000

2010 United Nations Climate Change Conference 0.000000 ... 0.000000

2007 United Nations Climate Change Conference 0.000000 ... 0.000000

Angelina Jolie 0.000000 ... 0.013785

Michael Fassbender 0.000000 ... 0.000000

Denzel Washington 0.000000 ... 0.000000

Catherine Zeta-Jones 0.000000 ... 0.000000

Jessica Biel 0.000000 ... 0.000000

Russell Crowe 0.098839 ... 0.005222

Mila Kunis 0.000000 ... 0.007135

Dakota Fanning 0.013147 ... 0.011631

Anne Hathaway 0.003845 ... 0.000000

Jennifer Aniston 0.002252 ... 0.014948

France national football team 0.000000 ... 0.000452

Cristiano Ronaldo 0.001534 ... 0.000440

Arsenal F.C. 0.005388 ... 0.002901

Radamel Falcao 0.000000 ... 0.000000

Zlatan Ibrahimović 0.000000 ... 0.000000

Colombia national football team 0.000000 ... 0.000000

2014 FIFA World Cup qualification 0.006105 ... 0.004743

Football 0.006474 ... 0.009521

Neymar 0.007991 ... 0.000000

Franck Ribéry 0.000000 ... 0.000000

Tonsillitis 0.000000 ... 0.000000

Hepatitis B 0.000000 ... 0.000000

Doxycycline 0.000000 ... 0.005838

Leukemia 0.000000 ... 0.000000

Gout 0.001781 ... 0.000000

Hepatitis C 0.000000 ... 0.000000

Prednisone 0.000000 ... 0.000000

Fever 0.000000 ... 0.000000

Gabapentin 0.011252 ... 0.037588

Lymphoma 0.000000 ... 0.000358

Chad Kroeger 0.308373 ... 0.000000

Nate Ruess 0.368182 ... 0.000000

The Wanted 0.397953 ... 0.021164

Stevie Nicks 0.375802 ... 0.005859

Arctic Monkeys 0.438037 ... 0.000000

Black Sabbath 0.457891 ... 0.000000

Skrillex 0.275483 ... 0.019100

Red Hot Chili Peppers 0.445203 ... 0.000000

Sepsis 0.292747 ... 0.009240

Adam Levine 0.378274 ... 0.000000

[60 rows x 6 columns]

<script.py> output:

0 0.003845

1 0.000000

2 0.000000

3 0.575711

4 0.000000

5 0.000000

Name: Anne Hathaway, dtype: float64

0 0.000000

1 0.005601

2 0.000000

3 0.422380

4 0.000000

5 0.000000

Name: Denzel Washington, dtype: float64

In [7]:

+100 XP

Great work! Notice that for both actors, the NMF feature 3 has by far the highest value. This means that both articles are reconstructed using mainly the 3rd NMF component. In the next video, you'll see why: NMF components represent topics (for instance, acting!).

**Exercise**

**Exercise**

**NMF reconstructs samples**

In this exercise, you'll check your understanding of how NMF reconstructs samples from its components using the NMF feature values. On the right are the components of an NMF model. If the NMF feature values of a sample are [2, 1], then which of the following is *most likely* to represent the original sample? A pen and paper will help here! You have to apply the same technique Ben used in the video to reconstruct the sample [0.1203 0.1764 0.3195 0.141].

**Instructions**

**50 XP**

**Possible Answers**

[2.2, 1.1, 2.1].

[0.5, 1.6, 3.1].

[-4.0, 1.0, -2.0].

**Submit Answer**

[**Take Hint (-15 XP)**](javascript:void(0))

[[1. 0.5 0. ]

[0.2 0.1 2.1]]

In [1]:

+50 XP

Well done, you've got it!

**Exercise**

**Exercise**

**NMF learns topics of documents**

In the video, you learned when NMF is applied to documents, the components correspond to topics of documents, and the NMF features reconstruct the documents from the topics. Verify this for yourself for the NMF model that you built earlier using the Wikipedia articles. Previously, you saw that the 3rd NMF feature value was high for the articles about actors Anne Hathaway and Denzel Washington. In this exercise, identify the topic of the corresponding NMF component.

The NMF model you built earlier is available as model, while words is a list of the words that label the columns of the word-frequency array.

After you are done, take a moment to recognise the topic that the articles about Anne Hathaway and Denzel Washington have in common!

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Import pandas as pd.
* Create a DataFrame components\_df from model.components\_, setting columns=words so that columns are labeled by the words.
* Print components\_df.shape to check the dimensions of the DataFrame.
* Use the .iloc[] accessor on the DataFrame components\_df to select row 3. Assign the result to component.
* Call the .nlargest() method of component, and print the result. This gives the five words with the highest values for that component.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Did you define the variable component without errors?

# Import pandas

import pandas as pd

# Create a DataFrame: components\_df

components\_df = pd.DataFrame(model.components\_, columns=words)

# Print the shape of the DataFrame

print(components\_df.shape)

# Select row 3: component

component = components\_df.iloc[3]

# Print result of nlargest

print(component.nlargest())

<script.py> output:

(6, 13125)

Traceback (most recent call last):

File "script.py", line 11, in <module>

component = component.nlargest()

NameError: name 'component' is not defined

<script.py> output:

(6, 13125)

Traceback (most recent call last):

File "script.py", line 11, in <module>

component = components\_df.nlargest()

TypeError: nlargest() missing 2 required positional arguments: 'n' and 'columns'

<script.py> output:

(6, 13125)

Traceback (most recent call last):

File "script.py", line 11, in <module>

component = model.components\_.nlargest()

AttributeError: 'numpy.ndarray' object has no attribute 'nlargest'

<script.py> output:

(6, 13125)

Traceback (most recent call last):

File "script.py", line 11, in <module>

component = components\_df.nlargest()

TypeError: nlargest() missing 2 required positional arguments: 'n' and 'columns'

<script.py> output:

(6, 13125)

film 0.627877

award 0.253131

starred 0.245284

role 0.211451

actress 0.186398

Name: 3, dtype: float64

In [1]:

+100 XP

Great work! Take a moment to recognise the topics that the articles about Anne Hathaway and Denzel Washington have in common!

**Exercise**

**Exercise**

**Explore the LED digits dataset**

In the following exercises, you'll use NMF to decompose grayscale images into their commonly occurring patterns. Firstly, explore the image dataset and see how it is encoded as an array. You are given 100 images as a 2D array samples, where each row represents a single 13x8 image. The images in your dataset are pictures of a LED digital display.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Import matplotlib.pyplot as plt.
* Select row 0 of samples and assign the result to digit. For example, to select column 2 of an array a, you could use a[:,2]. Remember that since samples is a NumPy array, you can't use the .loc[] or iloc[] accessors to select specific rows or columns.
* Print digit. This has been done for you. Notice that it is a 1D array of 0s and 1s.
* Use the .reshape() method of digit to get a 2D array with shape (13, 8). Assign the result to bitmap.
* Print bitmap, and notice that the 1s show the digit 7!
* Use the plt.imshow() function to display bitmap as an image.

[**Take Hint (-30 XP)**](javascript:void(0))

# Import pyplot

from matplotlib import pyplot as plt

# Select the 0th row: digit

digit = samples[0]

# Print digit

print(digit)

# Reshape digit to a 13x8 array: bitmap

bitmap = digit.reshape(13, 8)

# Print bitmap

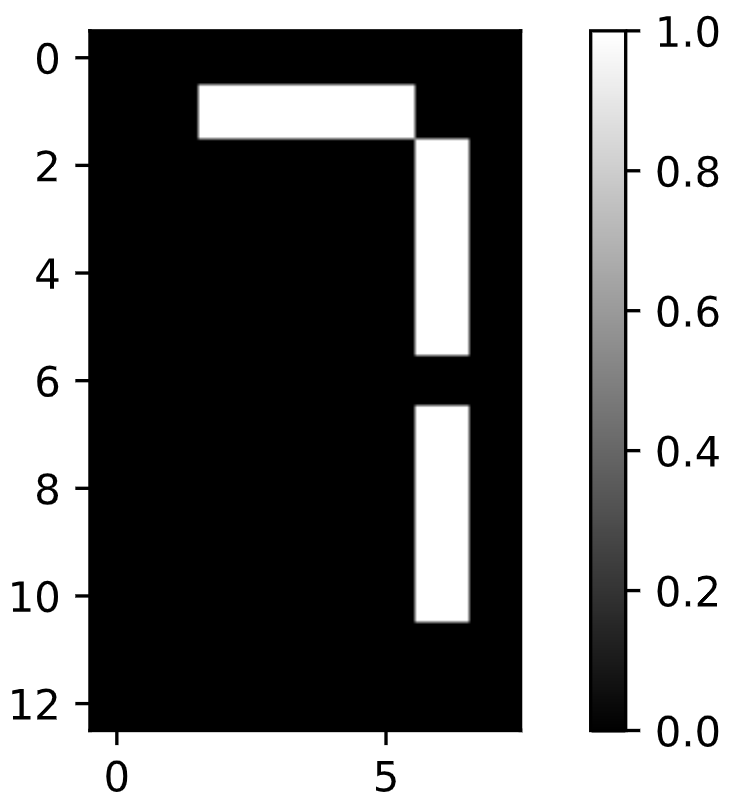
print(bitmap)

# Use plt.imshow to display bitmap

plt.imshow(bitmap, cmap='gray', interpolation='nearest')

plt.colorbar()

plt.show()



In [1]: samples

Out[1]:

array([[0., 0., 0., ..., 0., 0., 0.],

[0., 0., 0., ..., 0., 0., 0.],

[0., 0., 0., ..., 0., 0., 0.],

...,

[0., 0., 0., ..., 0., 0., 0.],

[0., 0., 0., ..., 0., 0., 0.],

[0., 0., 0., ..., 0., 0., 0.]])

In [2]: samples.loc

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

samples.loc

AttributeError: 'numpy.ndarray' object has no attribute 'loc'

<script.py> output:

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.

0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0.

0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0.]

[[0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 1. 1. 1. 1. 0. 0.]

[0. 0. 0. 0. 0. 0. 1. 0.]

[0. 0. 0. 0. 0. 0. 1. 0.]

[0. 0. 0. 0. 0. 0. 1. 0.]

[0. 0. 0. 0. 0. 0. 1. 0.]

[0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 1. 0.]

[0. 0. 0. 0. 0. 0. 1. 0.]

[0. 0. 0. 0. 0. 0. 1. 0.]

[0. 0. 0. 0. 0. 0. 1. 0.]

[0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0.]]

In [3]:

+100 XP

Excellent job! You'll explore this dataset further in the next exercise and see for yourself how NMF can learn the parts of images.

**Exercise**

**Exercise**

**NMF learns the parts of images**

Now use what you've learned about NMF to decompose the digits dataset. You are again given the digit images as a 2D array samples. This time, you are also provided with a function show\_as\_image() that displays the image encoded by any 1D array:

def show\_as\_image(sample):

bitmap = sample.reshape((13, 8))

plt.figure()

plt.imshow(bitmap, cmap='gray', interpolation='nearest')

plt.colorbar()

plt.show()

After you are done, take a moment to look through the plots and notice how NMF has expressed the digit as a sum of the components!

**Instructions**

**100 XP**

* Import NMF from sklearn.decomposition.
* Create an NMF instance called model with 7 components. (7 is the number of cells in an LED display).
* Apply the .fit\_transform() method of model to samples. Assign the result to features.
* To each component of the model (accessed via model.components\_), apply the show\_as\_image() function to that component inside the loop.
* Assign the row 0 of features to digit\_features.
* Print digit\_features.

[**Take Hint (-30 XP)**](javascript:void(0))

# Import NMF

from sklearn.decomposition import NMF

# Create an NMF model: model

model = NMF(7)

# Apply fit\_transform to samples: features

features = model.fit\_transform(samples)

# Call show\_as\_image on each component

for component in model.components\_:

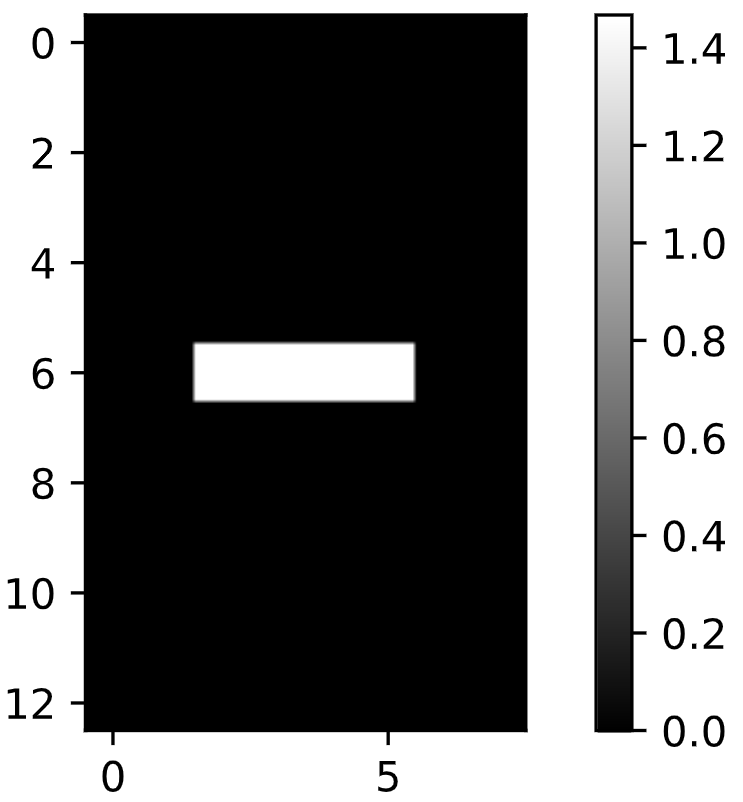
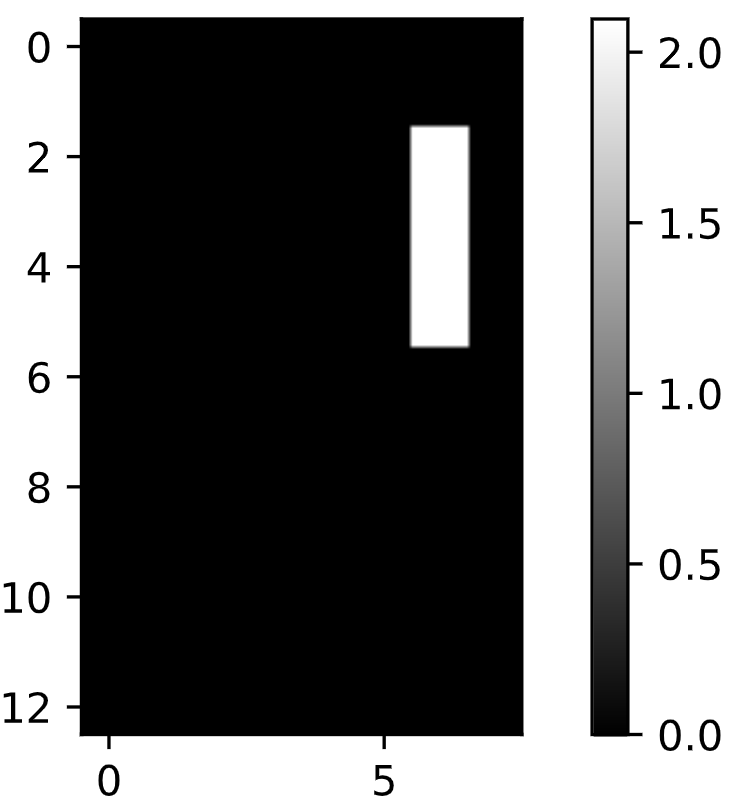
show\_as\_image(component)

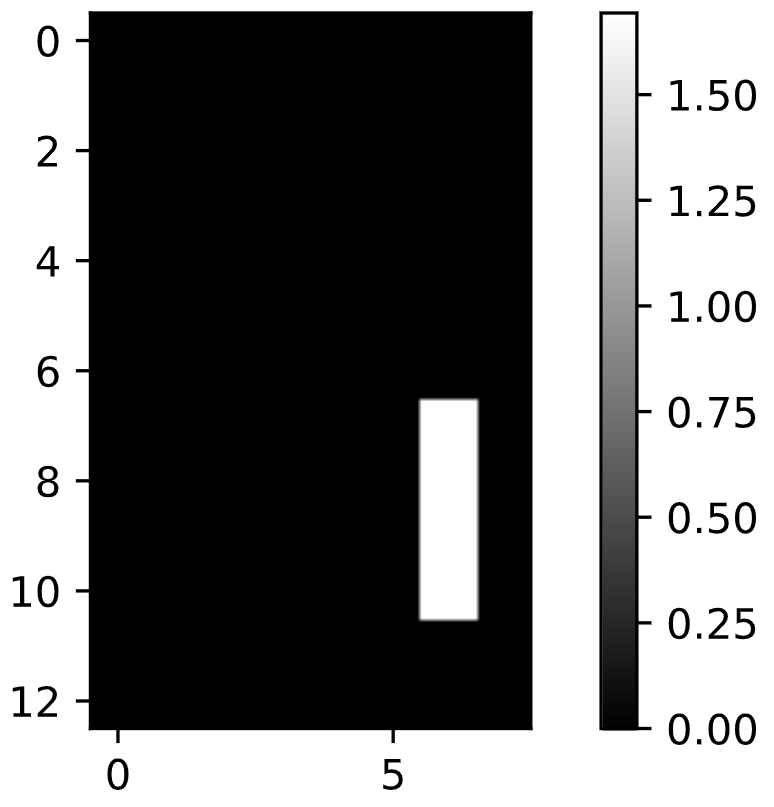
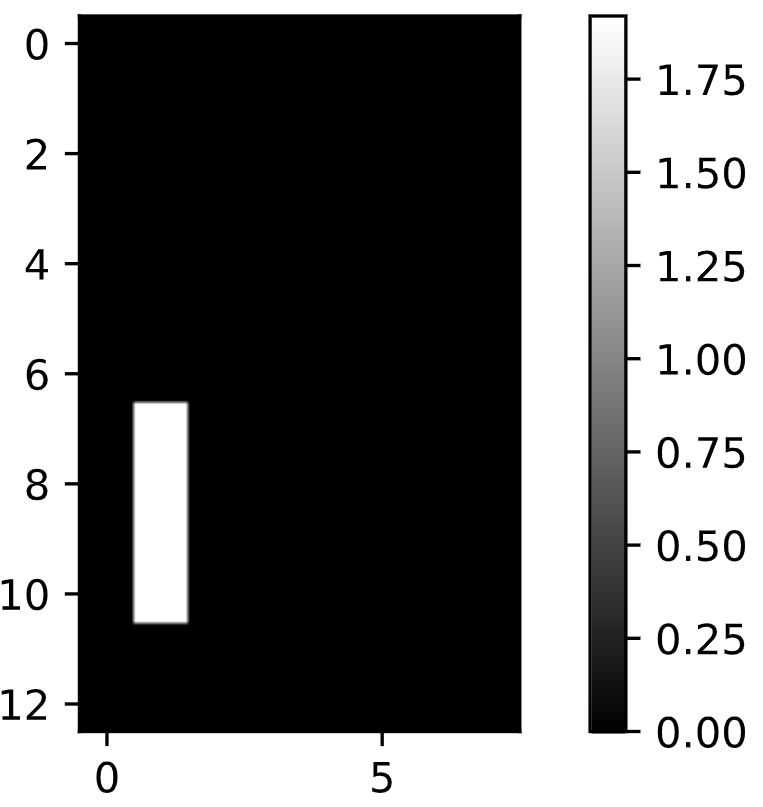
# Assign the 0th row of features: digit\_features

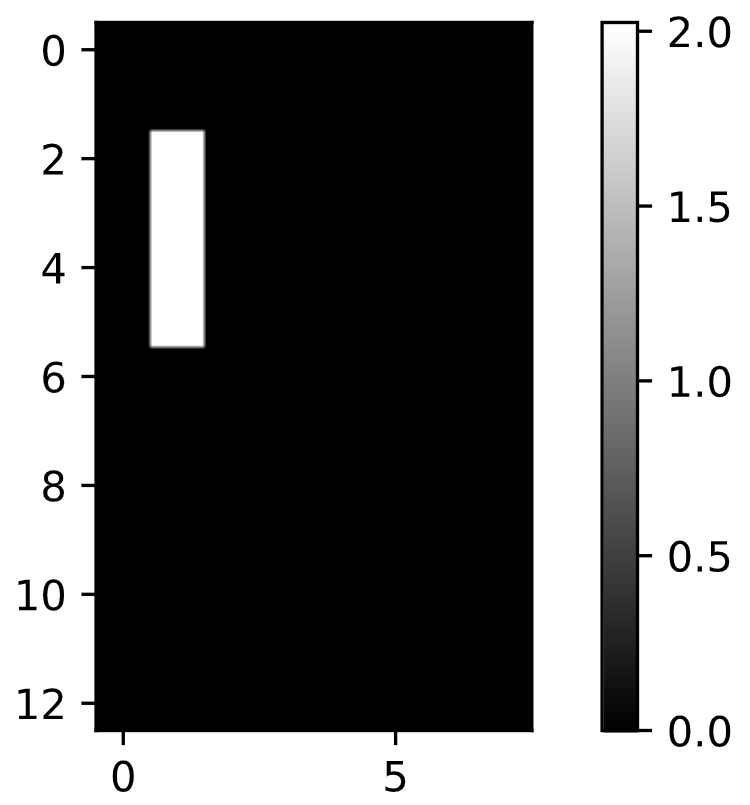
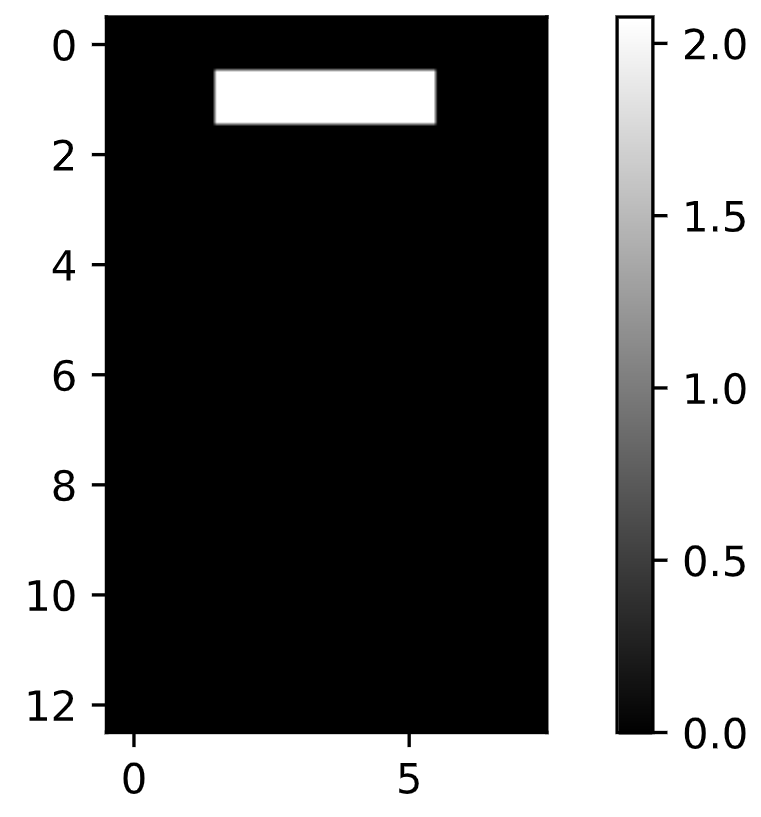
digit\_features = features[0]

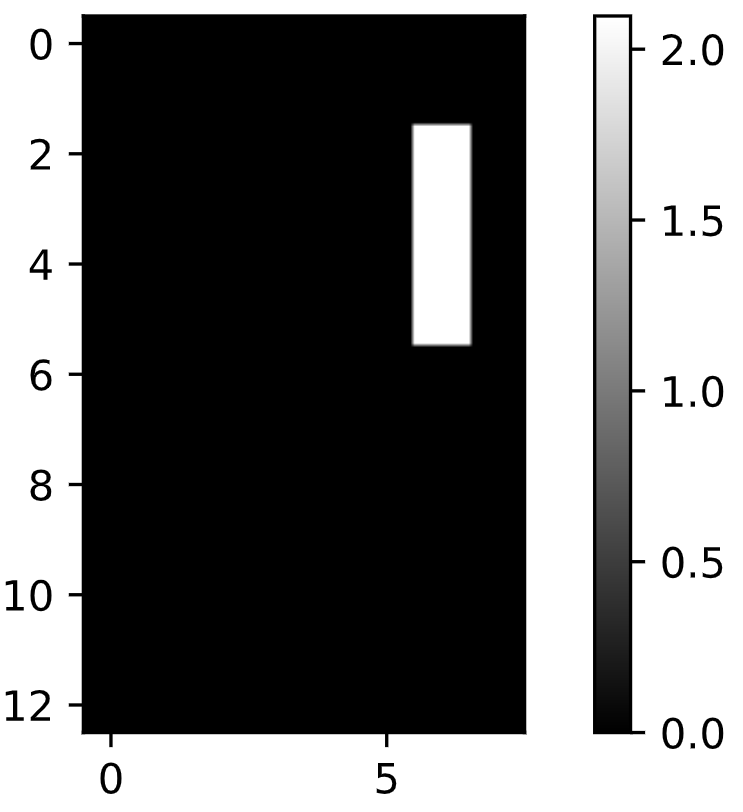
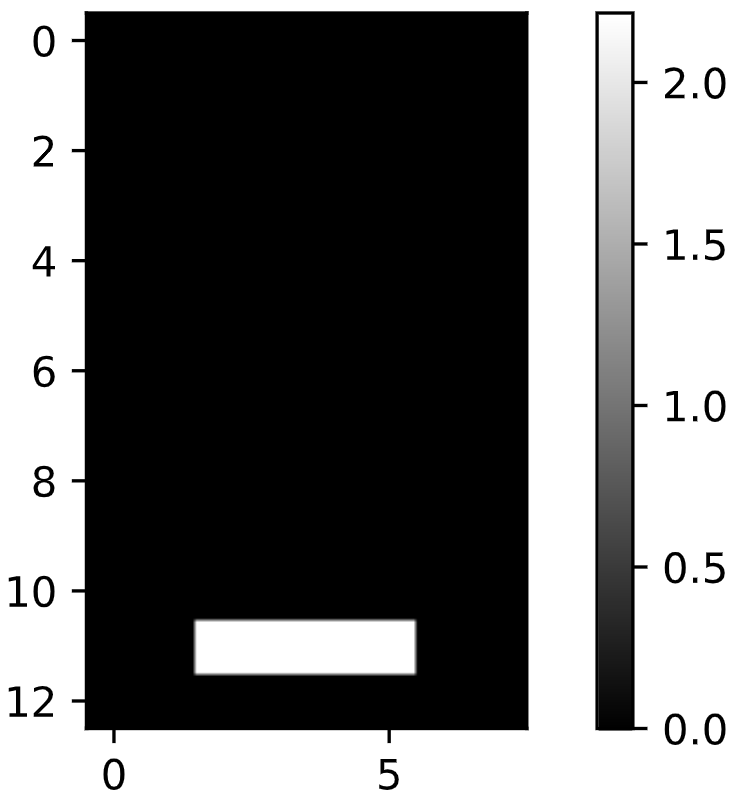
# Print digit\_features

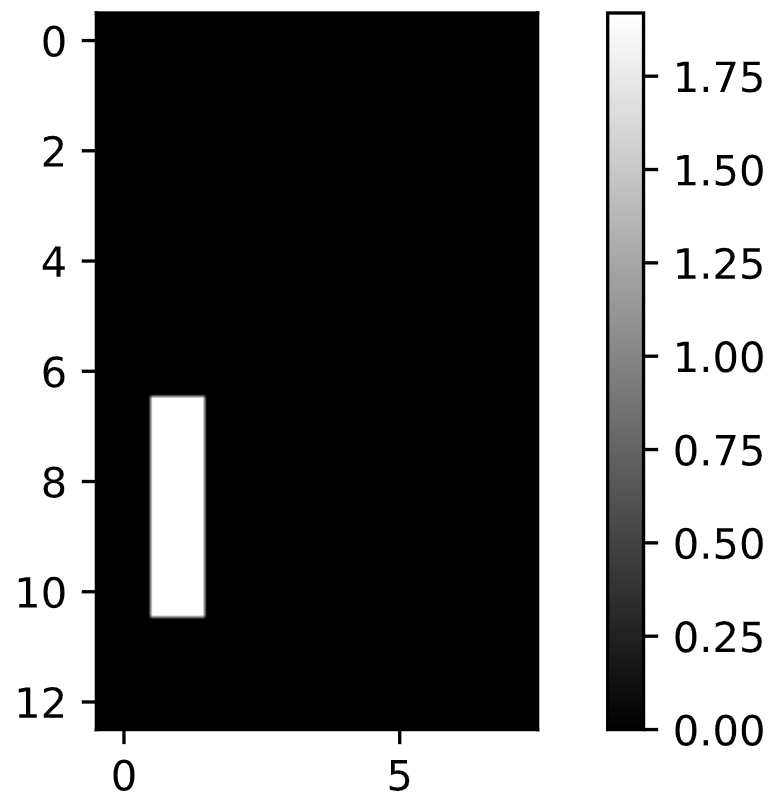
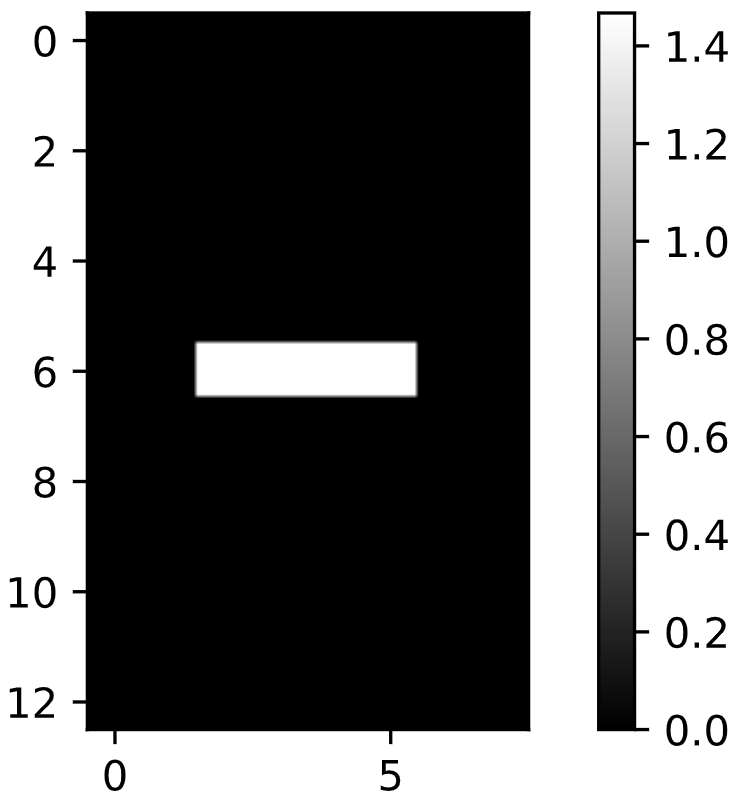
print(digit\_features)

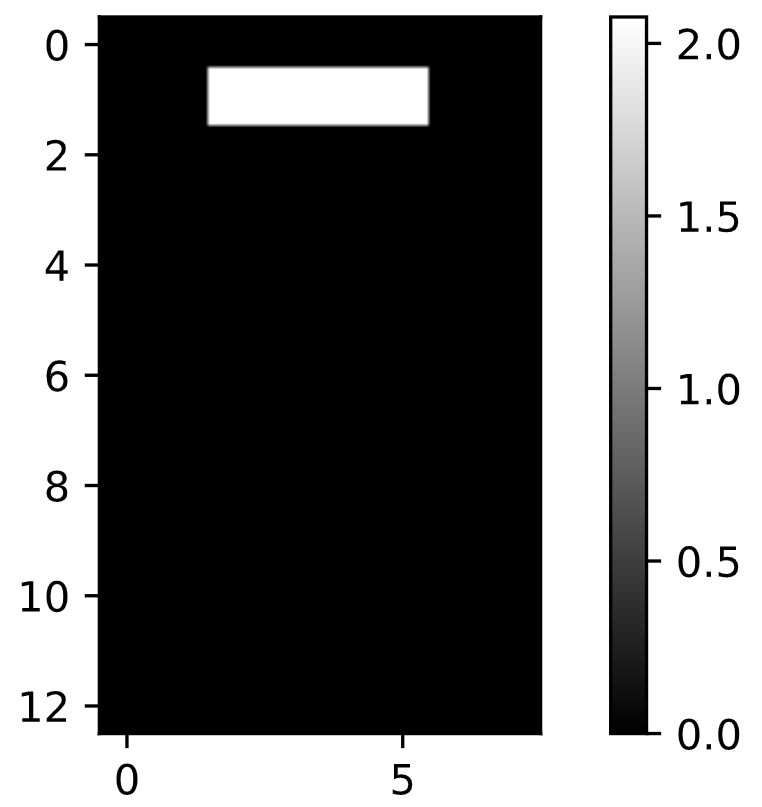
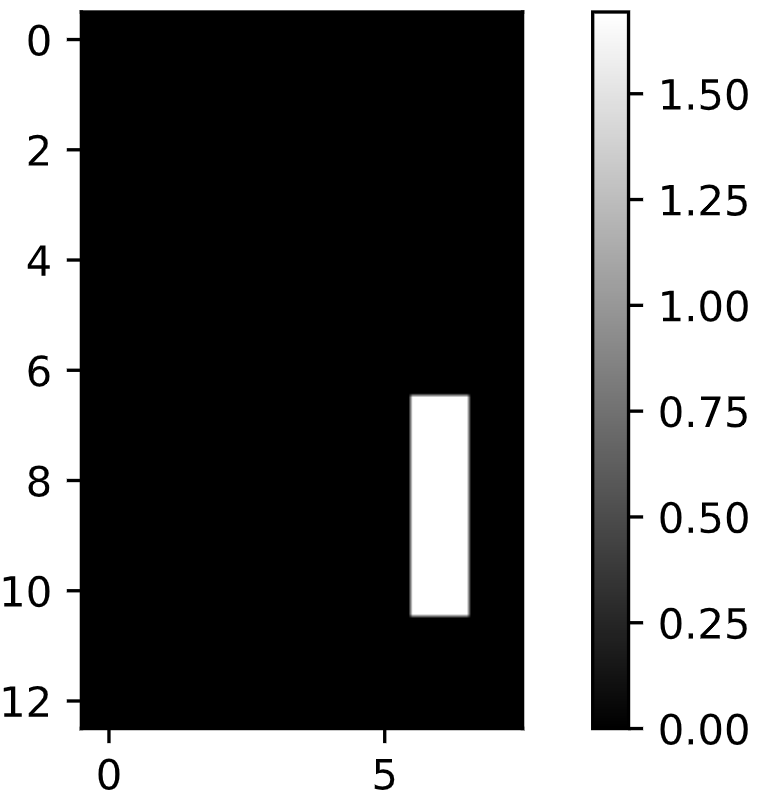


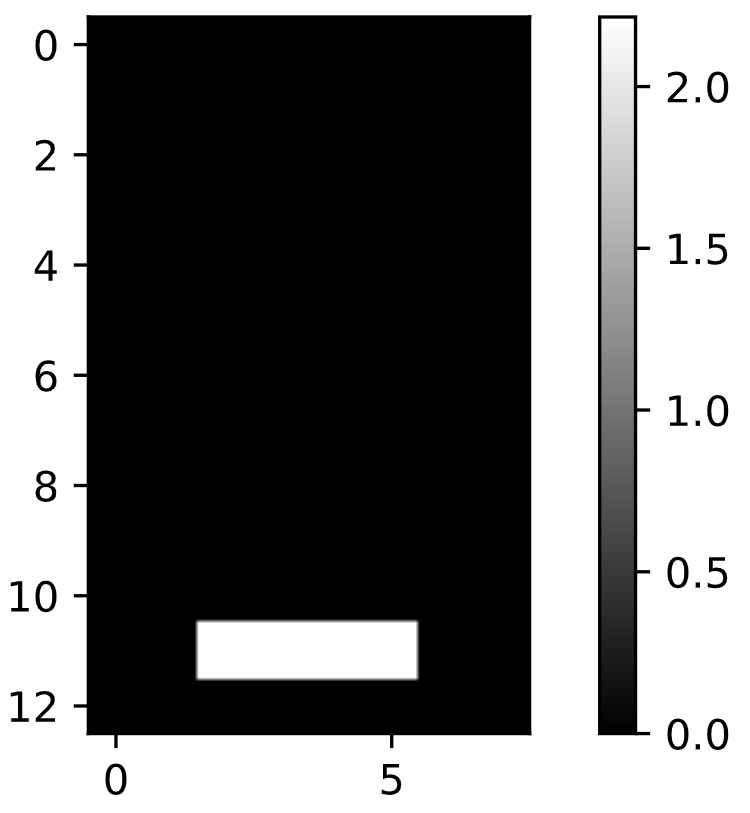
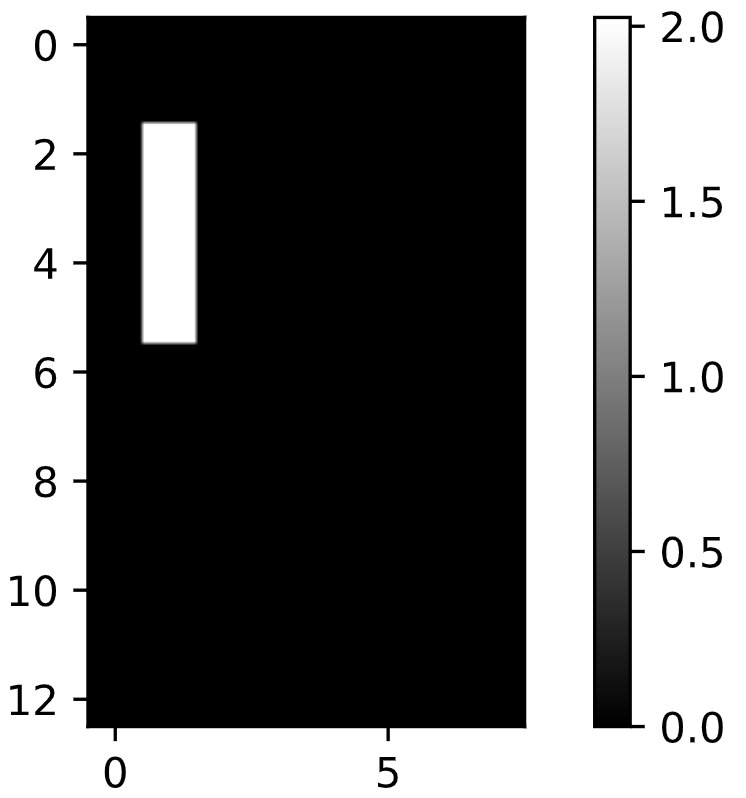












<script.py> output:

[4.76823559e-01 0.00000000e+00 0.00000000e+00 5.90605054e-01

4.81559442e-01 0.00000000e+00 7.37557191e-16]

<script.py> output:

[4.76823559e-01 0.00000000e+00 0.00000000e+00 5.90605054e-01

4.81559442e-01 0.00000000e+00 7.37557191e-16]

In [1]: samples

Out[1]:

array([[0., 0., 0., ..., 0., 0., 0.],

[0., 0., 0., ..., 0., 0., 0.],

[0., 0., 0., ..., 0., 0., 0.],

...,

[0., 0., 0., ..., 0., 0., 0.],

[0., 0., 0., ..., 0., 0., 0.],

[0., 0., 0., ..., 0., 0., 0.]])

In [2]: samples.tolist()

Out[2]:

[[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0],

[0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

1.0,

1.0,

1.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

1.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0,

0.0]]

In [3]: import pyperclip as ppc

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

import pyperclip as ppc

ModuleNotFoundError: No module named 'pyperclip'

In [4]:

+100 XP

Great work! Take a moment to look through the plots and notice how NMF has expressed the digit as a sum of the components!

**Exercise**

**Exercise**

**PCA doesn't learn parts**

Unlike NMF, PCA *doesn't* learn the parts of things. Its components do not correspond to topics (in the case of documents) or to parts of images, when trained on images. Verify this for yourself by inspecting the components of a PCA model fit to the dataset of LED digit images from the previous exercise. The images are available as a 2D array samples. Also available is a modified version of the show\_as\_image() function which colors a pixel red if the value is negative.

After submitting the answer, notice that the components of PCA do not represent meaningful parts of images of LED digits!

**Instructions**

**100 XP**

* Import PCA from sklearn.decomposition.
* Create a PCA instance called model with 7 components.
* Apply the .fit\_transform() method of model to samples. Assign the result to features.
* To each component of the model (accessed via model.components\_), apply the show\_as\_image() function to that component inside the loop.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Have you specified the arguments for model.fit\_transform() using the right syntax?

# Import PCA

from sklearn.decomposition import PCA

# Create a PCA instance: model

model = PCA(7)

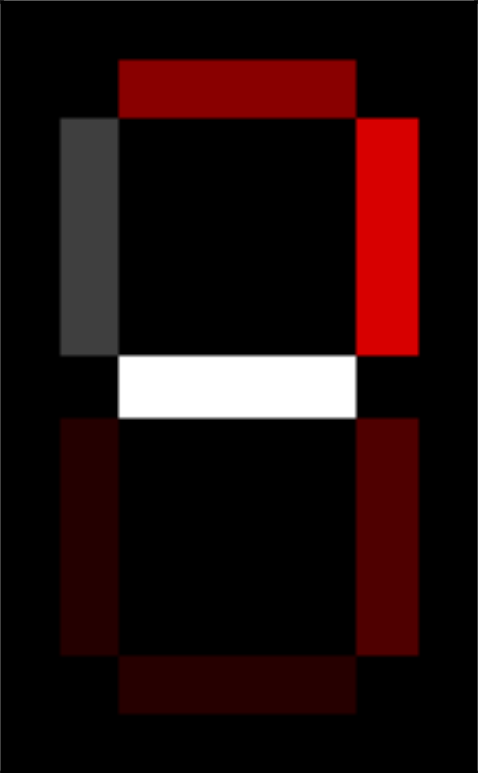
# Apply fit\_transform to samples: features

features = model.fit\_transform(samples)

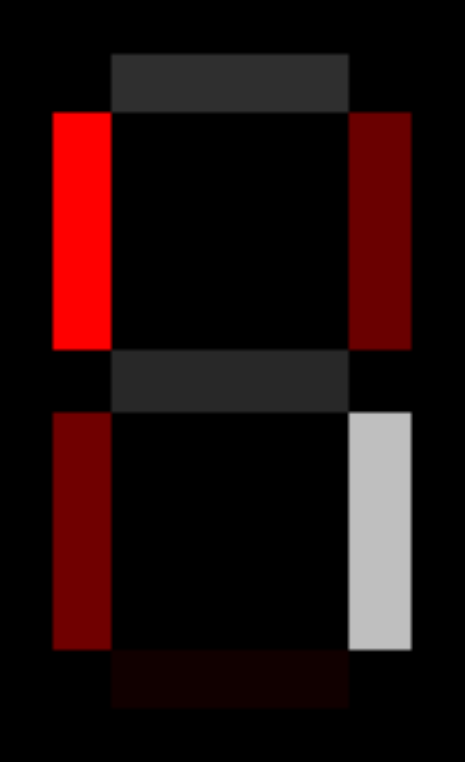
# Call show\_as\_image on each component

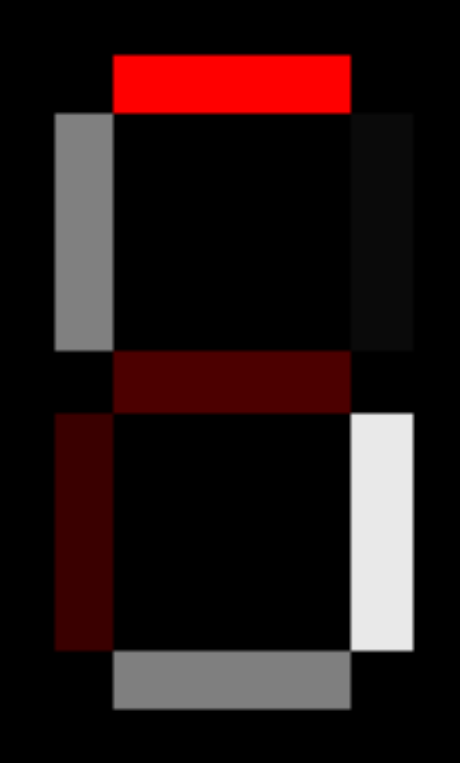
for component in model.components\_:

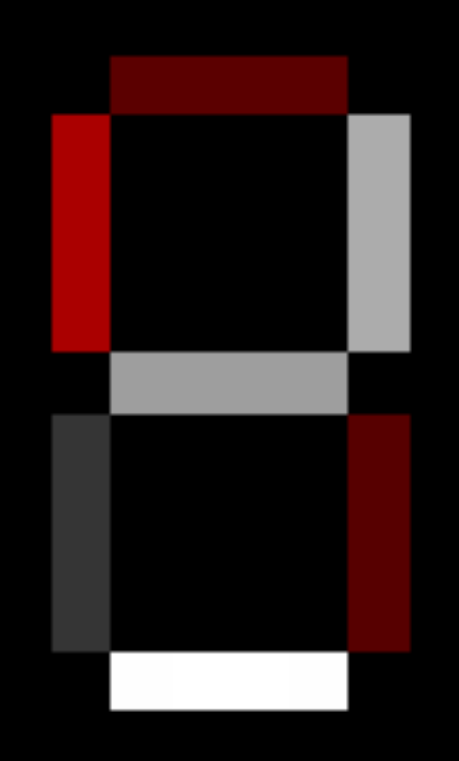
show\_as\_image(component)

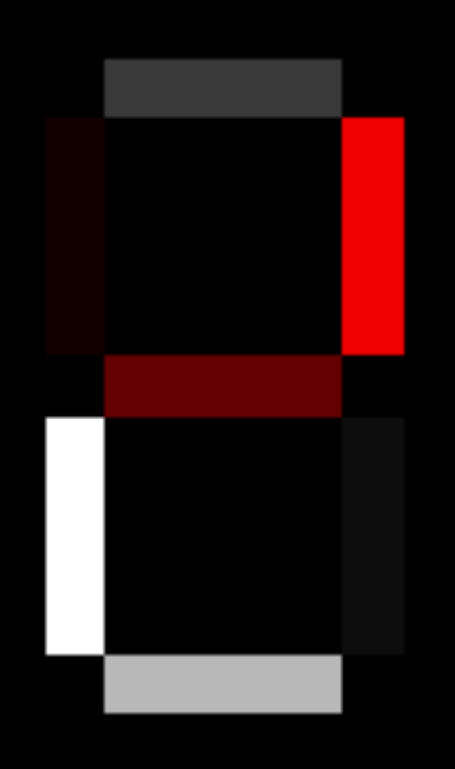














Traceback (most recent call last):

File "script.py", line 8, in <module>

features = model.fit\_transform()

TypeError: fit\_transform() missing 1 required positional argument: 'X'

In [1]:

+100 XP

Great work! Notice that the components of PCA do not represent meaningful parts of images of LED digits!

**Exercise**

**Exercise**

**Which articles are similar to 'Cristiano Ronaldo'?**

In the video, you learned how to use NMF features and the cosine similarity to find similar articles. Apply this to your NMF model for popular Wikipedia articles, by finding the articles most similar to the article about the footballer Cristiano Ronaldo. The NMF features you obtained earlier are available as nmf\_features, while titles is a list of the article titles.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Import normalize from sklearn.preprocessing.
* Apply the normalize() function to nmf\_features. Store the result as norm\_features.
* Create a DataFrame df from norm\_features, using titles as an index.
* Use the .loc[] accessor of df to select the row of 'Cristiano Ronaldo'. Assign the result to article.
* Apply the .dot() method of df to article to calculate the cosine similarity of every row with article.
* Print the result of the .nlargest() method of similarities to display the most similiar articles. This has been done for you, so hit 'Submit Answer' to see the result!

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Inside pd.DataFrame(), did you set the index to be titles?

# Perform the necessary imports

import pandas as pd

from sklearn.preprocessing import normalize

# Normalize the NMF features: norm\_features

norm\_features = normalize(nmf\_features)

# Create a DataFrame: df

df = pd.DataFrame(norm\_features, index=titles)

# Select the row corresponding to 'Cristiano Ronaldo': article

article = df.loc['Cristiano Ronaldo']

# Compute the dot products: similarities

similarities = df.dot(article)

# Display those with the largest cosine similarity

print(similarities.nlargest())

Traceback (most recent call last):

File "script.py", line 9, in <module>

df = pd.DataFrame(norm\_features, index=index)

NameError: name 'index' is not defined

In [1]:

<script.py> output:

Cristiano Ronaldo 1.000000

Franck Ribéry 0.999972

Radamel Falcao 0.999942

Zlatan Ibrahimović 0.999942

France national football team 0.999923

dtype: float64

In [1]:

+100 XP

Great work - although you may need to know a little about football (or soccer, depending on where you're from!) to be able to evaluate for yourself the quality of the computed similarities!

**Exercise**

**Exercise**

**Recommend musical artists part I**

In this exercise and the next, you'll use what you've learned about NMF to recommend popular music artists! You are given a sparse array artists whose rows correspond to artists and whose columns correspond to users. The entries give the number of times each artist was listened to by each user.

In this exercise, build a pipeline and transform the array into normalized NMF features. The first step in the pipeline, MaxAbsScaler, transforms the data so that all users have the same influence on the model, regardless of how many different artists they've listened to. In the next exercise, you'll use the resulting normalized NMF features for recommendation!

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Import:
  + NMF from sklearn.decomposition.
  + Normalizer and MaxAbsScaler from sklearn.preprocessing.
  + make\_pipeline from sklearn.pipeline.
* Create an instance of MaxAbsScaler called scaler.
* Create an NMF instance with 20 components called nmf.
* Create an instance of Normalizer called normalizer.
* Create a pipeline called pipeline that chains together scaler, nmf, and normalizer.
* Apply the .fit\_transform() method of pipeline to artists. Assign the result to norm\_features.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Did you import MaxAbsScaler from sklearn.preprocessing?

# Perform the necessary imports

from sklearn.decomposition import NMF

from sklearn.preprocessing import Normalizer, MaxAbsScaler

from sklearn.pipeline import make\_pipeline

# Create a MaxAbsScaler: scaler

scaler = MaxAbsScaler()

# Create an NMF model: nmf

nmf = NMF(20)

# Create a Normalizer: normalizer

normalizer = Normalizer()

# Create a pipeline: pipeline

pipeline = make\_pipeline(scaler, nmf, normalizer)

# Apply fit\_transform to artists: norm\_features

norm\_features = pipeline.fit\_transform(artists)

Traceback (most recent call last):

File "script.py", line 3, in <module>

from sklearn.preprocessing import Normalizer, \_MaxAbsScaler\_\_\_

ImportError: cannot import name '\_MaxAbsScaler\_\_\_'

In [1]:

+100 XP

Excellent work - now that you've computed the normalized NMF features, you'll use them in the next exercise to recommend musical artists!

**Exercise**

**Exercise**

**Recommend musical artists part II**

Suppose you were a big fan of Bruce Springsteen - which other musicial artists might you like? Use your NMF features from the previous exercise and the cosine similarity to find similar musical artists. A solution to the previous exercise has been run, so norm\_features is an array containing the normalized NMF features as rows. The names of the musical artists are available as the list artist\_names.

**Instructions**

**100 XP**

* Import pandas as pd.
* Create a DataFrame df from norm\_features, using artist\_names as an index.
* Use the .loc[] accessor of df to select the row of 'Bruce Springsteen'. Assign the result to artist.
* Apply the .dot() method of df to artist to calculate the dot product of every row with artist. Save the result as similarities.
* Print the result of the .nlargest() method of similarities to display the artists most similar to 'Bruce Springsteen'.

[**Take Hint (-30 XP)**](javascript:void(0))

# Import pandas

import pandas as pd

# Create a DataFrame: df

df = pd.DataFrame(norm\_features, index=artist\_names)

# Select row of 'Bruce Springsteen': artist

artist = df.loc['Bruce Springsteen']

# Compute cosine similarities: similarities

similarities = df.dot(artist)

# Display those with highest cosine similarity

print(similarities.nlargest())

Out[1]:

Out[1]:

[[0.0, 0.0, 0.0, 0.0, 0.0, 0.44046532207868505],

[0.0, 0.0, 0.0, 0.0, 0.0, 0.5666047555002354],

[0.00382058457138028, 0.0, 0.0, 0.0, 0.0, 0.39864642652118515],

[0.0, 0.0, 0.0, 0.0, 0.0, 0.38173975262897275],

[0.0, 0.0, 0.0, 0.0, 0.0, 0.4855171191657692],

[0.012929045895762043,

0.013788918430121082,

0.0077631373334542345,

0.033448732200107065,

0.0,

0.33452181849602475],

[0.0,

0.0,

0.020673865461719697,

0.0,

0.006044862831330426,

0.35906103776311227],

[0.0, 0.0, 0.0, 0.0, 0.0, 0.49097670261050264],

[0.015427415551281362,

0.01428171836433613,

0.0037662874837738215,

0.023711181660103425,

0.02626198931427636,

0.4807743181772023],

[0.01117382975790412,

0.031367681605420104,

0.03094804858961412,

0.06570003445923049,

0.019667739034662016,

0.33828864794249475],

[0.0, 0.0, 0.5307101119243957, 0.0, 0.028367934707988932, 0.0],

[0.0, 0.0, 0.3565030418990846, 0.0, 0.0, 0.0],

[0.012012725486257302,

0.0065003375601440224,

0.31223976926305175,

0.06097708182811259,

0.011386131793104088,

0.01926021023625837],

[0.00393485614917763,

0.006244322343097862,

0.3423672409222086,

0.011076893357076435,

0.0,

0.0],

[0.0046382139837896205,

0.0,

0.43490741444971426,

0.0,

0.03842749235220703,

0.0030813313962713617],

[0.0, 0.0, 0.4832806085149956, 0.0, 0.0, 0.0],

[0.005650165940335701,

0.01835324644326475,

0.37652638353449525,

0.03254609346705307,

0.0,

0.011333443763247264],

[0.0, 0.0, 0.48090531976257883, 0.0, 0.0, 0.0],

[0.0, 0.009018492740773527, 0.5509982437085191, 0.0, 0.0, 0.0],

[0.0, 0.0, 0.4659614344121262, 0.0, 0.0, 0.0],

[0.0,

0.011407893416278674,

0.02086519983591687,

0.5177673271987503,

0.0581408434535283,

0.01378538045062945],

[0.0, 0.0, 0.0, 0.510475272672127, 0.0, 0.0],

[0.0, 0.0056009476023087135, 0.0, 0.42237985381375764, 0.0, 0.0],

[0.0, 0.0, 0.0, 0.43675125265690334, 0.0, 0.0],

[0.0, 0.0, 0.0, 0.49809203614891995, 0.0, 0.0],

[0.09883940181018466,

0.08600294544055205,

0.003910288559057014,

0.3810175317340033,

0.00043923994230136857,

0.005221514812151791],

[0.0, 0.0, 0.0, 0.5721698884509835, 0.0, 0.007135419846343523],

[0.01314687151650309,

0.01048515807623334,

0.0,

0.46890604860455504,

0.0,

0.011630997128980572],

[0.003845486884479646, 0.0, 0.0, 0.5757105566045845, 0.0, 0.0],

[0.0022524416389245994,

0.001387340169568473,

0.0,

0.5279457779288029,

0.012026467070093276,

0.014948389534220842],

[0.0,

0.40754103833474753,

0.0018571141859079255,

0.0,

0.002966104966857928,

0.00045234230543179883],

[0.0015342122692038264,

0.6081623804517682,

0.0005222693281304437,

0.006248535412843345,

0.0011844473347220562,

0.00044007495536628564],

[0.0053881936605831935,

0.2650124103324353,

0.0005385018253337897,

0.018692592929174877,

0.006386511892148212,

0.002901048722165269],

[0.0, 0.6449046024759076, 0.0, 0.0, 0.0, 0.0],

[0.0, 0.6088963048952473, 0.0, 0.0, 0.0, 0.0],

[0.0, 0.3436792308350921, 0.0, 0.0, 0.003977948294432636, 0.0],

[0.006105088494898749,

0.3153072952791194,

0.015487729517922095,

0.0,

0.005062445630429158,

0.0047433544994469464],

[0.006473733275877798,

0.21332481760274732,

0.009494791742539567,

0.045698120688434866,

0.017191448147033796,

0.00952063620406544],

[0.007991472340747817,

0.4675869740943177,

0.0,

0.024342541339265385,

0.0,

0.0],

[0.0, 0.6428088498214294, 0.0, 0.0023585541278437028, 0.0, 0.0],

[0.0, 0.0, 0.0, 0.0, 0.47708011320299437, 0.0],

[0.0, 0.0, 0.0, 0.0, 0.4942532124079175, 0.0],

[0.0,

0.00029906378727979997,

0.002144848130326976,

0.0,

0.3817763176703919,

0.005837804790646698],

[0.0, 0.0, 0.0, 0.005646927714468547, 0.5422381699147378, 0.0],

[0.0017805965385384104,

0.0007843999456315008,

0.0141625537635917,

0.00045981350101621625,

0.42429984825752315,

0.0],

[0.0, 0.0, 0.0, 0.0, 0.5113888208556774, 0.0],

[0.0, 0.0, 0.003283810756315475, 0.0, 0.37288454310533353, 0.0],

[0.0,

0.0002620797347185202,

0.03610982949730708,

0.00023233606876857824,

0.23050933173441862,

0.0],

[0.011251769941793078,

0.0021232385014216594,

0.016096984073158176,

0.010248486596196755,

0.3254595912335115,

0.03758805960465889],

[0.0, 0.0, 0.0, 0.0, 0.41895570743277716, 0.00035769868874922333],

[0.30837346001395866, 0.0, 0.0, 0.0, 0.0, 0.0],

[0.36818157647323585, 0.0, 0.0, 0.0, 0.0, 0.0],

[0.39795320637872106,

0.02816981666879342,

0.003670059373601258,

0.01700668022459409,

0.0019596666837876538,

0.0211644500925536],

[0.37580248826394125,

0.002075170790429256,

0.0,

0.03721543734503068,

0.0,

0.00585927893915046],

[0.43803739443015166, 0.0, 0.0, 0.0, 0.0, 0.0],

[0.4578906257323118, 0.0, 0.0, 0.0, 0.0, 0.0],

[0.27548300641947054,

0.004469489357493207,

0.0,

0.052965548408715805,

0.0,

0.019099768226549983],

[0.4452032674032461, 0.0, 0.0, 0.0, 0.005486958127771287, 0.0],

[0.2927465332876542,

0.013366247528947177,

0.011426149783624887,

0.010520002972259758,

0.1876953685209926,

0.009239652229431601],

[0.37827442516407067,

0.01439677517528748,

0.0,

0.09852394938563469,

0.013589966587864558,

0.0]]

In [2]:

<script.py> output:

Bruce Springsteen 1.000000

Neil Young 0.955896

Van Morrison 0.872452

Leonard Cohen 0.864763

Bob Dylan 0.859047

dtype: float64

In [1]:

+100 XP

Well done, and congratulations on reaching the end of the course!