K Means Clustering

Notebook adapted from the 05.11 K-Means Clustering PythonDataScienceHandbook notebook.
Modified by: Gábor Major
Last Modified date: 2025-02-19

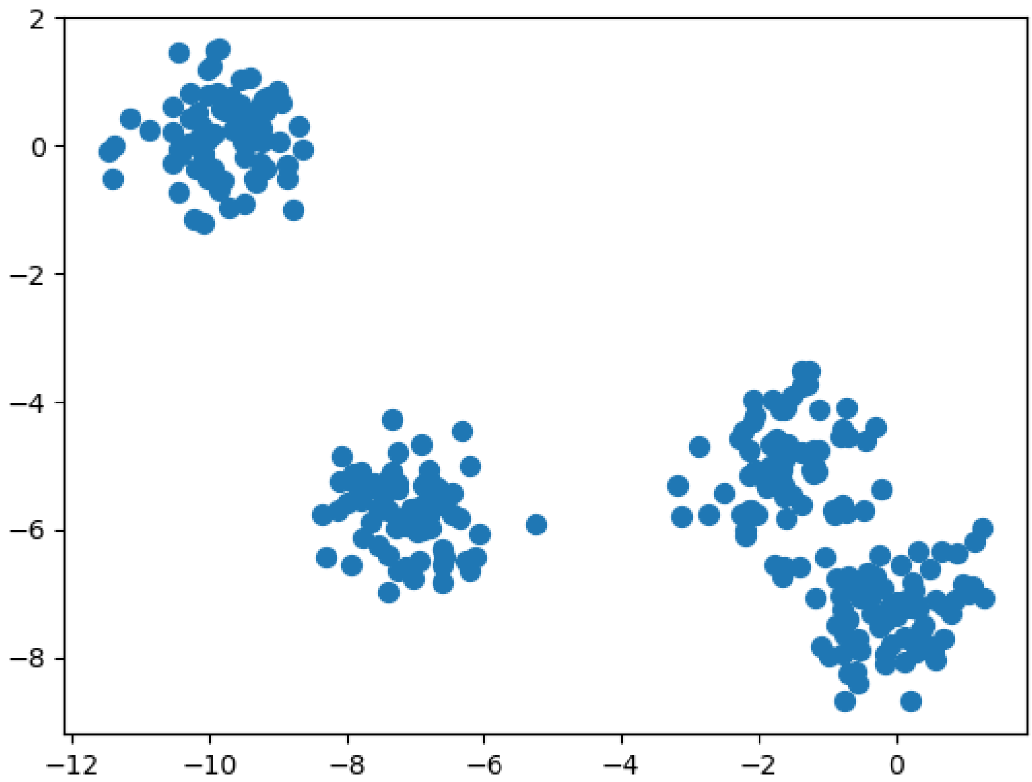
Import libraries.

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

K Means Modelling

First generate test data.

```
from sklearn.datasets import make_blobs
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.60,
random_state=9)
plt.scatter(X[:, 0], X[:, 1], s=50);
```

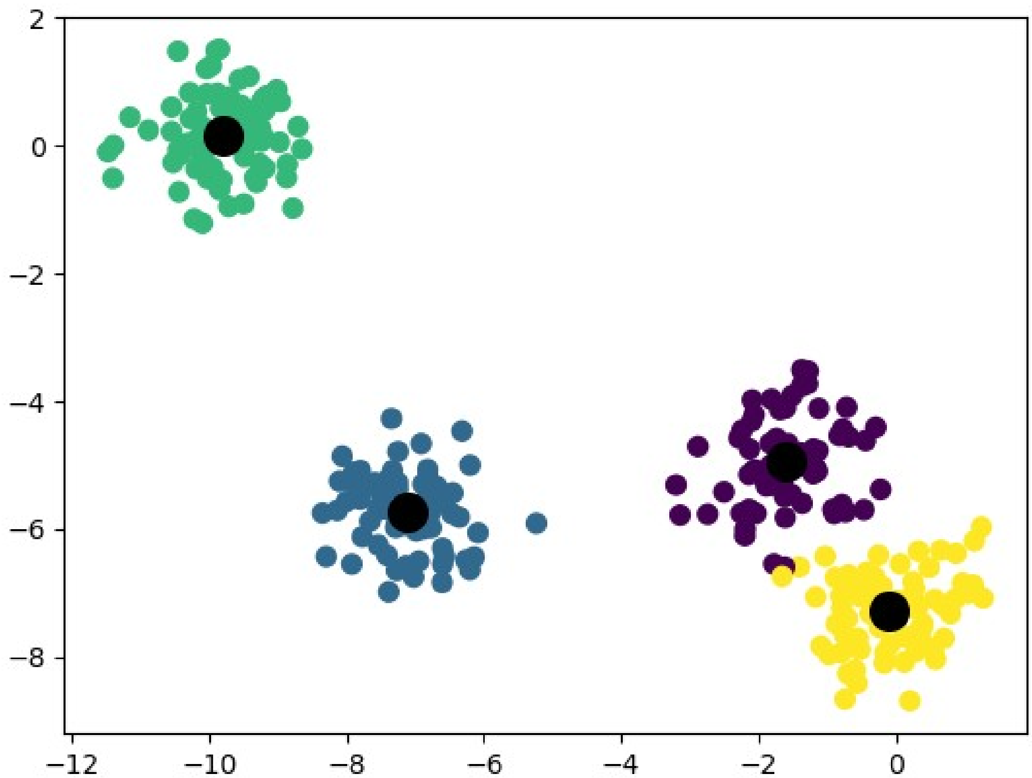


Create K Means model and set the cluster count to 4.

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
```

Plot the clusters that were estimated.

```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200);
```

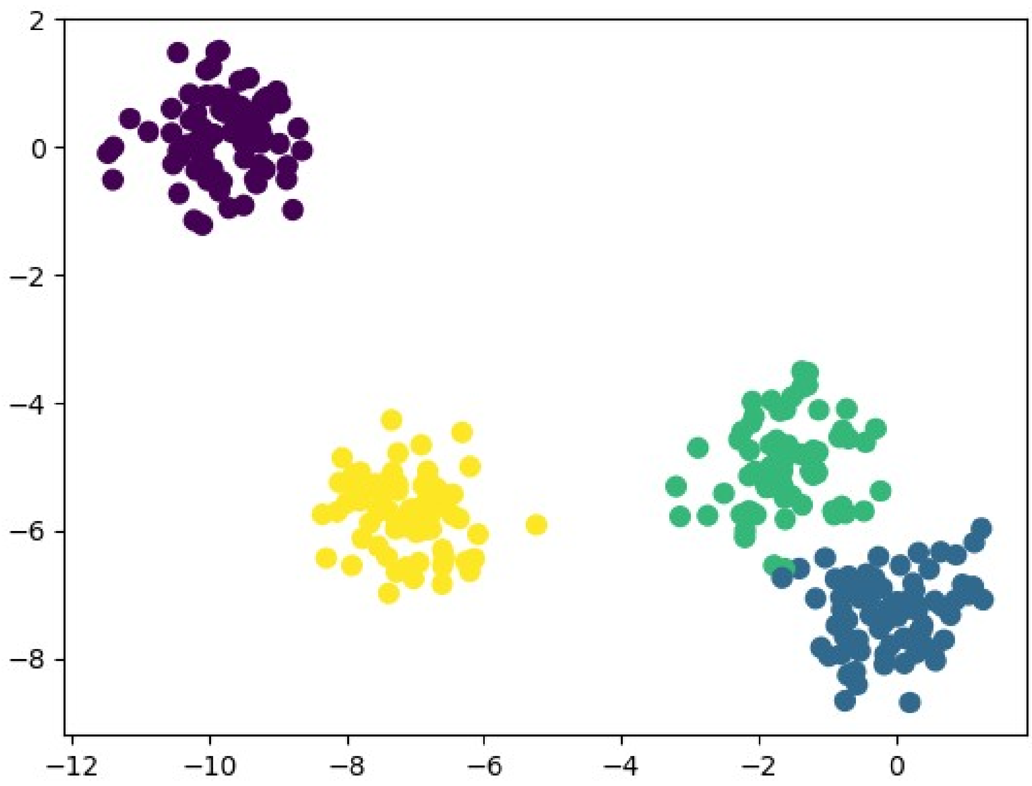


Expectation–Maximization

Implement the basics of K Means Clustering.

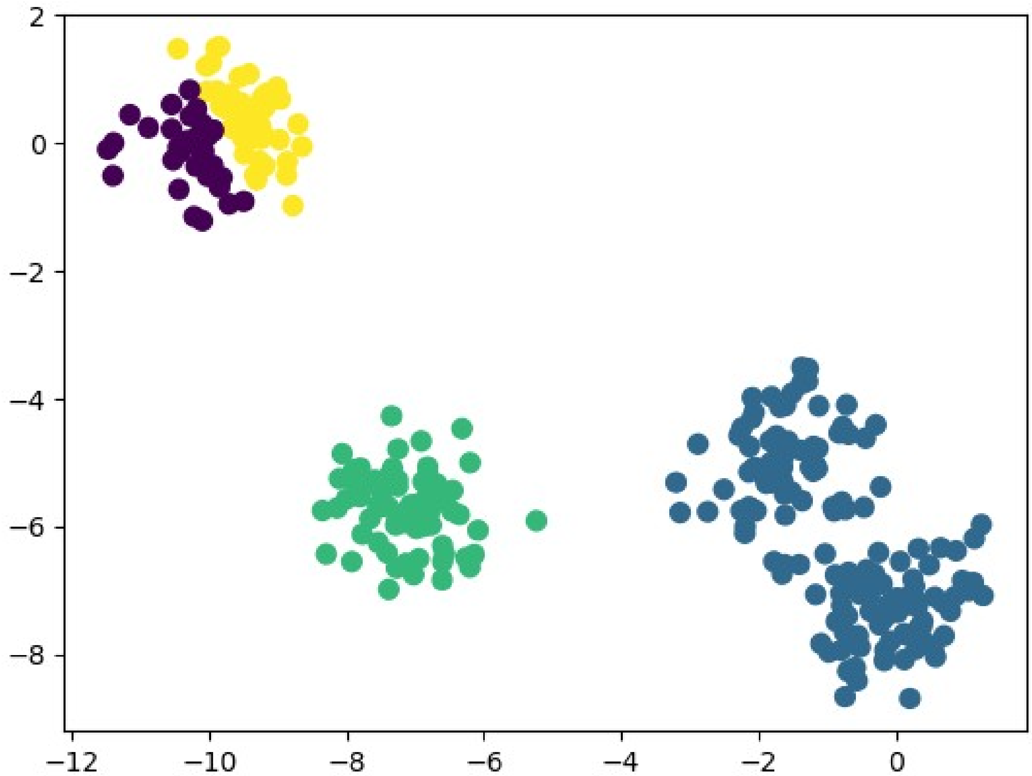
```
from sklearn.metrics import pairwise_distances_argmin

def find_clusters(X, n_clusters, rseed=2):
    # 1. Randomly choose clusters
    rng = np.random.RandomState(rseed)
    i = rng.permutation(X.shape[0])[:n_clusters]
    centers = X[i]
```

```
    while True:
        # 2a. Assign labels based on closest center
        labels = pairwise_distances_argmin(X, centers)

        # 2b. Find new centers from means of points
        new_centers = np.array([X[labels == i].mean(0) for i in
range(n_clusters)])

        # 2c. Check for convergence
        if np.all(centers == new_centers):
            break
        centers = new_centers

    return centers, labels

centers, labels = find_clusters(X, 4)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis');
```



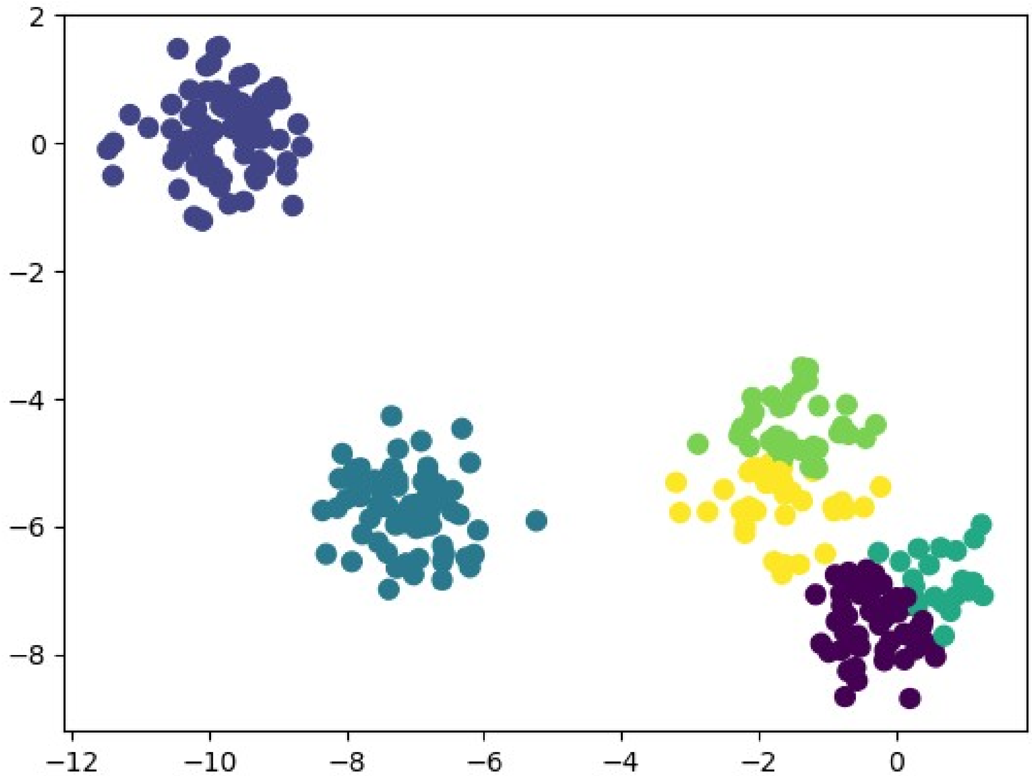
Depending on the seed the best clustering may not be found.

```
centers, labels = find_clusters(X, 4, rseed=0)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis');
```



The amount of clusters must also be properly defined otherwise the results may be incorrect.

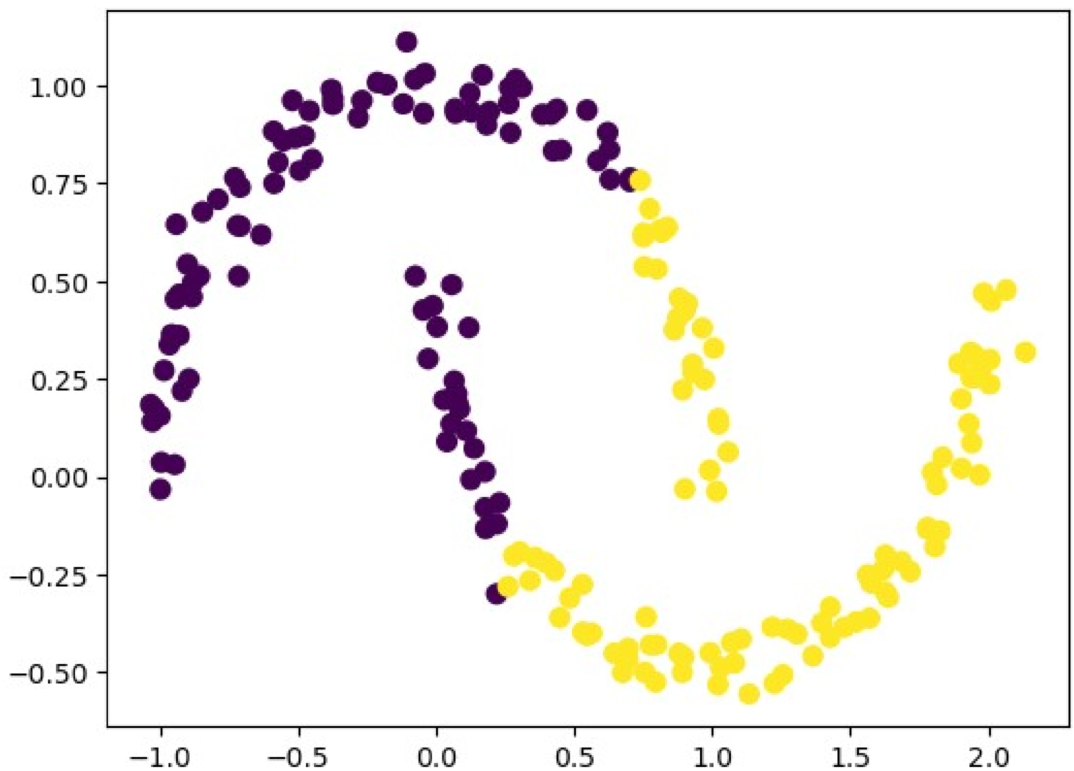
```
labels = KMeans(6, random_state=0).fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels,s=50, cmap='viridis');
```



Non Linear Clustering

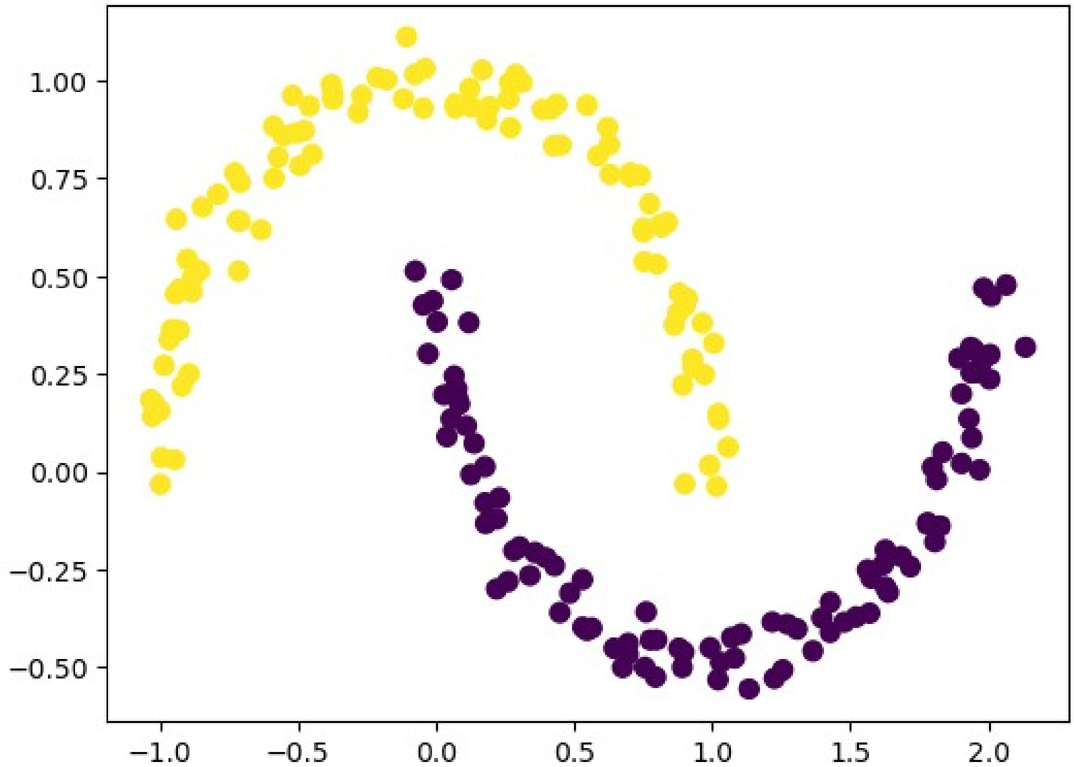
Data may not be nicely clumped together, so different kernel versions of K Means may be used. First generate data.

```
from sklearn.datasets import make_moons
X, y = make_moons(200, noise=.05, random_state=10)
labels = KMeans(2, random_state=0).fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis');
```



Use SpectralClustering instead for better clustering, to find more complicated patterns.

```
from sklearn.cluster import SpectralClustering
model = SpectralClustering(n_clusters=2, affinity='nearest_neighbors',
assign_labels='kmeans')
labels = model.fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis');
```



K Means Classification on Digits

Use K Means to classify different images of numbers without any labels present on the data. First load in the data.

```
from sklearn.datasets import load_digits
digits = load_digits()
print(digits.data.shape)
```

```
(1797, 64)
```

Perform clustering the same way

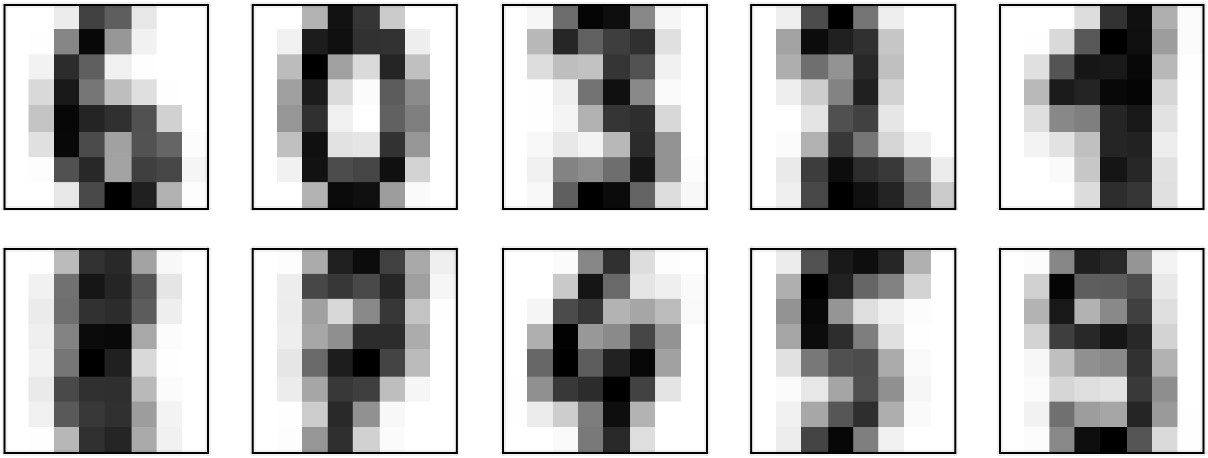
```
kmeans = KMeans(n_clusters=10, random_state=3)
clusters = kmeans.fit_predict(digits.data)
print(kmeans.cluster_centers_.shape)
```

```
(10, 64)
```

Show the centres the model has found.

```
fig, ax = plt.subplots(2, 5, figsize=(8, 3))
centers = kmeans.cluster_centers_.reshape(10, 8, 8)
for axi, center in zip(ax.flat, centers):
```

```python
    axi.set(xticks=[], yticks=[])
    axi.imshow(center, interpolation='nearest', cmap=plt.cm.binary)
```



Match predicted labels with the actual labels.

```python
from scipy.stats import mode

labels = np.zeros_like(clusters)
for i in range(10):
    mask = (clusters == i)
    labels[mask] = mode(digits.target[mask])[0]
```

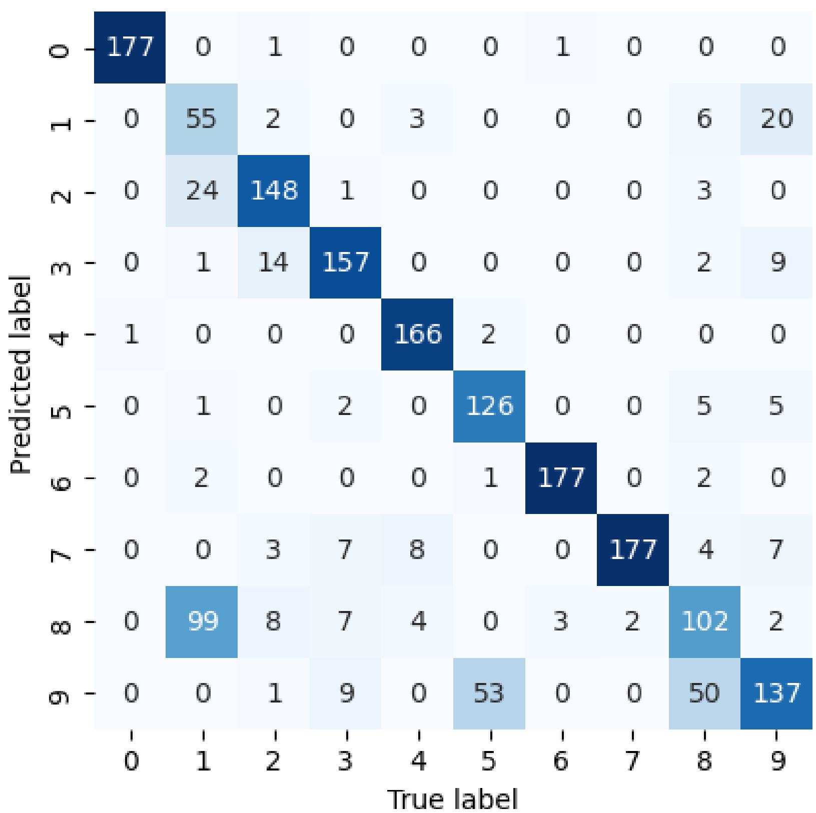
Check metric reports of the model.

```python
from sklearn.metrics import classification_report
print(classification_report(digits.target, labels))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       178
           1       0.64      0.30      0.41       182
           2       0.84      0.84      0.84       177
           3       0.86      0.86      0.86       183
           4       0.98      0.92      0.95       181
           5       0.91      0.69      0.79       182
           6       0.97      0.98      0.98       181
           7       0.86      0.99      0.92       179
           8       0.45      0.59      0.51       174
           9       0.55      0.76      0.64       180

    accuracy                           0.79      1797
   macro avg       0.80      0.79      0.79      1797
weighted avg       0.81      0.79      0.79      1797
```

Next check the confusion matrix of the model.

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns
mat = confusion_matrix(digits.target, labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
cmap='Blues', xticklabels=digits.target_names,
yticklabels=digits.target_names)
plt.xlabel('True label')
plt.ylabel('Predicted label');
```



Colour Compression

Use K Means model for reducing the colour amount of images. First load the data.

```python
# Note: this requires the PIL package to be installed
from sklearn.datasets import load_sample_image
china = load_sample_image("china.jpg")
ax = plt.axes(xticks=[], yticks=[])
ax.imshow(china);
```

Show shape of the data.
Then rescale the data.

```
print(china.shape)
data = china / 255.0  # use 0...1 scale
data = data.reshape(-1, 3)
print(data.shape)
```

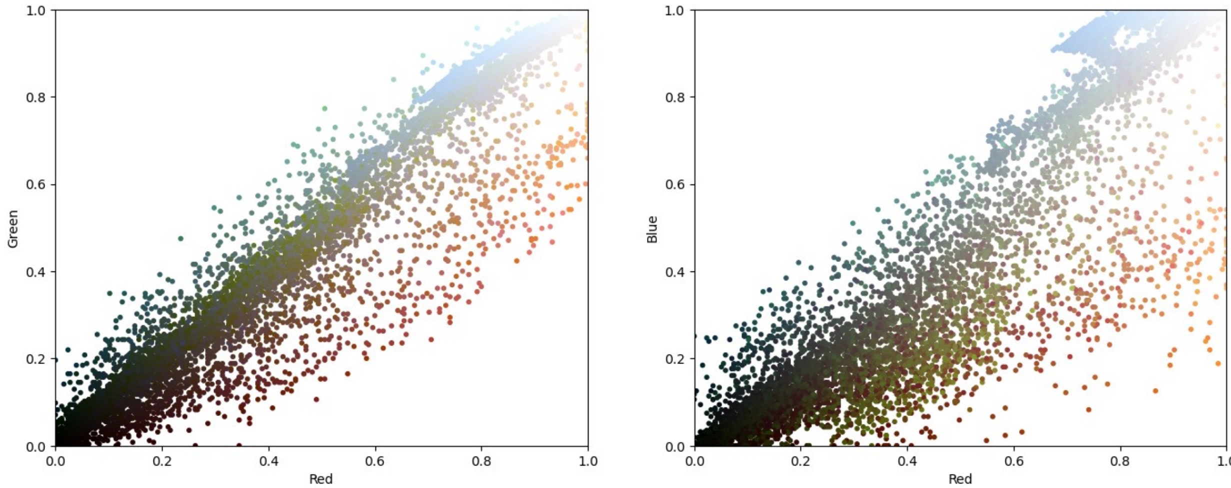
```
(427, 640, 3)
(273280, 3)
```

Visualise the pixels in a graph.

```
def plot_pixels(data, title, colours=None, N=10000):
    if colours is None:
        colours = data

    # choose a random subset
    rng = np.random.default_rng(0)
    i = rng.permutation(data.shape[0])[:N]
    colours = colours[i]
    R, G, B = data[i].T

    fig, ax = plt.subplots(1, 2, figsize=(16, 6))
    ax[0].scatter(R, G, color=colours, marker='.')
    ax[0].set(xlabel='Red', ylabel='Green', xlim=(0, 1), ylim=(0, 1))

    ax[1].scatter(R, B, color=colours, marker='.')
    ax[1].set(xlabel='Red', ylabel='Blue', xlim=(0, 1), ylim=(0, 1))
```

```
    fig.suptitle(title, size=20);
plot_pixels(data, 'Input colour space: 16 million possible colours')
```

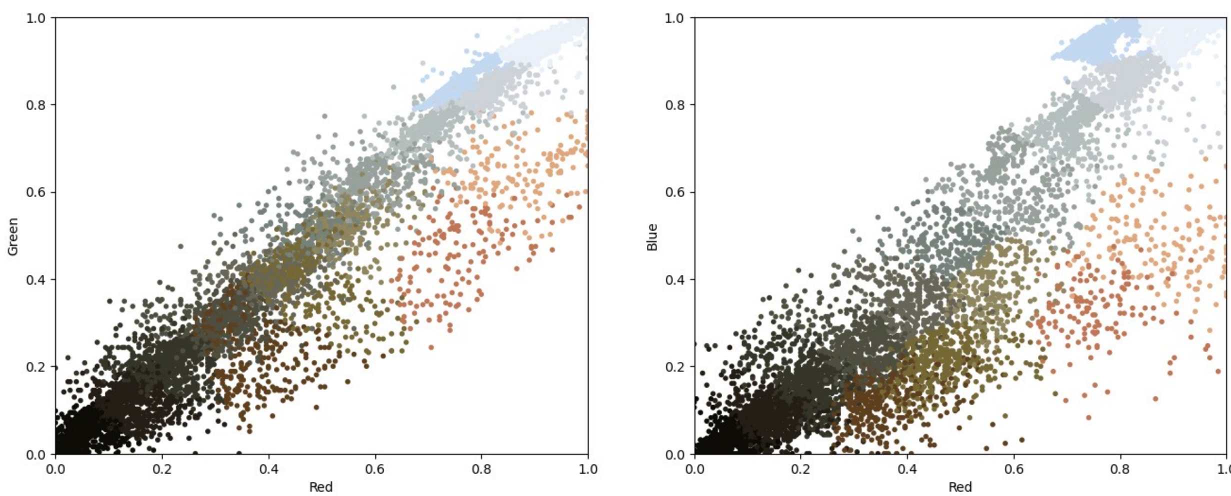
Input colour space: 16 million possible colours



Use MiniBatchKMeans to reduce the 16 million colours into 16 colours.

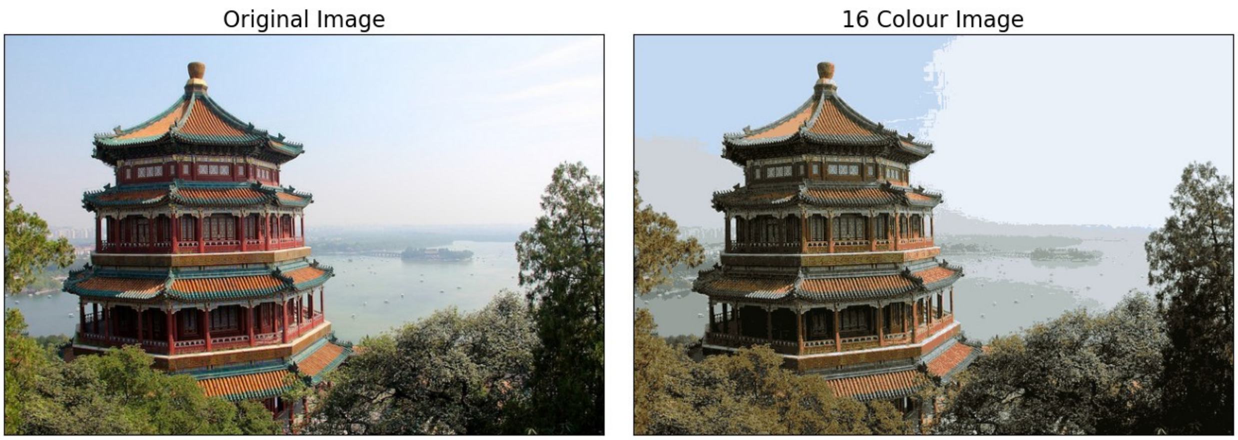
```
from sklearn.cluster import MiniBatchKMeans
kmeans = MiniBatchKMeans(16)
kmeans.fit(data)
new_colours = kmeans.cluster_centers_[kmeans.predict(data)]

plot_pixels(data, "Reduced colour space: 16 colours", new_colours)
```

Reduced colour space: 16 colours



Finally show the old and new images side by side.

```
china_recoloured = new_colours.reshape(china.shape)

fig, ax = plt.subplots(1, 2, figsize=(16, 6),
subplot_kw=dict(xticks=[], yticks=[]))
fig.subplots_adjust(wspace=0.05)
ax[0].imshow(china)
ax[0].set_title('Original Image', size=16)
ax[1].imshow(china_recoloured)
ax[1].set_title('16 Colour Image', size=16);
```



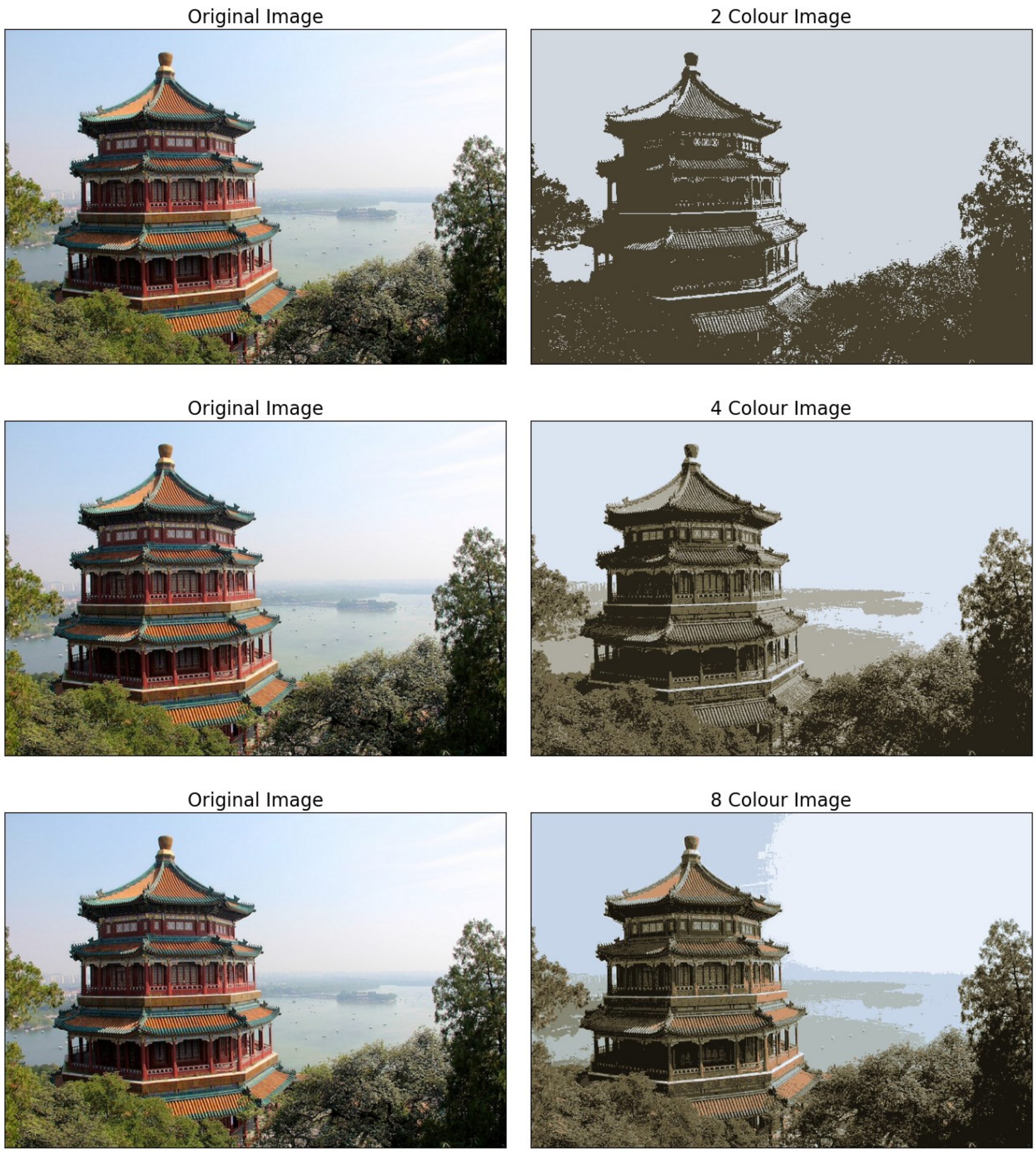
Save the original image.

```
import matplotlib.image
matplotlib.image.imsave('Images/china_original.png', china)
```

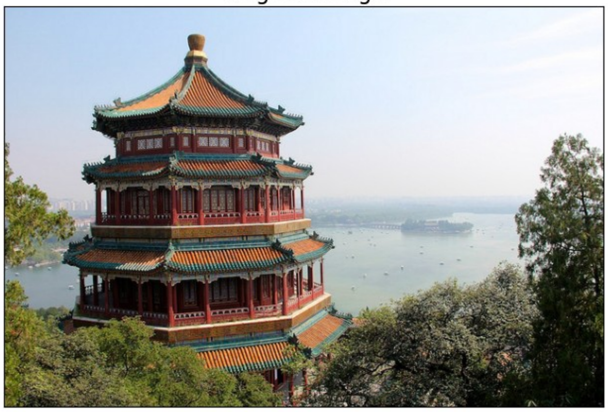
Compute different images from 2 colours all the way to 1024 colours, which ends up looking quite close to the original.
And compare the images to the original one.
The images are also saved in the PNG format so that file sizes can be compared.

```
number_of_colours = 1
for index in range(10):
    number_of_colours *= 2
    kmeans = MiniBatchKMeans(number_of_colours)
    kmeans.fit(data)
    new_colours = kmeans.cluster_centers_[kmeans.predict(data)]

    china_recoloured = new_colours.reshape(china.shape)

matplotlib.image.imsave(f'Images/china_{number_of_colours}_colours.png
', china_recoloured)

    fig, ax = plt.subplots(1, 2, figsize=(16, 6),
subplot_kw=dict(xticks=[], yticks=[]))
    fig.subplots_adjust(wspace=0.05)
```

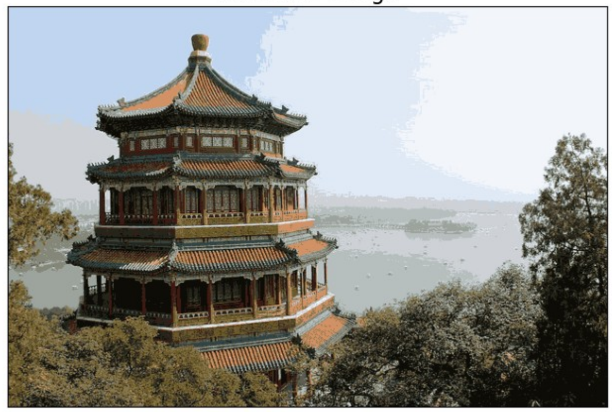
```
    ax[0].imshow(china)
    ax[0].set_title('Original Image', size=16)
    ax[1].imshow(china_recoloured)
    ax[1].set_title(f'{number_of_colours} Colour Image', size=16)
```



Original Image

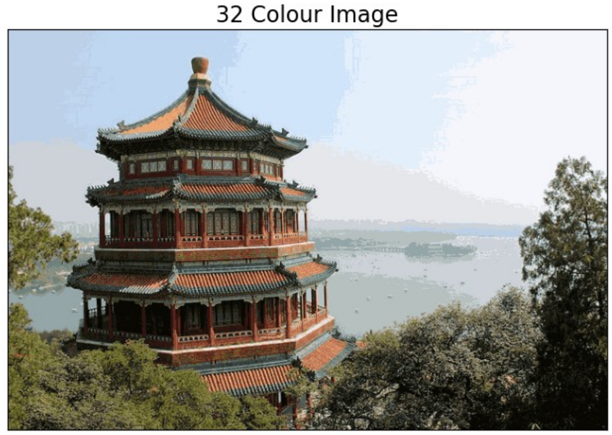


16 Colour Image



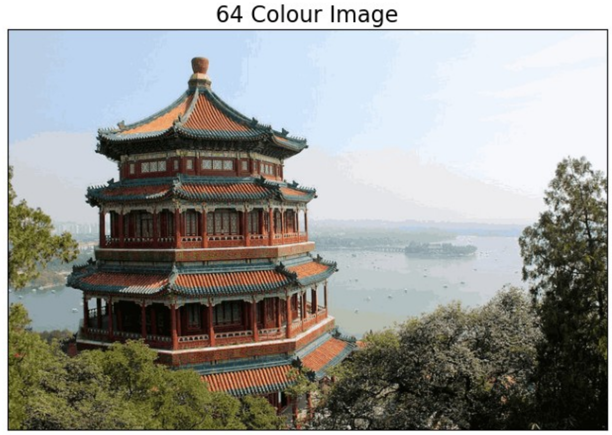
Original Image

32 Colour Image

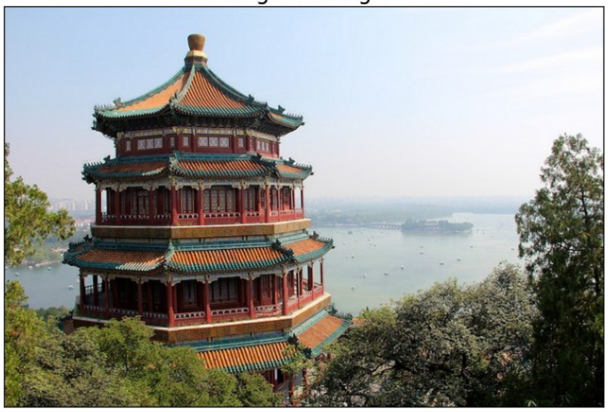


Original Image

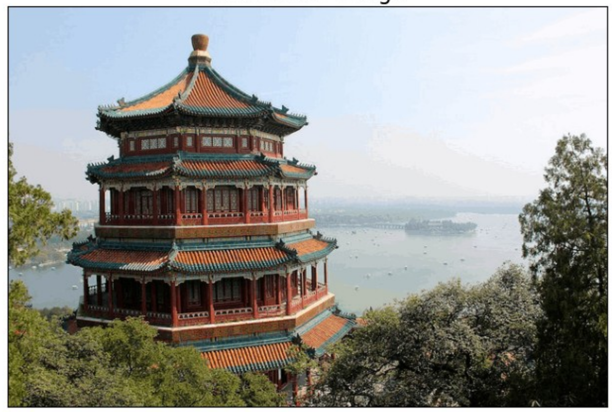
64 Colour Image



Original Image



128 Colour Image



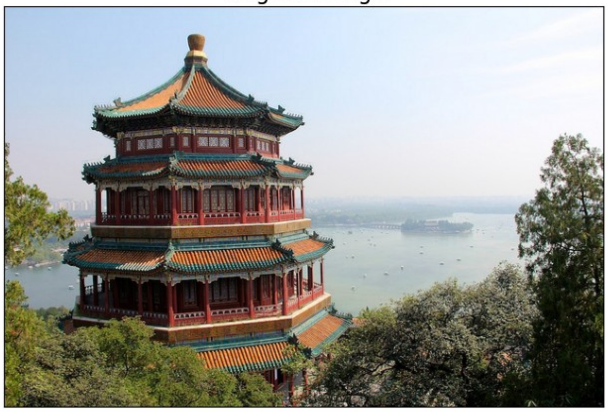
Original Image

256 Colour Image

Original Image

512 Colour Image

Original Image



1024 Colour Image