Implementation of K-Means Clustering Algorithm on a Simple Dataset

1. Importing necessary libraries

First, we need to import the libraries required for our analysis: `matplotlib` for visualising the data, and `sklearn` for performing K-means clustering.

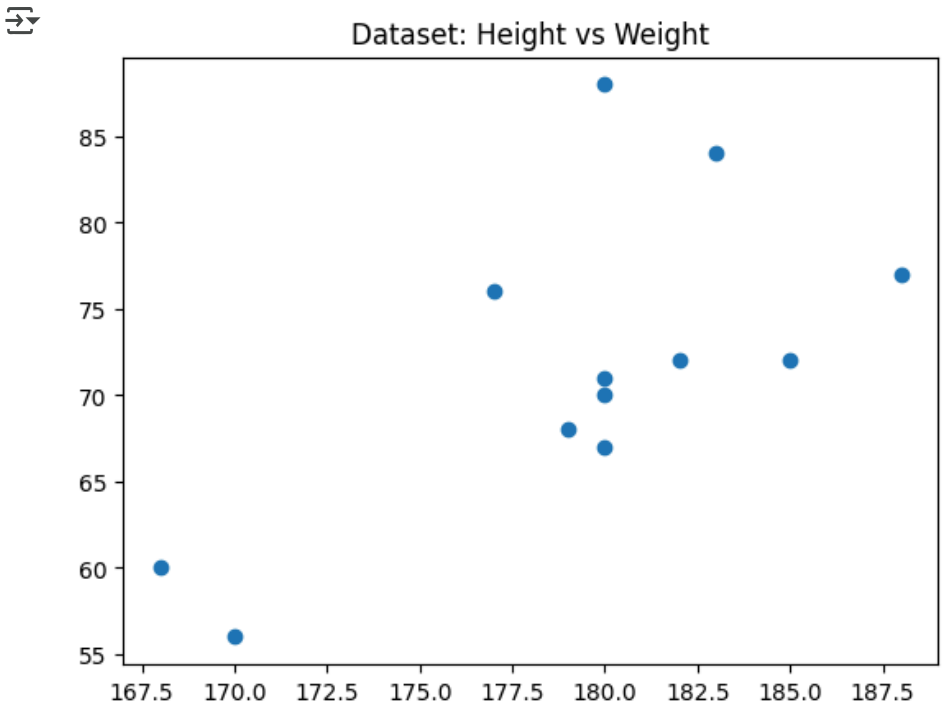
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

2. Defining the dataset

Then, we define two lists, `x` and `y`, to represent a simple dataset of heights and weights. We also create a scatterplot to visualise these points, displaying their initial distribution.

```
x = [185, 170, 168, 179, 182, 188, 180, 180, 183, 180, 180, 177]
y = [72, 56, 60, 68, 72, 77, 71, 70, 84, 88, 67, 76]

plt.scatter(x, y)
plt.title('Dataset: Height vs Weight')
plt.show()
```



3. Calculating and visualising inertia with the 'elbow method'

The **elbow method** is a heuristic used to determine the optimal number of clusters in a dataset. It involves plotting the inertia (the sum of squared distances from each point to its assigned cluster centre) for different values of `K` (i.e., number of clusters). The point where the inertia begins to decrease at a slower rate—forming an elbow—indicates the ideal number of clusters.

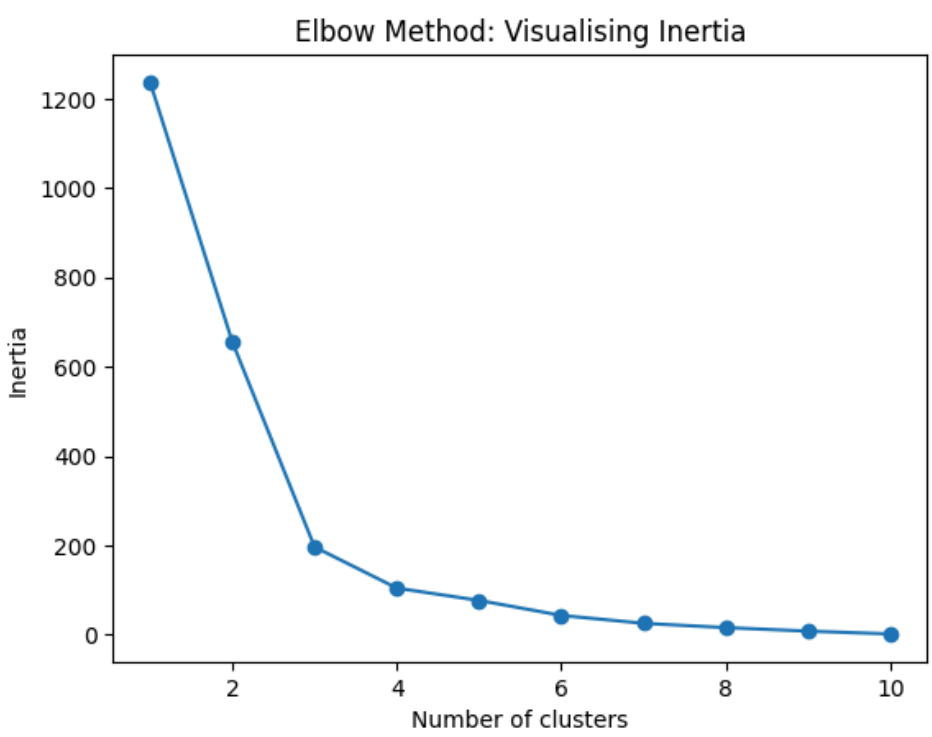
Inertia quantifies how well the data points are clustered. A lower inertia indicates that the points are closer to their cluster centres, suggesting better clustering quality. However, inertia alone does not define the best number of clusters, as it will always decrease with more clusters.

In this section, we first calculate and then visualise the inertia values against the number of clusters using a line plot.

```
data = list(zip(x, y))
inertias = []
```

```
for i in range(1,11):
  k_means = KMeans(n_clusters=i)
  k_means.fit(data)
  inertias.append(k_means.inertia_)

plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow Method: Visualising Inertia')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```



4. Fitting the K-means model with the chosen number of clusters

Finally, we fit the K-means model again using the optimal number of clusters determined from the above graph (in this case, 3). The data points are then plotted, coloured by their assigned cluster labels.

```
k_means = KMeans(n_clusters=3)
k_means.fit(data)

plt.scatter(x, y, c=k_means.labels_)
plt.show()
```