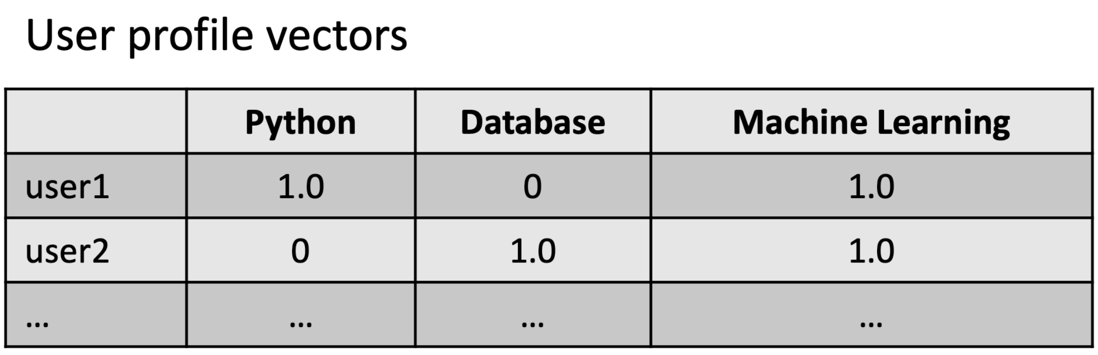
Clustering based Course Recommender System

Estimated time needed: **90** minutes

Previously, we have generated user profile vectors based on course ratings and genres.

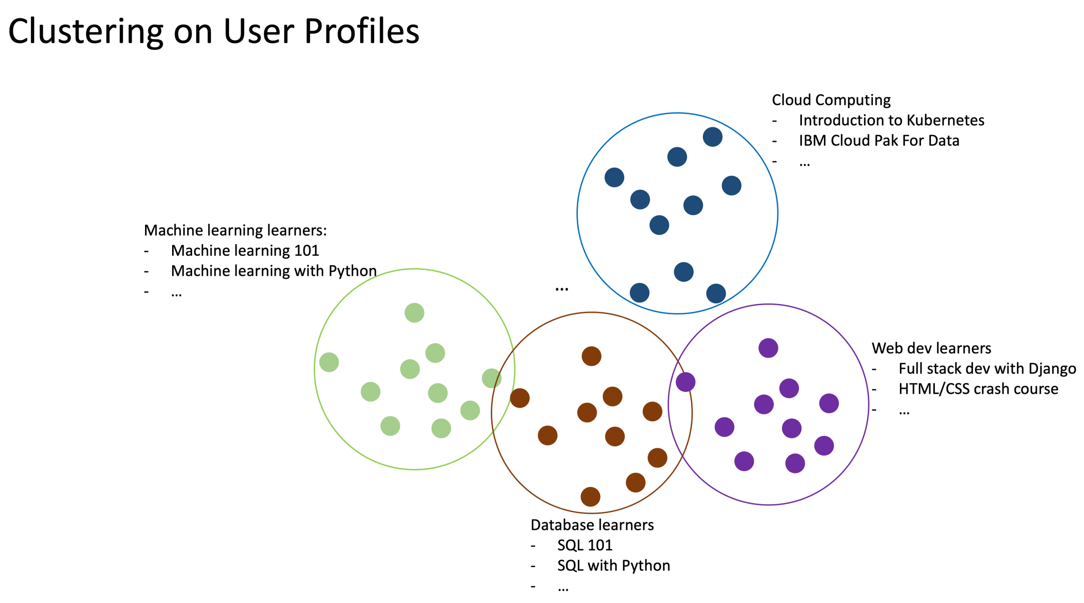
A user profile vector may look like a row vector in the following matrix, for example, we can see the Database column for user2 has a value 1 which means user2 is very interesting in courses related to the databases. With the user profile vectors generated, we can also easily compute the similarity among users based on their shared interests.



User profile vectors

	Python	Database	Machine Learning
user1	1.0	0	1.0
user2	0	1.0	1.0
...	...	...	...

Furthermore, we could perform clustering algorithms such as K-means or DBSCAN to group users with similar learning interests. For example, in the below user clusters, we have user clusters whom have learned courses related to machine learning, cloud computing, databases, and web development, etc.



For each user group, we can come up with a list of popular courses. For example, for the machine learning user cluster/learning group, we can count the most frequently enrolled courses, which are very likely to be the most popular and good machine learning courses because they are enrolled by many users who are interested in machine learning.

If we know a user belongs to the machine learning group, we may recommend the most enrolled courses to them and it is very likely the user will be interested in them.

Next in this lab, you will be implementing some clustering-based recommender system algorithms.

Objectives

After completing this lab you will be able to:

- Perform k-means clustering on the original user profile feature vectors
- Apply PCA (Principle Component Analysis) on user profile feature vectors to reduce dimensions
- Perform k-means clustering on the PCA transformed main components
- Generate course recommendations based on other group members' enrollment history

---

Prepare and setup lab environment

First install and import required libraries:

In [1]:
```
%pip install scikit-learn
%pip install seaborn
%pip install pandas
%pip install matplotlib
```

```
Collecting scikit-learn
  Downloading scikit_learn-1.8.0-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28
_x86_64.whl.metadata (11 kB)
Requirement already satisfied: numpy>=1.24.1 in /opt/conda/lib/python3.12/site-pa
ckages (from scikit-learn) (2.4.1)
Collecting scipy>=1.10.0 (from scikit-learn)
  Downloading scipy-1.17.0-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_6
4.whl.metadata (62 kB)
Collecting joblib>=1.3.0 (from scikit-learn)
  Downloading joblib-1.5.3-py3-none-any.whl.metadata (5.5 kB)
Collecting threadpoolctl>=3.2.0 (from scikit-learn)
  Downloading threadpoolctl-3.6.0-py3-none-any.whl.metadata (13 kB)
Downloading scikit_learn-1.8.0-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x
86_64.whl (8.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.9/8.9 MB 147.3 MB/s eta 0:00:00
Downloading joblib-1.5.3-py3-none-any.whl (309 kB)
Downloading scipy-1.17.0-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.
whl (35.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 35.0/35.0 MB 149.0 MB/s eta 0:00:000
0:01
Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, scipy, joblib, scikit-learn
Successfully installed joblib-1.5.3 scikit-learn-1.8.0 scipy-1.17.0 threadpoolctl
-3.6.0
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: seaborn in /opt/conda/lib/python3.12/site-packages
(0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /opt/conda/lib/python3.12/
site-packages (from seaborn) (2.4.1)
Requirement already satisfied: pandas>=1.2 in /opt/conda/lib/python3.12/site-pack
ages (from seaborn) (3.0.0)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /opt/conda/lib/python3.
12/site-packages (from seaborn) (3.10.8)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.12/site
-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.12/site-pac
kages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.12/sit
e-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.61.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/conda/lib/python3.12/sit
e-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.12/site-
packages (from matplotlib!=3.6.1,>=3.4->seaborn) (24.2)
Requirement already satisfied: pillow>=8 in /opt/conda/lib/python3.12/site-packag
es (from matplotlib!=3.6.1,>=3.4->seaborn) (12.1.0)
Requirement already satisfied: pyparsing>=3 in /opt/conda/lib/python3.12/site-pac
kages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.3.2)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python3.12/
site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-package
s (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.17.0)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: pandas in /opt/conda/lib/python3.12/site-packages
(3.0.0)
Requirement already satisfied: numpy>=1.26.0 in /opt/conda/lib/python3.12/site-pa
ckages (from pandas) (2.4.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/conda/lib/python3.1
2/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-package
s (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

```
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: matplotlib in /opt/conda/lib/python3.12/site-packa
ges (3.10.8)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.12/site
-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.12/site-pac
kages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.12/sit
e-packages (from matplotlib) (4.61.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/conda/lib/python3.12/sit
e-packages (from matplotlib) (1.4.9)
Requirement already satisfied: numpy>=1.23 in /opt/conda/lib/python3.12/site-pack
ages (from matplotlib) (2.4.1)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.12/site-
packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /opt/conda/lib/python3.12/site-packag
es (from matplotlib) (12.1.0)
Requirement already satisfied: pyparsing>=3 in /opt/conda/lib/python3.12/site-pac
kages (from matplotlib) (3.3.2)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python3.12/
site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-package
s (from python-dateutil>=2.7->matplotlib) (1.17.0)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```python
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

%matplotlib inline
```

In [3]:
```python
# also set a random state
rs = 123
```

Load the user profile dataset

Let's first load the original user profile feature vectors:

In [4]:
```python
# Importing the pandas library, which is commonly used for data manipulation and
import pandas as pd

# Defining the URL of the CSV file containing user profiles
user_profile_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain

# Using pandas' read_csv function to read the CSV file from the specified URL in
user_profile_df = pd.read_csv(user_profile_url)

# Displaying the first few rows of the DataFrame to inspect its contents
user_profile_df.head()
```

Out[4]:

	user	Database	Python	CloudComputing	DataAnalysis	Containers	MachineLearnin
0	2	52.0	14.0	6.0	43.0	3.0	33.
1	4	40.0	2.0	4.0	28.0	0.0	14.
2	5	24.0	8.0	18.0	24.0	0.0	30.
3	7	2.0	0.0	0.0	2.0	0.0	0.
4	8	6.0	0.0	0.0	4.0	0.0	0.

In [5]:
```
user_profile_df.shape
```

Out[5]:
```
(33901, 15)
```

we can then list the feature names, they are the user interested topics (course genres):

In [6]:
```
feature_names = list(user_profile_df.columns[1:])
feature_names
```

Out[6]:
```
['Database',
 'Python',
 'CloudComputing',
 'DataAnalysis',
 'Containers',
 'MachineLearning',
 'ComputerVision',
 'DataScience',
 'BigData',
 'Chatbot',
 'R',
 'BackendDev',
 'FrontendDev',
 'Blockchain']
```

As we can see from the user profile dataset, we have about 33K unique users with interests in areas like `Database` , `Python` , `CloudComputing` , etc. Then, let's check the summary statistics for each feature.

In [7]:
```
user_profile_df.describe()
```

Out[7]:

	user	Database	Python	CloudComputing	DataAnalysis	Co
count	3.390100e+04	33901.000000	33901.000000	33901.000000	33901.000000	3390
mean	1.064064e+06	5.518569	3.493791	2.307100	3.624701	(
std	4.972578e+05	7.611941	4.227254	3.841858	4.760135	(
min	2.000000e+00	0.000000	0.000000	0.000000	0.000000	(
25%	6.813480e+05	0.000000	0.000000	0.000000	0.000000	(
50%	1.043907e+06	3.000000	3.000000	0.000000	3.000000	(
75%	1.451159e+06	9.000000	6.000000	3.000000	6.000000	(
max	2.103039e+06	63.000000	18.000000	38.000000	48.000000	15

The original user profile feature vector is not normalized, which may cause issues when we perform clustering and Principal component analysis (PCA), therefor we standardize the data.

In [8]:
```
# Use StandardScaler to make each feature with mean 0, standard deviation 1
# Instantiating a StandardScaler object
scaler = StandardScaler()

# Standardizing the selected features (feature_names) in the user_profile_df Dat
user_profile_df[feature_names] = scaler.fit_transform(user_profile_df[feature_na

# Printing the mean and standard deviation of the standardized features
print("mean {} and standard deviation{} ".format(user_profile_df[feature_names].
mean Database          -2.682796e-17
Python            -5.365592e-17
CloudComputing    -1.341398e-17
DataAnalysis      -7.545364e-17
Containers         1.341398e-17
MachineLearning    3.353495e-17
ComputerVision    -7.545364e-18
DataScience       -2.012097e-17
BigData            6.706990e-17
Chatbot           -6.036291e-17
R                  5.700942e-17
BackendDev         2.012097e-17
FrontendDev        2.012097e-17
Blockchain        -6.706990e-17
dtype: float64 and standard deviationDatabase           1.000015
Python             1.000015
CloudComputing     1.000015
DataAnalysis       1.000015
Containers         1.000015
MachineLearning    1.000015
ComputerVision     1.000015
DataScience        1.000015
BigData            1.000015
Chatbot            1.000015
R                  1.000015
BackendDev         1.000015
FrontendDev        1.000015
Blockchain         1.000015
dtype: float64
```

In [9]:
```
user_profile_df.describe()
```

Out[9]:

	user	Database	Python	CloudComputing	DataAnalysis	C
count	3.390100e+04	3.390100e+04	3.390100e+04	3.390100e+04	3.390100e+04	3.39
mean	1.064064e+06	-2.682796e-17	-5.365592e-17	-1.341398e-17	-7.545364e-17	1.34
std	4.972578e+05	1.000015e+00	1.000015e+00	1.000015e+00	1.000015e+00	1.00
min	2.000000e+00	-7.249991e-01	-8.265040e-01	-6.005256e-01	-7.614816e-01	-4.
25%	6.813480e+05	-7.249991e-01	-8.265040e-01	-6.005256e-01	-7.614816e-01	-4.
50%	1.043907e+06	-3.308757e-01	-1.168130e-01	-6.005256e-01	-1.312380e-01	-4.
75%	1.451159e+06	4.573712e-01	5.928781e-01	1.803581e-01	4.990056e-01	-4.
max	2.103039e+06	7.551593e+00	3.431642e+00	9.290667e+00	9.322416e+00	5.95

The normalized user profile features are:

In [10]:
```
features = user_profile_df.loc[:, user_profile_df.columns != 'user']
features
```

Out[10]:

	Database	Python	CloudComputing	DataAnalysis	Containers	MachineLearni
0	6.106474	2.485388	0.961242	8.272010	0.850889	6.4775
1	4.529980	-0.353377	0.440653	5.120792	-0.424767	2.3685
2	2.427988	1.066006	4.084776	4.280467	-0.424767	5.8287
3	-0.462250	-0.826504	-0.600526	-0.341319	-0.424767	-0.6591
4	0.063248	-0.826504	-0.600526	0.078843	-0.424767	-0.6591
...	...	...	...	...	...	
33896	-0.330876	-0.116813	0.180358	0.499006	-0.424767	-0.6591
33897	-0.724999	-0.826504	-0.079936	-0.761482	-0.424767	-0.6591
33898	-0.330876	0.592878	0.961242	-0.761482	-0.424767	2.3685
33899	-0.724999	-0.826504	-0.600526	-0.761482	-0.424767	-0.2266
33900	-0.724999	-0.826504	-0.600526	-0.761482	-0.424767	-0.2266

33901 rows × 14 columns

we can also save the user ids for later recommendation tasks:

In [11]:
```
user_ids = user_profile_df.loc[:, user_profile_df.columns == 'user']
user_ids
```

Out[11]:

	user
0	2
1	4
2	5
3	7
4	8
...	...
33896	2102054
33897	2102356
33898	2102680
33899	2102983
33900	2103039

33901 rows × 1 columns

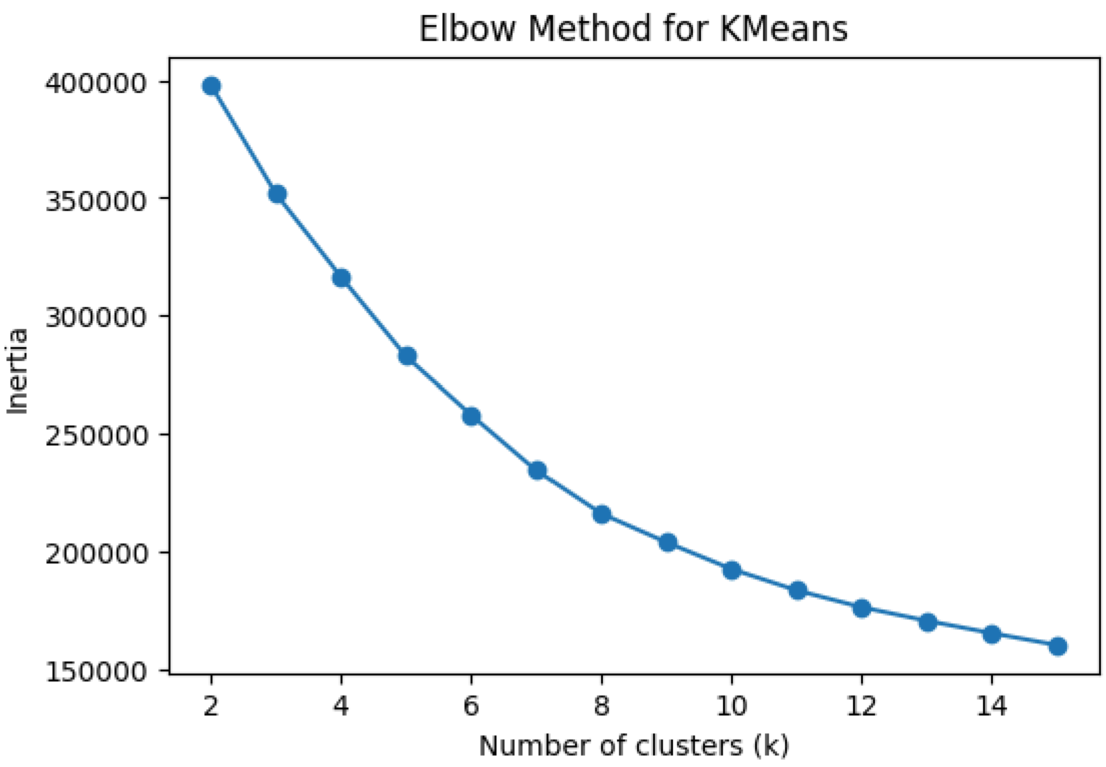
TASK: Perform K-means clustering algorithm on the user profile feature vectors

With the user profile dataset ready, you need to use the `KMeans` class provided by scikit-learn library to perform clustering on the user profile feature vectors.

For `KMeans` algorithm, one important hyperparameter is the number of clusters `n_cluster`, and a good way to find the optimized `n_cluster` is using to grid search a list of candidates and find the one with the best or optimized clustering evaluation metrics such as minimal `sum of squared distance`:

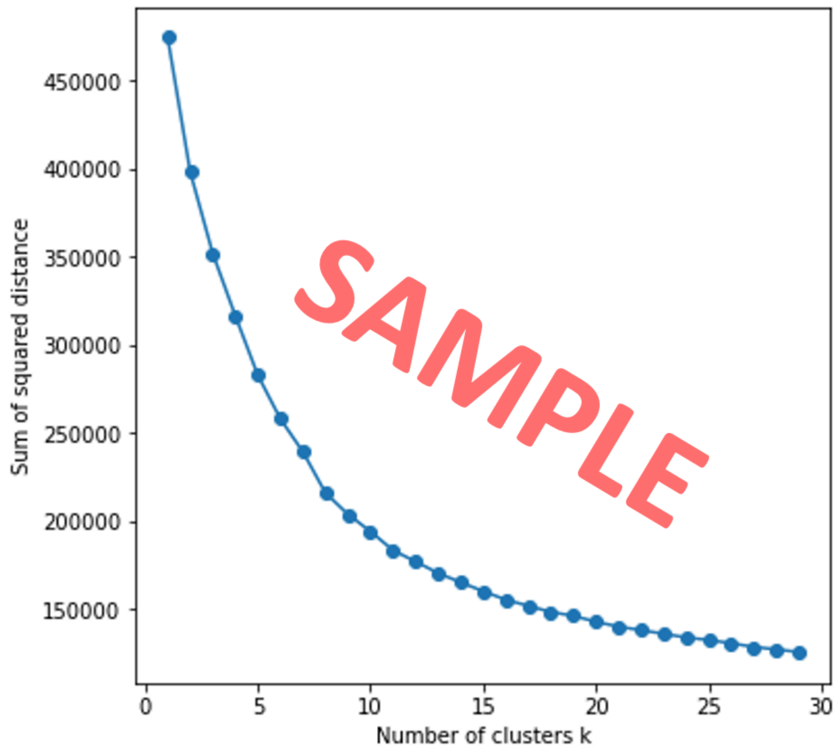
TODO: grid search the optimized n_cluster for KMeans() model

In [12]:
```python
# WRITE YOUR CODE HERE
from sklearn.cluster import KMeans

# Try different values of k
list_k = range(2, 16)
inertias = []

for k in list_k:
    model = KMeans(n_clusters=k, random_state=rs, n_init=10)
    model.fit(features)
    inertias.append(model.inertia_)

# Plot elbow curve
plt.figure(figsize=(6,4))
plt.plot(list_k, inertias, marker='o')
plt.xlabel("Number of clusters (k)")
plt.ylabel("Inertia")
plt.title("Elbow Method for KMeans")
plt.show()


# Find an optimized number of neighors k from a candidate list such as list_k =
```



▶ Click here for Hints

If you plot the grid search process, you may get a elbow plot like the following:



From the elbow plot, you should visualy identify the point where the metric starting to be flatten, which indicates the optimized number of clusters.

Once you have identified the best number of clusters, you can apply `KMeans()` again to generate cluster label for all users.

In [13]:
```
cluster_labels = [None] * len(user_ids)
```

_TODO: Apply KMeans() on the features with optimized n_cluster parameter after model fitting, you can find output cluster labels in `model.labels_` attribute_

In [17]:
```
## WRITE YOUR CODE HERE

optimal_k = 7

model = KMeans(n_clusters=optimal_k, random_state=rs, n_init=10)
model.fit(features)

cluster_labels = model.labels_
```

▶ Click here for Hints

The cluster labels you generated is a list of integers indicating cluster indices. You may use the following utility method to combine the cluster labels and user ids to a dataframe, so that you know which cluster a user belongs:

In [18]:
```python
def combine_cluster_labels(user_ids, labels):
    # Convert labels to a DataFrame
    labels_df = pd.DataFrame(labels)
    # Merge user_ids DataFrame with labels DataFrame based on index
    cluster_df = pd.merge(user_ids, labels_df, left_index=True, right_index=True
    # Rename columns to 'user' and 'cluster'
    cluster_df.columns = ['user', 'cluster']
    return cluster_df
```

Your clustering results may look like the following screenshot:

Now, each user finds its own cluster or we can say we have created many clusters of learning communities. Learners within each community share very similar learning interests.

TASK: Apply PCA on user profile feature vectors to reduce dimensions

In the previous step, we applied `KMeans` on the original user profile feature vectors which have 14 original features (the course genres).

In [19]:
```python
# Extracting features from the user_profile_df DataFrame, excluding the 'user' c
features = user_profile_df.loc[:, user_profile_df.columns != 'user']

# Extracting user IDs from the user_profile_df DataFrame
user_ids = user_profile_df.loc[:, user_profile_df.columns == 'user']

# Creating a list of feature names by excluding the 'user' column name
feature_names = list(user_profile_df.columns[1:])
```

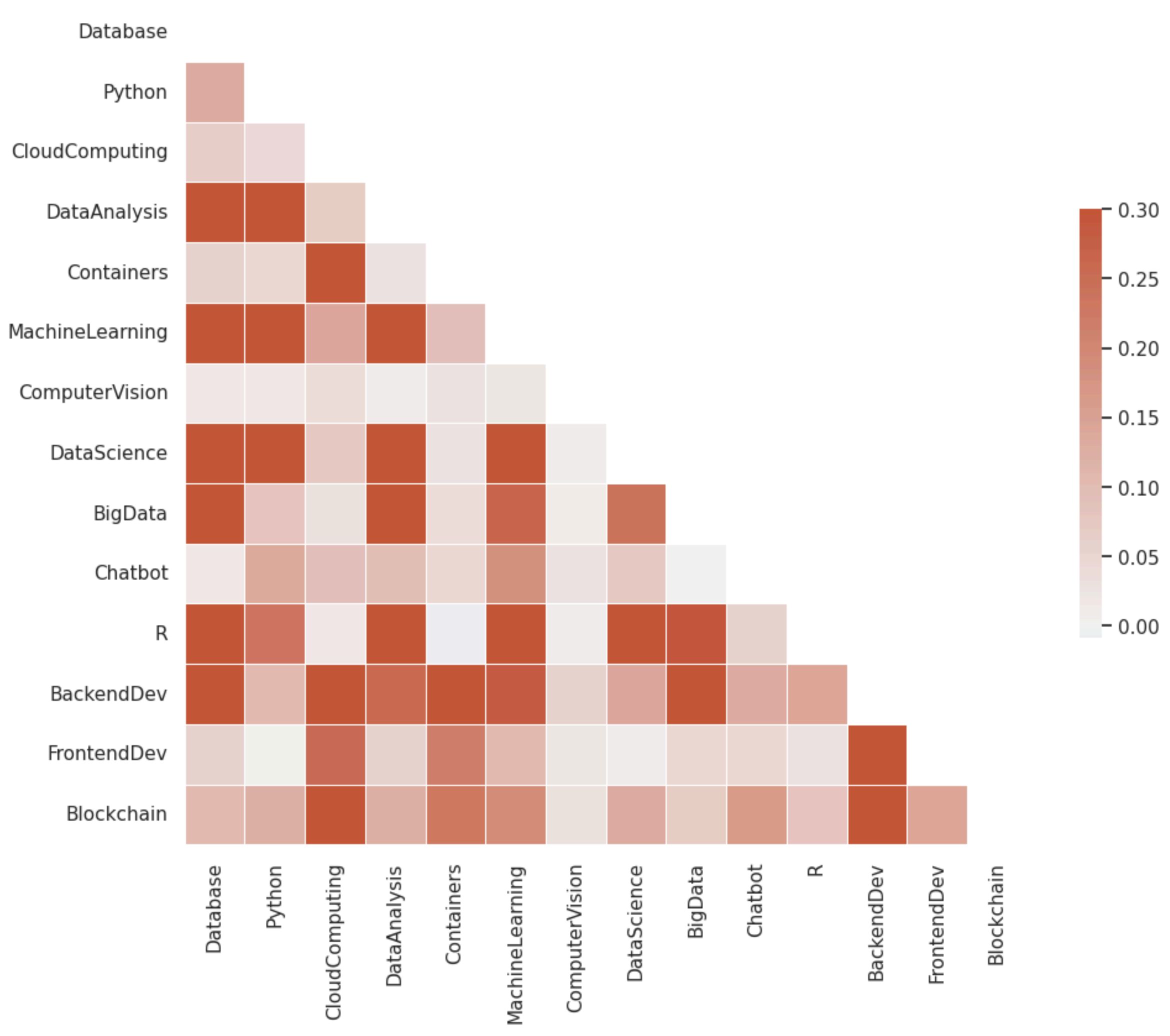
In [20]:
```python
print(f"There are {len(feature_names)} features for each user profile.")
```

```
There are 14 features for each user profile.
```

If we plot a covariance matrix of the user profile feature vectors with 14 features, we can observe that some features are actually correlated:

In [21]:
```python
sns.set_theme(style="white")

# Compute the correlation matrix
corr = features.cov()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

```
plt.show()
```



For example, the feature `MachineLearning` and the feature `DataScience` are correlated. Such covariances among features may indicate that we can apply PCA to find its main components (eigenvectors with max eigenvalues on the covariance matrix).

If we only keep the independent main components, then we can reduce the dimensions of our user profile feature vectors.

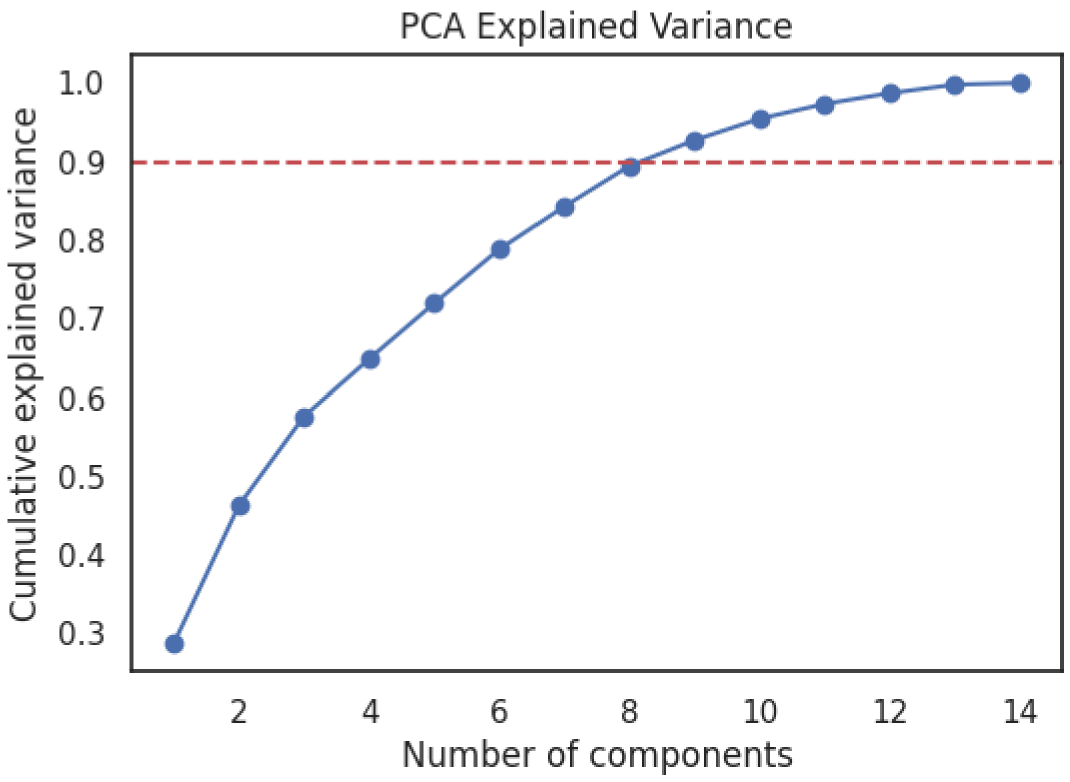
Now let's apply the `PCA()` provided by `scikit-learn` to find the main components in user profile feature vectors and see if we can reduce its dimensions by only keeping the main components.

Note that when calling the `PCA()` class, there is also an import argument called `n_components` which indicates how many components you want to keep in the PCA result. One way to find an optimized `n_components` is to do a grid search on a list of argument candidates (such as `range(1, 15)`) and calculate the ratio of the accumulated variance for each candidate.

If the accumulated variances ratio of a candidate `n_components` is larger than a threshold, e.g., 90%, then we can say the transformed `n_components` could explain about 90% of variances of the original data variance and can be considered as an optimized components size.

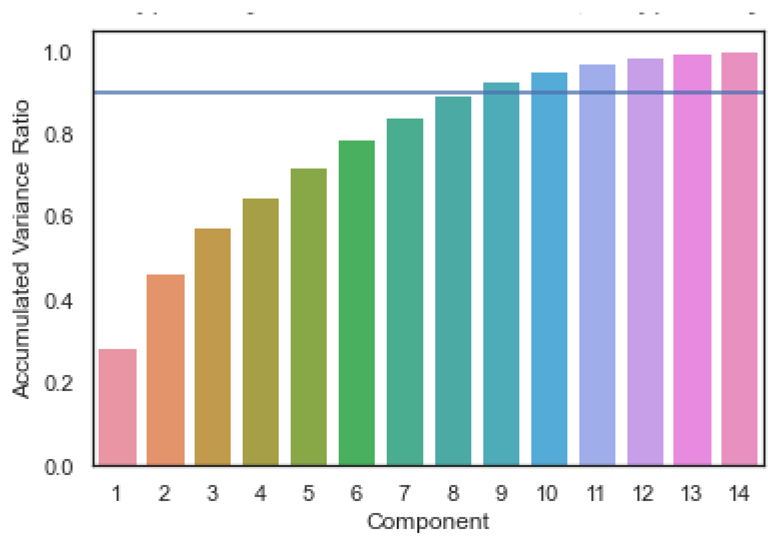
TODO: Find the optimized `n_components` for PCA

In [23]:
```
# WRITE YOUR CODE HERE

# - For a list of candidate `n_components` arguments such as 1 to 14, find out t
# - In the fitted PCA() model, you can find explained_variance_ratio_ and use th
from sklearn.decomposition import PCA

explained_variance = []

for n in range(1, 15):
    pca = PCA(n_components=n, random_state=rs)
    pca.fit(features)
    explained_variance.append(pca.explained_variance_ratio_.sum())

# Plot cumulative variance
plt.figure(figsize=(6,4))
plt.plot(range(1,15), explained_variance, marker='o')
plt.axhline(y=0.9, color='r', linestyle='--')
plt.xlabel("Number of components")
plt.ylabel("Cumulative explained variance")
plt.title("PCA Explained Variance")
plt.show()
```



▶ Click here for Hints

If you visualize your hyperparameter searching process, you may get a trend line like the following:



Once you found the optimized `n_component` argument value, you can apply PCA on the user profile feature vectors and reduce the 14 features into `n_component` features.

TODO: Perform PCA to transform original user profile features

In [24]:
```
# WRITE YOUR CODE HERE

# - For a list of candidate `n_components` arguments such as 1 to 14, find out t
# - In the fitted PCA() model, you can find explained_variance_ratio_ and use th
# - Merge the user ids and transformed features into a new dataframe
optimal_components = 6

pca = PCA(n_components=optimal_components, random_state=rs)
pca_features = pca.fit_transform(features)

# Create PCA dataframe
pca_df = pd.DataFrame(
    pca_features,
    columns=[f'PC{i+1}' for i in range(optimal_components)]
)

pca_df = pd.concat([user_ids.reset_index(drop=True), pca_df], axis=1)
pca_df.head()
```

Out[24]:

	user	PC1	PC2	PC3	PC4	PC5	PC6
0	2	17.772494	0.200681	1.730609	2.567359	-3.825814	-2.707154
1	4	7.145199	-2.847481	2.358636	-0.576654	0.398803	0.134533
2	5	11.363270	1.873619	-1.522077	1.076144	-1.711688	-0.883212
3	7	-1.834033	-0.277462	0.564905	0.053470	-0.064440	-0.165757
4	8	-1.049125	-0.684767	1.072765	0.006371	-0.005695	-0.118686

▶ Click here for Hints

Your PCA transformed dataframe may look like the following:

TASK: Perform k-means clustering on the PCA transformed feature vectors

Now, you have the PCA components of the original profile vectors. You can perform k-means on them again:

TODO: Perform K-means on the PCA transformed features

In [25]:
```
## WRITE YOUR CODE HERE

## - Apply KMeans() on the PCA features
## - Obtain the cluster label lists from model.labels_ attribute
## - Assign each user a cluster label by combining user ids and cluster labels
pca_features_only = pca_df.drop(columns=['user'])

model_pca = KMeans(n_clusters=optimal_k, random_state=rs, n_init=10)
model_pca.fit(pca_features_only)

pca_cluster_labels = model_pca.labels_

cluster_pca_df = combine_cluster_labels(user_ids, pca_cluster_labels)
cluster_pca_df.head()
```

Out[25]:

	user	cluster
0	2	4
1	4	4
2	5	4
3	7	0
4	8	0

Your clustering results should have the same format as the k-means on the original dataset:

Great, now all users find their learning interest groups, either based on their original or the PCA transformed user profile features.

When a user is in a group or a community, it is very likely that the user will be interested in the courses enrolled by other members within the same group.

TASK: Generate course recommendations based on the popular courses in the same cluster

The Intuition of clustering-based course recommendation is very simple and can be illustrated via the following example:

Suppose a user has joined a machine learning group (via clustering algorithm). In the group, he/she finds that the top-3 courses enrolled by all other group members are `Machine Learning for Everyone`, `Machine Learning with Python`, `Machine`

`Learning with Scikit-learn`. Since the user has already completed the `Machine Learning for Everyone` earlier, he/she decides to trust the group members' choices and enroll in other two unselected courses `Machine Learning with Python` and `Machine Learning with Scikit-learn`.

In summary, the clustering-based recommender system first groups all users based on their profiles, and maintains a popular courses list for each group.

For any group member who needs course recommendations, the algorithm recommends the unselected courses from the popular course lists.

Next, suppose we have a set of test users, and we want to recommend new courses to them using a clustering-based recommender system:

In [29]:
```
test_user_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cl

# Reading the test user data CSV file into a pandas DataFrame
# Selecting only the 'user' and 'item' columns using indexing
test_users_df = pd.read_csv(test_user_url)[['user', 'item']]

# Displaying the first few rows of the DataFrame to inspect its contents
test_users_df.head()
cluster_df = combine_cluster_labels(user_ids, cluster_labels)
```

The test users dataset has only two columns, the user id and the enrolled course id.

For each user, let's find its cluster label using the k-means results you have performed in previous steps, assuming it is named `cluster_df`.

You can assign the cluster label to all test users via merging the clustering labels (`cluster_df`):

In [30]:
```
test_users_labelled = pd.merge(test_users_df, cluster_df, left_on='user', right_
```

The merged the test dataset may look like the following:

	user	item	cluster
0	1502801	RP0105EN	9
1	1502801	BD0131EN	9
2	1502801	BD0212EN	9
3	1502801	BD0115EN	9
4	1502801	BD0211EN	9
...	...	...	...
9397	630511	BD0121EN	0
9398	630511	SC0101EN	0
9399	630511	BD0111EN	0
9400	630511	BD0115EN	0
9401	630511	PY0101EN	0

9402 rows × 3 columns

From the above dataframe, we know each user's enrolled courses and its cluster index.

If we use a `groupby` and `sum` aggregation, we can get the enrollments count for each course in each group, like the following code snippet:

In [31]:
```
# Extracting the 'item' and 'cluster' columns from the test_users_labelled DataF
courses_cluster = test_users_labelled[['item', 'cluster']]

# Adding a new column 'count' with a value of 1 for each row in the courses_clus
courses_cluster['count'] = [1] * len(courses_cluster)

# Grouping the DataFrame by 'cluster' and 'item', aggregating the 'count' column
# and resetting the index to make the result more readable
courses_cluster_grouped = courses_cluster.groupby(['cluster','item']).agg(enroll
```

TODO: For each test user, try to recommend any unseen courses based on the popular courses in his/her cluster. You may use an enrollment count threshold (such as larger than 10) to determine if it is a popular course in the cluster

In [38]:
```
courses_cluster_grouped.columns
```

Out[38]:
```
Index(['cluster', 'item', 'enrollments'], dtype='str')
```

In [39]:
```
POPULAR_THRESHOLD = 10

user_recommendations = {}

for user in test_users_labelled['user'].unique():

    user_data = test_users_labelled[test_users_labelled['user'] == user]

    # 1. Get user's cluster
    user_cluster = user_data['cluster'].iloc[0]
    # 2. Get popular courses in the same cluster
    popular_courses = courses_cluster_grouped[
        (courses_cluster_grouped['cluster'] == user_cluster) &
        (courses_cluster_grouped['enrollments'] > POPULAR_THRESHOLD)
    ]['item'].tolist()

    # 3. Get user's already enrolled courses
    enrolled_courses = set(user_data['item'].tolist())

    # 4. Recommend unseen popular courses
    recommended_courses = [
        course for course in popular_courses
        if course not in enrolled_courses
    ]

    # Limit recommendations (good practice)
    user_recommendations[user] = recommended_courses[:20]
```

▶ Click here for Hints

In [41]:
```
avg_recommendations = np.mean([len(v) for v in user_recommendations.values()])
avg_recommendations
```

Out[41]:
```
np.float64(19.989115365328455)
```

In [42]:
```
from collections import Counter

# Collect all recommended courses
all_recommended_courses = []

for courses in user_recommendations.values():
    all_recommended_courses.extend(courses)

# Count frequency of each course
course_frequency = Counter(all_recommended_courses)

# Get top-10 most frequently recommended courses
top_10_courses = course_frequency.most_common(10)

top_10_courses
```

Out[42]:
```
[('BD0137EN', 33324),
 ('BD0135EN', 33241),
 ('BC0202EN', 33111),
 ('BD0145EN', 32866),
 ('BD0153EN', 32808),
 ('BD0143EN', 32738),
 ('BD0123EN', 32539),
 ('BD0121EN', 32274),
 ('BD0212EN', 31892),
 ('BC0201EN', 31023)]
```

With the recommendation results, you also need to write some analytic code to answer the following two questions:

- On average, how many new/unseen courses have been recommended to each user?

- What are the most frequently recommended courses? Return the top-10 commonly recommended courses across all users.

For example, suppose we have only 3 test users, each user receives the following recommendations:

- User1: ['course1', 'course2']
- User2: ['course3', 'course4']
- User3: ['course3', 'course4', 'course5']

Then, the average recommended courses per user is $(2 + 2 + 3) / 3 = 2.33$. The top-2 recommended courses are: `course3`: 2 times, and `course4`: 2 times.

Note that the answers will depend on how you compute the popular courses for each cluster. A lower threshold yields more recommended courses but with smaller confidence so that some test users may receive very long course recommendation lists and feel overwhelmed.

Ideally, we should limit the maximum course recommendations for each user to be less than 20 courses per user.

Explore other clustering algorithms

As you have learned in previous unsupervised learning course, there are many other clustering algorithms such as `DBSCAN` and `Hierarchical Clustering`. You are encouraged to try them on the user profile feature vectors and compare the results with K-means.

Summary

Congratulations! In this lab, you have applied clustering algorithms to group users with similar interests and also tried PCA to reduce the dimensions of user feature vectors.

Furthermore, with each user finding its learning interest group, you have also implemented clustering-based course recommender system to make recommendations based on his/her group members' popular courses choices.

Authors

Yan Luo

Other Contributors

```
toggle##
```

```
toggle|Date

toggle|-|-|-|-|

toggle|2021-10-25|1.0|Yan|Created
```