# Course Recommendation System

By Aniruddh Kulkarni and Keval Shah

# Introduction

A course recommendation system is a software system that provides specific suggestions of courses to users according to their preferences, interests, and goals. A course recommendation system can help students and learners to choose the most suitable courses for their educational or professional development, as well as assist instructors and advisors to design and recommend effective curricula

# Methodology

1. Dataset
2. Methodology
3. Cosine Similarity and Collaborative filtering
4. Evaluation

The Coursera 2021 dataset is a collection of data scraped from the Coursera website that contains information about various courses offered by the online learning platform. The dataset consists of four columns: ‘Course Name’, ‘Difficulty Level’, ‘Course Description’, and ‘Skills’. The ‘Course Name’ column contains the title of the course, such as ‘Machine Learning’, ‘Python for Everybody’, or ‘Introduction to Psychology’. The ‘Difficulty Level’ column indicates the level of difficulty of the course, such as ‘Beginner’, ‘Intermediate’, or ‘Advanced’. The ‘Course Description’ column provides a brief summary of what the course is about, such as the topics covered, the learning objectives, and the prerequisites. The ‘Skills’ column lists the skills that the course aims to teach or enhance, such as ‘Data Science’, ‘Web Development’, or ‘Critical Thinking’. The dataset can be used for various purposes, such as analyzing the trends and patterns of online education, exploring the diversity and quality of courses, or building a course recommendation system. **The dataset can be found on Kaggle**.

### 0.0.1 Basic Data Analysis

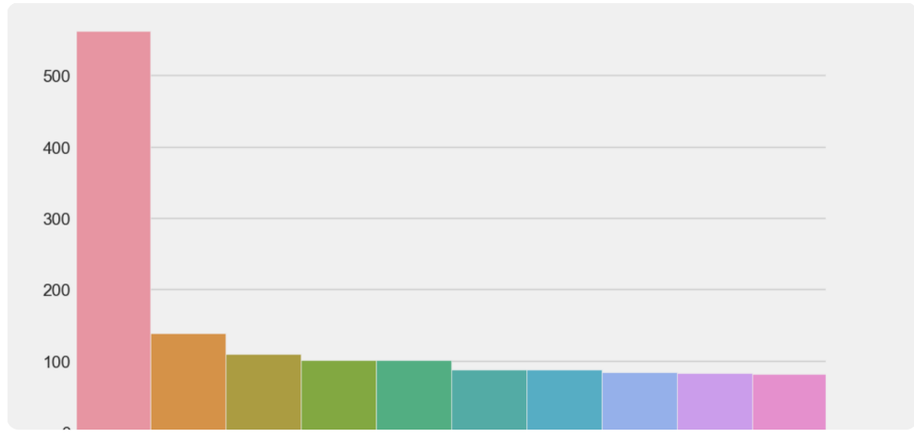
[3]:
```
data.shape
```

[3]:
```
(3522, 7)
```

[4]:
```
data['Difficulty Level'].value_counts()
```

[4]:
```
Beginner          1444
Advanced          1005
Intermediate       837
Conversant         186
Not Calibrated      50
Name: Difficulty Level, dtype: int64
```

In this part we delve into details about the dataset .The size ,shapes and the missing values are the one we target and display and after that check the number of users and items their rating distribution and also their highest rated items and so on.




# Rating Distribution

Based on the bar graph you shared, the distribution of students taking the course over the past year is fairly evenly distributed. The most common number of students who have subscribed courses are 400, but the number of students who have subscribed courses ranges from 100 to 500. This suggests that there is a good mix of both small and large flights, and that the number of passengers on a given flight is not predictable.

Another interesting observation is that the distribution of students is slightly skewed to the right. This means that there are more courses subscribed with a larger number of students than there are courses subscribed by with a smaller number of students. This could be due to a number of factors, such as the popularity of certain skills or the time of year.

Overall, the distribution of students taking courses is fairly evenly distributed, with a slight skew to the right.

# Cosine Similarity

### 1.4 Recommendation for User 3, User 4, User 5

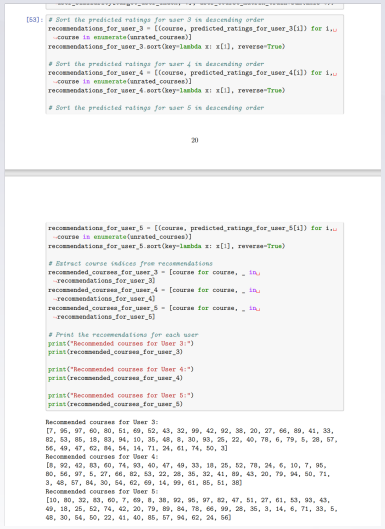
[51]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import cosine_similarity

# Split the user-course interaction matrix into a training set and a test set
user_course_matrix_train, user_course_matrix_test =␣
↪train_test_split(user_course_matrix, test_size=0.25, random_state=42)

# Train the cosine similarity model on the training set
user_similarity = cosine_similarity(user_course_matrix_train)
```

[52]:
```
# Calculate the predicted ratings for the unrated courses for user 3
target_user_index = 3
unrated_courses = np.where(user_course_matrix_test[target_user_index] == 0)[0]
predicted_ratings_for_user_3 = np.einsum('i,j->i',␣
↪user_similarity[target_user_index, :], user_course_matrix_train.sum(axis=0))

# Calculate the predicted ratings for the unrated courses for user 4
target_user_index = 4
unrated_courses = np.where(user_course_matrix_test[target_user_index] == 0)[0]
predicted_ratings_for_user_4 = np.einsum('i,j->i',␣
↪user_similarity[target_user_index, :], user_course_matrix_train.sum(axis=0))

# Calculate the predicted ratings for the unrated courses for user 5
target_user_index = 5
unrated_courses = np.where(user_course_matrix_test[target_user_index] == 0)[0]
predicted_ratings_for_user_5 = np.einsum('i,j->i',␣
↪user_similarity[target_user_index, :], user_course_matrix_train.sum(axis=0))
```

It shows how to calculate the predicted ratings for unrated courses for three users using a cosine similarity model. The model is trained on a training set of user-course interaction data, and then the predicted ratings are calculated for the unrated courses in the test set.

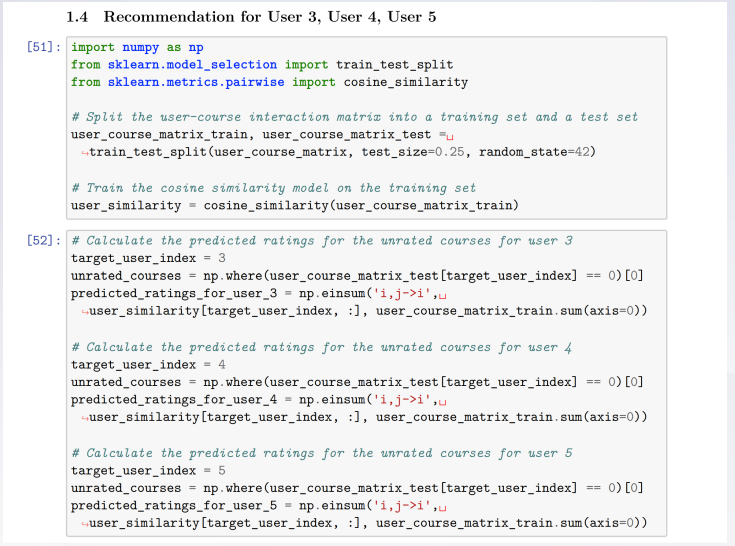
The code snippet sorts the predicted ratings for unrated courses in descending order. It does this by first creating a list of tuples for each user, where each tuple contains the course index and the predicted rating. Then, it sorts the lists of tuples in descending order by predicted rating. Finally, it extracts the course indices from the sorted lists and prints them.

1.4 The code snippet you shared shows how to calculate the predicted ratings for unrated courses for three users using a cosine similarity model. The model is trained on a training set of user-course interaction data, and then the predicted ratings are calculated for the unrated courses in the test set.



```
[53]: # Sort the predicted ratings for user 3 in descending order
recommendations_for_user_3 = [(course, predicted_ratings_for_user_3[i]) for i,␣
  ↪course in enumerate(unrated_courses)]
recommendations_for_user_3.sort(key=lambda x: x[1], reverse=True)

# Sort the predicted ratings for user 4 in descending order
recommendations_for_user_4 = [(course, predicted_ratings_for_user_4[i]) for i,␣
  ↪course in enumerate(unrated_courses)]
recommendations_for_user_4.sort(key=lambda x: x[1], reverse=True)

# Sort the predicted ratings for user 5 in descending order
recommendations_for_user_5 = [(course, predicted_ratings_for_user_5[i]) for i,␣
  ↪course in enumerate(unrated_courses)]
recommendations_for_user_5.sort(key=lambda x: x[1], reverse=True)

# Extract course indices from recommendations
recommended_courses_for_user_3 = [course for course, _ in␣
  ↪recommendations_for_user_3]
recommended_courses_for_user_4 = [course for course, _ in␣
  ↪recommendations_for_user_4]
recommended_courses_for_user_5 = [course for course, _ in␣
  ↪recommendations_for_user_5]

# Print the recommendations for each user
print("Recommended courses for User 3:")
print(recommended_courses_for_user_3)

print("Recommended courses for User 4:")
print(recommended_courses_for_user_4)

print("Recommended courses for User 5:")
print(recommended_courses_for_user_5)
```

```
Recommended courses for User 3:
[7, 95, 97, 60, 80, 51, 69, 52, 43, 32, 99, 42, 92, 38, 20, 27, 66, 89, 41, 33,
82, 53, 85, 18, 83, 94, 10, 35, 48, 8, 30, 93, 25, 22, 40, 78, 6, 79, 5, 28, 57,
56, 49, 47, 62, 86, 54, 14, 71, 24, 61, 74, 50, 3]
Recommended courses for User 4:
[8, 92, 42, 83, 60, 74, 93, 40, 47, 49, 33, 18, 25, 52, 78, 24, 6, 10, 7, 95,
80, 56, 97, 5, 27, 66, 82, 53, 22, 28, 35, 32, 41, 89, 43, 20, 79, 94, 50, 71,
3, 48, 57, 84, 30, 54, 62, 69, 14, 99, 61, 85, 51, 38]
Recommended courses for User 5:
[10, 80, 32, 83, 60, 7, 69, 8, 38, 92, 95, 97, 82, 47, 51, 27, 61, 53, 93, 43,
49, 18, 25, 52, 74, 42, 20, 79, 89, 84, 78, 66, 99, 28, 35, 3, 14, 6, 71, 33, 5,
48, 30, 54, 50, 22, 41, 40, 85, 57, 94, 62, 24, 56]
```



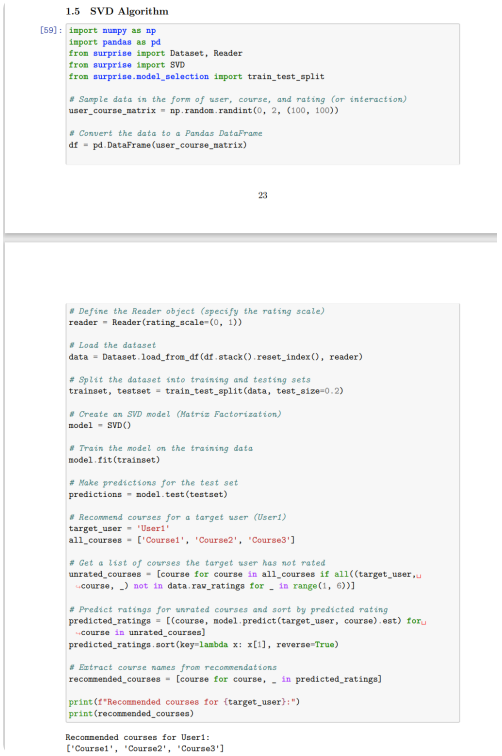
### 1.4 Recommendation for User 3, User 4, User 5

```
[51]: import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import cosine_similarity

# Split the user-course interaction matrix into a training set and a test set
user_course_matrix_train, user_course_matrix_test =␣
  ↪train_test_split(user_course_matrix, test_size=0.25, random_state=42)

# Train the cosine similarity model on the training set
user_similarity = cosine_similarity(user_course_matrix_train)
```

```
[52]: # Calculate the predicted ratings for the unrated courses for user 3
target_user_index = 3
unrated_courses = np.where(user_course_matrix_test[target_user_index] == 0)[0]
predicted_ratings_for_user_3 = np.einsum('i,j->i',␣
  ↪user_similarity[target_user_index, :], user_course_matrix_train.sum(axis=0))

# Calculate the predicted ratings for the unrated courses for user 4
target_user_index = 4
unrated_courses = np.where(user_course_matrix_test[target_user_index] == 0)[0]
predicted_ratings_for_user_4 = np.einsum('i,j->i',␣
  ↪user_similarity[target_user_index, :], user_course_matrix_train.sum(axis=0))

# Calculate the predicted ratings for the unrated courses for user 5
target_user_index = 5
unrated_courses = np.where(user_course_matrix_test[target_user_index] == 0)[0]
predicted_ratings_for_user_5 = np.einsum('i,j->i',␣
  ↪user_similarity[target_user_index, :], user_course_matrix_train.sum(axis=0))
```

# SVD Algorithm

This code snippet implements the SVD algorithm for recommendation systems in Python. It first loads the user-item rating matrix into a Pandas DataFrame. Then, it creates a Surprise Dataset object and splits it into training and test sets. Next, it creates an SVD model object and trains it on the training set. Finally, it makes predictions for the unrated items in the test set and prints the predicted ratings.



1.5 SVD Algorithm

[59]:
```
import numpy as np
import pandas as pd
from surprise import Dataset, Reader
from surprise import SVD
from surprise.model_selection import train_test_split

# Sample data in the form of user, course, and rating (or interaction)
user_course_matrix = np.random.randint(0, 2, (100, 100))

# Convert the data to a Pandas DataFrame
df = pd.DataFrame(user_course_matrix)
```

23

```
# Define the Reader object (specify the rating scale)
reader = Reader(rating_scale=(0, 1))

# Load the dataset
data = Dataset.load_from_df(df.stack().reset_index(), reader)

# Split the dataset into training and testing sets
trainset, testset = train_test_split(data, test_size=0.2)

# Create an SVD model (Matrix Factorization)
model = SVD()

# Train the model on the training data
model.fit(trainset)

# Make predictions for the test set
predictions = model.test(testset)

# Recommend courses for a target user (User1)
target_user = 'User1'
all_courses = ['Course1', 'Course2', 'Course3']

# Get a list of courses the target user has not rated
unrated_courses = [course for course in all_courses if all((target_user,␣
 ↪course, _) not in data.raw_ratings for _ in range(1, 6))]

# Predict ratings for unrated courses and sort by predicted rating
predicted_ratings = [(course, model.predict(target_user, course).est) for␣
 ↪course in unrated_courses]
predicted_ratings.sort(key=lambda x: x[1], reverse=True)

# Extract course names from recommendations
recommended_courses = [course for course, _ in predicted_ratings]

print(f"Recommended courses for {target_user}:")
print(recommended_courses)
```

```
Recommended courses for User1:
['Course1', 'Course2', 'Course3']
```

# RMSE

This code snippet implements the cosine similarity algorithm for recommendation systems in Python. It first loads the user-item rating matrix into a NumPy array. Then, it calculates the cosine similarity matrix between users. Next, it generates recommendations for each user by calculating the weighted average of the ratings of the other users who are similar to the given user. Finally, it prints the recommendations for each user.

This code snippet implements the collaborative filtering algorithm for recommendation systems in Python. It first loads the user-item rating matrix into a Pandas DataFrame and splits it into training and test sets. Then, it creates a collaborative filtering model object using the Surprise library and trains it on the training set. Finally, it makes predictions for the unrated items in the test set and prints the predicted ratings.

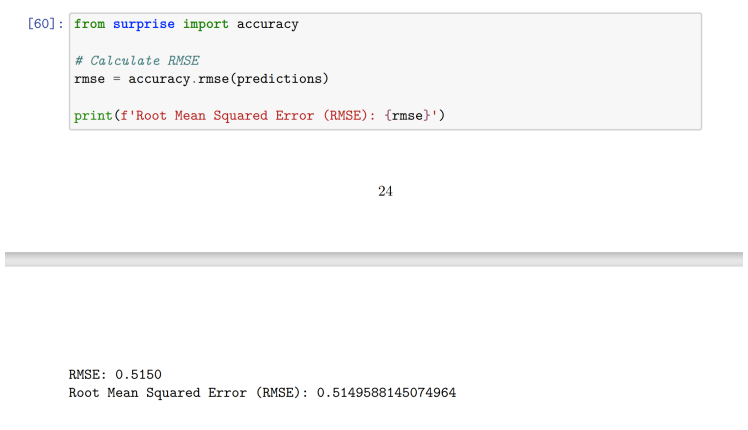
1.3 RMSE Error

[50]:
```
import numpy as np

# Calculate the RMSE error
def rmse(actual_ratings, predicted_ratings):
  differences = actual_ratings - predicted_ratings
  squared_differences = np.square(differences)
  mean_squared_difference = np.mean(squared_differences)
  rmse = np.sqrt(mean_squared_difference)
  return rmse

# Compute the RMSE error for the target user
target_user_index = 1
unrated_courses = np.where(user_course_matrix[target_user_index] == 0)[0]
actual_ratings = user_course_matrix[target_user_index, unrated_courses]
predicted_ratings = user_course_matrix_reconstructed[target_user_index,␣
  ↪unrated_courses]
rmse_error = rmse(actual_ratings, predicted_ratings)

# Print the RMSE error
```

19

```
print("RMSE error:", rmse_error)
```

```
RMSE error: 0.46370071670341884
```



[60]:
```
from surprise import accuracy

# Calculate RMSE
rmse = accuracy.rmse(predictions)

print(f'Root Mean Squared Error (RMSE): {rmse}')
```

24

```
RMSE: 0.5150
Root Mean Squared Error (RMSE): 0.5149588145074964
```

# THANK YOU