Course Recommendations System – White Paper

I. Business problem

In today's digital world, online learning sites like Udemy and Coursera have many courses on different topics. However, with so many choices, users often struggle to pick the right ones. The goal of this project is to solve this problem by creating a Course Recommendation System that gives personalized course suggestions based on what users like and what they want to learn.

II. Background / History

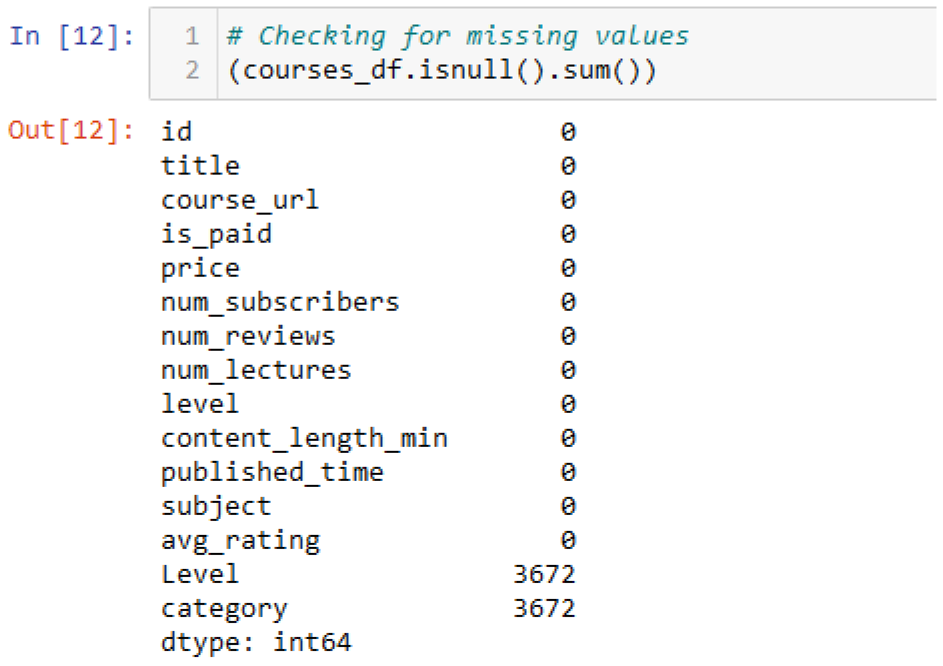
The rise of online learning platforms has made education more available, giving learners access to many courses. However, having so many options can overwhelm users. Previous research shows that personalized recommendations can greatly improve user satisfaction and engagement, resulting in better learning results.

III. Data Explanation

1. Data Preparation

Two datasets from Udemy and Coursera were used for this project. The data includes course details such as titles, descriptions, ratings, reviews, and enrollment numbers.

Cleaning and standardizing the data are important steps in getting the datasets ready for analysis. First, we checked for missing values and removed any duplicate entries from the Udemy and Coursera datasets to ensure that each course was listed only once.

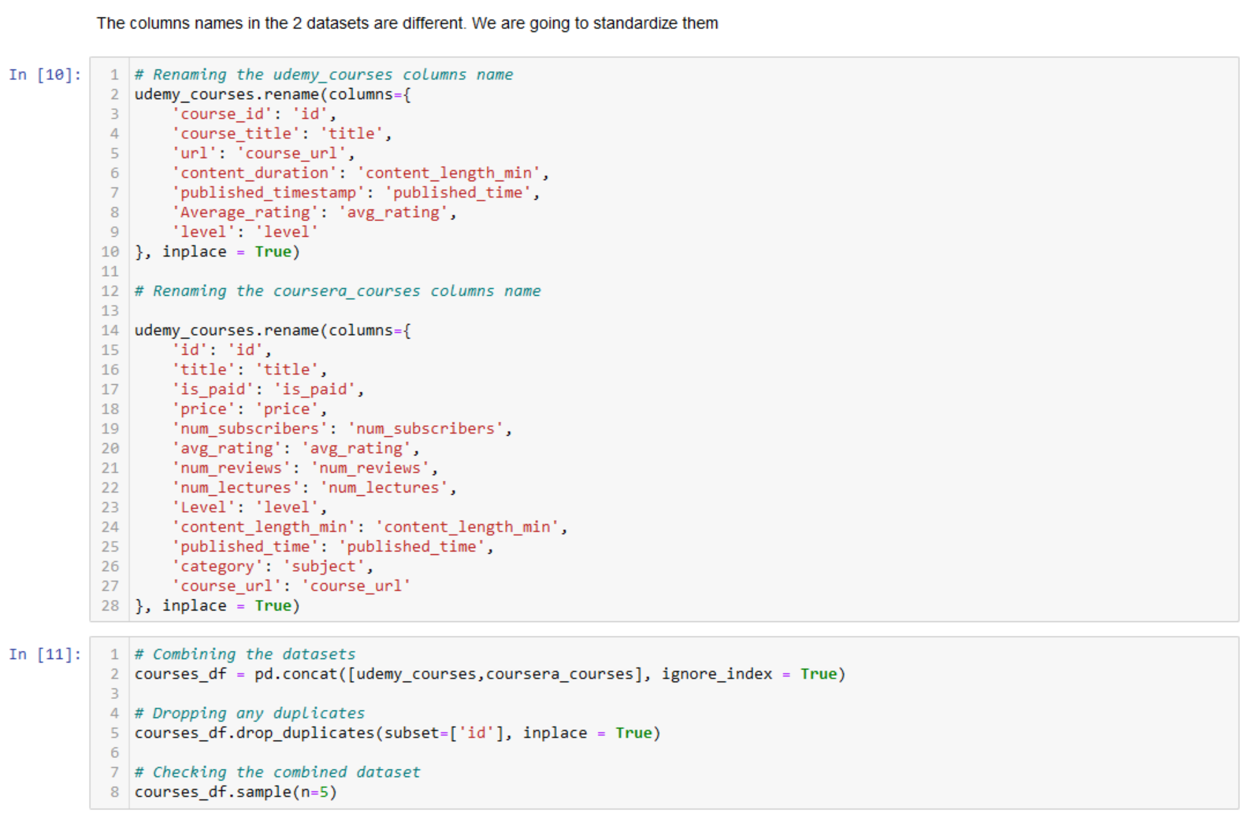


```
In [12]:  # Checking for missing values
          (courses_df.isnull().sum())
```

```
Out[12]:  id                       0
          title                    0
          course_url               0
          is_paid                  0
          price                    0
          num_subscribers          0
          num_reviews              0
          num_lectures             0
          level                    0
          content_length_min       0
          published_time           0
          subject                  0
          avg_rating               0
          Level                 3672
          category              3672
          dtype: int64
```

Fig1. Checking for missing values

Next, we made sure that the names of the columns in both datasets were the same. This helps us combine the two datasets more easily. For example, we changed *course_title* to *title* and *published_timestamp* to *published_time* so that they match. We also checked for missing information and fixed any gaps to make sure our analysis would be accurate.



The columns names in the 2 datasets are different. We are going to standardize them

```
In [10]:
# Renaming the udemy_courses columns name
udemy_courses.rename(columns={
    'course_id': 'id',
    'course_title': 'title',
    'url': 'course_url',
    'content_duration': 'content_length_min',
    'published_timestamp': 'published_time',
    'Average_rating': 'avg_rating',
    'level': 'level'
}, inplace = True)

# Renaming the coursera_courses columns name

udemy_courses.rename(columns={
    'id': 'id',
    'title': 'title',
    'is_paid': 'is_paid',
    'price': 'price',
    'num_subscribers': 'num_subscribers',
    'avg_rating': 'avg_rating',
    'num_reviews': 'num_reviews',
    'num_lectures': 'num_lectures',
    'Level': 'level',
    'content_length_min': 'content_length_min',
    'published_time': 'published_time',
    'category': 'subject',
    'course_url': 'course_url'
}, inplace = True)
```

```
In [11]:
# Combining the datasets
courses_df = pd.concat([udemy_courses,coursera_courses], ignore_index = True)

# Dropping any duplicates
courses_df.drop_duplicates(subset=['id'], inplace = True)

# Checking the combined dataset
courses_df.sample(n=5)
```

Fig2. Standardizing and combining the 2 datasets

2. Data Dictionary

id: Unique identifier for the course.

title: Title of the course.

course_url: URL to the course.

is_paid: Indicates whether the course is free or paid.

price: Price of the course.

num_subscribers: Number of users enrolled.

num_reviews: Number of reviews.

num_lectures: Number of lectures in the course.

level: Skill level (beginner, intermediate, advanced).

content_length_min: Duration of the course in minutes.

published_time: Date the course was published.

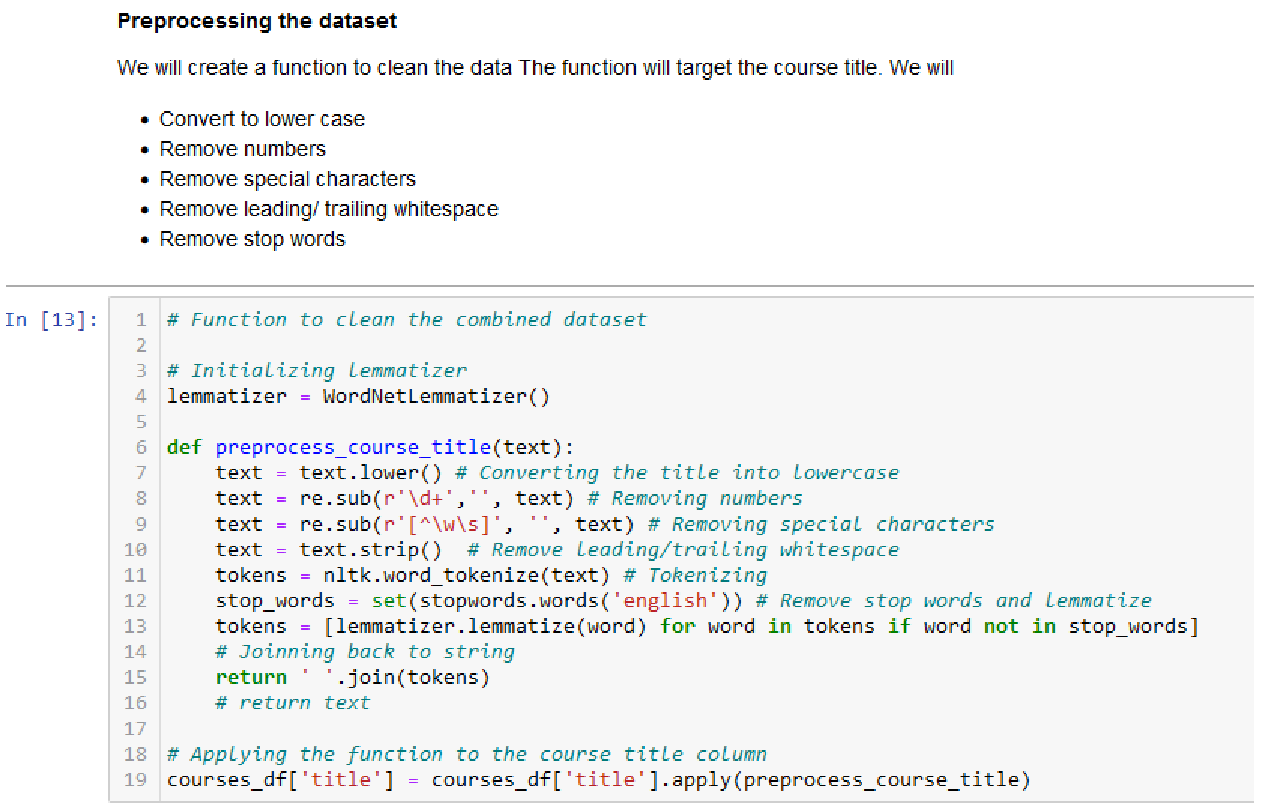
subject: Subject category.

avg_rating: Average course rating.

IV. Methods

To build the course recommendation system, we followed several steps:

- *Text preprocessing*: We used Natural Language Processing (NLP) techniques (tokenization, lemmatization, and removal of stop words) to process the Course Titles.



Preprocessing the dataset

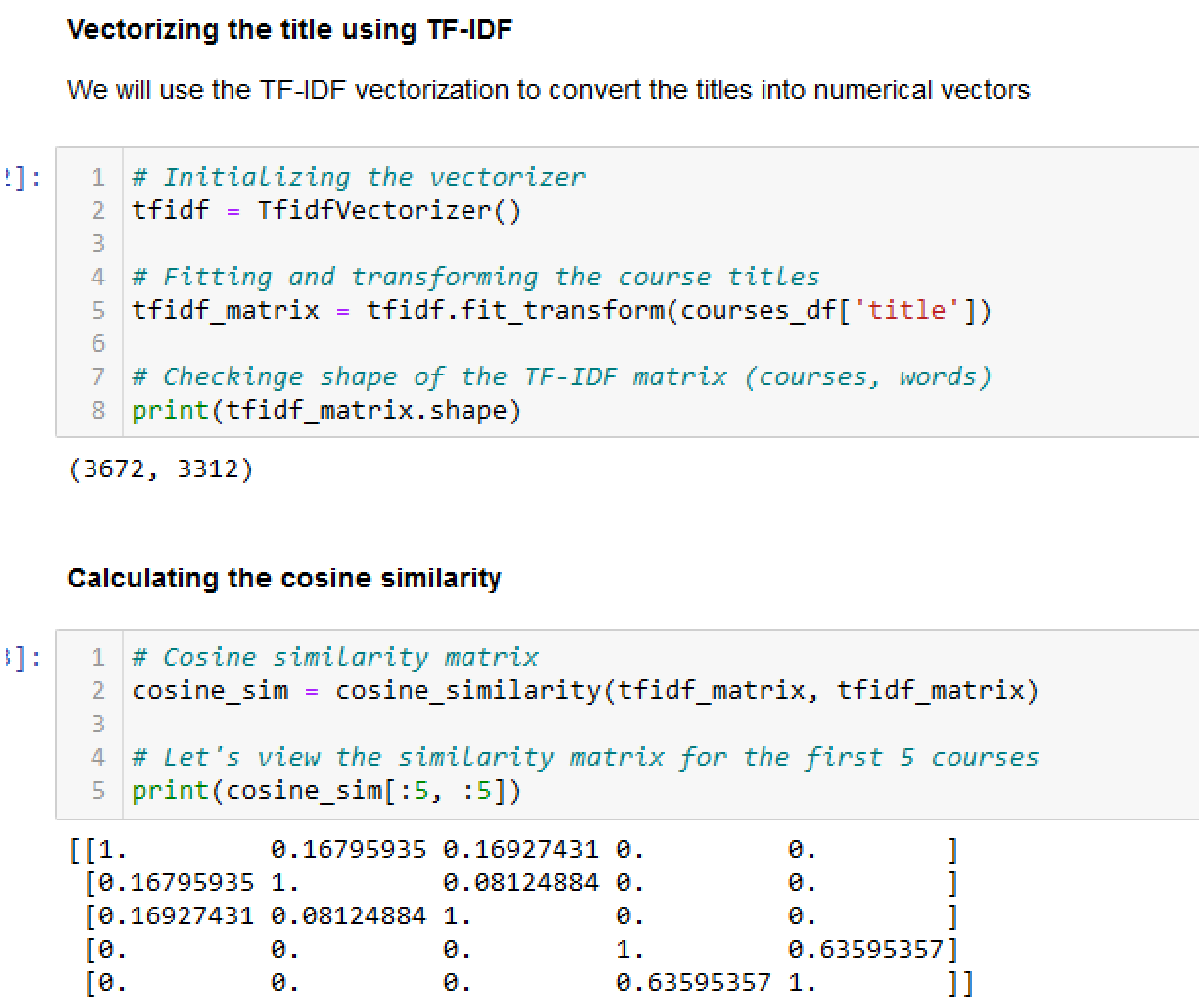
We will create a function to clean the data The function will target the course title. We will

- Convert to lower case
- Remove numbers
- Remove special characters
- Remove leading/ trailing whitespace
- Remove stop words

```
In [13]:
# Function to clean the combined dataset

# Initializing Lemmatizer
lemmatizer = WordNetLemmatizer()

def preprocess_course_title(text):
    text = text.lower() # Converting the title into lowercase
    text = re.sub(r'\d+','', text) # Removing numbers
    text = re.sub(r'[^\w\s]', '', text) # Removing special characters
    text = text.strip()  # Remove leading/trailing whitespace
    tokens = nltk.word_tokenize(text) # Tokenizing
    stop_words = set(stopwords.words('english')) # Remove stop words and lemmatize
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    # Joinning back to string
    return ' '.join(tokens)
    # return text

# Applying the function to the course title column
courses_df['title'] = courses_df['title'].apply(preprocess_course_title)
```

Fig 3: Text preprocessing

- *TF-IDF Vectorization*: We converted the course titles into numerical vectors using TF-IDF to capture the significance of each word.
- *Cosine Similarity calculation:* The cosine similarity was computed to evaluate the similarity between courses based on their titles.



Vectorizing the title using TF-IDF

We will use the TF-IDF vectorization to convert the titles into numerical vectors

```python
# Initializing the vectorizer
tfidf = TfidfVectorizer()

# Fitting and transforming the course titles
tfidf_matrix = tfidf.fit_transform(courses_df['title'])

# Checkinge shape of the TF-IDF matrix (courses, words)
print(tfidf_matrix.shape)
```

```
(3672, 3312)
```

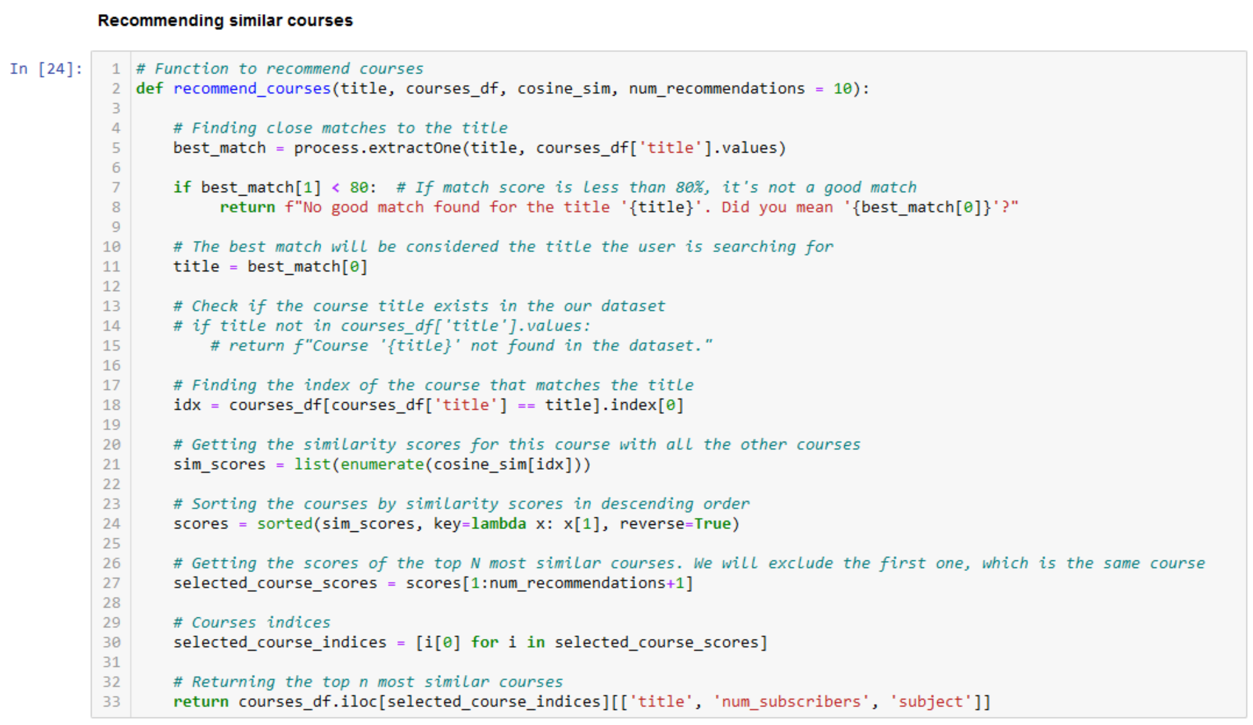
Calculating the cosine similarity

```python
# Cosine similarity matrix
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)

# Let's view the similarity matrix for the first 5 courses
print(cosine_sim[:5, :5])
```

```
[[1.         0.16795935 0.16927431 0.         0.        ]
 [0.16795935 1.         0.08124884 0.         0.        ]
 [0.16927431 0.08124884 1.         0.         0.        ]
 [0.         0.         0.         1.         0.63595357]
 [0.         0.         0.         0.63595357 1.        ]]
```

Fig 4: Vectorization and Cosine similarity

- *Recommendation algorithm*: We created a function to recommend similar courses based on the user input. The function is based on Cosine Similarity, which measures how similar two courses are based on their titles. When a user enters a course title, the algorithm first looks for the closest match in the dataset using a fuzzy matching algorithm. This helps find a title even if there are spelling differences or variations. Once the best match is identities, the algorithm retrieves the similarity scores of that course compared to all other courses in the dataset. By sorting these scores in order of similarity, we select the top courses that are most similar to the original title. The user receives personalized recommendations that closely align with their interests based on the title of the courses they are looking for.

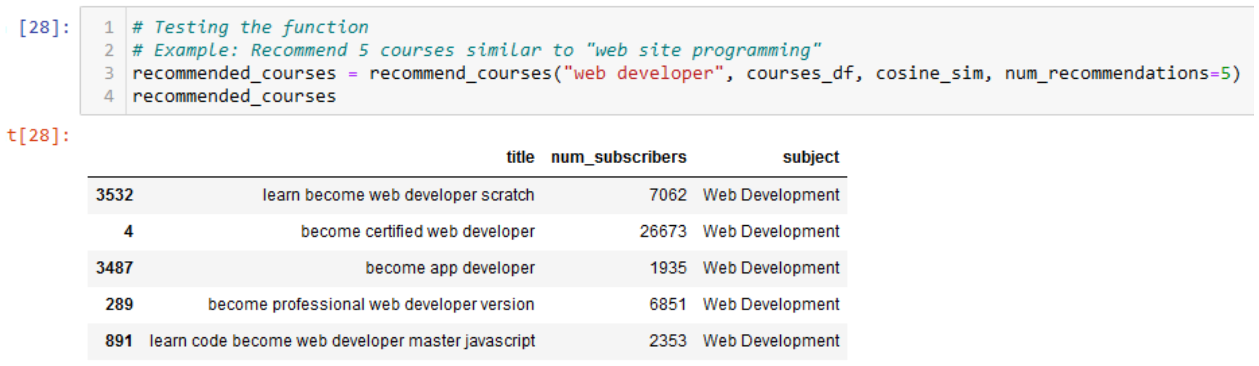


Recommending similar courses

```
In [24]:
# Function to recommend courses
def recommend_courses(title, courses_df, cosine_sim, num_recommendations = 10):

    # Finding close matches to the title
    best_match = process.extractOne(title, courses_df['title'].values)

    if best_match[1] < 80:  # If match score is less than 80%, it's not a good match
        return f"No good match found for the title '{title}'. Did you mean '{best_match[0]}'?"

    # The best match will be considered the title the user is searching for
    title = best_match[0]

    # Check if the course title exists in the our dataset
    # if title not in courses_df['title'].values:
        # return f"Course '{title}' not found in the dataset."

    # Finding the index of the course that matches the title
    idx = courses_df[courses_df['title'] == title].index[0]

    # Getting the similarity scores for this course with all the other courses
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sorting the courses by similarity scores in descending order
    scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Getting the scores of the top N most similar courses. We will exclude the first one, which is the same course
    selected_course_scores = scores[1:num_recommendations+1]

    # Courses indices
    selected_course_indices = [i[0] for i in selected_course_scores]

    # Returning the top n most similar courses
    return courses_df.iloc[selected_course_indices][['title', 'num_subscribers', 'subject']]
```

Fig 5: Course recommendation algorithm

- *Testing the algorithm*: We tested the algorithm by searching for 5 recommendations courses related to “Web developer”



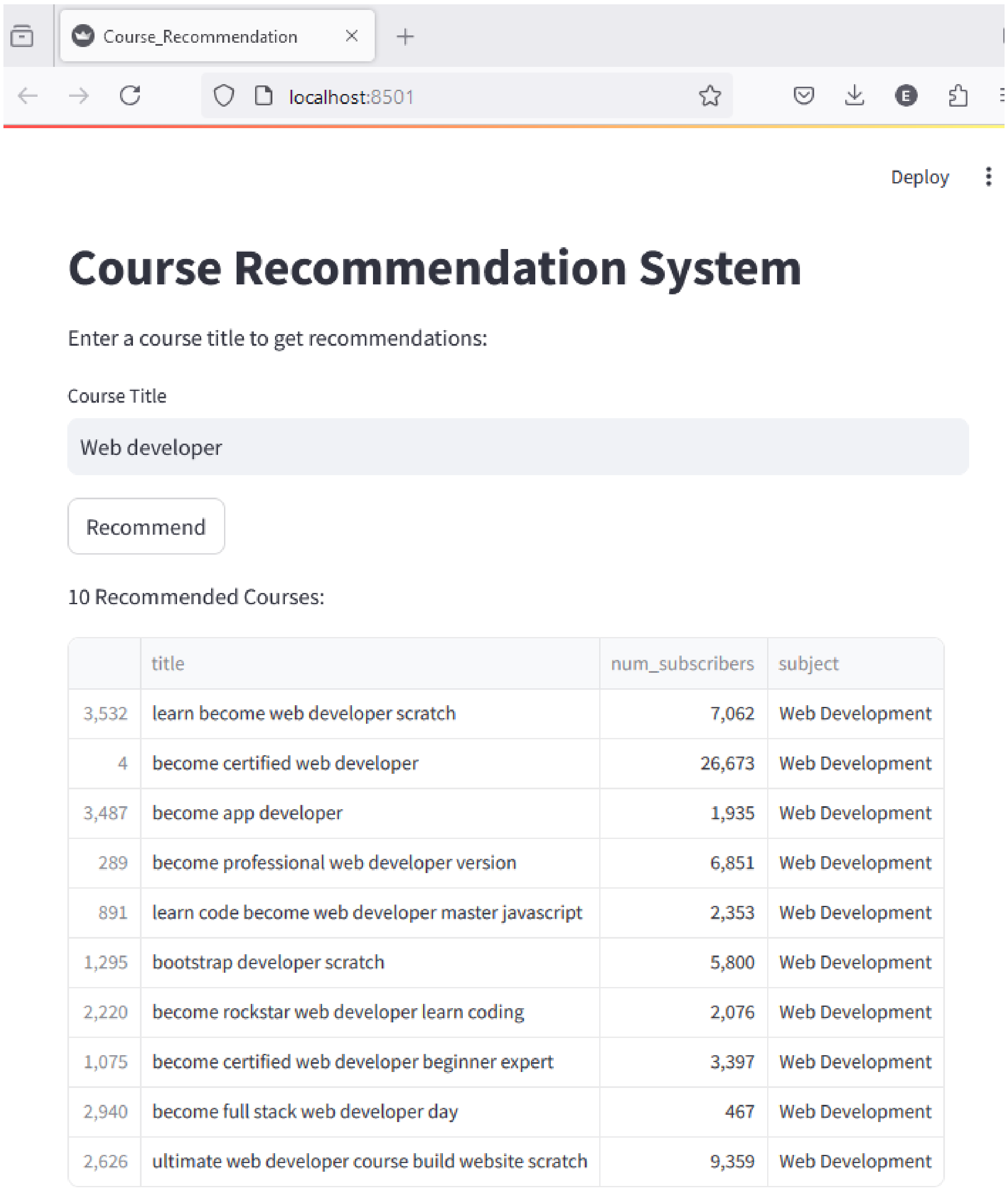
```
[28]:  # Testing the function
       # Example: Recommend 5 courses similar to "web site programming"
       recommended_courses = recommend_courses("web developer", courses_df, cosine_sim, num_recommendations=5)
       recommended_courses
```

t[28]:

	title	num_subscribers	subject
3532	learn become web developer scratch	7062	Web Development
4	become certified web developer	26673	Web Development
3487	become app developer	1935	Web Development
289	become professional web developer version	6851	Web Development
891	learn code become web developer master javascript	2353	Web Development

Fig 6: Testing the course recommendation function

- *Creating an interface using Streamlit*: To make the course recommendation user-friendly, we created an interface using Streamlit. It is a tool that allows to create interactive web applications quickly and easily. The user can enter the title of a course they are interested in and once they click the "Recommend" button, the application processes the input using the recommendation algorithm. It then displays a list of recommended courses based on the user's input.



Course Recommendation System

Enter a course title to get recommendations:

Course Title

Web developer

10 Recommended Courses:

	title	num_subscribers	subject
3,532	learn become web developer scratch	7,062	Web Development
4	become certified web developer	26,673	Web Development
3,487	become app developer	1,935	Web Development
289	become professional web developer version	6,851	Web Development
891	learn code become web developer master javascript	2,353	Web Development
1,295	bootstrap developer scratch	5,800	Web Development
2,220	become rockstar web developer learn coding	2,076	Web Development
1,075	become certified web developer beginner expert	3,397	Web Development
2,940	become full stack web developer day	467	Web Development
2,626	ultimate web developer course build website scratch	9,359	Web Development

Fig 7: Web interface of the course recommendation system using Streamlit

V. Analysis

We created several visualizations to analyze the distribution of the courses.

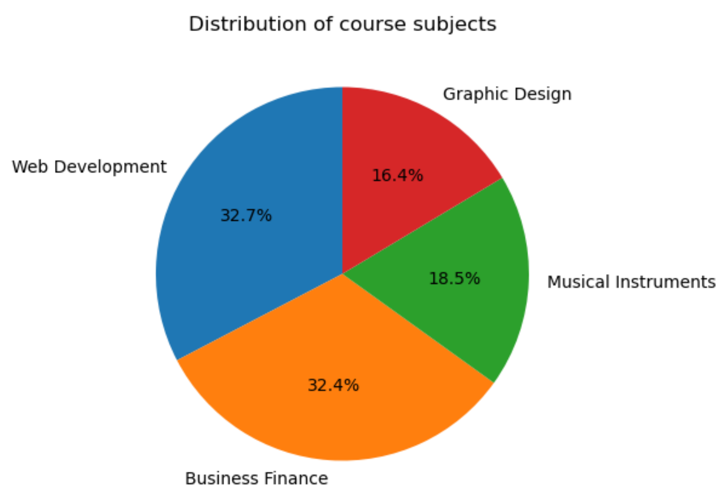


Fig 8: Pie chart of the distribution of courses

The pie chart shows the percentage of courses in different subjects. Web Development has the highest percentage at 32.7%, which means many people are interested in learning this skill. Business Finance is very close behind at 32.4%. Graphic Design makes up 16.4% of the total, showing some interest but less than the top two. Lastly, courses about Musical Instruments account for 18.5%, indicating a good interest in music and creativity.

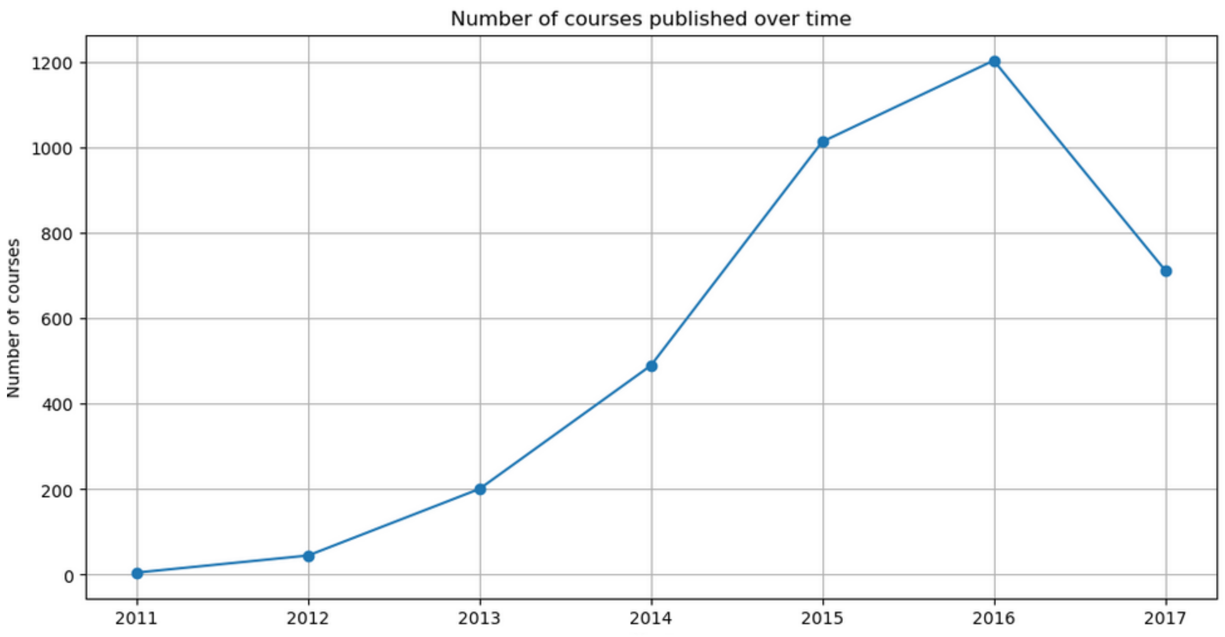


Fig 9: Line chart of the number of courses published over time

This graph shows that there is an increase in the number of courses published between 2011 and 2016 with a slight decrease in 2017.

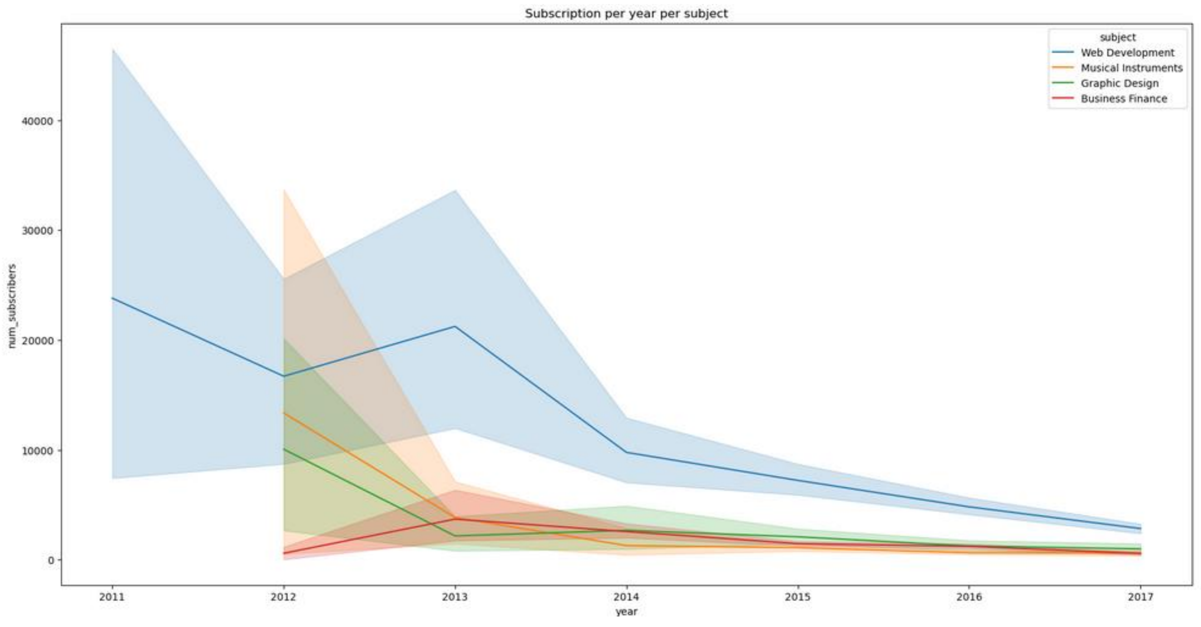


Fig 10: Line plot of the subscription per year and per subject

This graph shows that there has been a decline in the number of new subscriptions throughout the years. This is likely due to the plethora of different online platforms that are available now.

VI. Conclusion

The course recommendation system solves the problem by giving users personalized course suggestions, improving their learning experience. It uses Natural Language Processing techniques and machine learning to recommend courses based on what the user likes.

VII. Assumption

For this course recommendation system, we assume that:

- Users will provide accurate and relevant course titles for recommendations.
- The datasets are representative of the courses available on the platforms.
- Course ratings and reviews are reliable indicators of course quality.

VIII. Limitations

The recommendation system is limited to the courses available in the datasets. Newly released courses may not be included. The recommendations are primarily based on course titles, which may not capture all relevant content aspects.

IX. Challenges

One of the main challenges we faced was cleaning and organizing the data from two different platforms, Udemy and Coursera. The datasets had different column names and structures, so we had to standardize them to make sure they could be combined correctly. Another challenge was building an accurate recommendation system that could handle a scenario where the user entered a title that was not available. We had to use fuzzy matching to use the closest match instead.

X. Future Uses/Additional Applications

The recommendation system can be extended to include user feedback for continuous improvement. Machine learning algorithms can be incorporated to personalize recommendations further based on user interactions and preferences.

XI. Recommendations

We can implement user feedback mechanisms to refine the recommendation algorithm continuously. We can also incorporate additional data sources or past course completions, for more personalized recommendations.

XII. Implementation Plan

We can implement the course recommendation system by setting up a could platform like AWS or Microsoft Azure so that multiple users can use it at the same time. The Streamlit web interface will also be available online, allowing users to easily access it via a browser. We will also keep the data updated and improve the system based on user feedback.

XIII. Ethical Assessment

Data Privacy: We made sure to follow data protection rules and keep users' data safe. No personal data was used.

Bias Mitigation: We were vigilant against biases in recommendations that may favor popular courses over lesser-known but valuable options.

Transparency: To promote trust in the recommendation system, users should understand how recommendations are generated.

References

Aggarwal, C. C. (2016). *Recommender systems: The textbook*. Springer.

Flask documentation. (n.d.). Retrieved from https://flask.palletsprojects.com/en/2.0.x/

Manning, C. D. (2008). *Introduction to information retrieval*. MIT Press.

Natural Language Toolkit (NLTK). (n.d.). Retrieved from NLTK documentation: https://www.nltk.org/

Streamlit documentation. Retrieved from Streamlit: https://docs.streamlit.io/