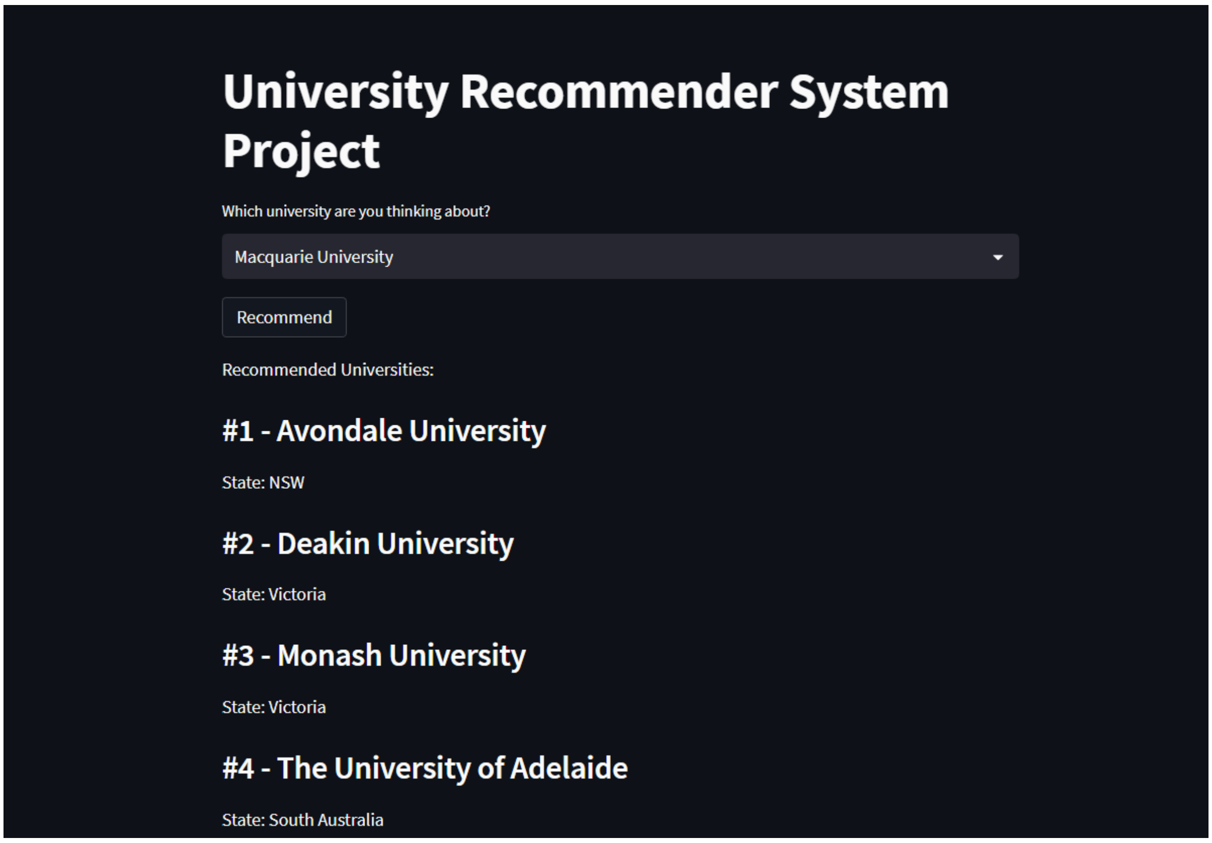
Course / University Recommendation System Project Documentation



University Recommender System Project

Which university are you thinking about?

Macquarie University

Recommend

Recommended Universities:

#1 - Avondale University

State: NSW

#2 - Deakin University

State: Victoria

#3 - Monash University

State: Victoria

#4 - The University of Adelaide

State: South Australia

Prepared By:
Aryan Karki

1. Introduction

The goal of this assignment is to develop a machine learning-based course recommendation system using a dataset of student preferences and course information.

In order to achieve this, we will first explore and preprocess the dataset, ensuring its quality and addressing any missing values or inconsistencies. Feature engineering and data transformations will be performed to extract meaningful information from the available data.

After that, an appropriate machine learning algorithm was selected for the recommendation system. The chosen model was implemented and trained using the dataset. Hyperparameter tuning and regularisation techniques was applied to optimise the model's performance.

Finally, a user-friendly interface will be developed to showcase the recommendation system. The trained model will be integrated into the interface to provide personalised course recommendations based on the user's preferences.

2. Data Preprocessing

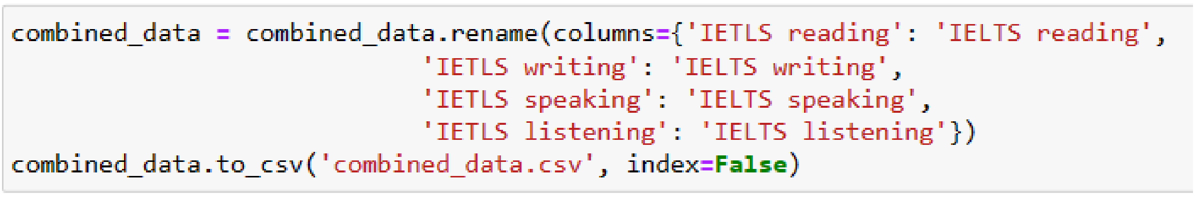
The data pre-processing involves taking the excel sheet titled ‘Australia Data’ and performing necessary changes and transformations to the data in order to make them ready for being fed into my recommendation system model.

The dataset contained metadata regarding the different features present, a list of all universities whose data was to be found there and comprehensive information about those universities, separated into seven different files pertaining to each state of Australia and containing a total of 36 columns (features).

All the datasets of each state in Australia were downloaded and combined into a single cs file, ‘combined_data.csv’. After this, the preprocessing process began with first getting information about missing values present throughout the dataset.

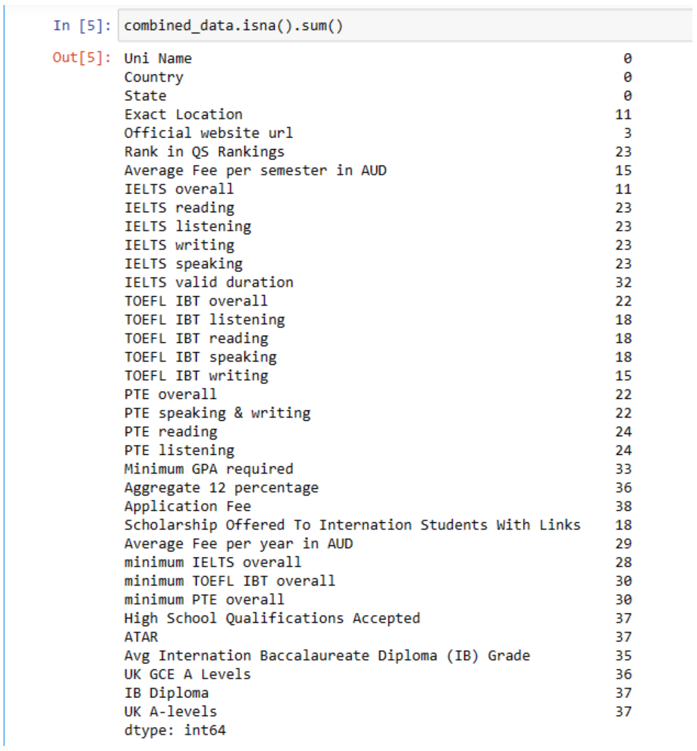
The combined dataset revealed 38 rows of University information and 36 columns.

Firstly, some changes of names of features were in order, for the sake of remaining grammatically correct and making things easier.



```
combined_data = combined_data.rename(columns={'IETLS reading': 'IELTS reading',
                          'IETLS writing': 'IELTS writing',
                          'IETLS speaking': 'IELTS speaking',
                          'IETLS listening': 'IELTS listening'})
combined_data.to_csv('combined_data.csv', index=False)
```

The check for missing values throughout the dataset revealed the following:

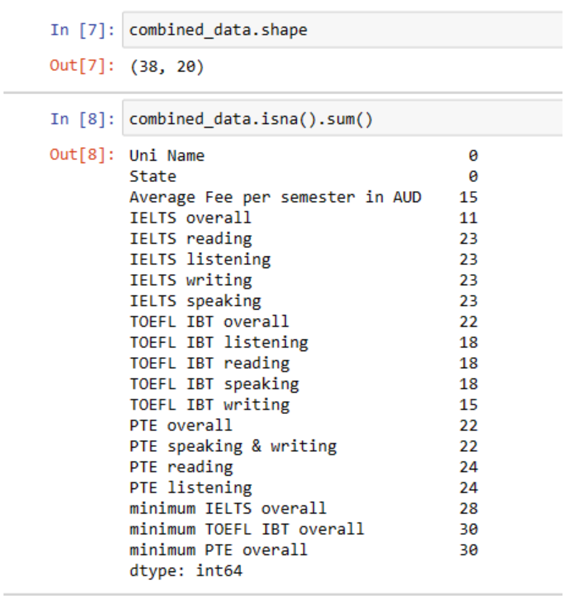


```
In [5]: combined_data.isna().sum()

Out[5]: Uni Name                                            0
        Country                                             0
        State                                               0
        Exact Location                                     11
        Official website url                                3
        Rank in QS Rankings                                23
        Average Fee per semester in AUD                    15
        IELTS overall                                      11
        IELTS reading                                      23
        IELTS listening                                    23
        IELTS writing                                      23
        IELTS speaking                                     23
        IELTS valid duration                               32
        TOEFL IBT overall                                  22
        TOEFL IBT listening                                18
        TOEFL IBT reading                                  18
        TOEFL IBT speaking                                 18
        TOEFL IBT writing                                  15
        PTE overall                                        22
        PTE speaking & writing                             22
        PTE reading                                        24
        PTE listening                                      24
        Minimum GPA required                               33
        Aggregate 12 percentage                            36
        Application Fee                                    38
        Scholarship Offered To Internation Students With Links    18
        Average Fee per year in AUD                        29
        minimum IELTS overall                              28
        minimum TOEFL IBT overall                          30
        minimum PTE overall                                30
        High School Qualifications Accepted                37
        ATAR                                               37
        Avg Internation Baccalaureate Diploma (IB) Grade   35
        UK GCE A Levels                                    36
        IB Diploma                                         37
        UK A-levels                                        37
        dtype: int64
```

Taking into account the number of missing values per column as well as the relevancy of the feature for my recommendation system, the following columns were chosen to be dropped.

'Country', 'Exact Location', 'Official website url', 'Rank in QS Rankings', 'IELTS valid duration', 'Minimum GPA required', 'Aggregate 12 percentage', 'Application Fee', 'Average Fee per year in AUD', 'Scholarship Offered To International Students With Links', 'IB Diploma', 'UK A-levels', 'UK GCE A Levels', 'IB Diploma', 'UK A-levels', 'High School Qualifications Accepted', 'ATAR', and 'Avg International Baccalaureate Diploma (IB) Grade'

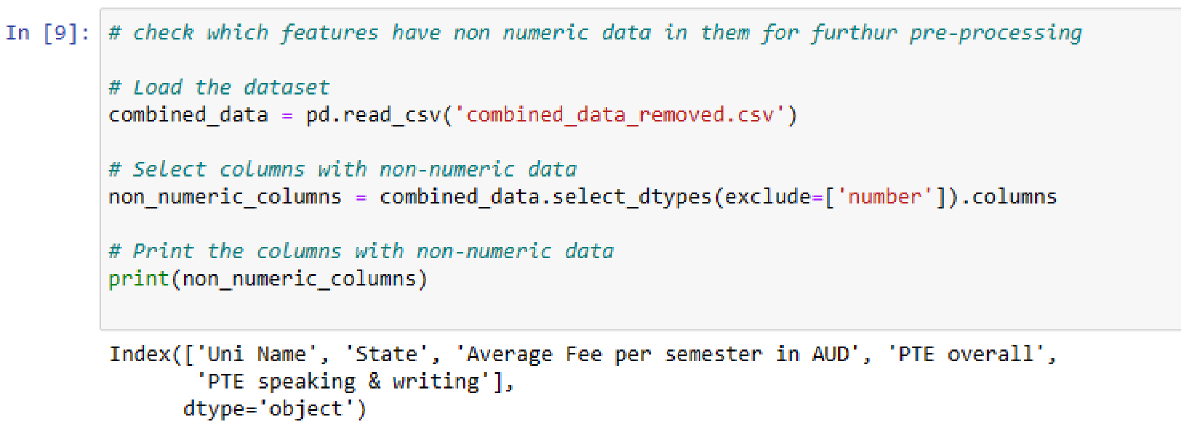
The new dataset looked as such and was stored as ‘combined_data_removed.csv’



```
In [7]: combined_data.shape

Out[7]: (38, 20)
```

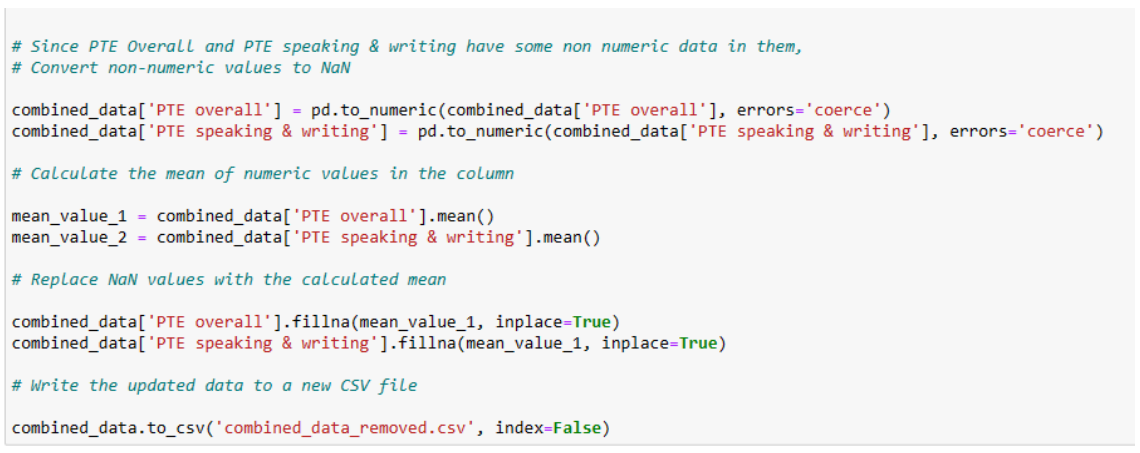
```
In [8]: combined_data.isna().sum()

Out[8]: Uni Name                          0
        State                             0
        Average Fee per semester in AUD  15
        IELTS overall                    11
        IELTS reading                    23
        IELTS listening                  23
        IELTS writing                    23
        IELTS speaking                   23
        TOEFL IBT overall                22
        TOEFL IBT listening              18
        TOEFL IBT reading                18
        TOEFL IBT speaking               18
        TOEFL IBT writing                15
        PTE overall                      22
        PTE speaking & writing           22
        PTE reading                      24
        PTE listening                    24
        minimum IELTS overall            28
        minimum TOEFL IBT overall        30
        minimum PTE overall              30
        dtype: int64
```

Then, I checked for all the columns that had numeric values and those that might have other than numeric values in them.



```
In [9]: # check which features have non numeric data in them for furthur pre-processing

        # Load the dataset
        combined_data = pd.read_csv('combined_data_removed.csv')

        # Select columns with non-numeric data
        non_numeric_columns = combined_data.select_dtypes(exclude=['number']).columns

        # Print the columns with non-numeric data
        print(non_numeric_columns)

        Index(['Uni Name', 'State', 'Average Fee per semester in AUD', 'PTE overall',
               'PTE speaking & writing'],
              dtype='object')
```

Now, processing for ‘Average Fee per Semester in AUD’, ‘PTE overall’ and ‘PTE speaking & writing’ was required before filling the missing numeric values in these fields.



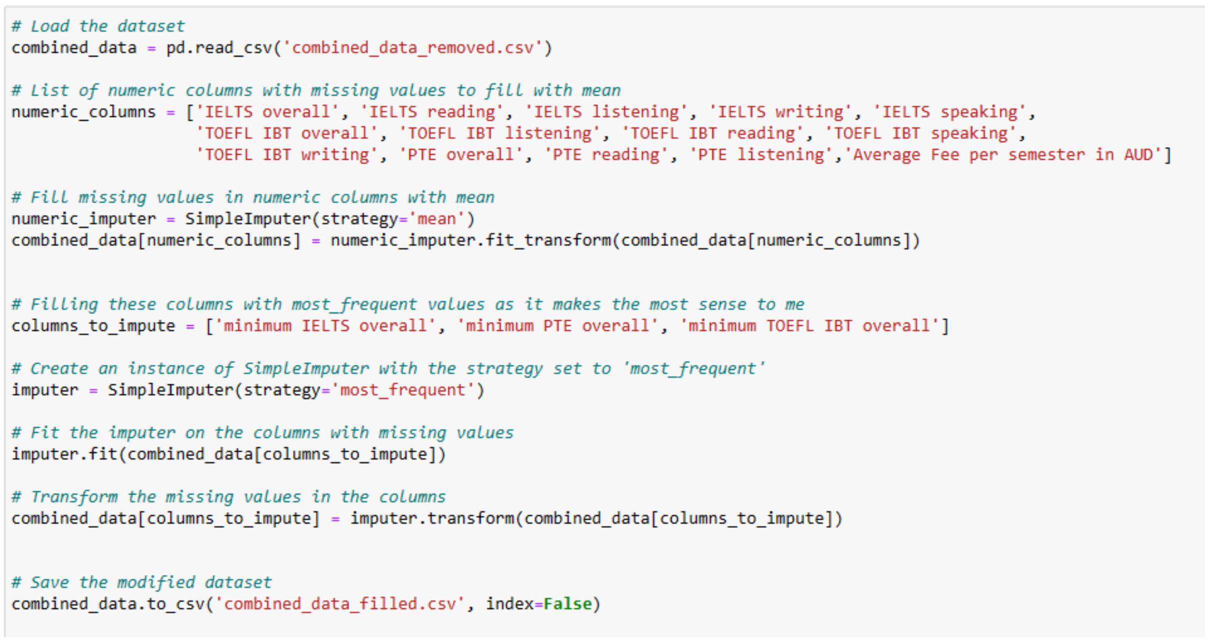
```
# Since PTE Overall and PTE speaking & writing have some non numeric data in them,
# Convert non-numeric values to NaN

combined_data['PTE overall'] = pd.to_numeric(combined_data['PTE overall'], errors='coerce')
combined_data['PTE speaking & writing'] = pd.to_numeric(combined_data['PTE speaking & writing'], errors='coerce')

# Calculate the mean of numeric values in the column

mean_value_1 = combined_data['PTE overall'].mean()
mean_value_2 = combined_data['PTE speaking & writing'].mean()

# Replace NaN values with the calculated mean

combined_data['PTE overall'].fillna(mean_value_1, inplace=True)
combined_data['PTE speaking & writing'].fillna(mean_value_1, inplace=True)

# Write the updated data to a new CSV file

combined_data.to_csv('combined_data_removed.csv', index=False)
```

# Investigating the 'Average Fee per semester in AUD' reveals '$' is redundant there and can hamper with furthur preprocessing..

# Remove '$' sign, commas, and 'AU' from the values

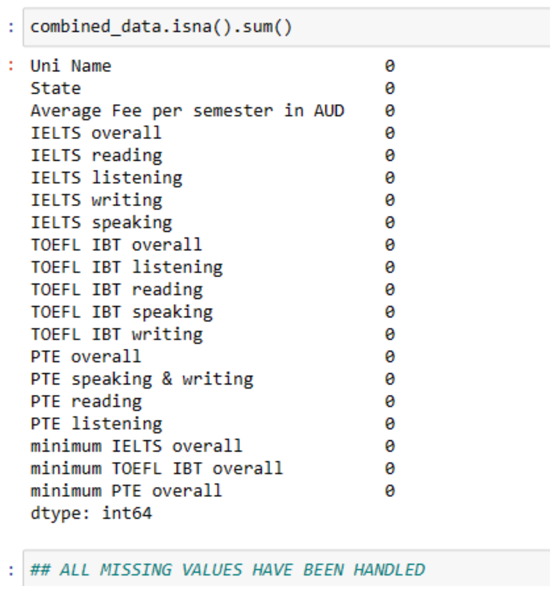
combined_data['Average Fee per semester in AUD'] = pd.to_numeric(combined_data['Average Fee per semester in AUD'].str.replace('$', '').str.replace(',', '').str.replace('AU', ''), errors='coerce')

At this point, the dataset was ready for having its numeric missing values dealt with.



```
# Load the dataset
combined_data = pd.read_csv('combined_data_removed.csv')

# List of numeric columns with missing values to fill with mean
numeric_columns = ['IELTS overall', 'IELTS reading', 'IELTS listening', 'IELTS writing', 'IELTS speaking',
                   'TOEFL IBT overall', 'TOEFL IBT listening', 'TOEFL IBT reading', 'TOEFL IBT speaking',
                   'TOEFL IBT writing', 'PTE overall', 'PTE reading', 'PTE listening','Average Fee per semester in AUD']

# Fill missing values in numeric columns with mean
numeric_imputer = SimpleImputer(strategy='mean')
combined_data[numeric_columns] = numeric_imputer.fit_transform(combined_data[numeric_columns])


# Filling these columns with most_frequent values as it makes the most sense to me
columns_to_impute = ['minimum IELTS overall', 'minimum PTE overall', 'minimum TOEFL IBT overall']

# Create an instance of SimpleImputer with the strategy set to 'most_frequent'
imputer = SimpleImputer(strategy='most_frequent')

# Fit the imputer on the columns with missing values
imputer.fit(combined_data[columns_to_impute])

# Transform the missing values in the columns
combined_data[columns_to_impute] = imputer.transform(combined_data[columns_to_impute])


# Save the modified dataset
combined_data.to_csv('combined_data_filled.csv', index=False)
```

Most of the fields could be dealt with filling the mean values into the missing values places. I chose to fill minimum IELTS overall, minimum TOEFL IBT overall and minimum PTE overall with mode values as it made the most sense to me. SimpleImputer from sklearn library was used for all these and stored the results in ‘combined_data_filled.csv’.



```
: combined_data.isna().sum()

: Uni Name                           0
  State                              0
  Average Fee per semester in AUD    0
  IELTS overall                      0
  IELTS reading                      0
  IELTS listening                    0
  IELTS writing                      0
  IELTS speaking                     0
  TOEFL IBT overall                  0
  TOEFL IBT listening                0
  TOEFL IBT reading                  0
  TOEFL IBT speaking                 0
  TOEFL IBT writing                  0
  PTE overall                        0
  PTE speaking & writing             0
  PTE reading                        0
  PTE listening                      0
  minimum IELTS overall              0
  minimum TOEFL IBT overall          0
  minimum PTE overall                0
  dtype: int64

: ## ALL MISSING VALUES HAVE BEEN HANDLED
```

As evident from above, I had no missing values in any of our chosen features and was ready for the next phase of the project.

3. Model Selection

I have 20 columns (features) after the completion of preprocessing, with 18 having numeric data type and 2, namely ‘Uni Name’ and ‘State’ having string datatype. Having such mixed data types requires careful selection of the recommender system model.
Model Chosen : Matrix Factorization using TruncatedSVD (Singular Value Decomposition)

In the course recommender system, I have chosen to implement matrix factorization using TruncatedSVD as the core model. This choice was based on several factors that make it suitable for the problem at hand.

Reasons why I saw this model as the best approach:

a) **Handling Mixed Data Types**: The dataset for the course recommender system includes both numeric and categorical variables. TruncatedSVD allows us to perform dimensionality reduction on the numeric variables while also accommodating the categorical variables through one-hot encoding. This flexibility makes TruncatedSVD a suitable choice for handling mixed data types.
b) **Capturing Latent Features**: TruncatedSVD decomposes the dataset into latent factors that capture the underlying structure and relationships between the courses and student preferences. By reducing the dimensionality of the dataset, TruncatedSVD helps in identifying the most important features and patterns, enabling effective recommendations.
c) **Computational Efficiency**: TruncatedSVD performs well on large-scale datasets with sparse matrices, making it computationally efficient for our course recommendation problem. It allows us to work with a reduced number of dimensions while still preserving the essence of the original data.

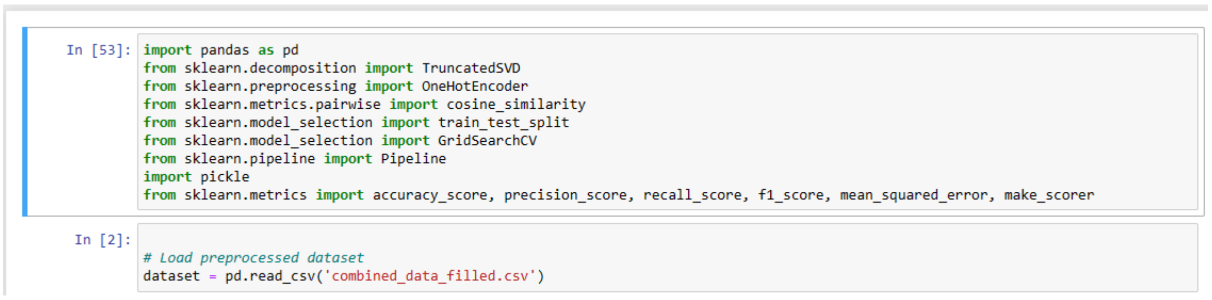
Alternative Models Considered:

a) **Collaborative Filtering**: Collaborative filtering is a widely used recommendation technique. However, it requires explicit user-item ratings or feedback, which were not available in our dataset. **Since we lacked labeled data or user feedback, collaborative filtering was not a viable option for our system.**

b) **Content-Based Filtering**: Content-based filtering recommends items based on their content attributes. While this approach can be effective, it heavily relies on accurate and comprehensive item descriptions or metadata. In **our case, the dataset did not contain detailed content attributes for courses, making content-based filtering less suitable.**

.

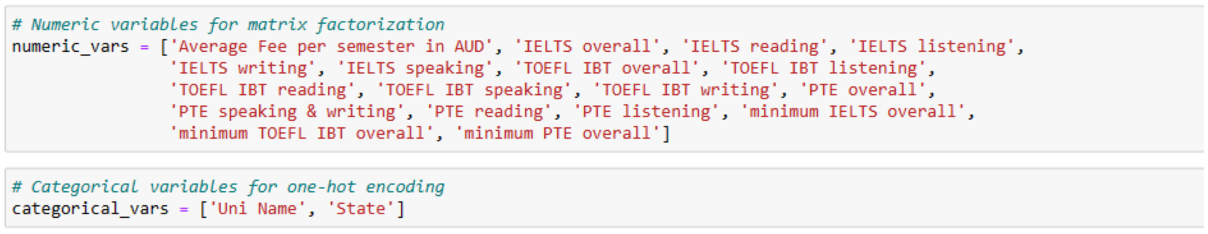
4. Model Building



```
In [53]: import pandas as pd
         from sklearn.decomposition import TruncatedSVD
         from sklearn.preprocessing import OneHotEncoder
         from sklearn.metrics.pairwise import cosine_similarity
         from sklearn.model_selection import train_test_split
         from sklearn.model_selection import GridSearchCV
         from sklearn.pipeline import Pipeline
         import pickle
         from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, mean_squared_error, make_scorer

In [2]:
         # Load preprocessed dataset
         dataset = pd.read_csv('combined_data_filled.csv')
```

The model is built out of **sklearn** library which is a great library for building recommender systems and provides a lot of crucial functions.

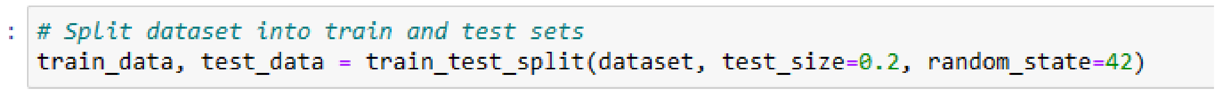
The features were divided into numerical variables and categorical variables, in order to deal with them separately as I needed some technique to deal with categorical data features in my dataset.



```
# Numeric variables for matrix factorization
numeric_vars = ['Average Fee per semester in AUD', 'IELTS overall', 'IELTS reading', 'IELTS listening',
                'IELTS writing', 'IELTS speaking', 'TOEFL IBT overall', 'TOEFL IBT listening',
                'TOEFL IBT reading', 'TOEFL IBT speaking', 'TOEFL IBT writing', 'PTE overall',
                'PTE speaking & writing', 'PTE reading', 'PTE listening', 'minimum IELTS overall',
                'minimum TOEFL IBT overall', 'minimum PTE overall']
```

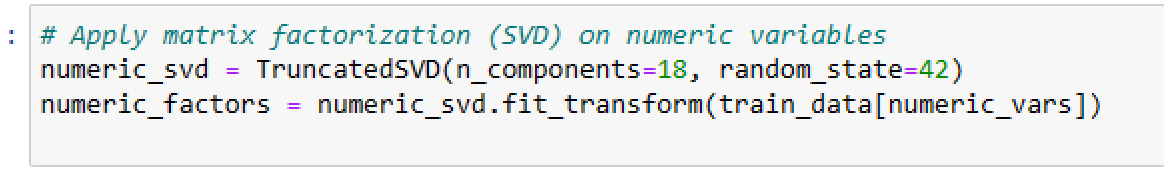
```
# Categorical variables for one-hot encoding
categorical_vars = ['Uni Name', 'State']
```

Then, necessary train-test split was performed. **(More discussion on this later)**

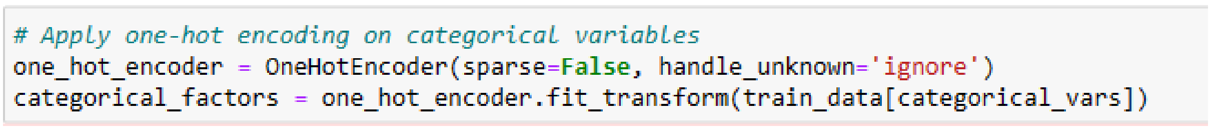


```
: # Split dataset into train and test sets
  train_data, test_data = train_test_split(dataset, test_size=0.2, random_state=42)
```

The Singular Value Decomposition (SVD) method was called with 18 components, pertaining to the 18 numerical type features that I had.



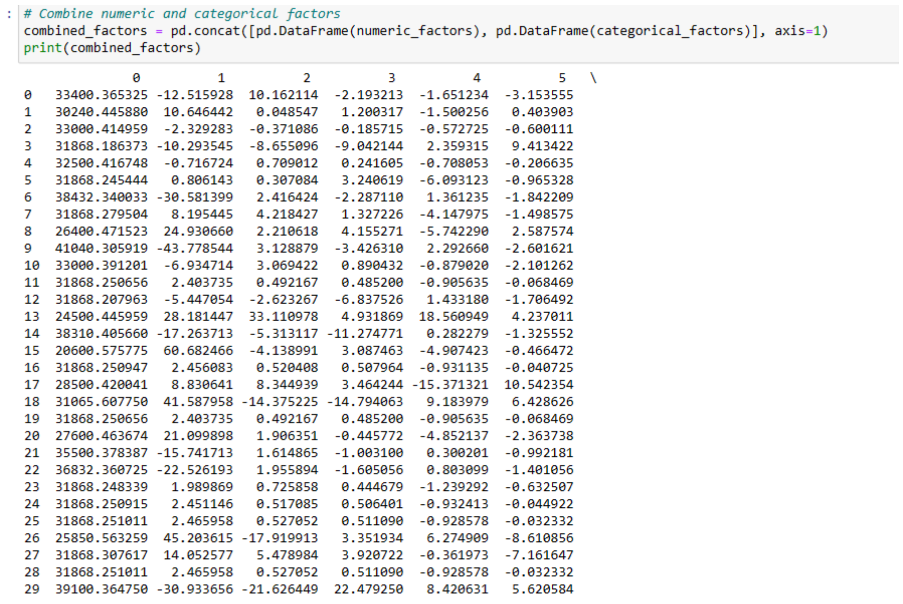
```
: # Apply matrix factorization (SVD) on numeric variables
  numeric_svd = TruncatedSVD(n_components=18, random_state=42)
  numeric_factors = numeric_svd.fit_transform(train_data[numeric_vars])
```



```
# Apply one-hot encoding on categorical variables
one_hot_encoder = OneHotEncoder(sparse=False, handle_unknown='ignore')
categorical_factors = one_hot_encoder.fit_transform(train_data[categorical_vars])
```

I decided to deal with categorical data issues using the **One Hot Encoding** technique.
This made the conversion of categorical data to numeric data with the addition of a few features. However, as the number of features were only 20, using this technique made the most sense to me.

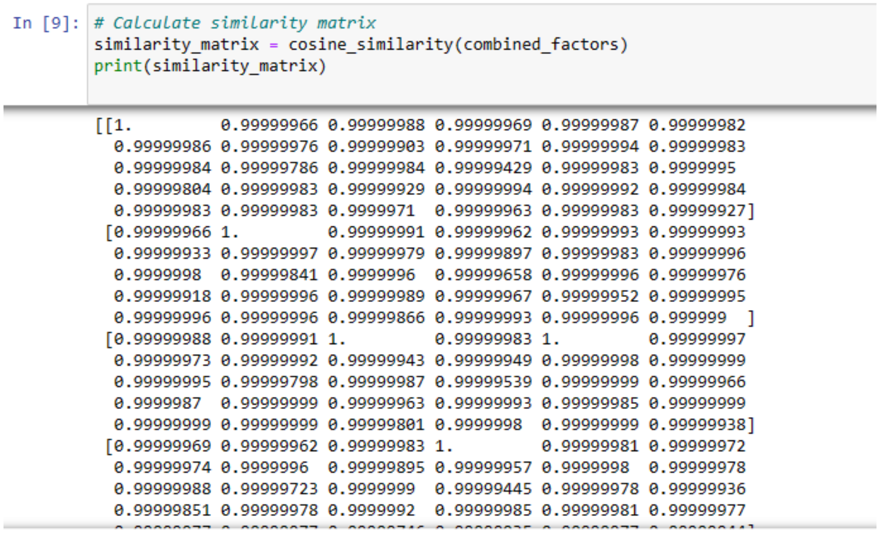
Finally, we concatenated the factors into ‘combined_factors’ which had 30 rows and 54 columns.



```
: # Combine numeric and categorical factors
  combined_factors = pd.concat([pd.DataFrame(numeric_factors), pd.DataFrame(categorical_factors)], axis=1)
  print(combined_factors)
```

```
              0          1          2          3          4          5  \
0   33400.365325 -12.515928  10.162114  -2.193213  -1.651234  -3.153555
1   30240.445880  10.646442   0.048547   1.200317  -1.500256   0.403903
2   33000.414959  -2.329283  -0.371086  -0.185715  -0.572725  -0.600111
3   31868.186373 -10.293545  -8.655096  -9.042144   2.359315   9.413422
4   32500.416748  -0.716724   0.709012   0.241605  -0.708053  -0.206635
5   31868.245444   0.806143   0.307084   3.240619  -6.093123  -0.965328
6   38432.340033 -30.581399   2.416424  -2.287110   1.361235  -1.842209
7   31868.279504   8.195445   4.218427   1.327226  -4.147975  -1.498575
8   26400.471523  24.930660   2.210618   4.155271  -5.742290   2.587574
9   41040.305919 -43.778544   3.128879  -3.426310   2.292660  -2.601621
10  33000.391201  -6.934714   3.069422   0.890432  -0.879020  -2.101262
11  31868.250656   2.403735   0.492167   0.485200  -0.905635  -0.068469
12  31868.207963  -5.447054  -2.623267  -6.837526   1.433180  -1.706492
13  24500.445959  28.181447  33.110978   4.931869  18.560949   4.237011
14  38310.405660 -17.263713  -5.313117 -11.274771   0.282279  -1.325552
15  20600.575775  60.682466  -4.138991   3.087463  -4.907423  -0.466472
16  31868.250947   2.456083   0.520408   0.507964  -0.931135  -0.040725
17  28500.420041   8.830641   8.344939   3.464244 -15.371321  10.542354
18  31065.607750  41.587958 -14.375225 -14.794063   9.183979   6.428626
19  31868.250656   2.403735   0.492167   0.485200  -0.905635  -0.068469
20  27600.463674  21.099898   1.906351  -0.445772  -4.852137  -2.363738
21  35500.378387 -15.741713   1.614865  -1.003100   0.300201  -0.992181
22  36832.360725 -22.526193   1.955894  -1.605056   0.803099  -1.401056
23  31868.248339   1.989869   0.725858   0.444679  -1.239292  -0.632507
24  31868.250915   2.451146   0.517085   0.506401  -0.932413  -0.044922
25  31868.251011   2.465958   0.527052   0.511090  -0.928578  -0.032332
26  25850.563259  45.203615 -17.919913   3.351934   6.274909  -8.610856
27  31868.307617  14.052577   5.478984   3.920722  -0.361973  -7.161647
28  31868.251011   2.465958   0.527052   0.511090  -0.928578  -0.032332
29  39100.364750 -30.933656 -21.626449  22.479250   8.420631   5.620584
```

The **cosine-similarity** technique offers the best way of measuring similarity between vector points in higher dimensional space, as in our case. Hence, this was chosen as a metric.



```
In [9]: # Calculate similarity matrix
        similarity_matrix = cosine_similarity(combined_factors)
        print(similarity_matrix)
```

```
[[1.         0.99999966 0.99999988 0.99999969 0.99999987 0.99999982
  0.99999986 0.99999976 0.99999903 0.99999971 0.99999994 0.99999983
  0.99999984 0.99999786 0.99999984 0.99999429 0.99999983 0.9999995
  0.99999804 0.99999983 0.99999929 0.99999994 0.99999992 0.99999984
  0.99999983 0.99999983 0.9999971  0.99999963 0.99999983 0.99999927]
 [0.99999966 1.         0.99999991 0.99999962 0.99999993 0.99999993
  0.99999933 0.99999997 0.99999979 0.99999897 0.99999983 0.99999996
  0.9999998  0.99999841 0.9999996  0.99999658 0.99999996 0.99999976
  0.99999918 0.99999996 0.99999989 0.99999967 0.99999952 0.99999995
  0.99999996 0.99999996 0.99999866 0.99999993 0.99999996 0.999999  ]
 [0.99999988 0.99999991 1.         0.99999983 1.         0.99999997
  0.99999973 0.99999992 0.99999943 0.99999949 0.99999998 0.99999999
  0.99999995 0.99999798 0.99999987 0.99999539 0.99999999 0.99999966
  0.9999987  0.99999999 0.99999963 0.99999993 0.99999985 0.99999999
  0.99999999 0.99999999 0.99999801 0.9999998  0.99999999 0.99999938]
 [0.99999969 0.99999962 0.99999983 1.         0.99999981 0.99999972
  0.99999974 0.9999996  0.99999895 0.99999957 0.9999998  0.99999978
  0.99999988 0.99999723 0.9999999  0.99999445 0.99999978 0.99999936
  0.99999851 0.99999978 0.9999992  0.99999985 0.99999981 0.99999977
In [45]: # Function to recommend courses for a given course ID
         def recommend_uni(uni):
             uni_index= dataset[dataset['Uni Name']== uni].index[0]
             distances= similarity_matrix[uni_index]
             uni_list = sorted(list(enumerate(distances)), reverse= True, key= lambda x: x[1])[1:6]
             c = 1
             for i in uni_list:
                 print(f"{c}) {dataset.iloc[i[0]]['Uni Name']}")
                 c += 1
```

The function recommed_uni takes a university name as input from a user, constructs a similarity matrix for it, sorts the top 5 closest vector values, then returns the name corresponding to those vectors.

Example usage:

```
print("The top 5 university recommendations are:")
print("\n")
recommended_courses = recommend_uni('The Australian National University')
```

```
The top 5 university recommendations are:


1) University of Technology Sydney
2) University of the Sunshine Coast
3) Flinders University
4) Avondale University
5) Macquarie University
```

For testing of the model, the following code, similar to for the testing data, were written.

```
In [36]: # Test the model
         test_numeric_factors = numeric_svd.transform(test_data[numeric_vars])
         test_categorical_factors = one_hot_encoder.transform(test_data[categorical_vars])
         test_combined_factors = pd.concat([pd.DataFrame(test_numeric_factors), pd.DataFrame(test_categorical_factors)], axis=1)
```

```
In [37]: # Calculate similarity matrix for test data
         test_similarity_matrix = cosine_similarity(test_combined_factors)
```

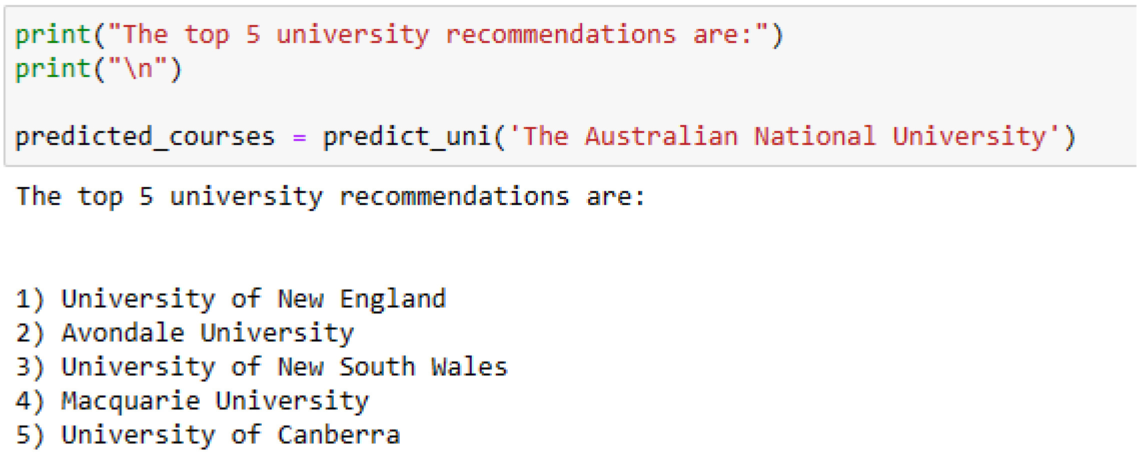
```
In [47]: # Function to recommend courses for a given course ID
         def predict_uni(uni):
             uni_index= dataset[dataset['Uni Name']== uni].index[0]
             distances= test_similarity_matrix[uni_index]
             uni_list = sorted(list(enumerate(distances)), reverse= True, key= lambda x: x[1])[1:6]

             c = 1
             for i in uni_list:
                 print(f"{c}) {dataset.iloc[i[0]]['Uni Name']}")
                 c += 1
```

5. Evaluation

As, the dataset provided to me **did not** have any **user feedback, labelled data or ground truth** to me, the expected measures of evaluation of accuracy, precision, recall, and F1-score was not possible.

To the best of my knowledge, evaluation of my recommender system's performance would require some sort of data of the above mentioned types provided.
With the provided dataset, the best I could muster up was performing a train-test split, training and testing my model with different sets of data.

The problem with this as well, was that since the model relies heavily on the similarity matrix it builds with respect to the 'combined_factors', which heavily relies on the dataset provided, following this approach would cause the model itself to be different and hence, the results would be different as well.



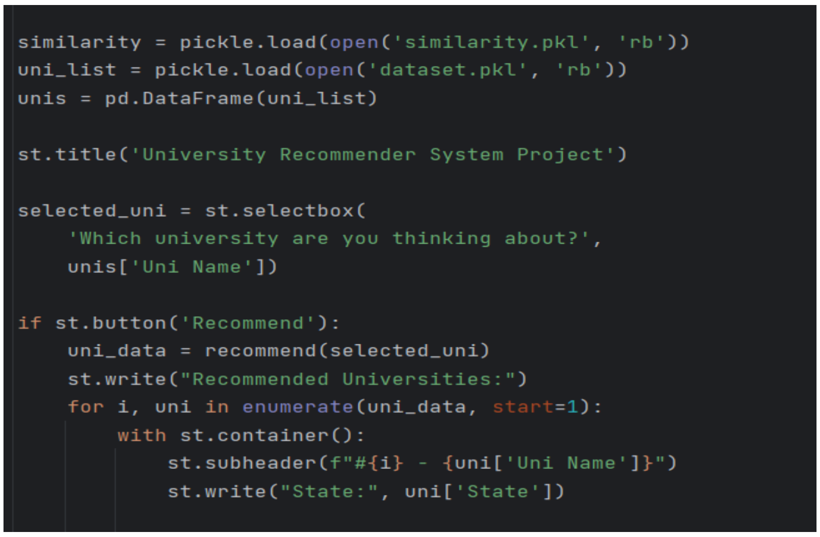
```
print("The top 5 university recommendations are:")
print("\n")

predicted_courses = predict_uni('The Australian National University')
```

```
The top 5 university recommendations are:


1) University of New England
2) Avondale University
3) University of New South Wales
4) Macquarie University
5) University of Canberra
```

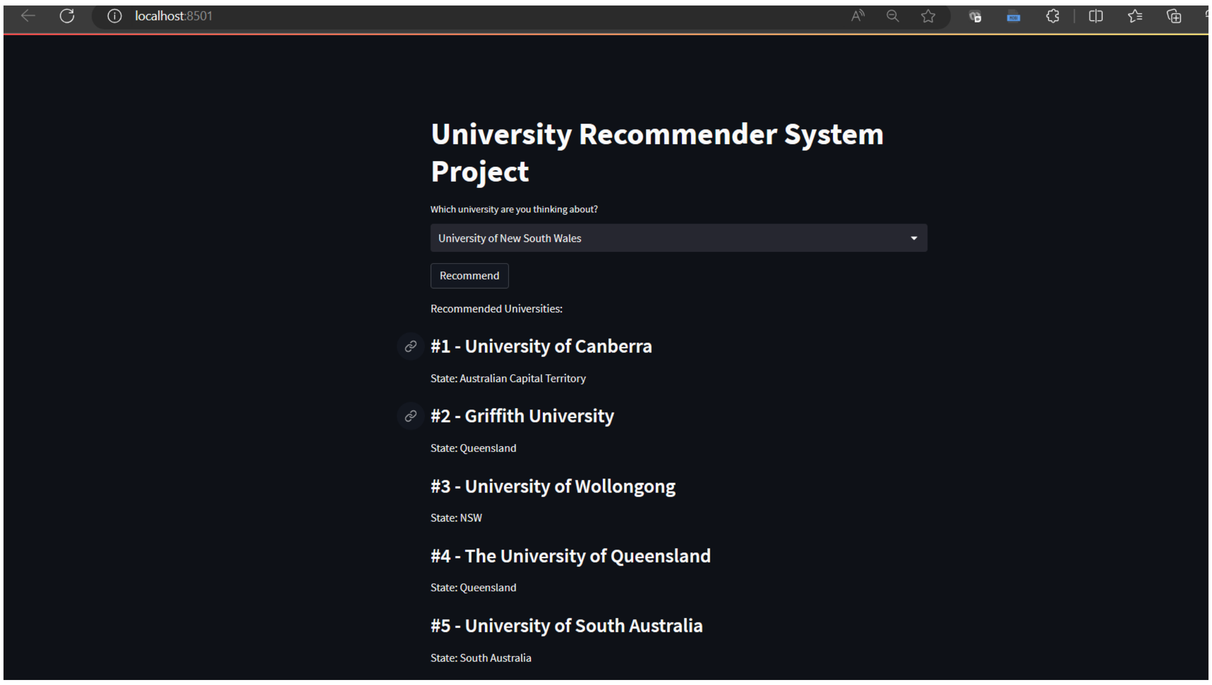
The variations in the recommendations can be observed. This served to be a major caveat for the project.

6. User-Interface

The user interface was built with **streamlit**, a library for quick and easy visualization of machine learning projects as such.



```
similarity = pickle.load(open('similarity.pkl', 'rb'))
uni_list = pickle.load(open('dataset.pkl', 'rb'))
unis = pd.DataFrame(uni_list)

st.title('University Recommender System Project')

selected_uni = st.selectbox(
    'Which university are you thinking about?',
    unis['Uni Name'])

if st.button('Recommend'):
    uni_data = recommend(selected_uni)
    st.write("Recommended Universities:")
    for i, uni in enumerate(uni_data, start=1):
        with st.container():
            st.subheader(f"#{i} - {uni['Uni Name']}")
            st.write("State:", uni['State'])
```



University Recommender System Project

Which university are you thinking about?

University of New South Wales

Recommend

Recommended Universities:

#1 - University of Canberra

State: Australian Capital Territory

#2 - Griffith University

State: Queensland

#3 - University of Wollongong

State: NSW

#4 - The University of Queensland

State: Queensland

#5 - University of South Australia

State: South Australia

7. Future Improvements

Further improvements to the project would include but would not be limited to:

1. **Incorporating Additional Features**: To enhance the course recommender system, additional relevant features can be incorporated. These could include student demographics, academic performance, course prerequisites, or student reviews.
2. **Hybrid Recommender Systems**: Combining multiple recommendation techniques, such as collaborative filtering and content-based filtering, can lead to more accurate and diverse recommendations.
3. **Incorporating User Feedback**: Collecting and incorporating user feedback can greatly enhance the recommendation system. It would also help in evaluation of the model as well.
4. **Contextual Recommendations**: Including contextual information such as time of the year, current trends in the job market, or personalised career goals can further enhance the relevance and timeliness of the recommendations.

8. Conclusion

In conclusion, the development of the course recommender system has involved various stages, including data preprocessing, model selection, and implementation of a user-friendly interface. The chosen approach of using matrix factorization with TruncatedSVD has shown promise in providing personalised course recommendations based on user preferences. However, it is important to acknowledge the limitations and challenges faced during the project. The lack of labelled data, ground truth information, and user feedback has hindered the evaluation of the model's performance. Although the system couldn't be evaluated quantitatively using metrics such as accuracy, precision, or recall, efforts were made to address data quality issues and ensure scalability. To further improve the recommender system, future extensions can involve incorporating additional features, leveraging user feedback, and exploring hybrid recommendation techniques. Overall, despite the limitations, the course recommender system presents a foundation for enhancing student experiences and facilitating informed decision-making in course selection.