Assignment

Name: Monu Yadav

Importing Required Libraries

In [47]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading Dataset

In [48]:
```
students_df = pd.read_csv(r"C:\Users\kmonu\Downloads\Zylentrix\students.csv")
course_activity_df = pd.read_csv(r"C:\Users\kmonu\Downloads\Zylentrix\course_activity.
feedback_df = pd.read_csv(r"C:\Users\kmonu\Downloads\Zylentrix\feedback.csv")
```

In [ ]:
```

```

In [49]:
```
print("Values in students.csv:\n",students_df)
```

```
Values in students.csv:
    Student_ID         Name  Age  Gender   Location Enrolment_Date
0         S001    Student_1   24  Female    Kolkata     24/11/2023
1         S002    Student_2   32   Other    Chennai     27/02/2023
2         S003    Student_3   28   Other     Mumbai     13/01/2023
3         S004    Student_4   25  Female  Bangalore     21/05/2023
4         S005    Student_5   24   Other      Delhi     06/05/2023
..         ...          ...  ...     ...        ...            ...
95        S096   Student_96   32   Other    Chennai     19/12/2023
96        S097   Student_97   31   Other    Chennai     13/10/2023
97        S098   Student_98   20   Other    Kolkata     23/04/2023
98        S099   Student_99   18    Male    Kolkata     17/12/2023
99        S100  Student_100   22   Other  Bangalore     16/06/2023

[100 rows x 6 columns]
```

In [50]:
```
print("Values in course_activity.csv:\n",course_activity_df)
```

```
Values in course_activity.csv:
     Student_ID Course_ID        Date  Time_Spent_Minutes  \
0          S001     PY202  05/01/2024                  90
1          S001     DM101  28/01/2024                 155
2          S001     UX303  28/01/2024                 177
3          S002     PY202  03/02/2024                  45
4          S002     UX303  15/03/2024                 119
..          ...       ...         ...                 ...
654        S100     PY202  03/03/2024                  83
655        S100     UX303  27/02/2024                 168
656        S100     UX303  02/01/2024                 134
657        S100     WD404  12/03/2024                 139
658        S100     WD404  28/03/2024                 135

     Completion_Percentage
0                    46.10
1                    88.87
2                    54.93
3                    32.20
4                    90.80
..                     ...
654                  22.63
655                  14.83
656                  22.29
657                  68.78
658                  47.41

[659 rows x 5 columns]
```

In [51]:
```
print("Values in feedback.csv:\n",feedback_df)
```

```
Values in feedback.csv:
    Student_ID Course_ID  Rating         Feedback_Text
0        S057     UX303       2        Too fast-paced
1        S063     PY202       2    Loved the examples
2        S022     PY202       4       Could be better
3        S011     PY202       5     Needs improvement
4        S073     WD404       4       Could be better
..        ...       ...     ...                   ...
75       S087     DM101       1       Well structured
76       S065     DM101       4    Loved the examples
77       S082     WD404       2        Too fast-paced
78       S067     DM101       5  Excellent instructor
79       S002     PY202       3       Well structured

[80 rows x 4 columns]
```

Data Inspection - Missing Values

In this section, we will check for missing values in the datasets: `students.csv`, `course_activity.csv`, and `feedback.csv`.

1. Checking for Missing Values in `students.csv`

In [52]:
```
print("Missing values in students.csv:\n", students_df.isnull().sum())
```

```
Missing values in students.csv:
 Student_ID        0
Name               0
Age                0
Gender             0
Location           0
Enrolment_Date     0
dtype: int64
```

1. Checking for Missing Values in `course_activity.csv`

In [53]:
```
print("\nMissing values in course_activity.csv:\n", course_activity_df.isnull().sum())
```

```
Missing values in course_activity.csv:
 Student_ID               0
Course_ID                 0
Date                      0
Time_Spent_Minutes        0
Completion_Percentage     0
dtype: int64
```

1. Checking for Missing Values in `feedback.csv`

In [54]:
```
print("\nMissing values in feedback.csv:\n", feedback_df.isnull().sum())
```

```
Missing values in feedback.csv:
 Student_ID       0
Course_ID         0
Rating            0
Feedback_Text     0
dtype: int64
```

Data Inspection - Duplicate Rows

In this section, we will check for duplicate rows in the datasets: `students.csv`, `course_activity.csv`, and `feedback.csv`.

1. Checking for Duplicate Rows in `students.csv`

In [55]:
```
print("\nDuplicate rows in students.csv:", students_df.duplicated().sum())
```

```
Duplicate rows in students.csv: 0
```

2. Checking for Duplicate Rows in `course_activity.csv`

In [56]:
```
print("Duplicate rows in course_activity.csv:", course_activity_df.duplicated().sum())
```

```
Duplicate rows in course_activity.csv: 0
```

3. Checking for Duplicate Rows in `feedback.csv`

In [57]:
```
print("Duplicate rows in feedback.csv:", feedback_df.duplicated().sum())
```

```
Duplicate rows in feedback.csv: 0
```

Data Conversion - Date Columns

In this section, we will convert the 'Enrolment_Date' column in `students.csv` and the 'Date' column in `course_activity.csv` to datetime format.

1. Converting 'Enrolment_Date' in `students.csv` to Datetime

In [58]:
```
students_df['Enrolment_Date'] = pd.to_datetime(students_df['Enrolment_Date'], format='
```

2. Converting 'Date' in course_activity.csv to Datetime

In [59]:
```
course_activity_df['Date'] = pd.to_datetime(course_activity_df['Date'], format='%d/%m/
```

Data Inspection - Data Types

In this section, we will display the data types of the columns in the cleaned datasets: `students.csv`, `course_activity.csv`, and `feedback.csv`.

1. Displaying Data Types in `students.csv`

In [60]:
```
print("\nCleaned students.csv Data Types:\n", students_df.dtypes)
```

```
Cleaned students.csv Data Types:
 Student_ID                object
Name                      object
Age                        int64
Gender                    object
Location                  object
Enrolment_Date    datetime64[ns]
dtype: object
```

2. Displaying Data Types in course_activity.csv

In [61]:
```
print("\nCleaned course_activity.csv Data Types:\n", course_activity_df.dtypes)
```

```
Cleaned course_activity.csv Data Types:
 Student_ID                    object
Course_ID                     object
Date                  datetime64[ns]
Time_Spent_Minutes             int64
Completion_Percentage        float64
dtype: object
```

3. Displaying Data Types in feedback.csv

In [62]:
```
print("\nfeedback.csv Data Types:\n", feedback_df.dtypes)
```

```
feedback.csv Data Types:
 Student_ID       object
Course_ID        object
Rating            int64
Feedback_Text    object
dtype: object
```

Exploratory Data Analysis (EDA)

Q1. What is the overall average completion rate across courses?

In this section, we calculate the overall average completion rate across all courses from the `course_activity.csv` dataset.

Calculation of Overall Average Completion Rate

In [63]:
```
overall_completion_rate = course_activity_df['Completion_Percentage'].mean()
print("Overall Average Completion Rate: {:.2f}%".format(overall_completion_rate))
```

```
Overall Average Completion Rate: 54.78%
```

Q2. Which course has the highest and lowest average engagement time?

In this section, we calculate the courses with the highest and lowest average engagement time based on the `Time_Spent_Minutes` column in the `course_activity.csv` dataset.

Highest and lowest average engagement time

In [64]:
```
engagement_by_course = course_activity_df.groupby('Course_ID')['Time_Spent_Minutes'].m
highest_engagement_course = engagement_by_course.idxmax(), engagement_by_course.max()
lowest_engagement_course = engagement_by_course.idxmin(), engagement_by_course.min()

print("Highest Engagement Course:", highest_engagement_course)
print("Lowest Engagement Course:", lowest_engagement_course)
```

```
Highest Engagement Course: ('DM101', 102.42767295597484)
Lowest Engagement Course: ('PY202', 93.90243902439025)
```

Q3. How does engagement differ by age group or location?

In this section, we first convert the `Age` column to numeric format, and then categorize the students into different age groups based on their age. We use this information to analyze engagement by age group.

1. Converting `Age` to Numeric

We convert the `Age` column in `students_df` to a numeric format. If there are any non-numeric values, they will be coerced into `NaN`.

In [65]:
```
students_df['Age'] = pd.to_numeric(students_df['Age'], errors='coerce')
```

2. Categorizing Students by Age Group

We define the function categorize_age, which categorizes students into different age groups based on their age. Then, we apply this function to create a new column 'Age_Group'.

In [66]:
```
def categorize_age(age):
    if age < 18:
        return '<18'
    elif 18 <= age <= 25:
        return '18-25'
    elif 26 <= age <= 35:
        return '26-35'
    elif 36 <= age <= 50:
        return '36-50'
    else:
        return '50+'

students_df['Age_Group'] = students_df['Age'].apply(categorize_age)
```

Merge Course Activity Data with Student Data

We merge the students_df dataset with the course_activity_df dataset based on Student_ID to have both student and course activity data together.

In [67]:
```
merged_df = course_activity_df.merge(students_df, on='Student_ID')
```

Engagement by Age Group

We calculate the average time spent (Time_Spent_Minutes) for each age group.

In [68]:
```
engagement_by_age = merged_df.groupby('Age_Group')['Time_Spent_Minutes'].mean()
print(" Engagement by Age Group:\n", engagement_by_age)
```

```
 Engagement by Age Group:
 Age_Group
18-25    100.758929
26-35     95.362229
Name: Time_Spent_Minutes, dtype: float64
```

Engagement by Location:

We will calculate the average engagement time for each location and display the top locations by engagement.

In [69]:
```
engagement_by_location = merged_df.groupby('Location')['Time_Spent_Minutes'].mean()
print("\n Engagement by Location:\n", engagement_by_location)
```

```
 Engagement by Location:
 Location
Bangalore     98.000000
Chennai       90.946746
Delhi        103.302857
Kolkata      104.384615
Mumbai        95.435484
Name: Time_Spent_Minutes, dtype: float64
```

Q4. What is the average feedback rating per course?

Calculate Average Feedback Rating per Course

We group the `feedback_df` dataset by `Course_ID` and calculate the mean of the `Rating` column for each course. The results are then sorted in descending order to display the highest-rated courses first.

In [70]:
```
avg_rating_per_course = feedback_df.groupby('Course_ID')['Rating'].mean().sort_values(
print("Average Feedback Rating per Course:\n", avg_rating_per_course)
```

```
Average Feedback Rating per Course:
 Course_ID
PY202    3.277778
UX303    2.923077
DM101    2.900000
WD404    2.789474
Name: Rating, dtype: float64
```

Q5. Is there a correlation between completion rate and feedback rating?

- Positive: Indicates better ratings as students complete more.
- Negative: Indicates dissatisfaction with completed content. #### Merge Course Activity Data with Student Data

In [71]:
```
merged_feedback = course_activity_df.merge(feedback_df, on=['Student_ID','Course_ID'])
```

In [72]:
```
correlation = merged_feedback[['Completion_Percentage', 'Rating']].corr().loc['Complet
print("Correlation between Completion % and Rating: {:.3f}".format(correlation))
```

```
Correlation between Completion % and Rating: -0.052
```

Q6. Identify top 3 student segments based on engagement and satisfaction

In this section, we identify the top 3 student segments based on their engagement (measured by `Time_Spent_Minutes`) and satisfaction (measured by `Rating`). We will calculate the average engagement and feedback rating for each student and merge this information with the student details.

Calculate Average Engagement and Satisfaction for Each Student

We group the merged feedback data by `Student_ID` and calculate the average `Time_Spent_Minutes` and `Rating` for each student.

In [73]:
```
student_scores = merged_feedback.groupby('Student_ID').agg({
    'Time_Spent_Minutes': 'mean',
    'Rating': 'mean'
}).reset_index()
print(student_scores)
```

```
   Student_ID  Time_Spent_Minutes  Rating
0        S002                90.5     3.0
1        S005                27.0     4.0
2        S006               119.0     1.0
3        S009                99.0     4.0
4        S011                30.5     5.0
..        ...                 ...     ...
58       S093                73.0     1.0
59       S094                60.5     4.0
60       S095               125.0     3.0
61       S097                87.0     5.0
62       S099               109.5     5.0

[63 rows x 3 columns]
```

Merge with Student Details

Next, we merge the calculated student scores with the student details (students_df) based on Student_ID.

In [74]:
```
student_segments = student_scores.merge(students_df, on='Student_ID')
```

Sorting and Identifying the Top 3 Student Segments

We sort the student segments by both Time_Spent_Minutes and Rating in descending order to identify the top 3 students who are the most engaged and satisfied.

In [75]:
```
top_segments = student_segments.sort_values(by=['Time_Spent_Minutes', 'Rating'], ascer
print("Top 3 Student Segments:\n", top_segments[['Student_ID', 'Name', 'Time_Spent_Mir
```

```
Top 3 Student Segments:
    Student_ID        Name  Time_Spent_Minutes  Rating  Age   Location
17       S036  Student_36               175.0     5.0   25    Chennai
49       S081  Student_81               174.0     4.0   24  Bangalore
39       S064  Student_64               173.0     4.0   30  Bangalore
```

Visualisations

In this section, we merge the datasets (`course_activity_df` , `students_df` , and `feedback_df`) for further analysis and visualization.

Merging the Datasets for Analysis

We first merge the `course_activity_df` with the `students_df` based on `Student_ID`, and then merge the resulting dataset with the `feedback_df` based on both `Student_ID` and `Course_ID`.

In [76]:
```
merged_df = course_activity_df.merge(students_df, on='Student_ID')
merged_df = merged_df.merge(feedback_df, on=['Student_ID', 'Course_ID'])
print(merged_df)
```

```
    Student_ID Course_ID       Date  Time_Spent_Minutes  \
0         S002     PY202 2024-02-03                  45
1         S002     PY202 2024-03-06                 136
2         S005     DM101 2024-01-30                  27
3         S006     DM101 2024-03-27                 177
4         S006     DM101 2024-01-08                 113
..         ...       ...        ...                 ...
117       S095     DM101 2024-02-26                 125
118       S097     PY202 2024-01-27                  81
119       S097     PY202 2024-02-16                  93
120       S099     WD404 2024-02-23                 162
121       S099     WD404 2024-02-24                  57

     Completion_Percentage        Name  Age Gender Location Enrolment_Date  \
0                    32.20   Student_2   32  Other  Chennai     2023-02-27
1                    18.18   Student_2   32  Other  Chennai     2023-02-27
2                    91.49   Student_5   24  Other    Delhi     2023-05-06
3                    98.57   Student_6   28  Other    Delhi     2023-04-25
4                    75.54   Student_6   28  Other    Delhi     2023-04-25
..                     ...         ...  ...    ...      ...            ...
117                  38.70  Student_95   22  Other    Delhi     2023-11-24
118                  77.71  Student_97   31  Other  Chennai     2023-10-13
119                  27.74  Student_97   31  Other  Chennai     2023-10-13
120                  22.41  Student_99   18   Male  Kolkata     2023-12-17
121                  53.80  Student_99   18   Male  Kolkata     2023-12-17

    Age_Group  Rating       Feedback_Text
0       26-35       3     Well structured
1       26-35       3     Well structured
2       18-25       4  Loved the examples
3       26-35       1  Loved the examples
4       26-35       1  Loved the examples
..        ...     ...                 ...
117     18-25       3      Too fast-paced
118     26-35       5  Loved the examples
119     26-35       5  Loved the examples
120     18-25       5   Needs improvement
121     18-25       5   Needs improvement

[122 rows x 13 columns]
```

1. Bar Chart: Average Time Spent by Course

In this section, we create a bar chart to visualize the average time spent by students in each course.

Compute Average Time Spent by Course

We first calculate the average time spent by students in each course by grouping the `merged_df` dataset by `Course_ID` and computing the mean of `Time_Spent_Minutes`.

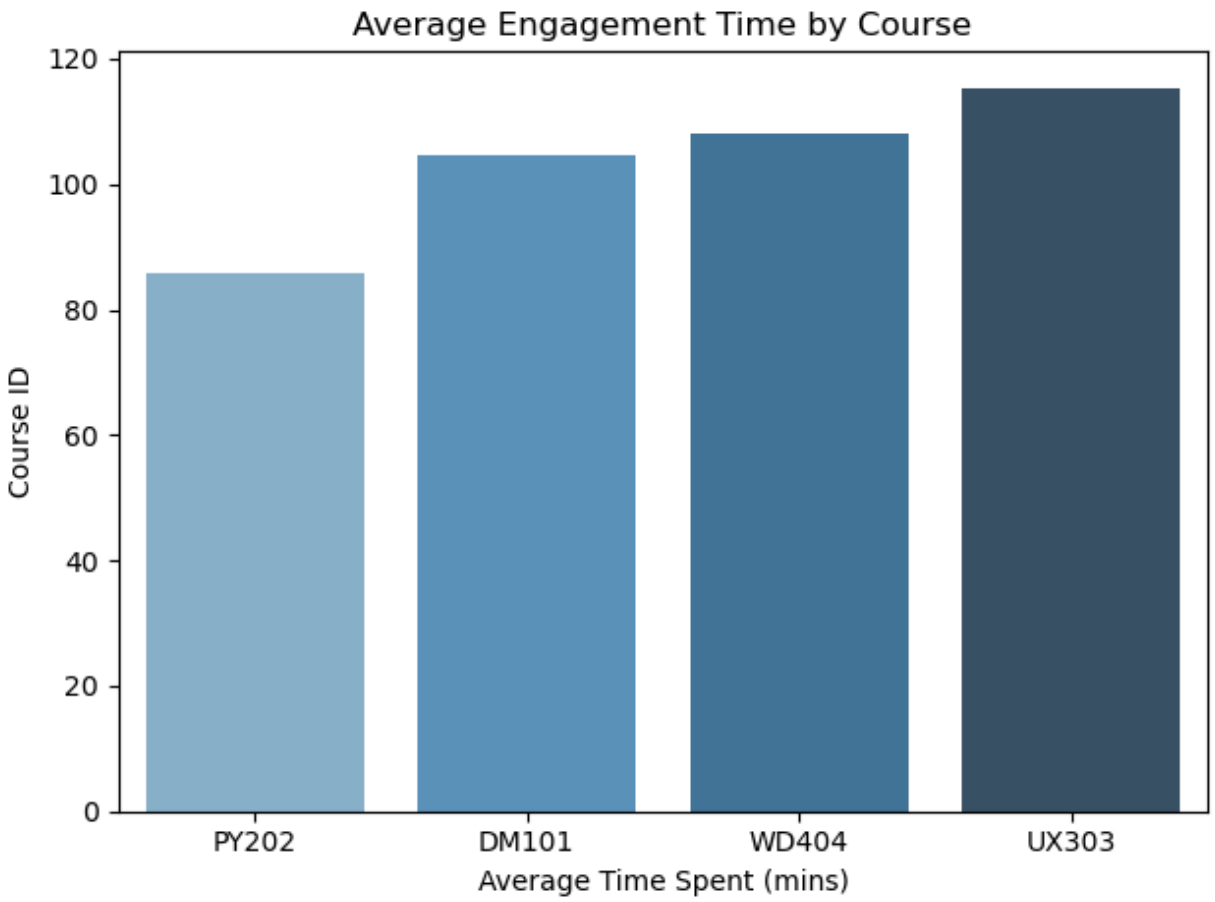
In [77]:
```
plt.figure(figsize=(10, 6))
avg_time_course = merged_df.groupby('Course_ID')['Time_Spent_Minutes'].mean().sort_val
print(avg_time_course)
```

```
Course_ID
PY202     85.947368
DM101    104.636364
WD404    108.130435
UX303    115.352941
Name: Time_Spent_Minutes, dtype: float64
<Figure size 1000x600 with 0 Axes>
```

Create a Vertical Bar Plot

We then create a vertical bar chart using seaborn to visualize the average time spent by course.

In [78]:
```
sns.barplot(x=avg_time_course.index,y=avg_time_course.values, palette='Blues_d')
plt.title('Average Engagement Time by Course')
plt.xlabel('Average Time Spent (mins)')
plt.ylabel('Course ID')
plt.tight_layout()
plt.show()
```



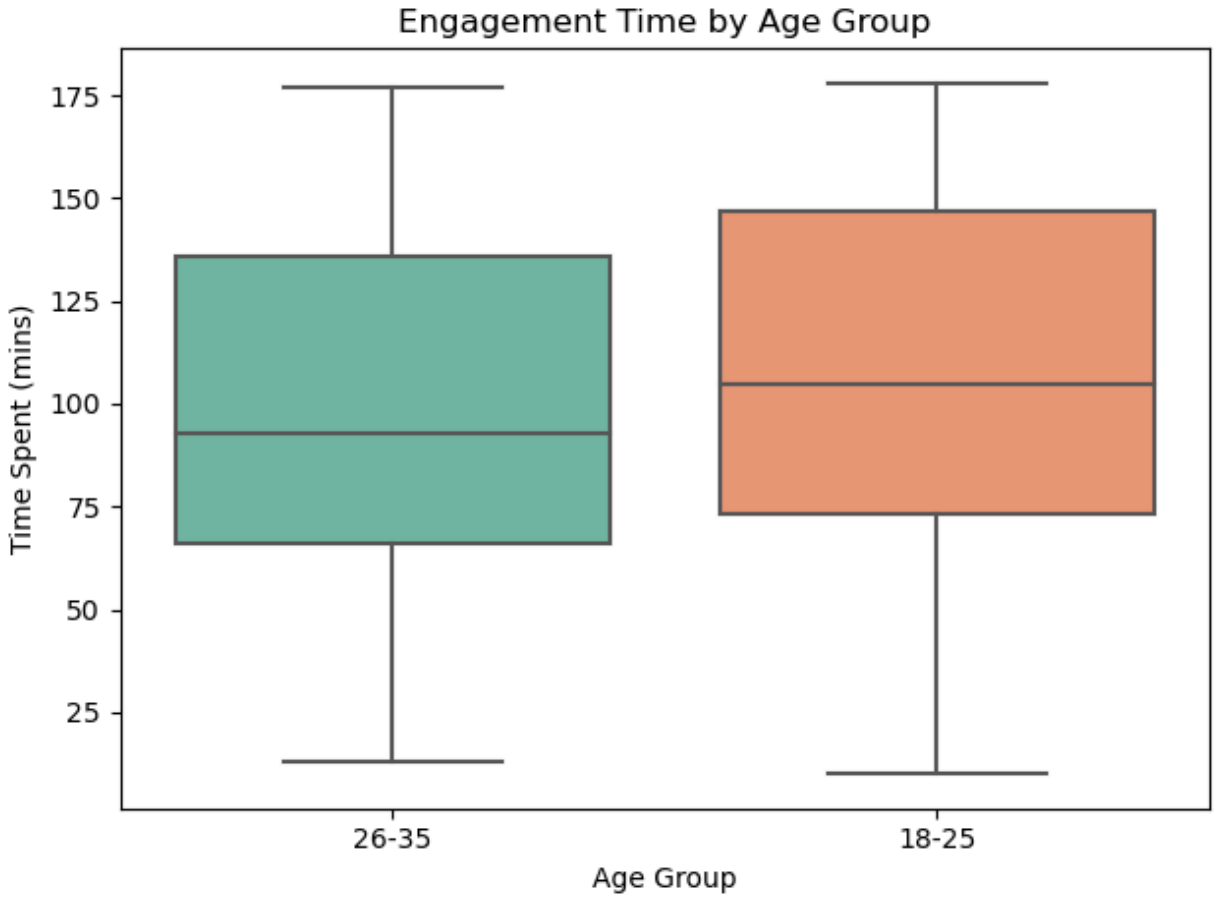
2. Box Plot: Time Spent by Age Group

In this section, we create a box plot to visualize the distribution of time spent by students in different age groups.

Creating the Box Plot

We use `seaborn` to create a box plot that shows the distribution of `Time_Spent_Minutes` for each `Age` group.

In [79]:
```
plt.figure()
sns.boxplot(data=merged_df, x='Age_Group', y='Time_Spent_Minutes', palette='Set2')
plt.title('Engagement Time by Age Group')
plt.xlabel('Age Group')
plt.ylabel('Time Spent (mins)')
plt.tight_layout()
plt.show()
```



In [80]:
```
print(merged_df['Age_Group'])
```

```
0      26-35
1      26-35
2      18-25
3      26-35
4      26-35
       ...
117    18-25
118    26-35
119    26-35
120    18-25
121    18-25
Name: Age_Group, Length: 122, dtype: object
```

3. Bar Chart: Average Feedback Rating by Course

In this section, we create a bar chart to visualize the average feedback rating for each course.

Compute Average Feedback Rating by Course

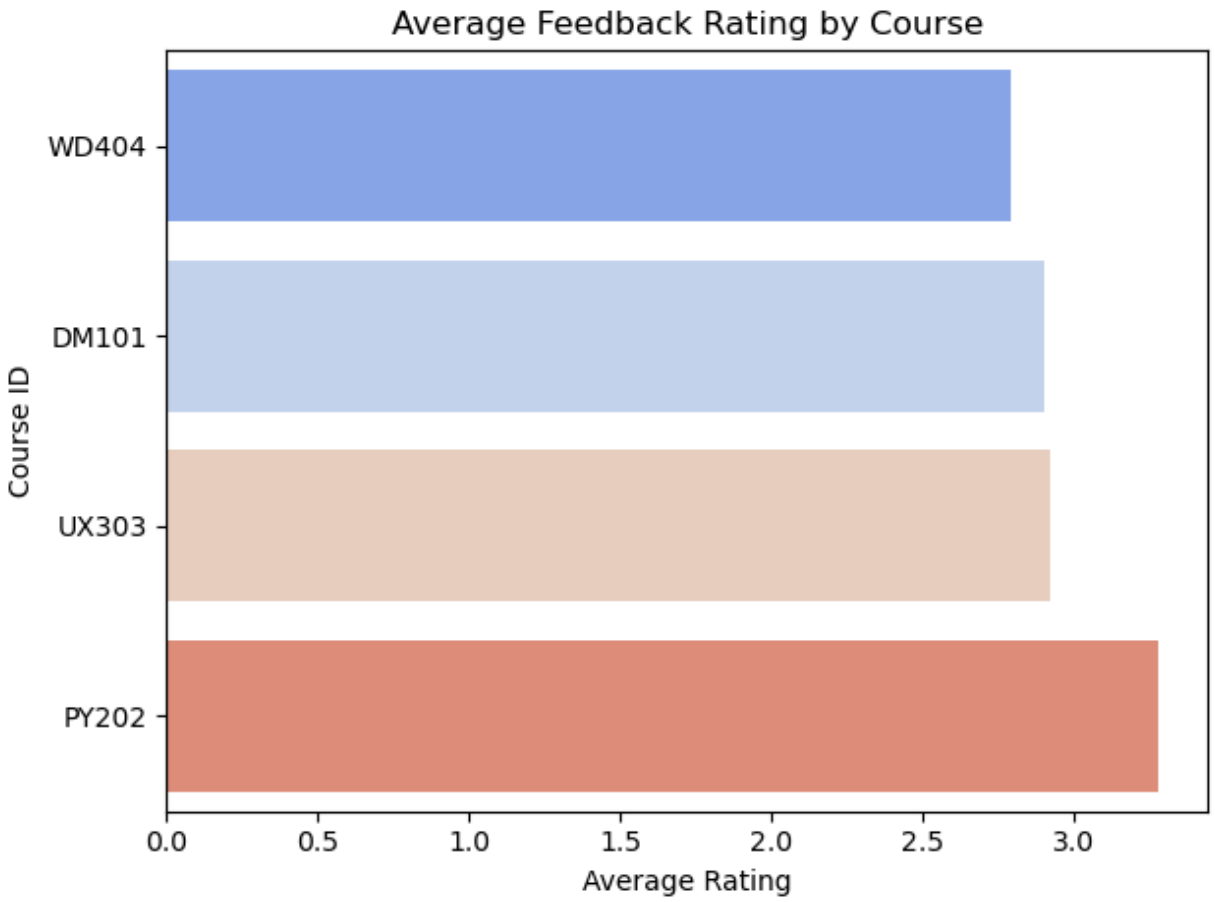
We calculate the average feedback rating for each course by grouping the `feedback_df` dataset by `Course_ID` and computing the mean of the `Rating`.

In [81]:
```
avg_rating = feedback_df.groupby('Course_ID')['Rating'].mean().sort_values()
```

Create a Bar Plot for Average Feedback Rating

We create a horizontal bar chart using seaborn to visualize the average feedback rating by course.

In [82]:
```
plt.figure()
avg_rating = feedback_df.groupby('Course_ID')['Rating'].mean().sort_values()
sns.barplot(x=avg_rating.values, y=avg_rating.index, palette='coolwarm')
plt.title('Average Feedback Rating by Course')
plt.xlabel('Average Rating')
plt.ylabel('Course ID')
plt.tight_layout()
plt.show()
```



4. Scatter Plot: Completion % vs Feedback Rating

In this section, we create a scatter plot to visualize the relationship between completion percentage (`Completion_Percentage`) and feedback rating (`Rating`). The data points are colored based on the age group.

Create the Scatter Plot

We use `seaborn` to create a scatter plot with `Completion_Percentage` on the x-axis and `Rating` on the y-axis. The data points are colored by `Age Group` to provide additional insights into how the relationship varies across different age groups.

In [83]:
```
plt.figure()
sns.scatterplot(data=merged_df, x='Completion_Percentage', y='Rating', hue='Age', pale
plt.title('Completion Percentage vs Feedback Rating')
plt.xlabel('Completion Percentage')
plt.ylabel('Rating')
plt.legend(title='Age Group')
plt.tight_layout()
plt.show()
```

5. Line Plot: Average Engagement Over Time

In this section, we create a line plot to visualize how the average engagement time (`Time_Spent_Minutes`) changes over time. We calculate the daily average engagement and plot it against the `Date` .

Calculate Daily Average Engagement Time

We first group the data by `Date` and calculate the mean of `Time_Spent_Minutes` for each day.

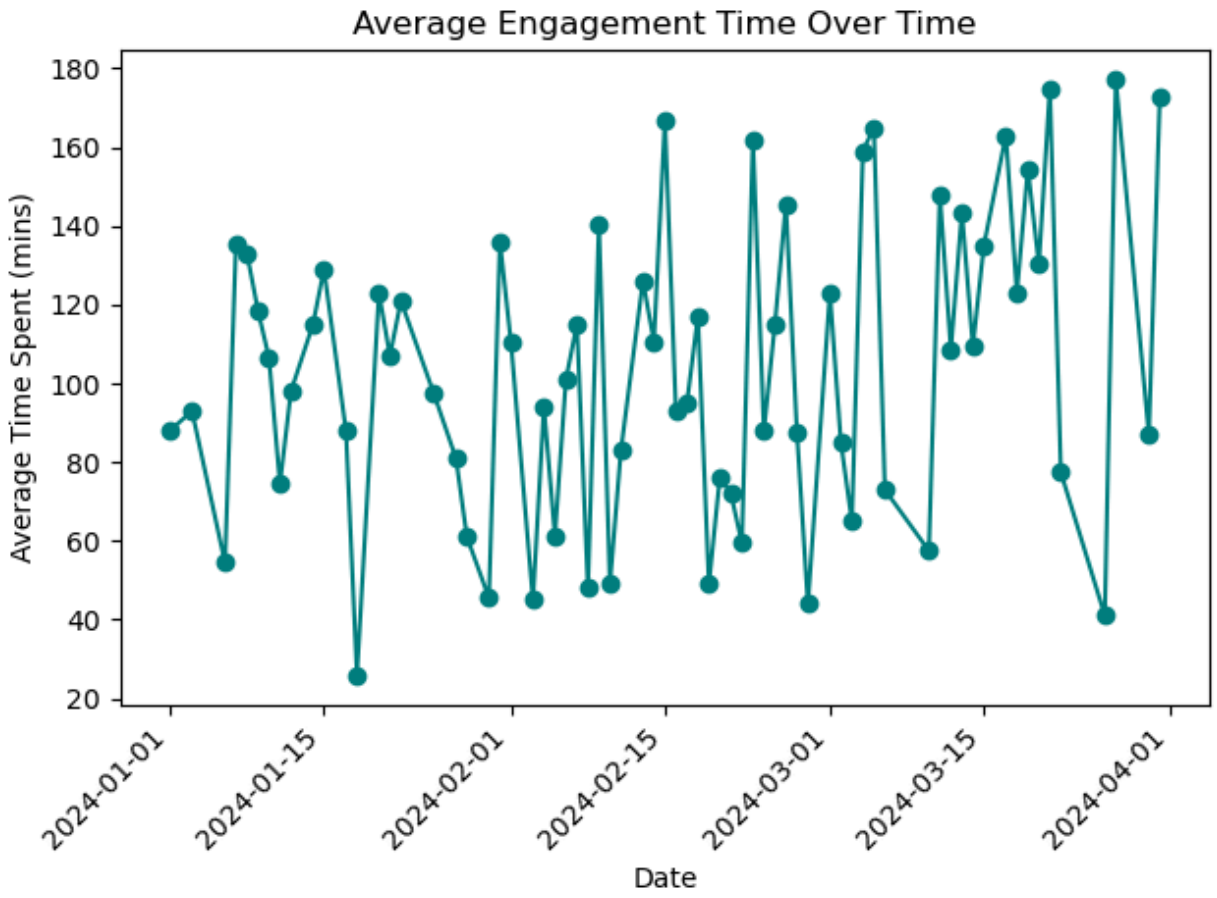
In [84]:
```
daily_engagement = merged_df.groupby('Date')['Time_Spent_Minutes'].mean()
```

Create the Line Plot

We then create a line plot to visualize the trend of average engagement time over time. The plot uses markers for each data point and rotates the x-axis labels for better readability.

In [85]:
```
plt.figure()
daily_engagement.plot(marker='o', color='teal')
plt.title('Average Engagement Time Over Time')
plt.xlabel('Date')
plt.ylabel('Average Time Spent (mins)')
plt.xticks(rotation=45)
```

```
plt.tight_layout()
plt.show()
```



Heatmap of Correlation Between Engagement Metrics

In this section, we create a heatmap to visualize the correlation between engagement metrics: `Time_Spent_Minutes`, `Completion_Percentage`, and `Rating`. The heatmap helps identify the relationships between these metrics.

Select Relevant Numerical Columns for the Heatmap

We first select the relevant numerical columns from the `merged_feedback` dataset: `Time_Spent_Minutes`, `Completion_Percentage`, and `Rating`.

Compute the Correlation Matrix

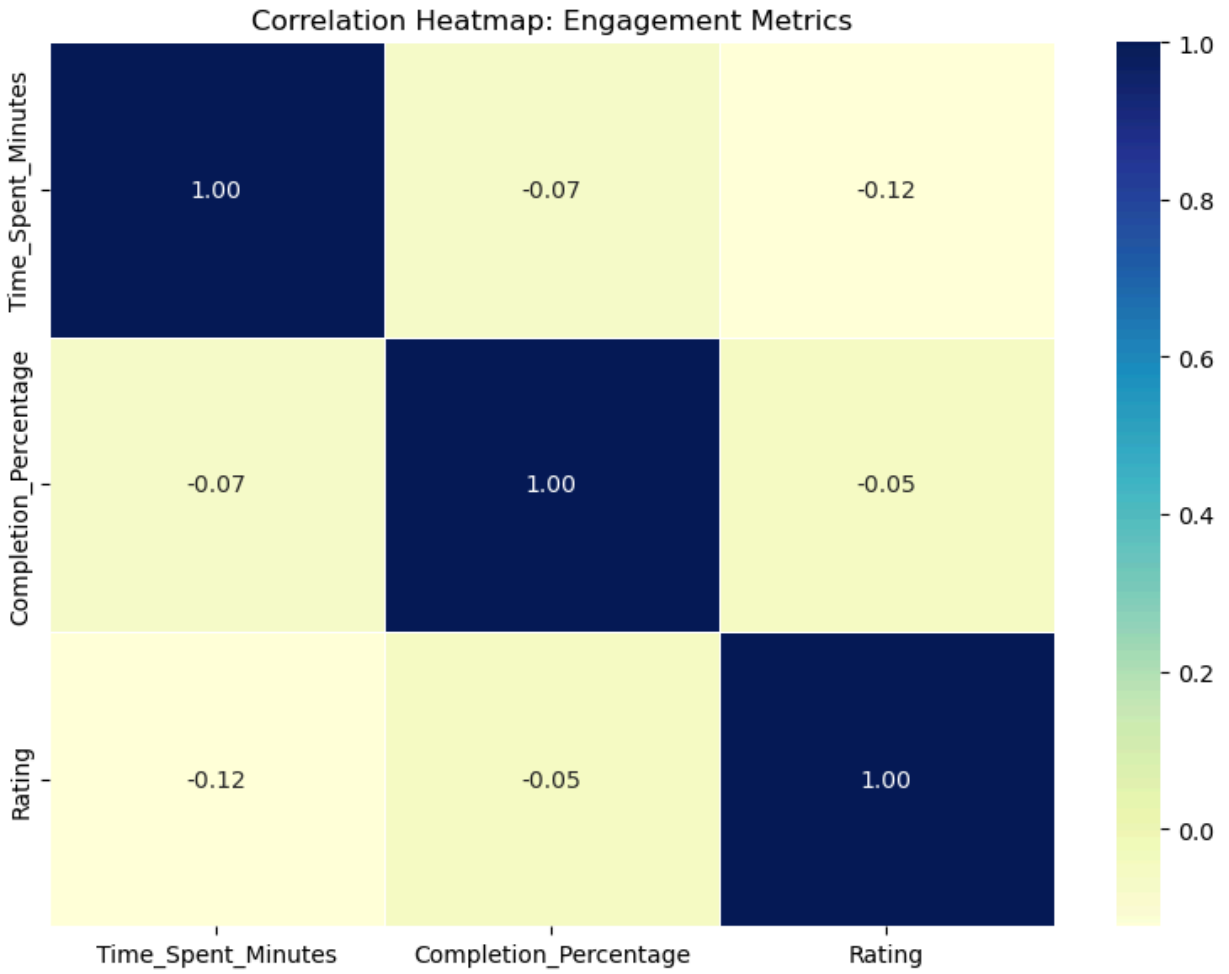
Next, we calculate the correlation matrix for these selected columns using the .corr() method.

In [86]:
```
engagement_df = merged_df[['Time_Spent_Minutes', 'Completion_Percentage', 'Rating']]
corr_matrix = engagement_df.corr()
```

Plot the Heatmap

Finally, we plot the correlation matrix using seaborn's heatmap function, with annotations and a color palette for better visualization.

In [87]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='YlGnBu', linewidths=0.5, fmt=".2f")
plt.title('Correlation Heatmap: Engagement Metrics')
plt.tight_layout()
plt.show()
```



2. Heatmap: Engagement Patterns by Course and Location (Demographic)

In this section, we create a heatmap to visualize the engagement patterns by course and location. We calculate the average time spent (`Time_Spent_Minutes`) for each course-location combination, and then display it using a heatmap.

Group Data by Course and Location to Find Average Time Spent

We group the data by `Course_ID` and `Location` , calculating the average time spent (`Time_Spent_Minutes`) for each combination. Then, we use `.unstack()` to reshape the data for better plotting.

Plot the Heatmap

We plot the heatmap using seaborn, with the reshaped data, color-coded by the average engagement time. The heatmap will show the engagement patterns across different courses and locations.

In [88]:
```
engagement_by_course_location = merged_df.groupby(['Course_ID', 'Location']).agg(
    avg_time_spent=('Time_Spent_Minutes', 'mean')
).unstack().reset_index()
```

Plot the Heatmap

We plot the heatmap using seaborn, with the reshaped data, color-coded by the average engagement time. The heatmap will show the engagement patterns across different courses and locations.

In [89]:
```
plt.figure(figsize=(12, 8))
sns.heatmap(engagement_by_course_location.drop('Course_ID', axis=1).transpose(), annot
plt.title('Engagement Patterns by Course and Location')
plt.xlabel('Location')
plt.ylabel('Course')
plt.show()
```

```
C:\Users\kmonu\AppData\Local\Temp\ipykernel_5676\39075386.py:2: PerformanceWarning: d
ropping on a non-lexsorted multi-index without a level parameter may impact performan
ce.
  sns.heatmap(engagement_by_course_location.drop('Course_ID', axis=1).transpose(), an
not=True, cmap='coolwarm', fmt=".1f")
```