Project Name - **Student Grades Prediction**

Name - Aman Mulla.

Batch - DS2307

Project Summary -

This project revolves around a comprehensive dataset encompassing students' grades across various university courses and their resultant Cumulative Grade Point Average (CGPA) over their four-year tenure. The dataset comprises 43 columns, notably featuring Seat Numbers identifying individual candidates and the CGPA, representing their overall academic performance.

Each column, except Seat No and CGPA, corresponds to course codes following a specific format denoting the department and year when the candidate took the exam. The objective here is to predict a student's CGPA based on their performance across these diverse courses throughout their academic journey.The predictive focus entails leveraging machine learning models to comprehend the relationship between the grades obtained in different courses over the four-year period and the resultant CGPA. By employing statistical analysis and predictive modeling techniques, this project aims to establish a predictive framework capable of estimating a student's CGPA based on their grades across multiple courses.

We have total 43 columns in which 1st one is Seat No. and last one is CGPA based on the four year total grade progress of each candidate.

All other columns are course codes in the format AB-XXX where AB are alphabets representing candidates' departments and XXX are numbers where first X represents the year the canditate took exam.

Below tables shows Year wise cource Code, in 1st, 2nd and 3rd year there was total 11 subject while for 4th year there was 8 subjects.

Year▼	Subjects										
4th Year Subject	CS-403	CS-421	CS-406	CS-414	CS-419	CS-423	CS-412	MT-442			
3rd Year Subject	MT-331	EF-303	HS-304	CS-301	CS-302	TC-383	EL-332	CS-318	CS-306	CS-312	CS-317
2nd Year Subject	HS-205/20	MT-222	EE-222	MT-224	CS-210	CS-211	CS-203	CS-214	EE-217	CS-212	CS-215
1st Year Subject	PH-121	HS-101	CY-105	HS-105/12	MT-111	CS-105	CS-106	EL-102	EE-119	ME-107	CS-107

Problem Statement

Develop a predictive model to accurately forecast a student's Cumulative Grade Point Average (CGPA) based on their cource grades of a four-year tenure. The challenge involves leveraging a dataset containing 43 columns, including individual Seat Numbers and course-specific grades denoted by unique codes reflecting departmental affiliation and examination year.

The objective is to construct a machine learning model capable of understanding the intricate relationship between the grades obtained in diverse courses and the resultant CGPA. The model should effectively analyze this multi-dimensional dataset, employing statistical analysis, feature engineering, and predictive modeling techniques to derive insights into the factors influencing a student's overall academic performance.

Knowing data and variable in dataset

```
# Importing Necessary Libraries.

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

pd.set_option('display.max_rows', None)
```

```
# loading dataset

grades_data = pd.read_csv('/content/drive/MyDrive/DataSets/Grades.csv')

grades_data.head()
```

	Seat No.	PH-121	HS-101	CY-105	HS-105/12	MT-111	CS-105	CS-106	EL-102	EE-119	...	CS-317	MT-442	CS-403	CS-421	CS-406	CS-414	CS-419	CS-423	CS-412	CGPA
0	CS-97001	B-	D+	C-	C	C-	D+	D	C-	B-	...	C-	B+	C-	C-	A-	A	C-	B	A-	2.205
1	CS-97002	A	D	D+	D	B-	C	D	A	D+	...	D	C-	C	D	A-	B-	C	C	B	2.008
2	CS-97003	A	B	A	B-	B+	A	B-	B+	A-	...	B	A	A	C	A	A	A	A-	A	3.608
3	CS-97004	D	C+	D+	D	D	A-	D+	C-	D	...	C	C-	D+	C-	B-	B	C+	C+	C+	1.906

```
grades_data.shape
```

```
(571, 43)
```

```
grades_data.columns
```

```
Index(['Seat No.', 'PH-121', 'HS-101', 'CY-105', 'HS-105/12', 'MT-111',
       'CS-105', 'CS-106', 'EL-102', 'EE-119', 'ME-107', 'CS-107', 'HS-205/20',
       'MT-222', 'EE-222', 'MT-224', 'CS-210', 'CS-211', 'CS-203', 'CS-214',
       'EE-217', 'CS-212', 'CS-215', 'MT-331', 'EF-303', 'HS-304', 'CS-301',
       'CS-302', 'TC-383', 'EL-332', 'CS-318', 'CS-306', 'CS-312', 'CS-317',
       'MT-442', 'CS-403', 'CS-421', 'CS-406', 'CS-414', 'CS-419', 'CS-423',
       'CS-412', 'CGPA'],
      dtype='object')
```

We have total 571 student records with 43 rows including seat no. and CGPA.

For simplicity in further will rename for some column names.

```
grades_data.rename(columns={'HS-105/12':'HS-105'},inplace=True)
```

```
grades_data.rename(columns={'HS-205/20':'HS-205'},inplace=True)
```

```
grades_data.rename(columns={'Seat No.':'Seat_no'},inplace=True)
```

Dataset Information

```
grades_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 571 entries, 0 to 570
Data columns (total 43 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Seat_no  571 non-null    object
 1   PH-121   571 non-null    object
 2   HS-101   571 non-null    object
 3   CY-105   570 non-null    object
 4   HS-105   570 non-null    object
 5   MT-111   569 non-null    object
 6   CS-105   571 non-null    object
 7   CS-106   569 non-null    object
 8   EL-102   569 non-null    object
 9   EE-119   569 non-null    object
 10  ME-107   569 non-null    object
 11  CS-107   569 non-null    object
 12  HS-205   566 non-null    object
 13  MT-222   566 non-null    object
 14  EE-222   564 non-null    object
 15  MT-224   564 non-null    object
 16  CS-210   564 non-null    object
 17  CS-211   566 non-null    object
 18  CS-203   566 non-null    object
 19  CS-214   565 non-null    object
 20  EE-217   565 non-null    object
 21  CS-212   565 non-null    object
 22  CS-215   565 non-null    object
 23  MT-331   562 non-null    object
 24  EF-303   561 non-null    object
 25  HS-304   561 non-null    object
 26  CS-301   561 non-null    object
 27  CS-302   561 non-null    object
 28  TC-383   561 non-null    object
 29  EL-332   562 non-null    object
 30  CS-318   562 non-null    object
 31  CS-306   562 non-null    object
 32  CS-312   561 non-null    object
 33  CS-317   559 non-null    object
 34  MT-442   561 non-null    object
 35  CS-403   559 non-null    object
 36  CS-421   559 non-null    object
 37  CS-406   486 non-null    object
 38  CS-414   558 non-null    object
 39  CS-419   558 non-null    object
 40  CS-423   557 non-null    object
 41  CS-412   492 non-null    object
 42  CGPA     571 non-null    float64
dtypes: float64(1), object(42)
memory usage: 191.9+ KB
```

From above we can observe that we have 2 types of dataset object and float type.

Will Check for Null Values present in dataset

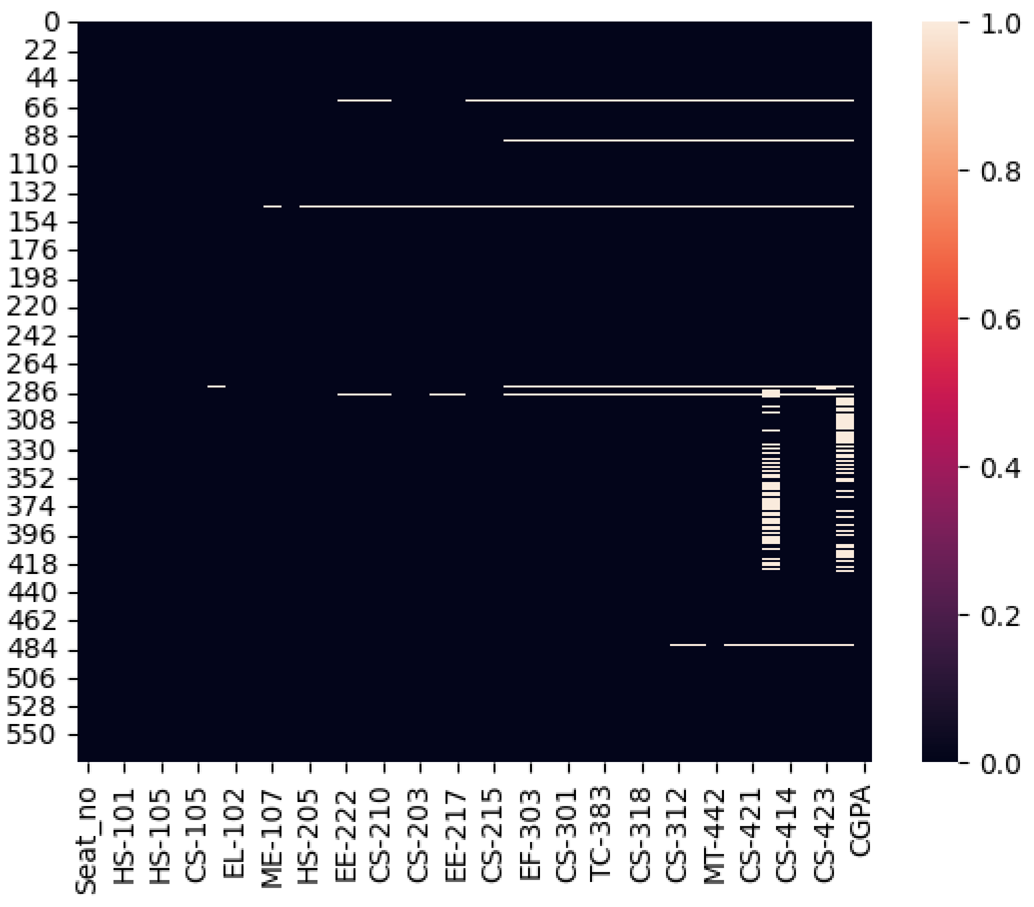
```
grades_data.isnull().sum()
```

```
Seat_no     0
PH-121      0
HS-101      0
CY-105      1
HS-105      1
MT-111      2
CS-105      0
CS-106      2
EL-102      2
EE-119      2
ME-107      2
CS-107      2
HS-205      5
MT-222      5
EE-222      7
MT-224      7
CS-210      7
CS-211      5
CS-203      5
CS-214      6
```

```
EE-217      6
CS-212      6
CS-215      6
MT-331      9
EF-303     10
HS-304     10
CS-301     10
CS-302     10
TC-383     10
EL-332      9
CS-318      9
CS-306      9
CS-312     10
CS-317     12
MT-442     10
CS-403     12
CS-421     12
CS-406     85
CS-414     13
CS-419     13
CS-423     14
CS-412     79
CGPA        0
dtype: int64
```

```
sns.heatmap(grades_data.isnull())
```

```
<Axes: >
```



We have few Null values in some of course code, will replace same with 0.

```
grades_data.replace(np.nan,0, inplace = True)
```

```
grades_data.isnull().sum()
```

```
Seat_no    0
PH-121     0
HS-101     0
CY-105     0
HS-105     0
MT-111     0
CS-105     0
CS-106     0
EL-102     0
EE-119     0
ME-107     0
CS-107     0
HS-205     0
MT-222     0
EE-222     0
MT-224     0
CS-210     0
CS-211     0
CS-203     0
CS-214     0
EE-217     0
CS-212     0
CS-215     0
MT-331     0
EF-303     0
HS-304     0
CS-301     0
CS-302     0
TC-383     0
EL-332     0
CS-318     0
CS-306     0
CS-312     0
CS-317     0
MT-442     0
CS-403     0
CS-421     0
CS-406     0
CS-414     0
CS-419     0
CS-423     0
CS-412     0
CGPA       0
dtype: int64
```

We have all data in object type datatype and need to convert textual information into numerical types through encoding

```
grades_data.reset_index(drop=True,inplace=True)


for column in grades_data.columns:

    grades_data[column]=grades_data[column].replace('A+',4.0)
    grades_data[column]=grades_data[column].replace('A',4.0)
    grades_data[column]=grades_data[column].replace('A-',3.7)
    grades_data[column]=grades_data[column].replace('B+',3.4)
    grades_data[column]=grades_data[column].replace('B',3.0)
    grades_data[column]=grades_data[column].replace('B-',2.7)
    grades_data[column]=grades_data[column].replace('C+',2.4)
    grades_data[column]=grades_data[column].replace('C',2.0)
    grades_data[column]=grades_data[column].replace('C-',1.7)
    grades_data[column]=grades_data[column].replace('D+',1.4)
    grades_data[column]=grades_data[column].replace('D',1.0)
    grades_data[column]=grades_data[column].replace('F',0.0)
    grades_data[column]=grades_data[column].replace('WU',0.0)
    grades_data[column]=grades_data[column].replace('W',0.0)
    grades_data[column]=grades_data[column].replace('I',0.0)


grades_data.head(2)
```

	Seat_no	PH-121	HS-101	CY-105	HS-105	MT-111	CS-105	CS-106	EL-102	EE-119	...	CS-317	MT-442	CS-403	CS-421	CS-406	CS-414	CS-419	CS-423	CS-412	CGPA
0	CS-97001	2.7	1.4	1.7	2.0	1.7	1.4	1.0	1.7	2.7	...	1.7	3.4	1.7	1.7	3.7	4.0	1.7	3.0	3.7	2.205
1	CS-97002	4.0	1.0	1.4	1.0	2.7	2.0	1.0	4.0	1.4	...	1.0	1.7	2.0	1.0	3.7	2.7	2.0	2.0	3.0	2.008

2 rows × 43 columns

Chart - 1

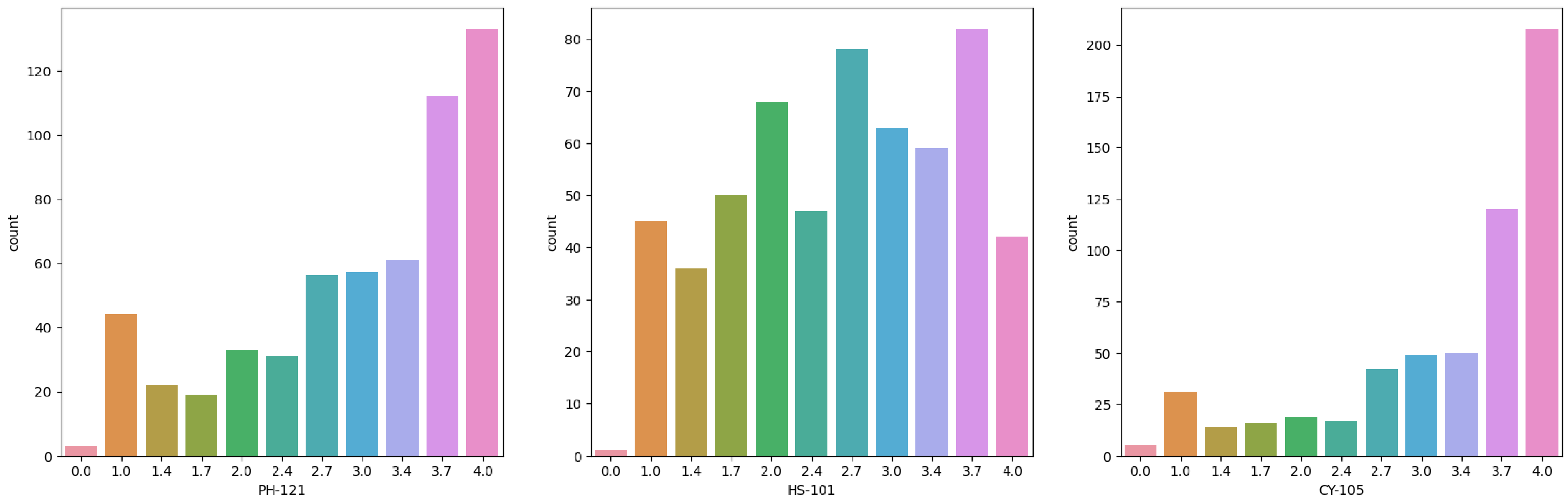
Count of Seat Numbers for cource code 'PH-121', 'HS-101', 'CY-105'

```
# Will get value count and average CGPA for each mentioned course code.

courses = ['PH-121', 'HS-101', 'CY-105']

# Creating dictionaries to hold average CGPA and value counts for each course
course_avg_cgpa = {}
course_value_counts = {}

for course in courses:
    course_avg_cgpa[course] = grades_data.groupby(course)['CGPA'].mean()

    course_value_counts[course] = grades_data[course].value_counts().sort_index()

for course in courses:
    print(f"Course: {course}")
    print(f"Average CGPA for {course}:")
    print(course_avg_cgpa[course])
    print("\nValue counts for {course}:")
    print(course_value_counts[course])
    print("\n")
```

```
1.0    2.268178
1.4    2.424667
1.7    2.654820
2.0    2.813088
2.4    2.872064
2.7    2.997987
3.0    3.059540
3.4    3.282186
3.7    3.268195
4.0    3.567429
Name: CGPA, dtype: float64

Value counts for {course}:
0.0     1
1.0    45
1.4    36
1.7    50
2.0    68
2.4    47
2.7    78
3.0    63
3.4    59
3.7    82
4.0    42
Name: HS-101, dtype: int64


Course: CY-105
Average CGPA for CY-105:
CY-105
0.0    1.731200
1.0    2.026161
1.4    2.104786
1.7    2.281125
2.0    2.453684
2.4    2.682471
2.7    2.574738
3.0    2.675776
3.4    2.896060
3.7    3.064600
4.0    3.393173
Name: CGPA, dtype: float64

Value counts for {course}:
0.0      5
1.0     31
1.4     14
1.7     16
2.0     19
2.4     17
2.7     42
3.0     49
3.4     50
3.7    120
4.0    208
Name: CY-105, dtype: int64
```

```
fig,axs = plt.subplots(1,3,figsize=(20,6))

sns.countplot(grades_data,x='PH-121',ax=axs[0])

sns.countplot(grades_data,x='HS-101',ax=axs[1])

sns.countplot(grades_data,x='CY-105',ax=axs[2])
```

```
<Axes: xlabel='CY-105', ylabel='count'>
```



From above graph we have below insights:

- Each subplot represents the count of occurrences for a specific course code ('PH-121', 'HS-101', 'CY-105') within the dataset.
- The height of each bar in the plot indicates the frequency or count of appearances of each category (in this case, grades or course-related data) within the respective course column.
- **From above each subplot,**
  1. **For course code PH-121, the highest student count is observed for grades 4.0 (A+) and 3.7 (A-), whereas notably fewer students received grades 0.0 (F) and 1.7 (C-).**
  2. **Regarding course code HS-101, the highest student count is for grades 3.7 (A-) and 2.7 (B-), while there's a lower count for grades 0.0 (F) and 1.4 (D+).**
  3. **Concerning course code CY-105, the highest student count is seen for grades 4.0 (A+) and 3.7 (A-), with a lower count for grades 0.0 (F) and 1.4 (D+).**

Chart - 2

Count of Seat Numbers for cource code 'HS-105', 'MT-111', 'CS-105'

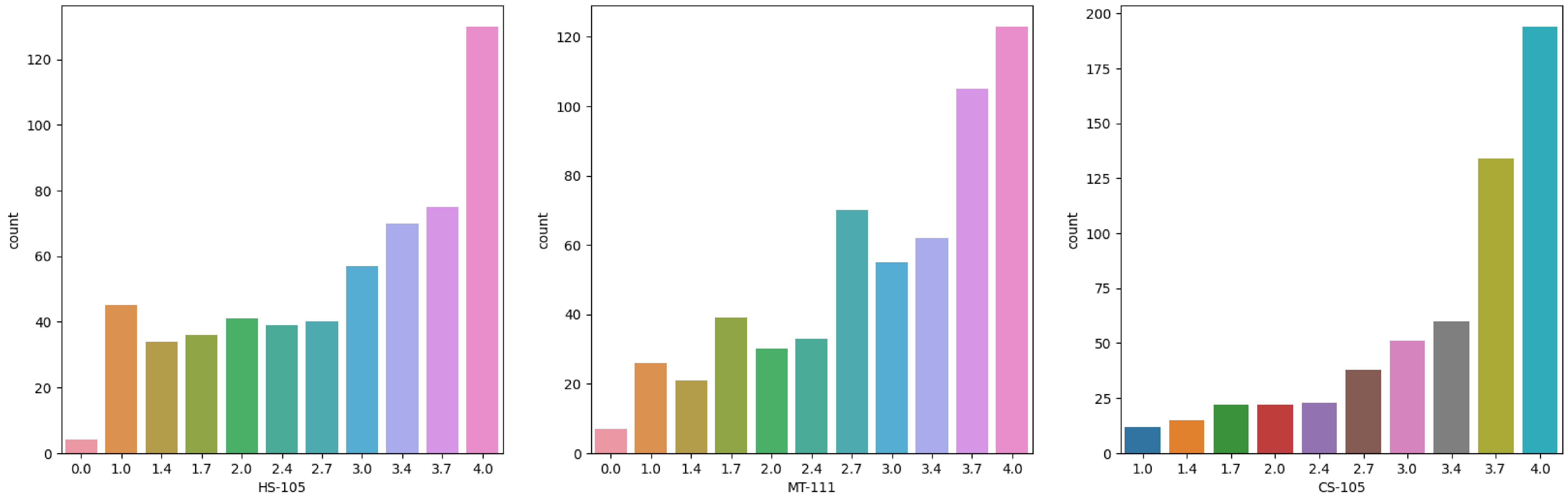
```
# Will get value count and average CGPA for each mentioned course code.

courses = ['HS-105', 'MT-111', 'CS-105']

# Creating dictionaries to hold average CGPA and value counts for each course
course_avg_cgpa = {}
course_value_counts = {}

for course in courses:
    course_avg_cgpa[course] = grades_data.groupby(course)['CGPA'].mean()

    course_value_counts[course] = grades_data[course].value_counts().sort_index()

for course in courses:
    print(f"Course: {course}")
    print(f"Average CGPA for {course}:")
    print(course_avg_cgpa[course])
    print("\nValue counts for {course}:")
    print(course_value_counts[course])
    print("\n")
```

```
MT-111
0.0    1.382714
```

```
2.0    2.526273
2.4    2.509609
2.7    2.580632
3.0    2.747314
3.4    2.829033
3.7    3.049769
4.0    3.377160
Name: CGPA, dtype: float64

Value counts for {course}:
1.0     12
1.4     15
1.7     22
2.0     22
2.4     23
2.7     38
3.0     51
3.4     60
3.7    134
4.0    194
Name: CS-105, dtype: int64
```

```
fig,axs = plt.subplots(1,3,figsize=(20,6))

sns.countplot(grades_data,x='HS-105',ax=axs[0])

sns.countplot(grades_data,x='MT-111',ax=axs[1])

sns.countplot(grades_data,x='CS-105',ax=axs[2])
```

```
<Axes: xlabel='CS-105', ylabel='count'>
```



- **From above each subplot,**
    1. **For course code HS-105, the highest student count is observed for grades 4.0 (A+) and 3.7 (A-), whereas notably fewer students received grades 0.0 (F) and 1.4 (D+).**
    2. **Regarding course code MT-111, the highest student count is for grades 3.7 (A-) and 2.7 (B-), while there's a lower count for grades 0.0 (F) and 1.4 (D+).**
    3. **Concerning course code CY-105, the highest student count is seen for grades 4.0 (A+) and 3.7 (A-), with a lower count for grades 1.0 (D) and 1.4 (D+).**

Chart - 3

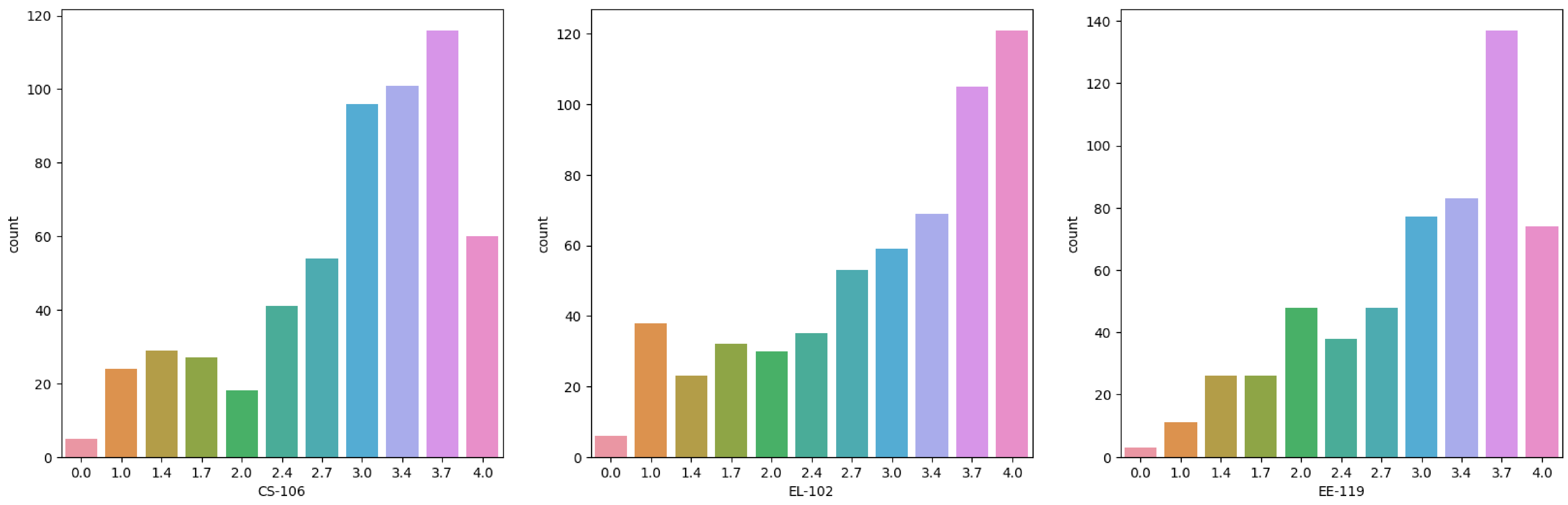
Count of Seat Numbers for cource code 'CS-106', 'EL-102', 'EE-119'

```
# Will get value count and average CGPA for each mentioned course code.

courses = ['CS-106', 'EL-102', 'EE-119']

# Creating dictionaries to hold average CGPA and value counts for each course
course_avg_cgpa = {}
course_value_counts = {}

for course in courses:
    course_avg_cgpa[course] = grades_data.groupby(course)['CGPA'].mean()

    course_value_counts[course] = grades_data[course].value_counts().sort_index()

for course in courses:
    print(f"Course: {course}")
    print(f"Average CGPA for {course}:")
    print(course_avg_cgpa[course])
    print("\nValue counts for {course}:")
    print(course_value_counts[course])
    print("\n")
```

```
3.0     59
3.4     69
3.7    105
4.0    121
Name: EL-102, dtype: int64


Course: EE-119
Average CGPA for EE-119:
EE-119
0.0    1.326000
1.0    2.159000
1.4    2.111654
1.7    2.353423
2.0    2.363208
2.4    2.670921
2.7    2.766437
3.0    2.983247
3.4    3.103735
3.7    3.216905
4.0    3.617135
Name: CGPA, dtype: float64

Value counts for {course}:
0.0      3
1.0     11
1.4     26
1.7     26
2.0     48
2.4     38
2.7     48
3.0     77
3.4     83
3.7    137
4.0     74
Name: EE-119, dtype: int64
```

```
fig,axs = plt.subplots(1,3,figsize=(20,6))

sns.countplot(grades_data,x='CS-106',ax=axs[0])

sns.countplot(grades_data,x='EL-102',ax=axs[1])

sns.countplot(grades_data,x='EE-119',ax=axs[2])
```

```
<Axes: xlabel='EE-119', ylabel='count'>
```



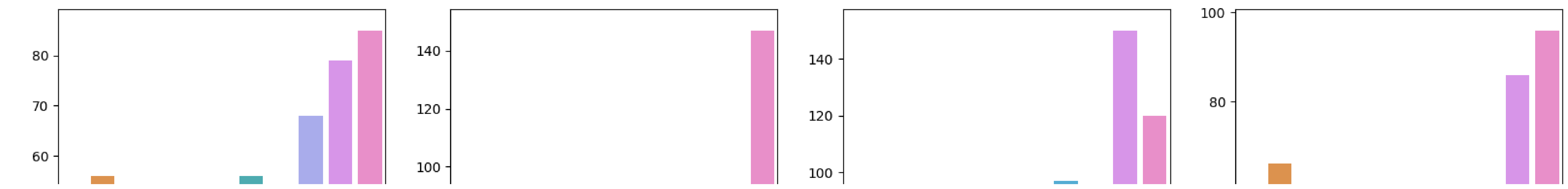
- **From above each subplot,**
    1. **For course code CS-106, the highest student count is observed for grades 3.7 (A-) and 3.4 (B+), whereas notably fewer students received grades 0.0 (F) and 2.0 (C).**
    2. **Regarding course code EL-102, the highest student count is for grades 4.0 (A+) and 3.7 (A-), while there's a lower count for grades 0.0 (F) and 1.4 (D+).**
    3. **Concerning course code CY-105, the highest student count is seen for grades 3.7 (A-) and 3.4 (B+), with a lower count for grades 0.0 (F) and 1.0 (D).**

Chart - 4

Count of Seat Numbers for cource code 'ME-107', 'CS-107', 'HS-205','MT-222'

```
fig,axs = plt.subplots(1,4,figsize=(20,6))

sns.countplot(grades_data,x='ME-107',ax=axs[0])

sns.countplot(grades_data,x='CS-107',ax=axs[1])

sns.countplot(grades_data,x='HS-205',ax=axs[2])

sns.countplot(grades_data,x='MT-222',ax=axs[3])
```

```
<Axes: xlabel='MT-222', ylabel='count'>
```



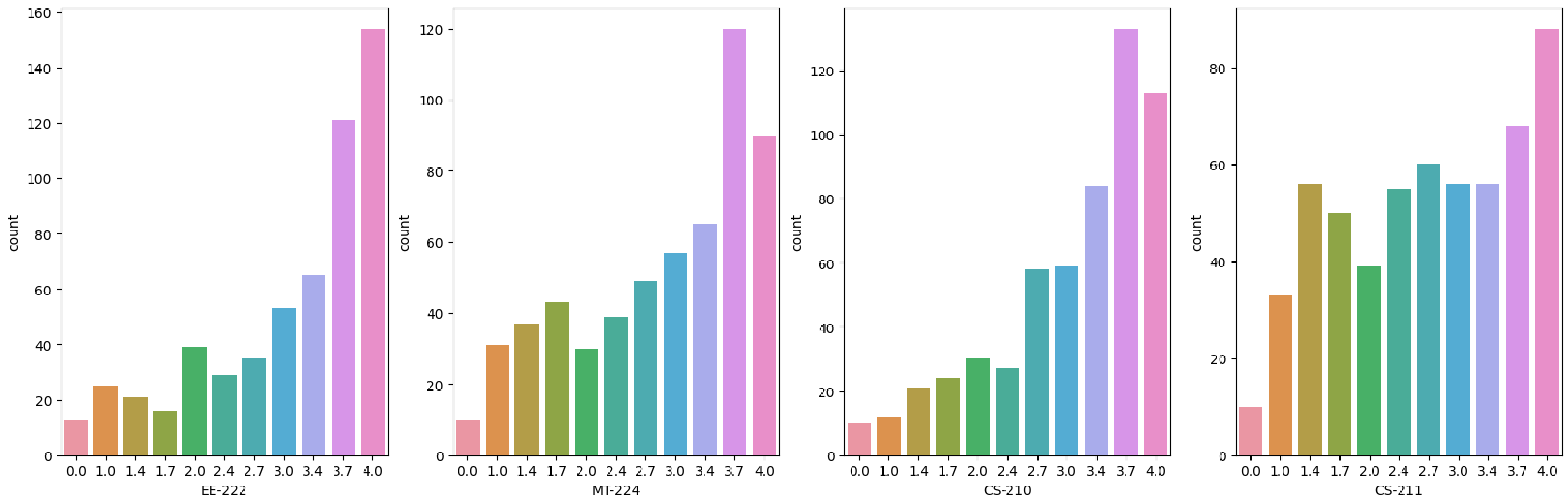
- **From above each subplot,**
  1. **For course code ME-107, the highest student count is observed for grades 4.0 (A+) and 3.7 (A), whereas notably fewer students received grades 0.0 (F) and 2.4 (C+).**
  2. **Regarding course code CS-107, the highest student count is for grades 4.0 (A+) and 3.7 (A-), while there's a lower count for grades 0.0 (F) and 1.0 (D).**
  3. **Concerning course code HS-205, the highest student count is seen for grades 3.7 (A-) and 4.0 (A+), with a lower count for grades 1.0 (D) and 0.0 (F).**
  4. **For course code MT-222, the highest student count is observed for grades 4.0 (A+) and 3.7 (A-), whereas notably fewer students received grades 0.0 (F) and W**

Chart - 5

Count of Seat Numbers for cource code 'EE-222', 'MT-224', 'CS-210', 'CS-211'

```
fig,axs = plt.subplots(1,4,figsize=(20,6))

sns.countplot(grades_data,x='EE-222',ax=axs[0])

sns.countplot(grades_data,x='MT-224',ax=axs[1])

sns.countplot(grades_data,x='CS-210',ax=axs[2])

sns.countplot(grades_data,x='CS-211',ax=axs[3])
```

```
<Axes: xlabel='CS-211', ylabel='count'>
```



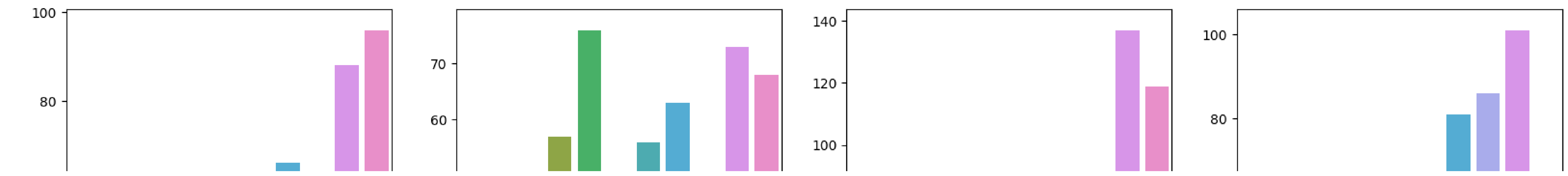
- **From above each subplot,**
  1. **For course code EE-222, the highest student count is observed for grades 4.0 (A+) and 3.7 (A), whereas notably fewer students received grades 0.0 (F) and 2.4 (C+).**
  2. **Regarding course code MT-224, the highest student count is for grades 3.7 (A-) and 4.0 (A+), while there's a lower count for grades 0.0 (F) and W**
  3. **Concerning course code HS-205, the highest student count is seen for grades 3.7 (A-) and 4.0 (A+), with a lower count for grades 0.0 (F) and W**
  4. **For course code MT-222, the highest student count is observed for grades 4.0 (A+) and 3.7 (A-), whereas notably fewer students received grades 0.0 (F) and W**

Chart - 6

Count of Seat Numbers for cource code 'CS-203', 'CS-214', 'EE-217','CS-212'

```
fig,axs = plt.subplots(1,4,figsize=(20,6))

sns.countplot(grades_data,x='CS-203',ax=axs[0])

sns.countplot(grades_data,x='CS-214',ax=axs[1])

sns.countplot(grades_data,x='EE-217',ax=axs[2])

sns.countplot(grades_data,x='CS-212',ax=axs[3])
```

```
<Axes: xlabel='CS-212', ylabel='count'>
```



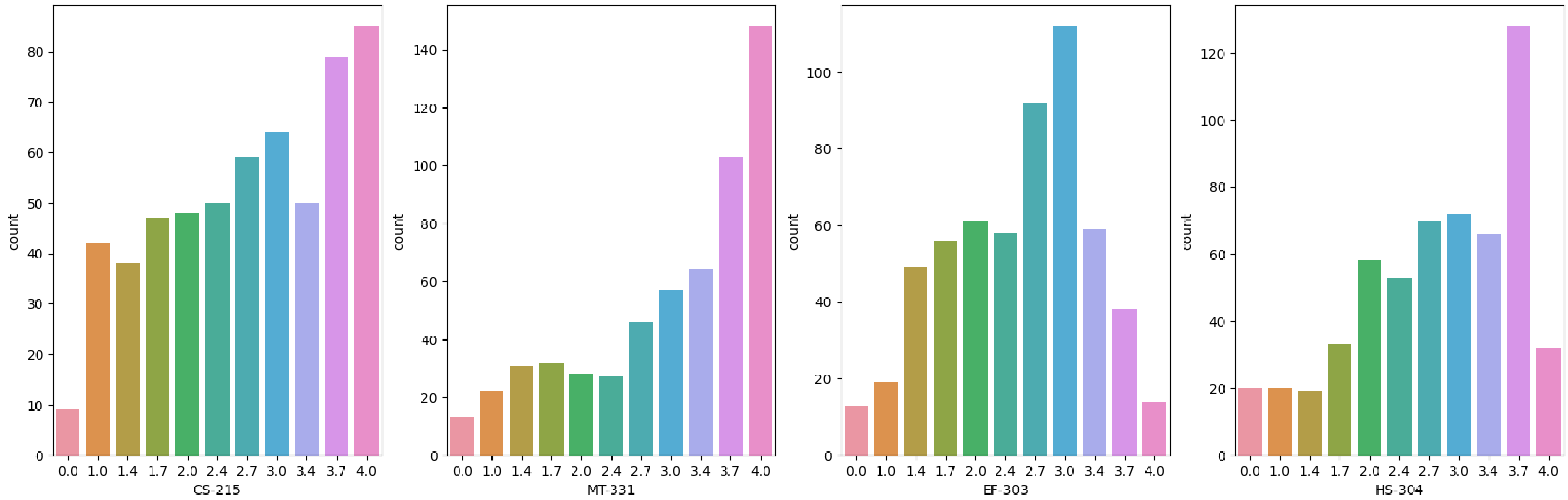
- **From above each subplot,**
    1. **For course code CS-203, the highest student count is observed for grades 4.0 (A+) and 3.7 (A), whereas notably fewer students received grades 0.0 (F) and I**
    2. **Regarding course code CS-214, the highest student count is for grades 2.0 (C) and 3.7 (A), while there's a lower count for grades 0.0 (F) and I**
    3. **Concerning course code HS-205, the highest student count is seen for grades 3.7 (A-) and 4.0 (A+), with a lower count for grades 0.0 (F) and 1.0(D)**
    4. **For course code MT-222, the highest student count is observed for grades 3.7 (A-) and 3.4 (B+), whereas notably fewer students received grades 0.0 (F) and 1.0(D)**

Chart - 7

Count of Seat Numbers for cource code 'CS-215', 'MT-331', 'EF-303', 'HS-304'

```
fig,axs = plt.subplots(1,4,figsize=(20,6))

sns.countplot(grades_data,x='CS-215',ax=axs[0])

sns.countplot(grades_data,x='MT-331',ax=axs[1])

sns.countplot(grades_data,x='EF-303',ax=axs[2])

sns.countplot(grades_data,x='HS-304',ax=axs[3])
```

```
<Axes: xlabel='HS-304', ylabel='count'>
```



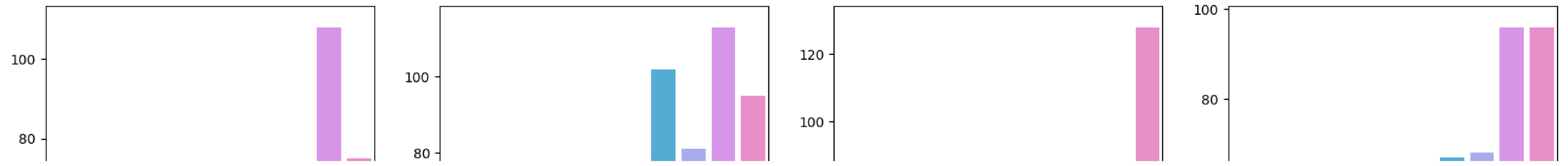
- **From above each subplot,**
    1. **For course code CS-215, the highest student count is observed for grades 4.0 (A+) and 3.7 (A), whereas notably fewer students received grades 0.0 (F) and W**
    2. **Regarding course code MT-331, the highest student count is for grades 4.0 (A+) and 3.7 (A), while there's a lower count for grades 0.0 (F) and 1.0(D)**
    3. **Concerning course code E-303, the highest student count is seen for grades 3.0 (B) and 2.7 (B-), with a lower count for grades 0.0 (F) and 4.0(A+)**
    4. **For course code HS-304, the highest student count is observed for grades 3.7 (A-) and 3.0 (B), whereas notably fewer students received grades 0.0 (F) and W**

Chart - 8

Count of Seat Numbers for cource code 'CS-301', 'CS-302','TC-383', 'EL-332'

```
fig,axs = plt.subplots(1,4,figsize=(20,6))

sns.countplot(grades_data,x='CS-301',ax=axs[0])

sns.countplot(grades_data,x='CS-302',ax=axs[1])

sns.countplot(grades_data,x='TC-383',ax=axs[2])

sns.countplot(grades_data,x='EL-332',ax=axs[3])
```

```
<Axes: xlabel='EL-332', ylabel='count'>
```



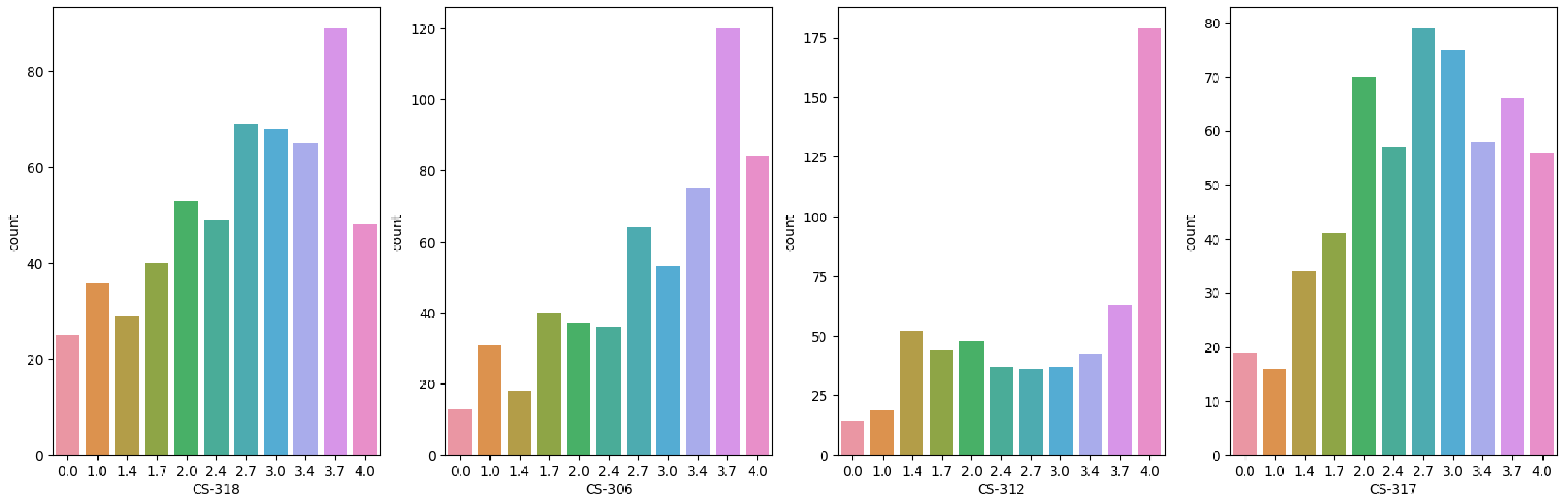
- **From above each subplot,**
  1. **For course code CS-301, the highest student count is observed for grades 3.7 (A) and 4.0 (A+), whereas notably fewer students received grades 0.0 (F) and 1.0(D)**
  2. **Regarding course code CS-302, the highest student count is for grades 4.0 (A+) and 3.7 (A), while there's a lower count for grades 0.0 (F) and 1.4(D+)**
  3. **Concerning course code TC-383, the highest student count is seen for grades 4.0 (A+) and 3.7 (A), with a lower count for grades 0.0 (F) and 1.0(D)**
  4. **For course code EL-332, the highest student count is observed for grades 4.0 (A+) and 3.7 (A-), whereas notably fewer students received grades 0.0 (F) and 1.0(D)**

Chart - 9

Count of Seat Numbers for cource code 'CS-318', 'CS-306', 'CS-312', 'CS-317'

```
fig,axs = plt.subplots(1,4,figsize=(20,6))

sns.countplot(grades_data,x='CS-318',ax=axs[0])

sns.countplot(grades_data,x='CS-306',ax=axs[1])

sns.countplot(grades_data,x='CS-312',ax=axs[2])

sns.countplot(grades_data,x='CS-317',ax=axs[3])
```

```
<Axes: xlabel='CS-317', ylabel='count'>
```



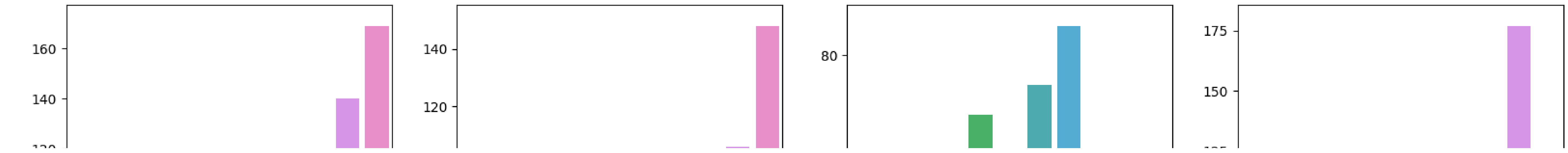
- **From above each subplot,**
  1. **For course code CS-318, the highest student count is observed for grades 3.7 (A) and 2.7 (B-), whereas notably fewer students received grades 0.0 (F) and W**
  2. **Regarding course code CS-302, the highest student count is for grades 3.7 (A-) and 4.0 (A+), while there's a lower count for grades 0.0 (F) and 1.4(D+)**
  3. **Concerning course code CS-312, the highest student count is seen for grades 4.0 (A+) and 3.7 (A), with a lower count for grades 0.0 (F) and W**
  4. **For course code CS-317, the highest student count is observed for grades 2.7 (B-) and 3.0 (B), whereas notably fewer students received grades 0.0 (F) and 1.0(D)**

Chart - 10

Count of Seat Numbers for cource code 'MT-442','CS-403', 'CS-421', 'CS-406'

```
fig,axs = plt.subplots(1,4,figsize=(20,6))

sns.countplot(grades_data,x='MT-442',ax=axs[0])

sns.countplot(grades_data,x='CS-403',ax=axs[1])

sns.countplot(grades_data,x='CS-421',ax=axs[2])

sns.countplot(grades_data,x='CS-406',ax=axs[3])
```

```
<Axes: xlabel='CS-406', ylabel='count'>
```



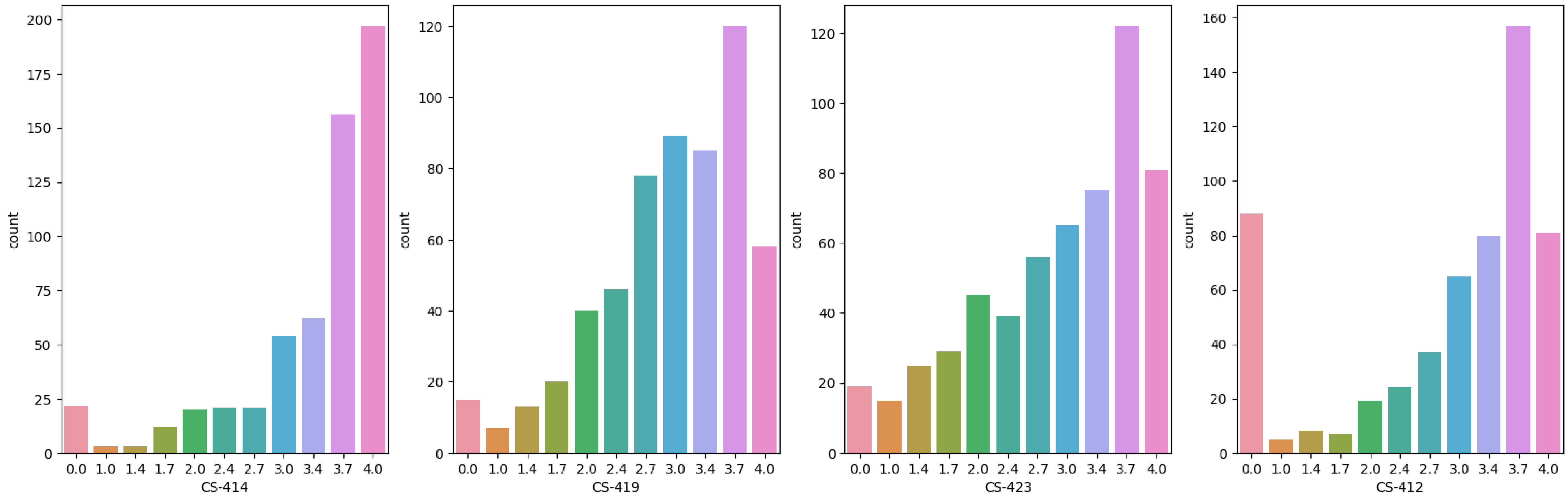
- **From above each subplot,**
  1. **For course code MT-442, the highest student count is observed for grades 4.0 (A+) and 3.7 (A-), whereas notably fewer students received grades 0.0 (F) and 1.4(D+)**
  2. **Regarding course code CS-403, the highest student count is for grades 4.0 (A+) and 3.7 (A-), while there's a lower count for grades 0.0 (F) and 1.0(D)**
  3. **Concerning course code CS-421, the highest student count is seen for grades 3.0 (B) and 2.7 (B-), with a lower count for grades 1.0 (D) and W**
  4. **For course code CS-406, the highest student count is observed for grades 3.7 (A-) and 0.0 (F), whereas notably fewer students received grades 1.0 (D) and W**

Chart - 11

Count of Seat Numbers for cource code 'CS-414', 'CS-419', 'CS-423', 'CS-412'

```
fig,axs = plt.subplots(1,4,figsize=(20,6))

sns.countplot(grades_data,x='CS-414',ax=axs[0])

sns.countplot(grades_data,x='CS-419',ax=axs[1])

sns.countplot(grades_data,x='CS-423',ax=axs[2])

sns.countplot(grades_data,x='CS-412',ax=axs[3])
```

```
<Axes: xlabel='CS-412', ylabel='count'>
```

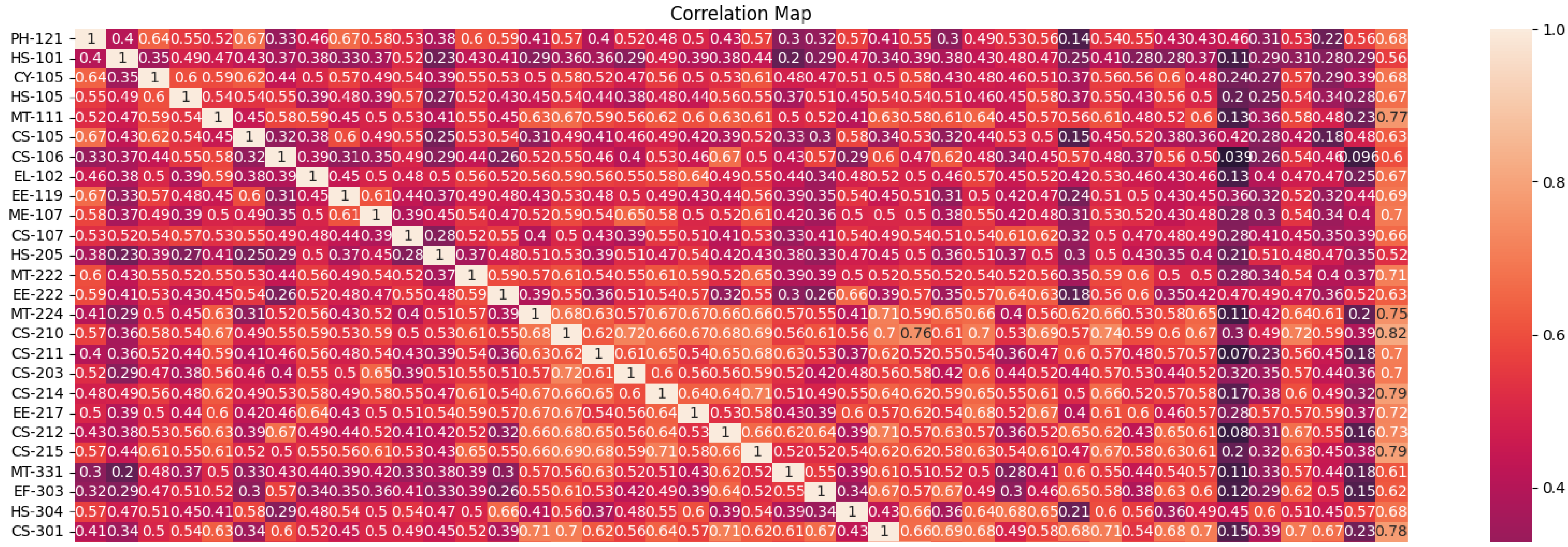


- **From above each subplot,**
  1. **For course code CS-414, the highest student count is observed for grades 4.0 (A+) and 3.7 (A-), whereas notably fewer students received grades 1.0 (D) and W**
  2. **Regarding course code CS-419, the highest student count is for grades 3.7 (A-) and 3.0 (B), while there's a lower count for grades 1.0 (D) and 1.4(D+)**
  3. **Concerning course code CS-423, the highest student count is seen for grades 3.7 (A-) and 4.0 (A+), with a lower count for grades 0.0 (F) and 1.0(D)**
  4. **For course code CS-412, the highest student count is observed for grades 3.7 (A-) and 0.0 (F), whereas notably fewer students received grades 1.0 (D) and W**

Heatmap

```
correlation_data = grades_data

correlation_matrix = correlation_data.corr()

plt.figure(figsize=(20,10))

sns.heatmap(correlation_matrix,annot=True)
plt.title('Correlation Map')
plt.show()
```

```
<ipython-input-227-8c5395578256>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid colum
  correlation_matrix = correlation_data.corr()
```



Pair Plot


```
sns.pairplot(grades_data)

plt.show()
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-228-b094ce3dd55e> in <cell line: 1>()
----> 1 sns.pairplot(grades_data)
      2
      3 plt.show()

                                  9 frames
/usr/local/lib/python3.10/dist-packages/matplotlib/cbook/__init__.py in <listcomp>(.0)
    832         """Clean dead weak references from the dictionary."""
    833         mapping = self._mapping
--> 834         to_drop = [key for key in mapping if key() is None]
    835         for key in to_drop:
    836             val = mapping.pop(key)

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

```
Error in callback <function _draw_all_if_interactive at 0x7fd9ed0fe050> (for post_execute):
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py in _draw_all_if_interactive()
    118 def _draw_all_if_interactive():
    119     if matplotlib.is_interactive():
--> 120         draw_all()
    121
    122

                                  23 frames
/usr/local/lib/python3.10/dist-packages/matplotlib/cbook/__init__.py in <listcomp>(.0)
    832         """Clean dead weak references from the dictionary."""
    833         mapping = self._mapping
--> 834         to_drop = [key for key in mapping if key() is None]
    835         for key in to_drop:
    836             val = mapping.pop(key)

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

```
Error in callback <function flush_figures at 0x7fd9ed0fd2d0> (for post_execute):
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/matplotlib_inline/backend_inline.py in flush_figures()
    124             # ignore the tracking, just draw and close all figures
    125             try:
--> 126                 return show(True)
    127             except Exception as e:
    128                 # safely show traceback if in IPython, else raise

                                  25 frames
<decorator-gen-2> in __call__(self, obj)

/usr/local/lib/python3.10/dist-packages/matplotlib/cbook/__init__.py in <listcomp>(.0)
    832         """Clean dead weak references from the dictionary."""
    833         mapping = self._mapping
--> 834         to_drop = [key for key in mapping if key() is None]
    835         for key in to_drop:
    836             val = mapping.pop(key)

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

```
grades_data.columns
```

```
Index(['Seat_no', 'PH-121', 'HS-101', 'CY-105', 'HS-105', 'MT-111', 'CS-105',
       'CS-106', 'EL-102', 'EE-119', 'ME-107', 'CS-107', 'HS-205', 'MT-222',
       'EE-222', 'MT-224', 'CS-210', 'CS-211', 'CS-203', 'CS-214', 'EE-217',
       'CS-212', 'CS-215', 'MT-331', 'EF-303', 'HS-304', 'CS-301', 'CS-302',
       'TC-383', 'EL-332', 'CS-318', 'CS-306', 'CS-312', 'CS-317', 'MT-442',
       'CS-403', 'CS-421', 'CS-406', 'CS-414', 'CS-419', 'CS-423', 'CS-412',
       'CGPA'],
      dtype='object')
```

KDE plot For all columns

```
num_columns = grades_data[['PH-121', 'HS-101', 'CY-105', 'HS-105', 'MT-111', 'CS-105',
```

```
num_columns = grades_data[['PH-121', 'HS-101', 'CY-105', 'HS-105', 'MT-111', 'CS-105',
       'CS-106', 'EL-102', 'EE-119', 'ME-107', 'CS-107', 'HS-205', 'MT-222',
       'EE-222', 'MT-224', 'CS-210', 'CS-211', 'CS-203', 'CS-214', 'EE-217',
       'CS-212', 'CS-215', 'MT-331', 'EF-303', 'HS-304', 'CS-301', 'CS-302',
       'TC-383', 'EL-332', 'CS-318', 'CS-306', 'CS-312', 'CS-317', 'MT-442',
       'CS-403', 'CS-421', 'CS-406', 'CS-414', 'CS-419', 'CS-423', 'CS-412']]

plt.figure(figsize=(20,15),facecolor='red')
plotnumber=1

for column in num_columns:
  if plotnumber<=45:
    ax=plt.subplot(9,5,plotnumber)
    sns.distplot(num_columns[column])
    plt.xlabel(column,fontsize=20)
  plotnumber+=1
plt.tight_layout()
```

```
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
```

```
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```

```
  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:
```

```
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(num_columns[column])
<ipython-input-230-44703309d835>:14: UserWarning:
```

ML Model Implementation

ML Model - 1

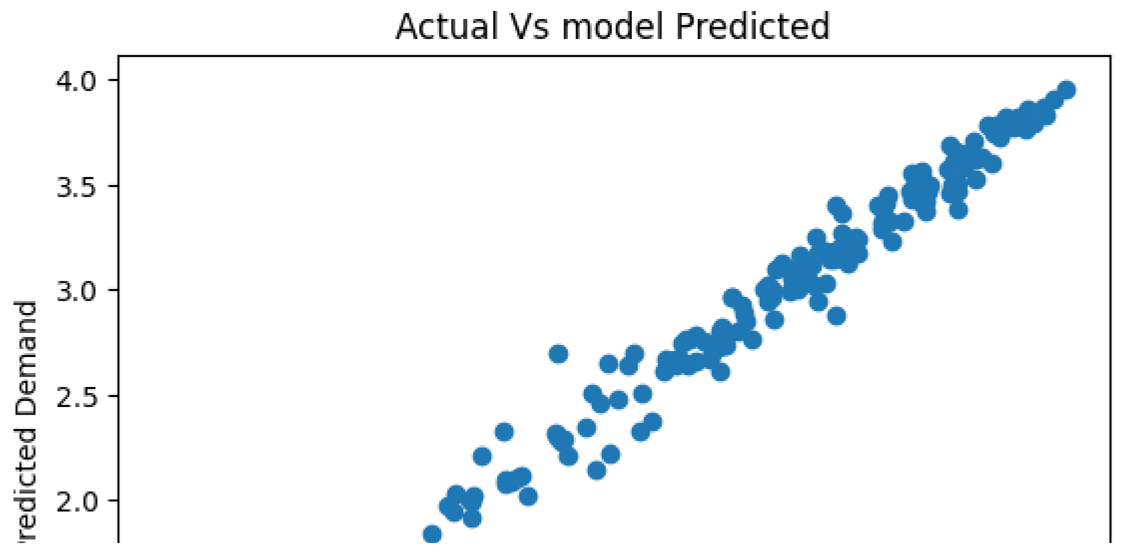
Linear regression

```
# Importing Necessary Libraries

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import Ridge
import math
```

```
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.
```

```
x = grades_data[['PH-121', 'HS-101', 'CY-105', 'HS-105', 'MT-111', 'CS-105',
       'CS-106', 'EL-102', 'EE-119', 'ME-107', 'CS-107', 'HS-205', 'MT-222',
       'EE-222', 'MT-224', 'CS-210', 'CS-211', 'CS-203', 'CS-214', 'EE-217',
       'CS-212', 'CS-215', 'MT-331', 'EF-303', 'HS-304', 'CS-301', 'CS-302',
       'TC-383', 'EL-332', 'CS-318', 'CS-306', 'CS-312', 'CS-317', 'MT-442',
       'CS-403', 'CS-421', 'CS-406', 'CS-414', 'CS-419', 'CS-423', 'CS-412']]

y = grades_data['CGPA']

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.30,random_state=248)

print("Shape of x_train",x_train.shape)
print("Shape of x_test",x_test.shape)
print("Shape of y_train",y_train.shape)
print("Shape of y_train",y_test.shape)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting linear regressio to training set
LR = LinearRegression()
LR.fit(x_train, y_train)

# Predicting on test set results

y_pred = LR.predict(x_test)

y_pred

# Evaluate the Linear Regression model
LR_predictions = LR.predict(x_test)
LR_mse = mean_squared_error(y_test, LR_predictions)
LR_RMSE = math.sqrt(mean_squared_error(y_test,y_pred))
LR_r2 = r2_score(y_test, LR_predictions)
print("Linear Regression MSE For Model 1: ", LR_mse)
print("Linear Regression RMSE For Model 1: ", LR_RMSE)
print("Linear Regression R-squared For Model 1: ", LR_r2)


plt.scatter(y_test,y_pred)
plt.xlabel('Actual Demand')
plt.ylabel('Predicted Demand')
plt.title('Actual Vs model Predicted')
plt.show()
```

```
Shape of x_train (399, 41)
Shape of x_test (172, 41)
Shape of y_train (399,)
Shape of y_train (172,)
Linear Regression MSE For Model 1:  0.0146459886611635
Linear Regression RMSE For Model 1:  0.1210206125466381
Linear Regression R-squared For Model 1:  0.9579167057947054
```



Insights from ML Model 1:

1. The MSE for Model 1 is 0.0146. Lower MSE indicates better fit; in this case, the model's average squared error between predicted and actual values is quite small.
2. The RMSE for Model 1 is 0.121. RMSE is the square root of MSE and measures the average magnitude of errors. It appears to be relatively low, suggesting the model's predictions are generally close to the actual values.
3. The R-squared value for Model 1 is 0.958, indicating the proportion of variance in the dependent variable (target) explained by the independent variables (features). A value closer to 1 signifies a better fit.

```
# Importing Necessary Libraries

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression, Lasso, Ridge

# Lasso Regression model with hyperparameter tuning
lasso_model = Lasso()
lasso_param_grid = {'alpha': [0.01, 0.1, 1, 10]}
lasso_grid = GridSearchCV(lasso_model, lasso_param_grid, cv=5)
lasso_grid.fit(x_train, y_train)

# Evaluate the Lasso Regression model
lasso_predictions = lasso_grid.predict(x_test)
lasso_mse = mean_squared_error(y_test, lasso_predictions)
lasso_r2 = r2_score(y_test, lasso_predictions)
print("Lasso Regression MSE: ", lasso_mse)
print("Lasso Regression R-squared: ", lasso_r2)
print("Best Lasso Alpha: ", lasso_grid.best_params_['alpha'])


# Similarly for ridge regression

# Ridge Regression model with hyperparameter tuning
ridge_model = Ridge()
ridge_param_grid = {'alpha': [0.01, 0.1, 1, 10]}
ridge_grid = GridSearchCV(ridge_model, ridge_param_grid, cv=5)
ridge_grid.fit(x_train, y_train)

# Evaluate the Ridge Regression model
ridge_predictions = ridge_grid.predict(x_test)
ridge_mse = mean_squared_error(y_test, ridge_predictions)
ridge_r2 = r2_score(y_test, ridge_predictions)
print("Ridge Regression MSE: ", ridge_mse)
print("Ridge Regression R-squared: ", ridge_r2)
print("Best Ridge Alpha: ", ridge_grid.best_params_['alpha'])
```

```
Lasso Regression MSE:  0.019170083207307717
Lasso Regression R-squared:  0.9449173237657676
Best Lasso Alpha:  0.01
Ridge Regression MSE:  0.01588072094227876
Ridge Regression R-squared:  0.954368867335105
Best Ridge Alpha:  10
```

Ridge regression shows slightly better performance in terms of MSE and R-squared compared to Lasso regression. This implies that Ridge might be better at fitting the data.Lasso with a smaller alpha of 0.01 and Ridge with a larger alpha of 10 indicate different levels of regularization. Lasso, being a feature selection method, might have eliminated some less impactful variables due to its stronger regularization.

ML Model - 2

Considering subjects of only 1st year

```
x = grades_data[['PH-121', 'HS-101', 'CY-105', 'HS-105', 'MT-111', 'CS-105',
       'CS-106', 'EL-102', 'EE-119', 'ME-107', 'CS-107']]

y = grades_data['CGPA']

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.30,random_state=248)

print("Shape of x_train",x_train.shape)
print("Shape of x_test",x_test.shape)
print("Shape of y_train",y_train.shape)
print("Shape of y_train",y_test.shape)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting linear regressio to training set
LR = LinearRegression()
LR.fit(x_train, y_train)

# Predicting on test set results

y_pred = LR.predict(x_test)

y_pred

# Evaluate the Linear Regression model
LR_predictions = LR.predict(x_test)
LR_mse = mean_squared_error(y_test, LR_predictions)
LR_RMSE = math.sqrt(mean_squared_error(y_test,y_pred))
LR_r2 = r2_score(y_test, LR_predictions)
print("Linear Regression MSE For Model 2: ", LR_mse)
print("Linear Regression RMSE For Model 2: ", LR_RMSE)
print("Linear Regression R-squared For Model 2: ", LR_r2)


plt.scatter(y_test,y_pred)
plt.xlabel('Actual Demand')
plt.ylabel('Predicted Demand')
plt.title('Actual Vs model Predicted')
plt.show()

# Will check for Lasso and ridge regression on model for better performance

# Lasso Regression model with hyperparameter tuning
lasso_model = Lasso()
lasso_param_grid = {'alpha': [0.01, 0.1, 1, 10]}
lasso_grid = GridSearchCV(lasso_model, lasso_param_grid, cv=5)
lasso_grid.fit(x_train, y_train)

# Evaluate the Lasso Regression model
lasso_predictions = lasso_grid.predict(x_test)
lasso_mse = mean_squared_error(y_test, lasso_predictions)
lasso_r2 = r2_score(y_test, lasso_predictions)
print("Lasso Regression MSE for ML Model 2: ", lasso_mse)
print("Lasso Regression R-squared for ML Model 2: ", lasso_r2)
print("Best Lasso Alpha For ML Model 2: ", lasso_grid.best_params_['alpha'])

# Doing regularization with lasso and ridge regression

# Ridge Regression model with hyperparameter tuning
ridge_model = Ridge()
ridge_param_grid = {'alpha': [0.01, 0.1, 1, 10]}
ridge_grid = GridSearchCV(ridge_model, ridge_param_grid, cv=5)
ridge_grid.fit(x_train, y_train)

# Evaluate the Ridge Regression model
ridge_predictions = ridge_grid.predict(x_test)
ridge_mse = mean_squared_error(y_test, ridge_predictions)
ridge_r2 = r2_score(y_test, ridge_predictions)
print("Ridge Regression MSE For ML Model 2: ", ridge_mse)
print("Ridge Regression R-squared For ML Model 2: ", ridge_r2)
print("Best Ridge Alpha For ML Model 2: ", ridge_grid.best_params_['alpha'])
```

```
Shape of x_train (399, 11)
Shape of x_test (172, 11)
Shape of y_train (399,)
Shape of y_train (172,)
Linear Regression MSE For Model 2:  0.051468854255514104
```

Insights from linear Model 2:

MSE: 0.0514 RMSE: 0.2269 R-squared: 0.8521 It explains around 85.2% of the variance and has a moderate error rate.

Lasso Regression: Slightly lower performance: MSE: 0.0561 R-squared: 0.8387 Best Alpha: 0.01 Penalizes coefficients more, offering some feature selection.

Ridge Regression: Performance similar to Linear Regression: MSE: 0.0514 R-squared: 0.8522 Best Alpha: 1 Moderate regularization without eliminating coefficients.

Linear and Ridge regressions show similar performance, explaining about 85.2%

2.5

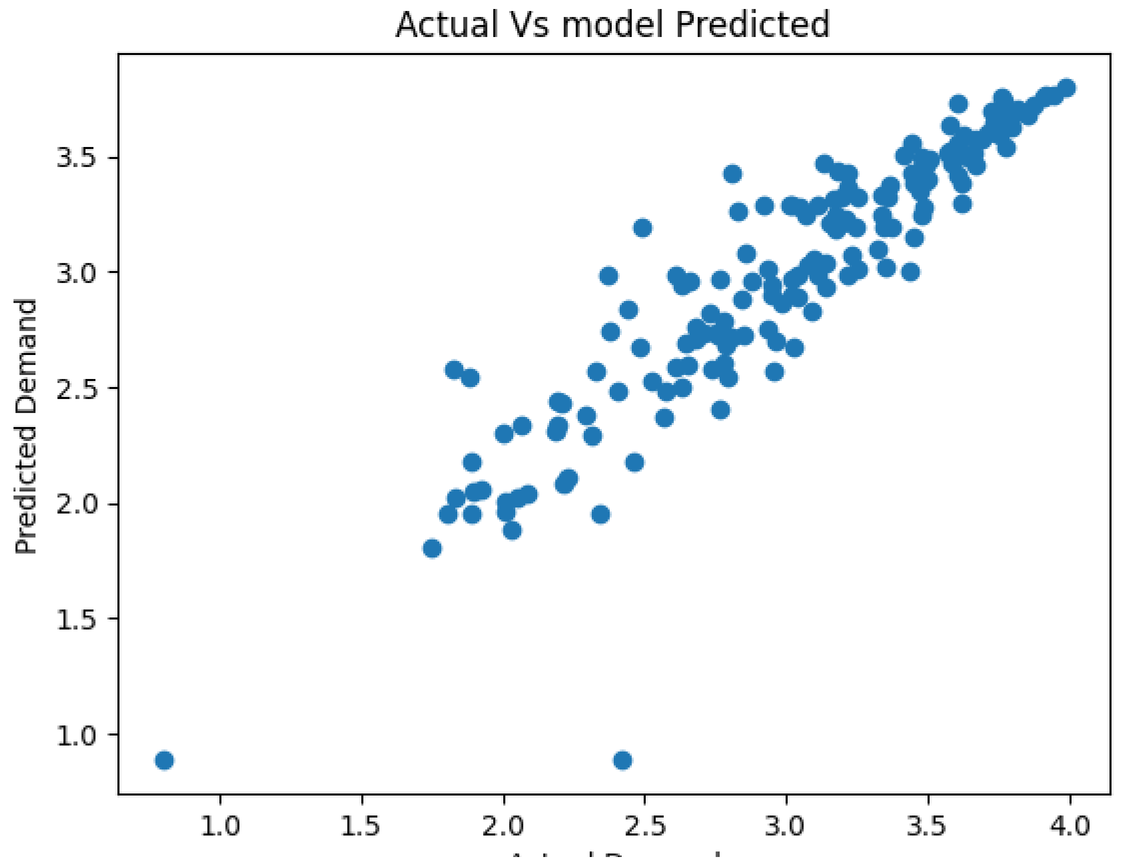
ML Model - 3

Considering subjects of only 2nd year

1.5

```
x = grades_data[['HS-205', 'MT-222',
       'EE-222', 'MT-224', 'CS-210', 'CS-211', 'CS-203', 'CS-214', 'EE-217',
       'CS-212', 'CS-215']]

y = grades_data['CGPA']

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.30,random_state=248)

print("Shape of x_train",x_train.shape)
print("Shape of x_test",x_test.shape)
print("Shape of y_train",y_train.shape)
print("Shape of y_train",y_test.shape)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting linear regressio to training set
LR = LinearRegression()
LR.fit(x_train, y_train)

# Predicting on test set results

y_pred = LR.predict(x_test)

y_pred

# Evaluate the Linear Regression model
LR_predictions = LR.predict(x_test)
LR_mse = mean_squared_error(y_test, LR_predictions)
LR_RMSE = math.sqrt(mean_squared_error(y_test,y_pred))
LR_r2 = r2_score(y_test, LR_predictions)
print("Linear Regression MSE For Model 3: ", LR_mse)
print("Linear Regression RMSE For Model 3: ", LR_RMSE)
print("Linear Regression R-squared For Model 3: ", LR_r2)


plt.scatter(y_test,y_pred)
plt.xlabel('Actual Demand')
plt.ylabel('Predicted Demand')
plt.title('Actual Vs model Predicted')
plt.show()

# Will check for Lasso and ridge regression on model for better performance

# Lasso Regression model with hyperparameter tuning
lasso_model = Lasso()
lasso_param_grid = {'alpha': [0.01, 0.1, 1, 10]}
lasso_grid = GridSearchCV(lasso_model, lasso_param_grid, cv=5)
lasso_grid.fit(x_train, y_train)

# Evaluate the Lasso Regression model
lasso_predictions = lasso_grid.predict(x_test)
lasso_mse = mean_squared_error(y_test, lasso_predictions)
lasso_r2 = r2_score(y_test, lasso_predictions)
print("Lasso Regression MSE for ML Model 3: ", lasso_mse)
print("Lasso Regression R-squared for ML Model 3: ", lasso_r2)
print("Best Lasso Alpha For ML Model 3: ", lasso_grid.best_params_['alpha'])

# Doing regularization with lasso and ridge regression

# Ridge Regression model with hyperparameter tuning
ridge_model = Ridge()
ridge_param_grid = {'alpha': [0.01, 0.1, 1, 10]}
ridge_grid = GridSearchCV(ridge_model, ridge_param_grid, cv=5)
ridge_grid.fit(x_train, y_train)

# Evaluate the Ridge Regression model
ridge_predictions = ridge_grid.predict(x_test)
ridge_mse = mean_squared_error(y_test, ridge_predictions)
ridge_r2 = r2_score(y_test, ridge_predictions)
print("Ridge Regression MSE For ML Model 3: ", ridge_mse)
print("Ridge Regression R-squared For ML Model 3: ", ridge_r2)
print("Best Ridge Alpha For ML Model 3: ", ridge_grid.best_params_['alpha'])
```

```
Shape of x_train (399, 11)
Shape of x_test (172, 11)
Shape of y_train (399,)
Shape of y_train (172,)
Linear Regression MSE For Model 3:  0.05602293438336937
Linear Regression RMSE For Model 3:  0.2366916440928352
Linear Regression R-squared For Model 3:  0.8390255731829883
```



Insights from linar model 3:

Linear Regression Performance: Mean Squared Error (MSE): 0.056 Root Mean Squared Error (RMSE): 0.237 R-squared: 0.839 **Insight:** The linear regression model explains approximately 83.9% of the variance in the dependent variable.

Lasso Regression Performance: MSE: 0.057 R-squared: 0.835 Best Alpha: 0.01 **Insight:** Lasso regression performs slightly lower than linear regression with a slightly higher MSE and a similar R-squared value.

Ridge Regression Performance: MSE: 0.056 R-squared: 0.840 Best Alpha: 1 **Insight:** Ridge regression yields results comparable to linear regression but with a slightly lower MSE and a slightly better R-squared value.

ML Model - 4

Considering subjects of only 3rd year

```
x = grades_data[['MT-331', 'EF-303', 'HS-304', 'CS-301', 'CS-302',
       'TC-383', 'EL-332', 'CS-318', 'CS-306', 'CS-312', 'CS-317']]

y = grades_data['CGPA']

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.30,random_state=248)

print("Shape of x_train",x_train.shape)
print("Shape of x_test",x_test.shape)
print("Shape of y_train",y_train.shape)
print("Shape of y_train",y_test.shape)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting linear regressio to training set
LR = LinearRegression()
LR.fit(x_train, y_train)

# Predicting on test set results

y_pred = LR.predict(x_test)

y_pred

# Evaluate the Linear Regression model
LR_predictions = LR.predict(x_test)
LR_mse = mean_squared_error(y_test, LR_predictions)
LR_RMSE = math.sqrt(mean_squared_error(y_test,y_pred))
LR_r2 = r2_score(y_test, LR_predictions)
print("Linear Regression MSE For Model 4: ", LR_mse)
print("Linear Regression RMSE For Model 4: ", LR_RMSE)
print("Linear Regression R-squared For Model 4: ", LR_r2)


plt.scatter(y_test,y_pred)
plt.xlabel('Actual Demand')
plt.ylabel('Predicted Demand')
plt.title('Actual Vs model Predicted')
plt.show()

# Will check for Lasso and ridge regression on model for better performance

# Lasso Regression model with hyperparameter tuning
lasso_model = Lasso()
lasso_param_grid = {'alpha': [0.01, 0.1, 1, 10]}
lasso_grid = GridSearchCV(lasso_model, lasso_param_grid, cv=5)
lasso_grid.fit(x_train, y_train)

# Evaluate the Lasso Regression model
lasso_predictions = lasso_grid.predict(x_test)
lasso_mse = mean_squared_error(y_test, lasso_predictions)
lasso_r2 = r2_score(y_test, lasso_predictions)
print("Lasso Regression MSE for ML Model 4: ", lasso_mse)
print("Lasso Regression R-squared for ML Model 4: ", lasso_r2)
print("Best Lasso Alpha For ML Model 4: ", lasso_grid.best_params_['alpha'])

# Doing regularization with lasso and ridge regression

# Ridge Regression model with hyperparameter tuning
ridge_model = Ridge()
ridge_param_grid = {'alpha': [0.01, 0.1, 1, 10]}
ridge_grid = GridSearchCV(ridge_model, ridge_param_grid, cv=5)
ridge_grid.fit(x_train, y_train)

# Evaluate the Ridge Regression model
ridge_predictions = ridge_grid.predict(x_test)
ridge_mse = mean_squared_error(y_test, ridge_predictions)
ridge_r2 = r2_score(y_test, ridge_predictions)
print("Ridge Regression MSE For ML Model 4: ", ridge_mse)
print("Ridge Regression R-squared For ML Model 4: ", ridge_r2)
print("Best Ridge Alpha For ML Model 4: ", ridge_grid.best_params_['alpha'])
```

```
Shape of x_train (399, 11)
Shape of x_test (172, 11)
Shape of y_train (399,)
Shape of y_train (172,)
Linear Regression MSE For Model 4:  0.05166467837965144
```

Insights from Linear Model 4:

Linear Regression Performance: MSE: 0.052 RMSE: 0.227 R-squared: 0.852 **Insight:** The linear regression model explains approximately 85.2% of the variance in the dependent variable.

Lasso Regression Performance: MSE: 0.055 R-squared: 0.841 Best Alpha: 0.01 **Insight:** Lasso regression slightly underperforms compared to linear regression with a slightly higher MSE and a slightly lower R-squared value.

Ridge Regression Performance: MSE: 0.051 R-squared: 0.854 Best Alpha: 1 **Insight:** Ridge regression outperforms both Linear and Lasso regression with a slightly lower MSE and a slightly better R-squared value.

ML Model - 5

Considering subjects of only 4th year

```
x = grades_data[['MT-442',
       'CS-403', 'CS-421', 'CS-406', 'CS-414', 'CS-419', 'CS-423', 'CS-412']]

y = grades_data['CGPA']

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.30,random_state=248)

print("Shape of x_train",x_train.shape)
print("Shape of x_test",x_test.shape)
print("Shape of y_train",y_train.shape)
print("Shape of y_train",y_test.shape)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting linear regressio to training set
LR = LinearRegression()
LR.fit(x_train, y_train)

# Predicting on test set results

y_pred = LR.predict(x_test)

y_pred

# Evaluate the Linear Regression model
LR_predictions = LR.predict(x_test)
LR_mse = mean_squared_error(y_test, LR_predictions)
LR_RMSE = math.sqrt(mean_squared_error(y_test,y_pred))
LR_r2 = r2_score(y_test, LR_predictions)
print("Linear Regression MSE For Model 5: ", LR_mse)
print("Linear Regression RMSE For Model 5: ", LR_RMSE)
print("Linear Regression R-squared For Model 5: ", LR_r2)


plt.scatter(y_test,y_pred)
plt.xlabel('Actual Demand')
plt.ylabel('Predicted Demand')
plt.title('Actual Vs model Predicted')
plt.show()

# Will check for Lasso and ridge regression on model for better performance

# Lasso Regression model with hyperparameter tuning
lasso_model = Lasso()
lasso_param_grid = {'alpha': [0.01, 0.1, 1, 10]}
lasso_grid = GridSearchCV(lasso_model, lasso_param_grid, cv=5)
lasso_grid.fit(x_train, y_train)

# Evaluate the Lasso Regression model
lasso_predictions = lasso_grid.predict(x_test)
lasso_mse = mean_squared_error(y_test, lasso_predictions)
lasso_r2 = r2_score(y_test, lasso_predictions)
print("Lasso Regression MSE for ML Model 5: ", lasso_mse)
print("Lasso Regression R-squared for ML Model 5: ", lasso_r2)
print("Best Lasso Alpha For ML Model 5: ", lasso_grid.best_params_['alpha'])

# Doing regularization with lasso and ridge regression

# Ridge Regression model with hyperparameter tuning
ridge_model = Ridge()
ridge_param_grid = {'alpha': [0.01, 0.1, 1, 10]}
ridge_grid = GridSearchCV(ridge_model, ridge_param_grid, cv=5)
ridge_grid.fit(x_train, y_train)

# Evaluate the Ridge Regression model
ridge_predictions = ridge_grid.predict(x_test)
ridge_mse = mean_squared_error(y_test, ridge_predictions)
ridge_r2 = r2_score(y_test, ridge_predictions)
print("Ridge Regression MSE For ML Model 5: ", ridge_mse)
print("Ridge Regression R-squared For ML Model 5: ", ridge_r2)
print("Best Ridge Alpha For ML Model 5: ", ridge_grid.best_params_['alpha'])
```