About The Dataset

The dataset contains the academic performance of students in various courses and their final CGPA. Each row represents a student and each column represents a course or the CGPA. The courses are coded by the department and the year of study. The CGPA is calculated based on the grades in all the courses. The goal is to predict the CGPA of a student based on their grades.

In [1]:
```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
import matplotlib.pyplot as plt
```

In [3]:
```
df=pd.read_csv('Grades.csv')
df
```

Out[3]:

	Seat No.	PH-121	HS-101	CY-105	HS-105/12	MT-111	CS-105	CS-106	EL-102	EE-119	...	CS-312	CS-317	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-	C-	C-	C-	A-	A	C-	B	A-	2.205
1	CS-97002	A	D	D+	D	B-	C	D	A	D+	...	D+	D	C	D	A-	B-	C	C	B	2.008
2	CS-97003	A	B	A	B-	B+	A	B-	B+	A-	...	B	B	A	C	A	A	A	A-	A	3.608
3	CS-97004	D	C+	D+	D	D	A-	D+	C-	D	...	D+	C	D+	C-	B-	B	C+	C+	C+	1.906
4	CS-97005	A-	A-	A-	B+	A	A	A-	B+	A	...	B-	B+	B+	B-	A-	A	A-	A-	A	3.448
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
566	CS-97567	B	A	A	A-	A+	A	A-	A-	A+	...	A-	A-	A	A	A	B+	B+	B	A	3.798
567	CS-97568	A+	A	A	A	A	A	A	A-	A	...	B+	B+	A	A	A-	B	A-	C	A-	3.772
568	CS-97569	B	A	A-	B+	A	A	A	A	A	...	A-	B	A	B+	A	C	B+	A-	A-	3.470
569	CS-97570	A	B+	D	A	D	D+	B-	C-	B-	...	D	B	B	C-	D	C	B	B-	C	2.193
570	CS-97571	C	D	D	C	C	D+	B	C+	C	...	C+	C	B-	D	F	C-	B+	D	C-	1.753

571 rows × 43 columns

As can be seen, there are more columns than rows.

Let's examine the data shape.

In [4]:
```
df.shape
```

Out[4]:
```
(571, 43)
```

As we can see, there are 43 columns and 571 rows overall.

In [5]:
```
df.shape[0] #overall rows
```

Out[5]:
```
571
```

In [6]:
```
df.shape[1] #overall columns
```

Out[6]:
```
43
```

Now determinig whether this data set contains any null values.

In [7]:
```
df.isnull()
```

Out[7]:

	Seat No.	PH-121	HS-101	CY-105	HS-105/12	MT-111	CS-105	CS-106	EL-102	EE-119	...	CS-312	CS-317	CS-403	CS-421	CS-406	CS-414	CS-419	CS-423	CS-412	CGP
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
566	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal
567	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal
568	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal
569	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal
570	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fal

571 rows × 43 columns

As we can see in the current data frame, observing in sum.

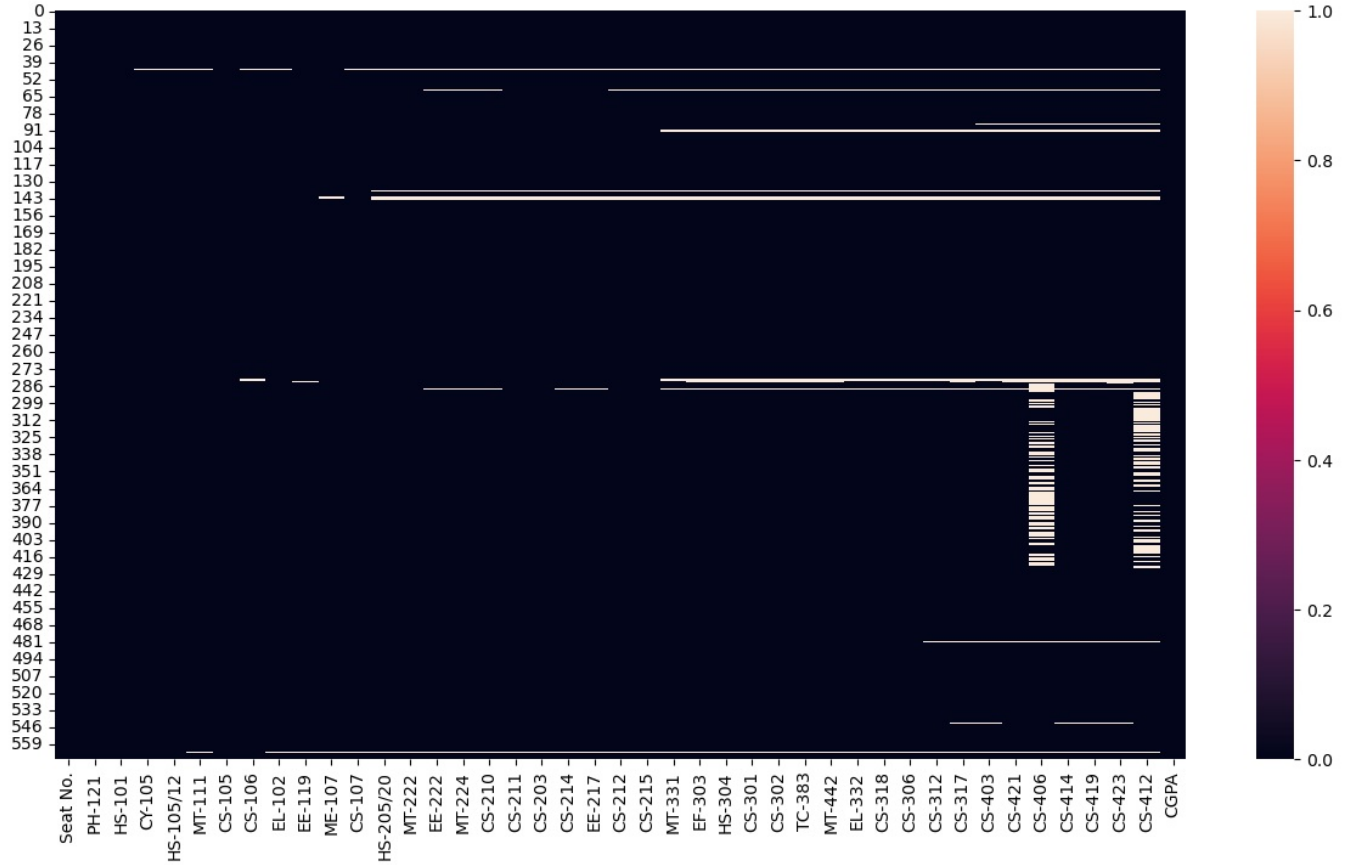
In [8]:
```
df.isnull().sum()
```

Out[8]:
```
Seat No.      0
PH-121        0
HS-101        0
CY-105        1
HS-105/12     1
MT-111        2
CS-105        0
CS-106        2
EL-102        2
EE-119        2
ME-107        2
CS-107        2
HS-205/20     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
MT-442       10
EL-332        9
CS-318        9
CS-306        9
CS-312       10
CS-317       12
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
```

Taking a look at the heat map now that we have seen that several columns have null values.

In [9]:
```
plt.figure(figsize=(15,8))
sns.heatmap(df.isnull())
```

Out[9]:
```
<AxesSubplot:>
```



As we can see, the CS-406 and CS-412 have a large number of null values.

Now we use its mode to fill in all of the null values due to the object type of the data.

In [10]:
```
df.dtypes
```

Out[10]:
```
Seat No.      object
PH-121        object
HS-101        object
CY-105        object
HS-105/12     object
MT-111        object
CS-105        object
CS-106        object
EL-102        object
EE-119        object
ME-107        object
CS-107        object
HS-205/20     object
MT-222        object
EE-222        object
MT-224        object
CS-210        object
CS-211        object
CS-203        object
CS-214        object
EE-217        object
CS-212        object
CS-215        object
MT-331        object
EF-303        object
HS-304        object
CS-301        object
CS-302        object
TC-383        object
MT-442        object
EL-332        object
CS-318        object
CS-306        object
CS-312        object
CS-317        object
CS-403        object
CS-421        object
CS-406        object
CS-414        object
CS-419        object
CS-423        object
CS-412        object
CGPA         float64
dtype: object
```

In [11]:
```
df['CS-406'].value_counts()
```

Out[11]:
```
A-    177
A      79
B+     64
B      58
B-     22
C+     22
C      19
D+     14
C-      8
A+      8
F       6
D       5
W       3
WU      1
Name: CS-406, dtype: int64
```

In [12]:
```
df['CS-406']=df['CS-406'].fillna('A-')
```

In [13]:
```
df['CS-301'].value_counts()
```

Out[13]:
```
A-    108
B+     74
B      71
A      66
B-     60
C      41
C+     37
C-     36
D      29
D+     29
A+      9
F       1
Name: CS-301, dtype: int64
```

In [14]:
```
df['CS-301']=df['CS-301'].fillna('A-')
```

In [15]:
```
df['HS-304'].value_counts()
```

Out[15]:
```
A-    128
B      72
B-     70
B+     66
C      58
C+     53
C-     33
A      28
D      20
D+     19
F       6
A+      4
WU      2
W       2
Name: HS-304, dtype: int64
```

In [16]:
```
df['HS-304']=df['HS-304'].fillna('A-')
```

In [17]:
```
df['EF-303'].value_counts()
```

Out[17]:
```
B     112
B-     92
C      61
B+     59
C+     58
C-     56
D+     49
A-     38
D      19
A      14
F       2
WU      1
Name: EF-303, dtype: int64
```

In [18]:
```
df['EF-303']=df['EF-303'].fillna('B')
```

In [19]:
```
df['CS-302'].value_counts()
```

Out[19]:
```
A-    113
B     102
A      86
B+     81
B-     60
C+     32
C      28
D      21
C-     19
D+     10
A+      9
Name: CS-302, dtype: int64
```

In [20]:
```
df['CS-302']=df['CS-302'].fillna('A-')
```

In [21]:
```
df['TC-383'].value_counts()
```

Out[21]:
```
A     105
A-     73
B+     68
B      59
C+     44
D+     44
C-     42
C      42
B-     40
A+     23
D      20
F       1
Name: TC-383, dtype: int64
```

In [22]:
```
df['TC-383']=df['TC-383'].fillna('A')
```

In [23]:
```
df['TC-383'].mode()
```

Out[23]:
```
0    A
Name: TC-383, dtype: object
```

In [24]:
```
df['MT-442'].value_counts()
```

Out[24]:
```
A-    140
A     130
B+     65
B      47
A+     39
B-     30
C-     28
C+     24
C      24
D      20
D+     13
F       1
Name: MT-442, dtype: int64
```

In [25]:
```
df['MT-442'].mode()
```

Out[25]:
```
0    A-
Name: MT-442, dtype: object
```

In [26]:
```
df['MT-442']=df['MT-442'].fillna('A-')
```

In [27]:
```
df['EL-332'].value_counts()
```

Out[27]:
```
A-    96
A     76
B+    68
B     67
B-    62
C     49
C+    38
C-    32
D+    22
D     22
A+    20
F      9
WU     1
Name: EL-332, dtype: int64
```

In [28]:
```
df['EL-332']=df['EL-332'].fillna('A-')
```

In [29]:
```
df['CS-318'].value_counts()
```

Out[29]:
```
A-    89
B-    69
B     68
B+    65
C     53
C+    49
A     42
C-    40
D     36
D+    29
F     10
A+     6
WU     5
W      1
Name: CS-318, dtype: int64
```

In [30]:
```
df['CS-318']=df['CS-318'].fillna('A-')
```

In [31]:
```
df['CS-306'].value_counts()
```

Out[31]:
```
A-    120
B+     75
A      74
B-     64
B      53
C-     40
C      37
C+     36
D      31
D+     18
A+     10
F       3
WU      1
Name: CS-306, dtype: int64
```

In [32]:
```
df['CS-306']=df['CS-306'].fillna('A-')
```

In [33]:
```
df['CS-312'].value_counts()
```

Out[33]:
```
A+    93
A     86
A-    63
D+    52
C     48
C-    44
B+    42
B     37
C+    37
B-    36
D     19
F      2
W      1
WU     1
Name: CS-312, dtype: int64
```

In [34]:
```
df['CS-312']=df['CS-312'].fillna('A+')
```

In [35]:
```
df['CS-317'].value_counts()
```

Out[35]:
```
B-    79
B     75
C     70
A-    66
B+    58
C+    57
A     47
C-    41
D+    34
D     16
A+     9
F      7
Name: CS-317, dtype: int64
```

In [36]:
```
df['CS-317']=df['CS-317'].fillna('B-')
```

In [37]:
```
df['CS-403'].value_counts()
```

Out[37]:
```
A     133
A-    106
B+     62
B-     55
B      54
D+     35
C      33
C+     32
C-     30
A+     15
D       4
Name: CS-403, dtype: int64
```

In [38]:
```
df['CS-403']=df['CS-403'].fillna('A')
```

In [39]:
```
df['CS-421'].value_counts()
```

Out[39]:
```
B     86
B-    74
C     68
C+    61
C-    60
B+    60
A-    47
D+    36
A     25
D     21
F     17
A+     2
W      2
Name: CS-421, dtype: int64
```

In [40]:
```
df['CS-421']=df['CS-421'].fillna('B')
```

In [41]:
```
df['CS-414'].value_counts()
```

Out[41]:
```
A     176
A-    156
B+     62
B      54
B-     21
A+     21
C+     21
C      20
C-     12
F       7
D+      3
D       3
W       2
Name: CS-414, dtype: int64
```

In [42]:
```
df['CS-414']=df['CS-414'].fillna('A')
```

In [43]:
```
df['CS-419'].value_counts()
```

Out[43]:
```
A-    120
B      89
B+     85
B-     78
A      56
C+     46
C      40
C-     20
D+     13
D       7
A+      2
F       2
Name: CS-419, dtype: int64
```

In [44]:
```
df['CS-419']=df['CS-419'].fillna('A-')
```

In [45]:
```
df['CS-423'].value_counts()
```

Out[45]:
```
A-    122
A      78
B+     75
B      65
B-     56
C      45
C+     39
C-     29
D+     25
D      15
F       5
A+      3
Name: CS-423, dtype: int64
```

In [46]:
```
df['CS-423']=df['CS-423'].fillna('A-')
```

In [47]:
```
df['CS-412'].value_counts()
```

Out[47]:
```
A-    157
B+     80
A      77
B      65
B-     37
C+     24
C      19
D+      8
C-      7
F       6
D       5
A+      4
W       3
Name: CS-412, dtype: int64
```

In [48]:
```
df['CS-412']=df['CS-412'].fillna('A-')
```

In [49]:
```
df.dropna(inplace=True)
```

In [50]:
```
df.isnull().sum()
```

Out[50]:
```
Seat No.     0
PH-121       0
HS-101       0
CY-105       0
HS-105/12    0
MT-111       0
CS-105       0
CS-106       0
EL-102       0
EE-119       0
ME-107       0
CS-107       0
HS-205/20    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
MT-442       0
EL-332       0
CS-318       0
CS-306       0
CS-312       0
CS-317       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
```

When treating a null result, we choose the most frequent grade, which is above 10 and below 10, meaning that 2,5,7 are all removed from the dataset.

In [51]:
```
df.shape
```

Out[51]:
```
(561, 43)
```

Our dataset currently comprises 561 rows and 43 columns, as can be seen.

In [52]:
```
df.columns
```

Out[52]:
```
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', 'MT-442', 'EL-332', 'CS-318', 'CS-306', 'CS-312',
       'CS-317', 'CS-403', 'CS-421', 'CS-406', 'CS-414', 'CS-419', 'CS-423',
       'CS-412', 'CGPA'],
      dtype='object')
```

Checking the details in our data now.

In [53]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 561 entries, 0 to 570
Data columns (total 43 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Seat No.   561 non-null    object
 1   PH-121     561 non-null    object
 2   HS-101     561 non-null    object
 3   CY-105     561 non-null    object
 4   HS-105/12  561 non-null    object
 5   MT-111     561 non-null    object
 6   CS-105     561 non-null    object
 7   CS-106     561 non-null    object
 8   EL-102     561 non-null    object
 9   EE-119     561 non-null    object
 10  ME-107     561 non-null    object
 11  CS-107     561 non-null    object
 12  HS-205/20  561 non-null    object
 13  MT-222     561 non-null    object
 14  EE-222     561 non-null    object
 15  MT-224     561 non-null    object
 16  CS-210     561 non-null    object
 17  CS-211     561 non-null    object
 18  CS-203     561 non-null    object
 19  CS-214     561 non-null    object
 20  EE-217     561 non-null    object
 21  CS-212     561 non-null    object
 22  CS-215     561 non-null    object
 23  MT-331     561 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  MT-442     561 non-null    object
 30  EL-332     561 non-null    object
 31  CS-318     561 non-null    object
 32  CS-306     561 non-null    object
 33  CS-312     561 non-null    object
 34  CS-317     561 non-null    object
 35  CS-403     561 non-null    object
 36  CS-421     561 non-null    object
 37  CS-406     561 non-null    object
 38  CS-414     561 non-null    object
 39  CS-419     561 non-null    object
 40  CS-423     561 non-null    object
 41  CS-412     561 non-null    object
 42  CGPA       561 non-null    float64
dtypes: float64(1), object(42)
memory usage: 192.8+ KB
```

As shown, there are 561 rows total, all of which have non-null data.

Now, examining the dataset's data types.

In [54]:
```
df.dtypes
```

Out[54]:
```
Seat No.      object
PH-121        object
HS-101        object
CY-105        object
HS-105/12     object
MT-111        object
CS-105        object
CS-106        object
EL-102        object
EE-119        object
ME-107        object
CS-107        object
HS-205/20     object
MT-222        object
EE-222        object
MT-224        object
CS-210        object
CS-211        object
CS-203        object
CS-214        object
EE-217        object
CS-212        object
CS-215        object
MT-331        object
EF-303        object
HS-304        object
CS-301        object
CS-302        object
TC-383        object
MT-442        object
EL-332        object
CS-318        object
CS-306        object
CS-312        object
CS-317        object
CS-403        object
CS-421        object
CS-406        object
CS-414        object
CS-419        object
CS-423        object
CS-412        object
CGPA         float64
dtype: object
```

As can be seen, all features or input columns are of the object type, but the target data type is float.

Since the seat number column is no longer necessary, we should get rid of it.

In [55]:
```
df.drop(['Seat No.'],axis=1,inplace=True)
df.head()
```

Out[55]:

	PH-121	HS-101	CY-105	HS-105/12	MT-111	CS-105	CS-106	EL-102	EE-119	ME-107	...	CS-312	CS-317	CS-403	CS-421	CS-406	CS-414	CS-419	CS-423	CS-412	CGPA
0	B-	D+	C-	C	C-	D+	D	C-	B-	C-	...	C-	C-	C-	C-	A-	A	C-	B	A-	2.205
1	A	D	D+	D	B-	C	D	A	D+	D	...	D+	D	C	D	A-	B-	C	C	B	2.008
2	A	B	A	B-	B+	A	B-	B+	A-	A-	...	B	B	A	C	A	A	A	A-	A	3.608
3	D	C+	D+	D	D	A-	D+	C-	D	C+	...	D+	C	D+	C-	B-	B	C+	C+	C+	1.906
4	A-	A-	A-	B+	A	A	A-	B+	A	A-	...	B-	B+	B+	B-	A-	A	A-	A-	A	3.448

5 rows × 42 columns

Currently, there are 42 total columns: 41 input columns, 1 target column.

Now, examining each column's unique values.

In [56]:
```
df.columns
```

Out[56]:
```
Index(['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',
       'MT-442', 'EL-332', 'CS-318', 'CS-306', 'CS-312', 'CS-317', 'CS-403',
       'CS-421', 'CS-406', 'CS-414', 'CS-419', 'CS-423', 'CS-412', 'CGPA'],
      dtype='object')
```

In [57]:
```
uniques=['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',
       'MT-442', 'EL-332', 'CS-318', 'CS-306', 'CS-312', 'CS-317', 'CS-403',
```

```
       'CS-421', 'CS-406', 'CS-414', 'CS-419', 'CS-423', 'CS-412']

for i in uniques:
    print(str(i),'=',df[i].unique(),'in this column has total',len(df[i].unique()),'unique values.','\n')
```

```
PH-121 = ['B-' 'A' 'D' 'A-' 'B+' 'B' 'C+' 'C' 'C-' 'D+' 'A+'] in this column has total 11 unique values.

HS-101 = ['D+' 'D' 'B' 'C+' 'A-' 'B-' 'C-' 'B+' 'C' 'A' 'A+'] in this column has total 11 unique values.

CY-105 = ['C-' 'D+' 'A' 'A-' 'B' 'C+' 'B+' 'B-' 'C' 'D' 'A+'] in this column has total 11 unique values.

HS-105/12 = ['C' 'D' 'B-' 'B+' 'D+' 'B' 'C-' 'C+' 'A-' 'A' 'A+'] in this column has total 11 unique values.

MT-111 = ['C-' 'B-' 'B+' 'D' 'A' 'C+' 'A-' 'C' 'B' 'D+' 'A+' 'F'] in this column has total 12 unique values.

CS-105 = ['D+' 'C' 'A' 'A-' 'B' 'B-' 'B+' 'C+' 'C-' 'A+' 'D'] in this column has total 11 unique values.

CS-106 = ['D' 'B-' 'D+' 'A-' 'C+' 'A' 'C-' 'C' 'B' 'B+' 'A+'] in this column has total 11 unique values.

EL-102 = ['C-' 'A' 'B+' 'B' 'A-' 'B-' 'C' 'C+' 'D+' 'D' 'A+'] in this column has total 11 unique values.

EE-119 = ['B-' 'D+' 'A-' 'D' 'A' 'B+' 'B' 'C-' 'C+' 'C' 'A+'] in this column has total 11 unique values.

ME-107 = ['C-' 'D' 'A-' 'C+' 'B+' 'A' 'D+' 'C' 'B' 'B-' 'A+'] in this column has total 11 unique values.

CS-107 = ['C-' 'B+' 'B-' 'D' 'C' 'A-' 'C+' 'D+' 'B' 'A' 'A+'] in this column has total 11 unique values.

HS-205/20 = ['B+' 'C+' 'A-' 'B' 'A' 'B-' 'D' 'A+' 'C' 'D+' 'C-'] in this column has total 11 unique values.

MT-222 = ['D' 'B-' 'A' 'D+' 'C' 'B+' 'A-' 'B' 'C+' 'C-' 'F' 'A+'] in this column has total 12 unique values.

EE-222 = ['A-' 'C-' 'A' 'B-' 'B+' 'C+' 'D+' 'B' 'C' 'A+' 'D' 'F'] in this column has total 12 unique values.

MT-224 = ['B-' 'D+' 'A' 'D' 'A-' 'C-' 'C' 'B+' 'C+' 'B' 'A+' 'F'] in this column has total 12 unique values.

CS-210 = ['C+' 'D+' 'A' 'C' 'A-' 'B+' 'B-' 'B' 'D' 'A+' 'C-' 'F'] in this column has total 12 unique values.

CS-211 = ['D+' 'D' 'A+' 'B+' 'B-' 'B' 'A-' 'C-' 'C+' 'C' 'A' 'F'] in this column has total 12 unique values.

CS-203 = ['D+' 'C' 'A' 'C+' 'A-' 'B+' 'B' 'B-' 'C-' 'D' 'A+'] in this column has total 11 unique values.

CS-214 = ['D' 'A-' 'B' 'C' 'B-' 'C+' 'A' 'C-' 'B+' 'D+' 'A+'] in this column has total 11 unique values.

EE-217 = ['A' 'B-' 'D+' 'A-' 'A+' 'C' 'B+' 'B' 'C-' 'D' 'F' 'C+'] in this column has total 12 unique values.

CS-212 = ['D' 'C' 'A-' 'C+' 'B-' 'D+' 'B+' 'B' 'A' 'C-' 'A+'] in this column has total 11 unique values.

CS-215 = ['C-' 'D' 'A' 'A-' 'C+' 'C' 'B+' 'B-' 'D+' 'B' 'A+'] in this column has total 11 unique values.

MT-331 = ['C' 'D+' 'A' 'C+' 'B+' 'A-' 'B' 'B-' 'A+' 'D' 'C-' 'F'] in this column has total 12 unique values.

EF-303 = ['C-' 'D' 'C+' 'B+' 'C' 'D+' 'A-' 'B-' 'A' 'B' 'WU' 'F'] in this column has total 12 unique values.

HS-304 = ['C+' 'C-' 'A' 'B-' 'A-' 'B+' 'B' 'C' 'D' 'D+' 'A+' 'WU' 'F' 'W'] in this column has total 14 unique v
alues.

CS-301 = ['B' 'A-' 'B+' 'D' 'B-' 'D+' 'C' 'C+' 'C-' 'A' 'F' 'A+'] in this column has total 12 unique values.

CS-302 = ['C' 'D+' 'A-' 'B-' 'C+' 'B' 'D' 'B+' 'A' 'C-' 'A+'] in this column has total 11 unique values.

TC-383 = ['C+' 'C-' 'B' 'D+' 'C' 'B+' 'B-' 'D' 'A-' 'A' 'F' 'A+'] in this column has total 12 unique values.

MT-442 = ['B+' 'C-' 'A' 'A-' 'B' 'A+' 'C+' 'B-' 'C' 'D+' 'D' 'F'] in this column has total 12 unique values.

EL-332 = ['C' 'B+' 'D+' 'A-' 'B' 'A' 'B-' 'C-' 'C+' 'D' 'A+' 'F' 'WU'] in this column has total 13 unique value
s.

CS-318 = ['C-' 'A-' 'B+' 'B-' 'C' 'B' 'C+' 'D' 'D+' 'A' 'A+' 'WU' 'F' 'W'] in this column has total 14 unique v
alues.

CS-306 = ['C' 'D' 'A-' 'C-' 'B+' 'C+' 'B-' 'B' 'A' 'D+' 'A+' 'F' 'WU'] in this column has total 13 unique value
s.

CS-312 = ['C-' 'D+' 'B' 'B-' 'C' 'B+' 'C+' 'A-' 'D' 'A' 'A+' 'W' 'F' 'WU'] in this column has total 14 unique v
alues.

CS-317 = ['C-' 'D' 'B' 'C' 'B+' 'B-' 'A-' 'D+' 'C+' 'A' 'A+' 'F'] in this column has total 12 unique values.

CS-403 = ['C-' 'C' 'A' 'D+' 'B+' 'C+' 'B-' 'A-' 'B' 'A+' 'D'] in this column has total 11 unique values.

CS-421 = ['C-' 'D' 'C' 'B-' 'C+' 'B+' 'D+' 'B' 'A' 'A-' 'F' 'A+' 'W'] in this column has total 13 unique values
.

CS-406 = ['A-' 'A' 'B-' 'B' 'B+' 'C+' 'D' 'D+' 'C-' 'C' 'A+' 'F' 'W' 'WU'] in this column has total 14 unique v
alues.

CS-414 = ['A' 'B-' 'B' 'A-' 'B+' 'A+' 'C' 'C-' 'C+' 'F' 'D+' 'W' 'D'] in this column has total 13 unique values
.
```

```
CS-419 = ['C-' 'C' 'A' 'C+' 'A-' 'B+' 'B-' 'B' 'D+' 'A+' 'D' 'F'] in this column has total 12 unique values.

CS-423 = ['B' 'C' 'A-' 'C+' 'B-' 'B+' 'C-' 'A' 'D+' 'D' 'A+' 'F'] in this column has total 12 unique values.

CS-412 = ['A-' 'B' 'A' 'C+' 'B+' 'C-' 'B-' 'A+' 'C' 'D+' 'F' 'W' 'D'] in this column has total 13 unique values
.
```

As we can see from the data above, each column has an average of 10 to 12 different grades.

We have now completed checking for unique column values.

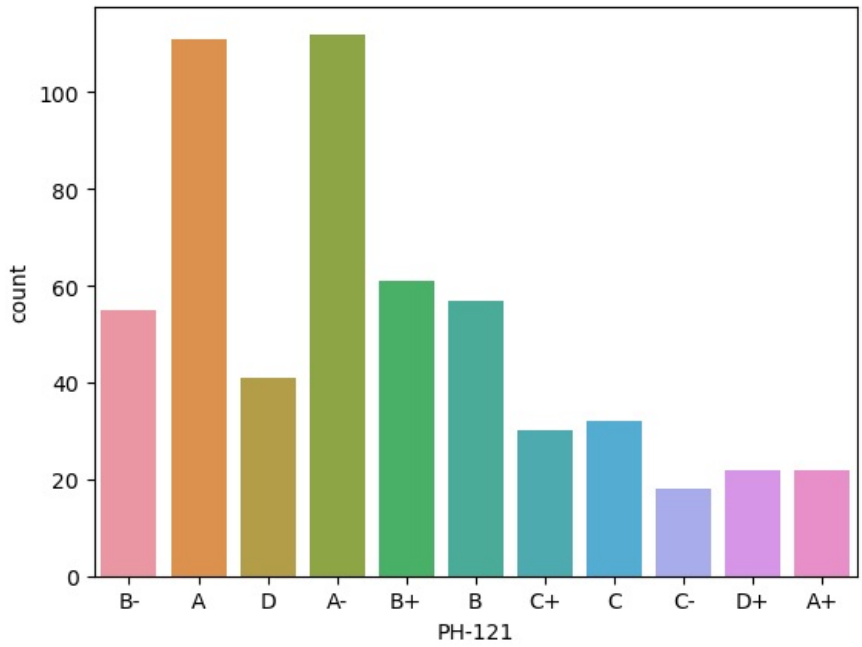
Now, data is being visualised.

Univariable Analysis

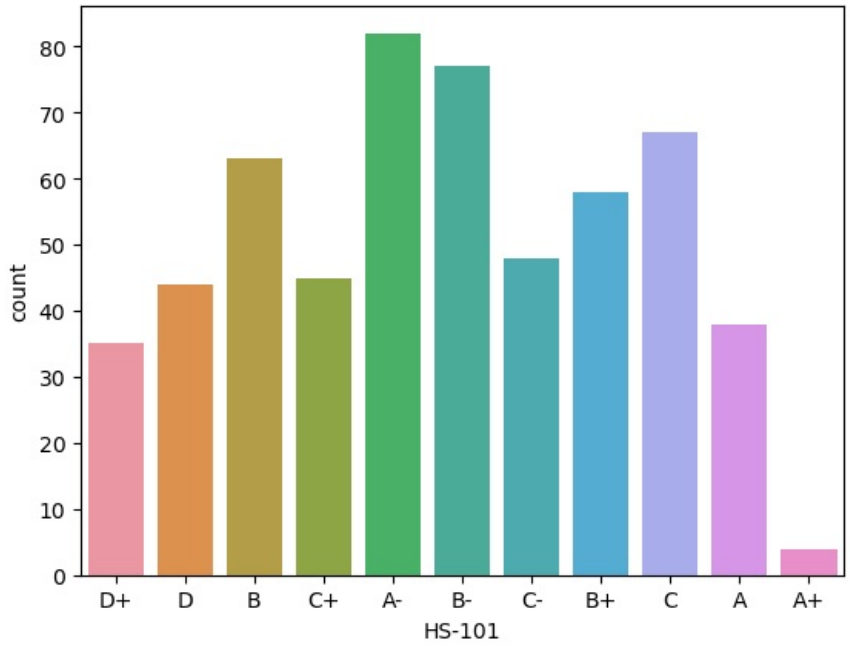
We can use the count plot and its unique values for univariate analysis.

In [58]:
```
Count=['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',
       'MT-442', 'EL-332', 'CS-318', 'CS-306', 'CS-312', 'CS-317', 'CS-403',
       'CS-421', 'CS-406', 'CS-414', 'CS-419', 'CS-423', 'CS-412']

for i in Count:
    print(str(i))
    sns.countplot(df[i])
    plt.show()
```

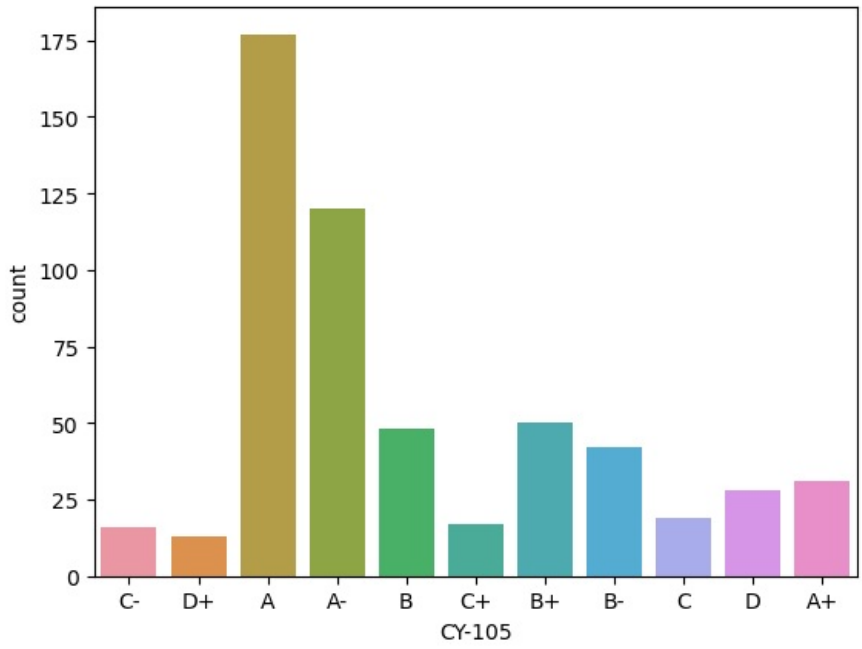
```
PH-121
```



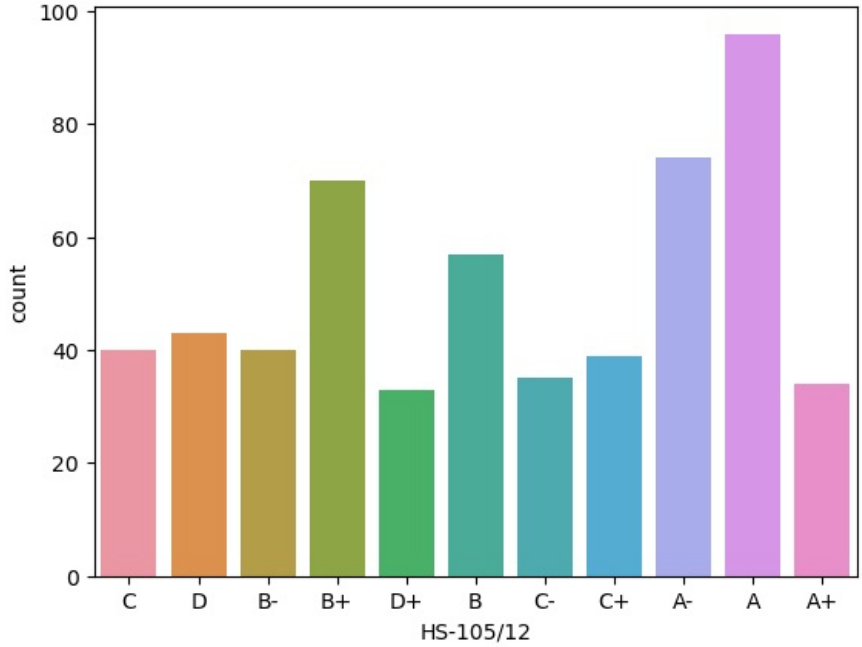

```
HS-101
```



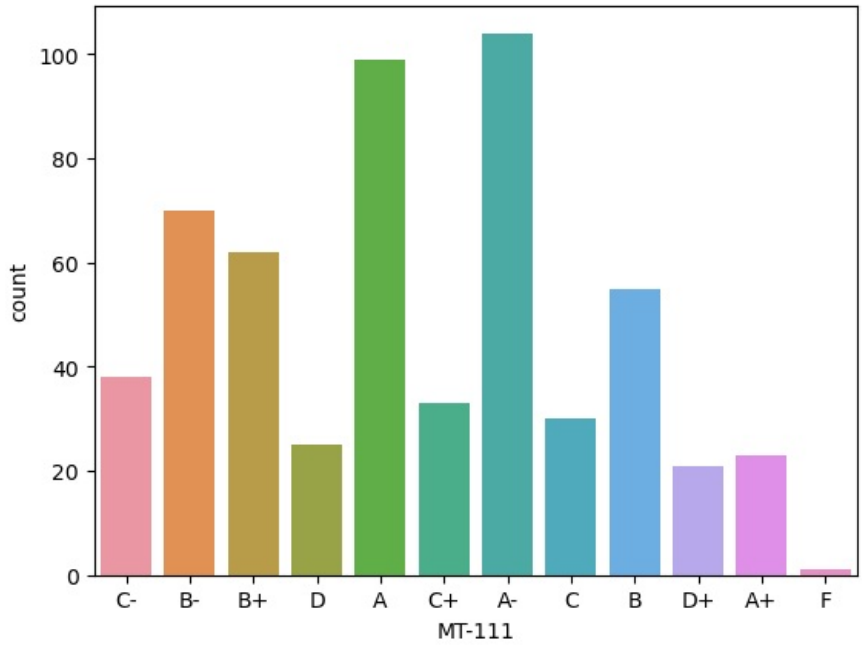
CY-105



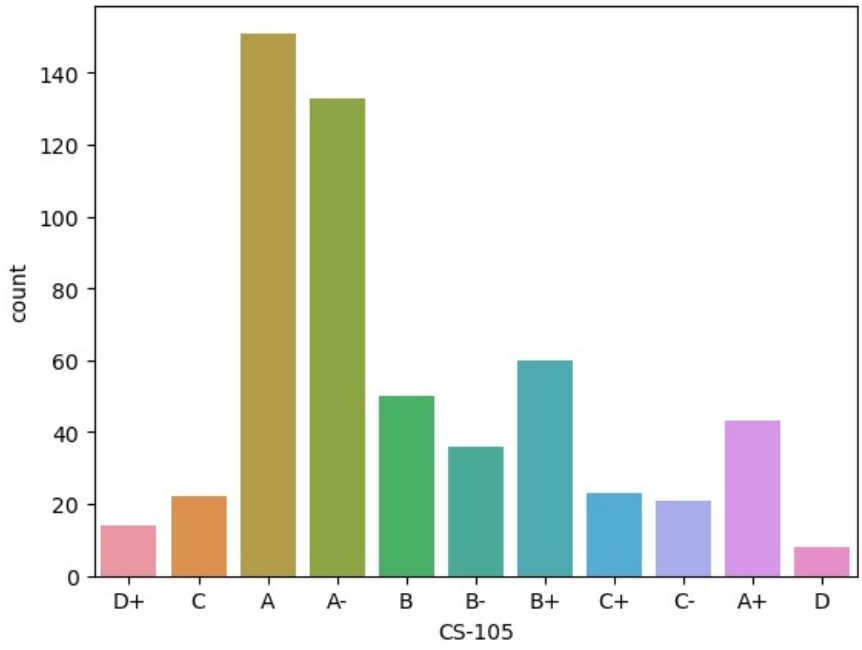
HS-105/12


100
80
60
40
20
0
count
C
D
B-
B+
D+
B
C-
C+
A-
A
A+
HS-105/12

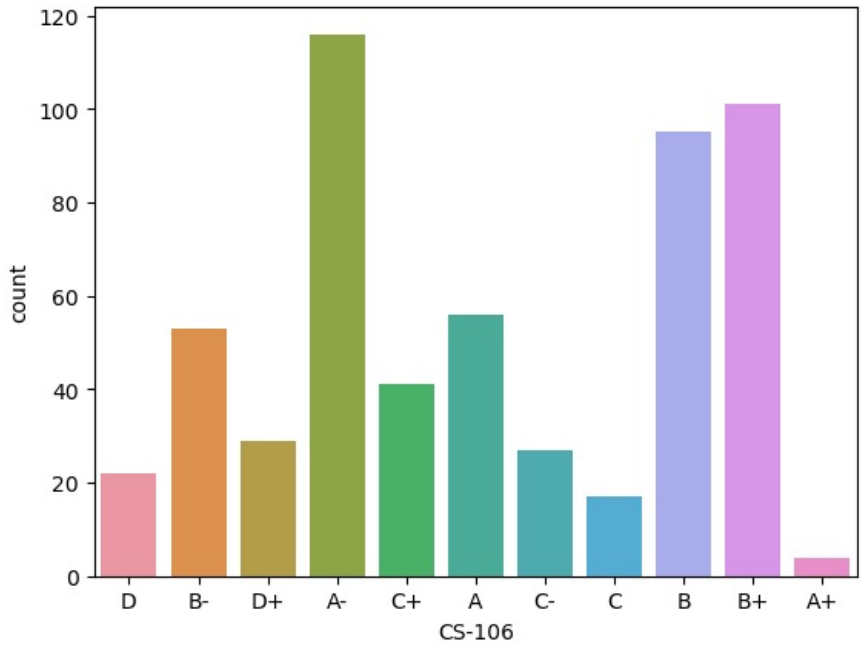
MT-111


100
80
60
40
20
0
count
C-
B-
B+
D
A
C+
A-
C
B
D+
A+
F
MT-111

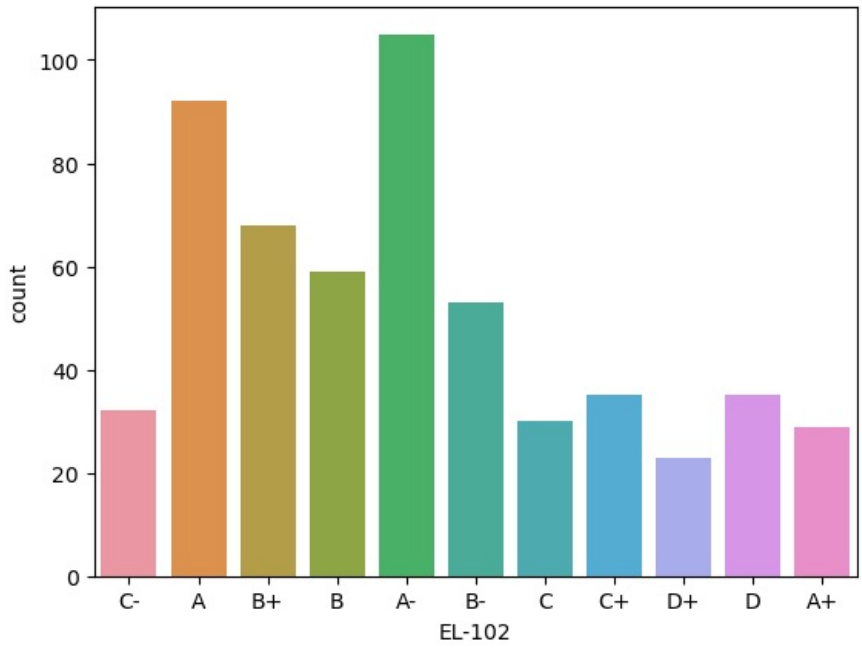
CS-105


count
0
20
40
60
80
100
120
140
D+
C
A
A-
B
B-
B+
C+
C-
A+
D
CS-105

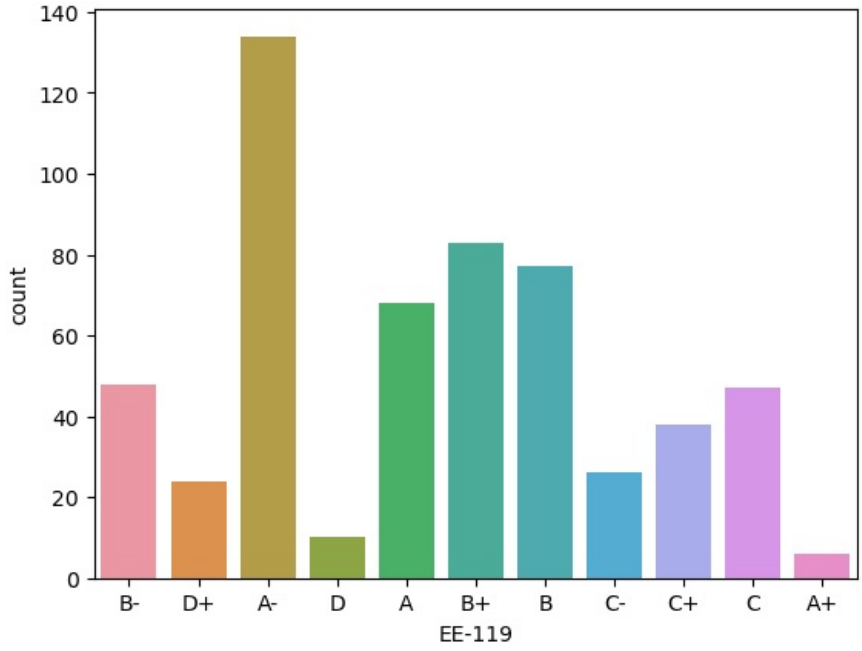
CS-106


count
0
20
40
60
80
100
120
D
B-
D+
A-
C+
A
C-
C
B
B+
A+
CS-106

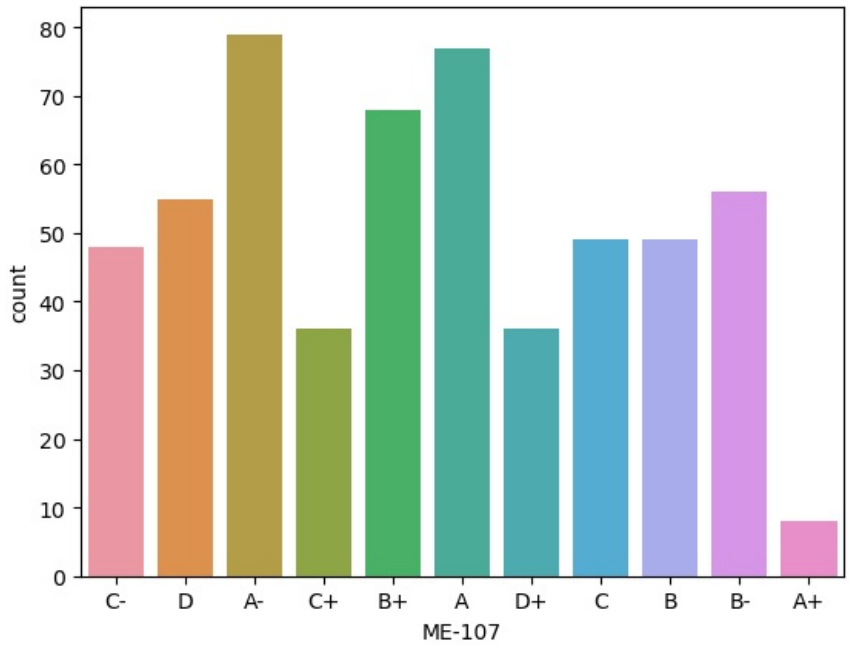
EL-102


count
100
80
60
40
20
0
C-
A
B+
B
A-
B-
C
C+
D+
D
A+
EL-102

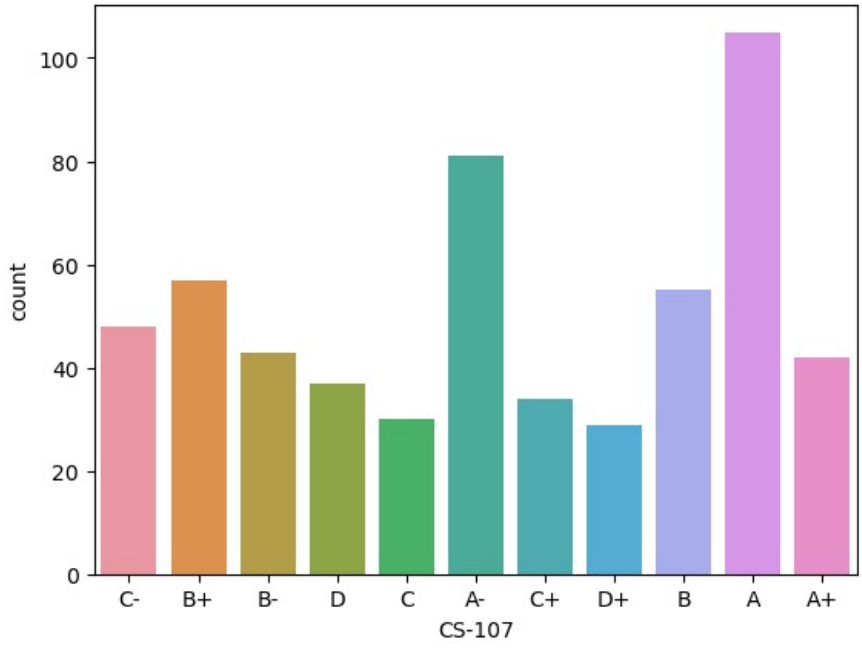
EE-119


count
140
120
100
80
60
40
20
0
B-
D+
A-
D
A
B+
B
C-
C+
C
A+
EE-119

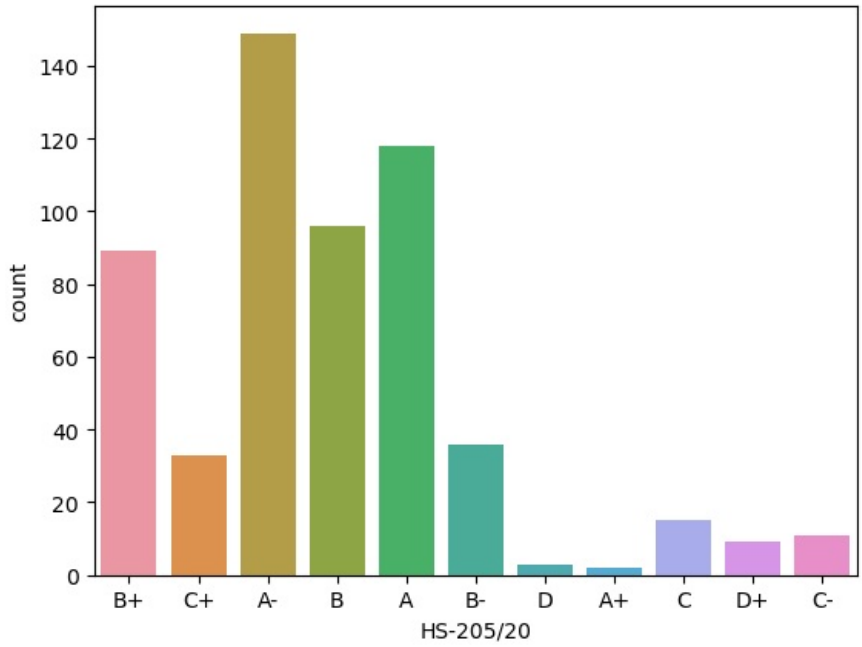
ME-107



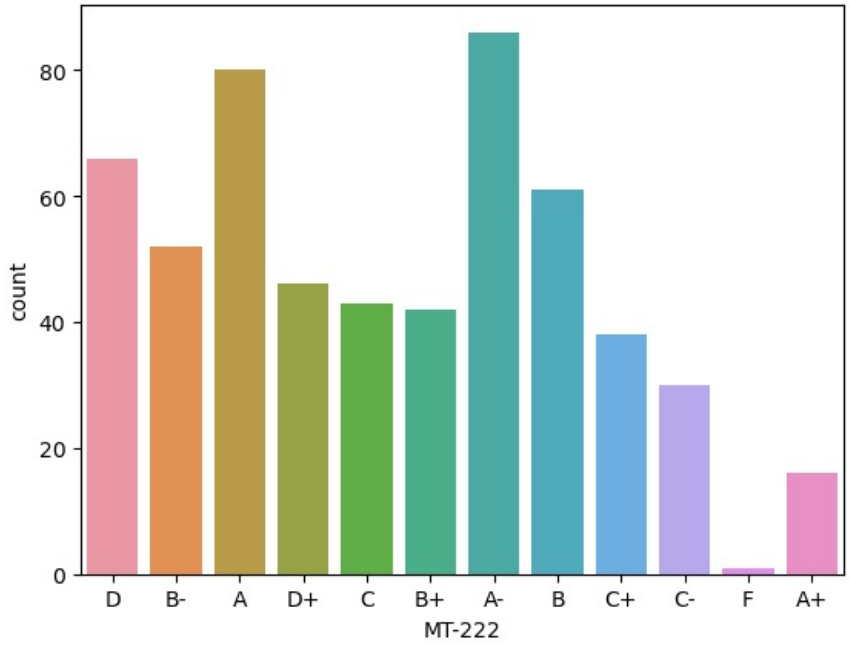
CS-107



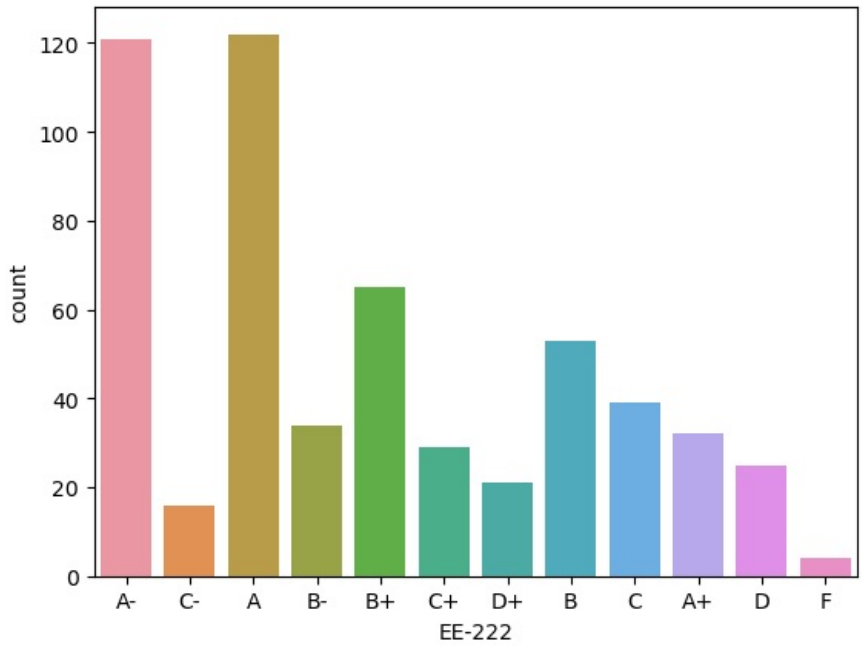
HS-205/20


count
140
120
100
80
60
40
20
0
B+
C+
A-
B
A
B-
D
A+
C
D+
C-
HS-205/20

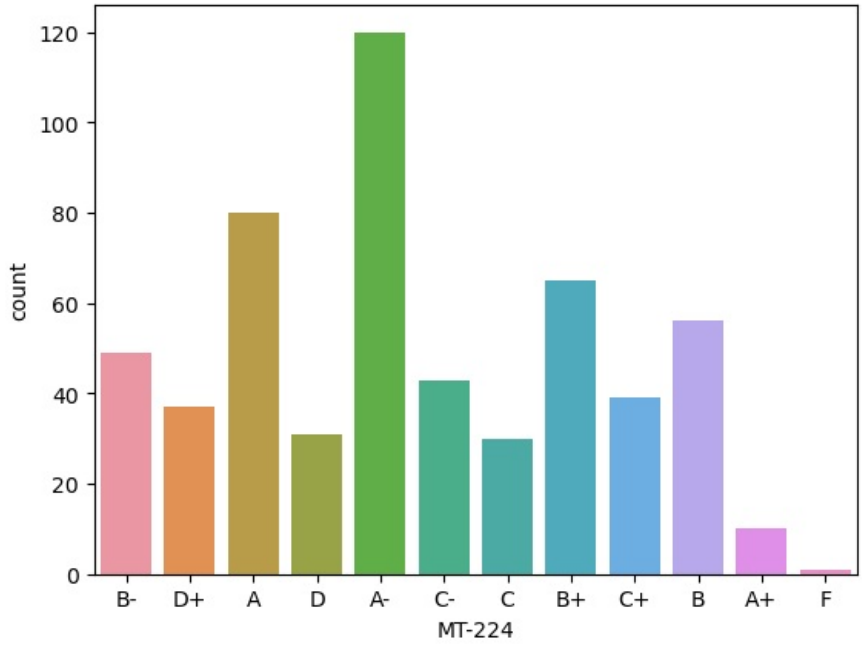
MT-222


count
80
60
40
20
0
D
B-
A
D+
C
B+
A-
B
C+
C-
F
A+
MT-222

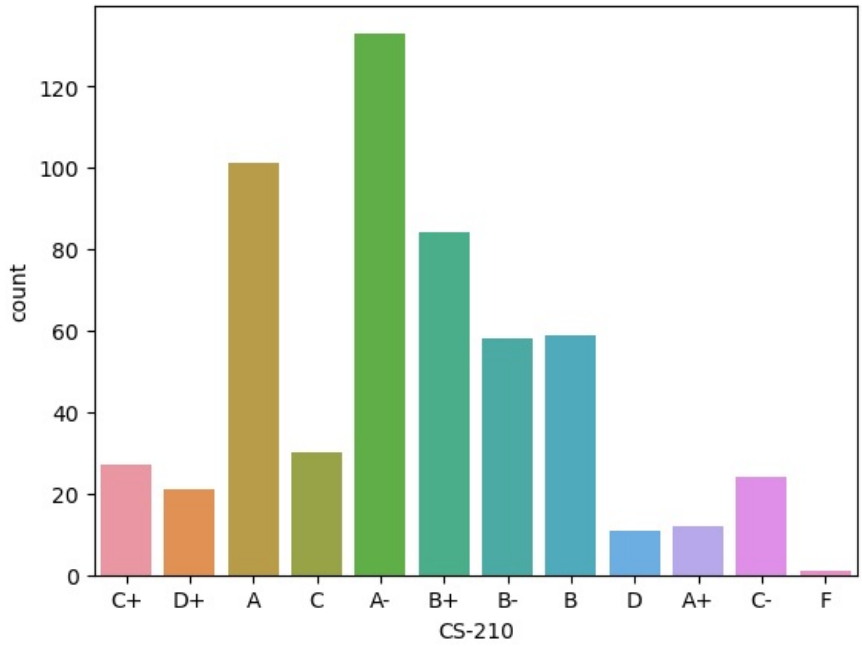
EE-222


120
100
80
60
40
20
0
count
A-
C-
A
B-
B+
C+
D+
B
C
A+
D
F
EE-222

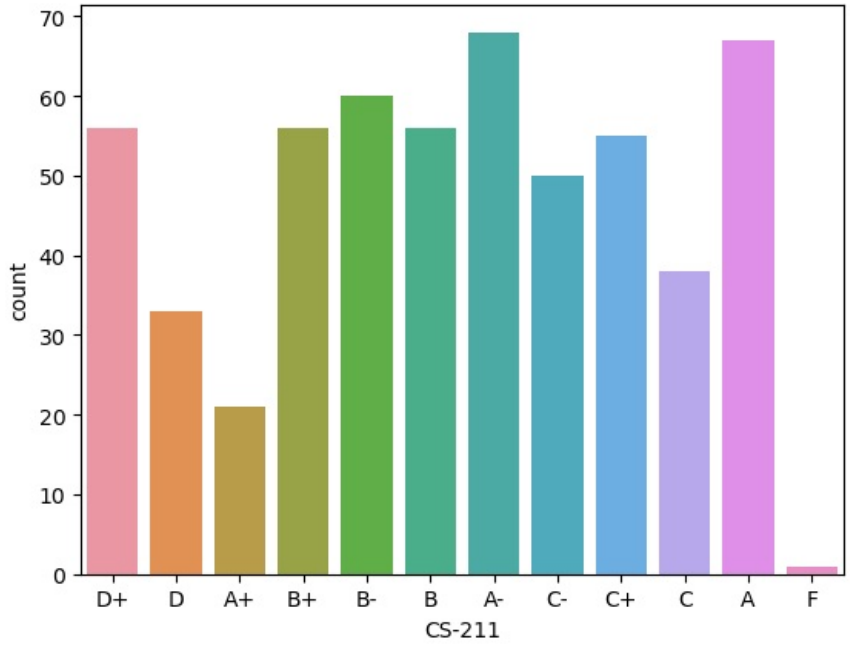
MT-224


120
100
80
60
40
20
0
count
B-
D+
A
D
A-
C-
C
B+
C+
B
A+
F
MT-224

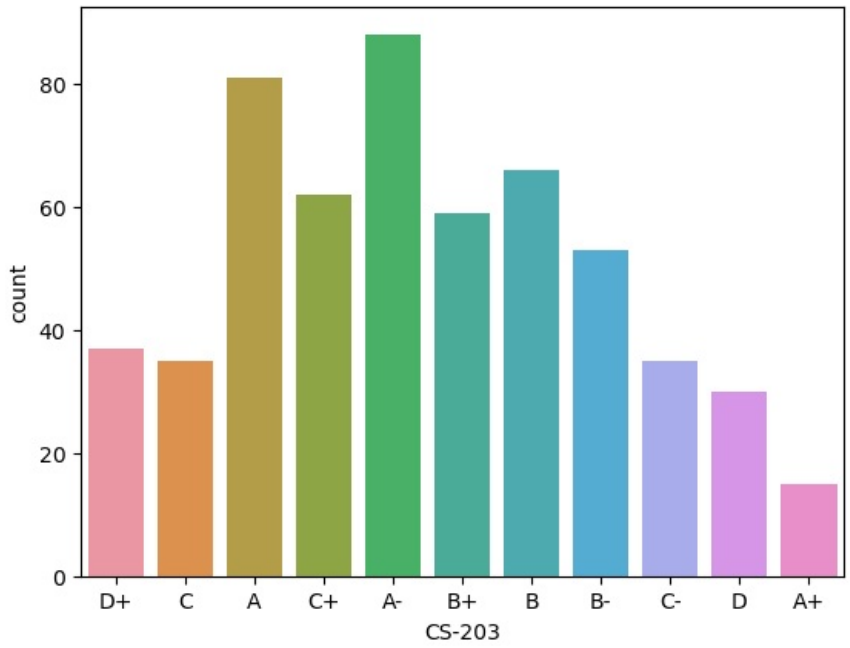
CS-210


120
100
80
60
40
20
0
count
C+
D+
A
C
A-
B+
B-
B
D
A+
C-
F
CS-210

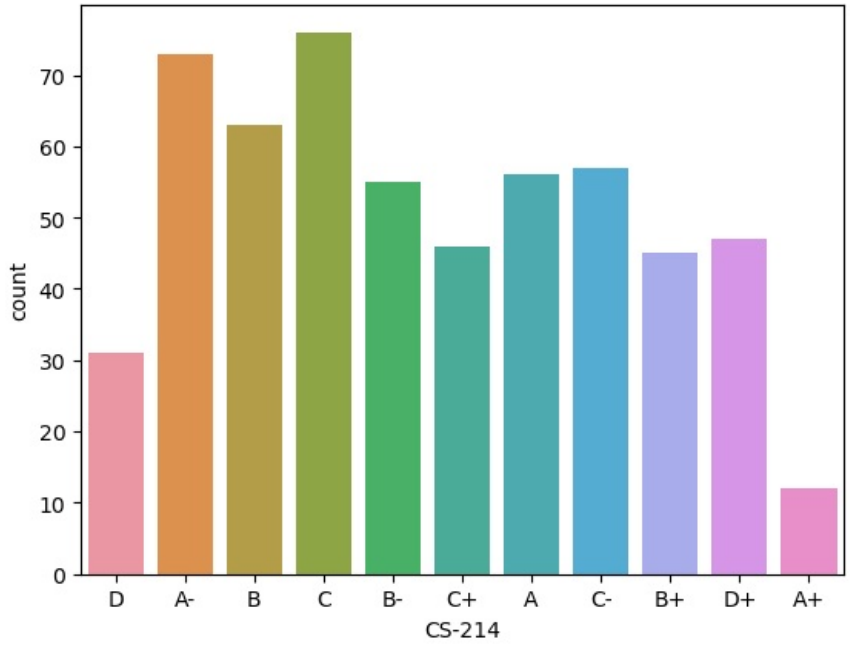
CS-211


70
60
50
40
30
20
10
0
count
D+
D
A+
B+
B-
B
A-
C-
C+
C
A
F
CS-211

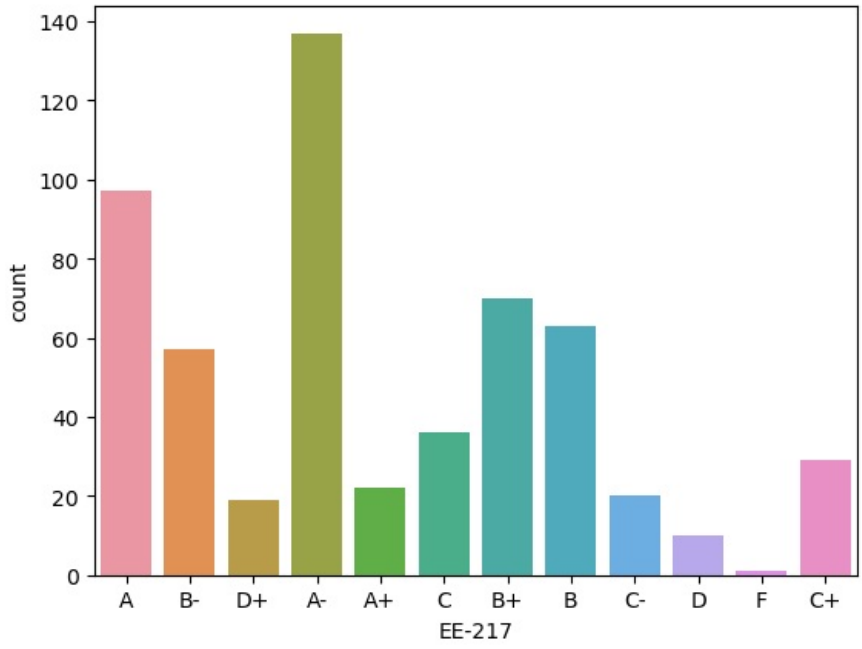
CS-203


count
0
20
40
60
80
D+
C
A
C+
A-
B+
B
B-
C-
D
A+
CS-203

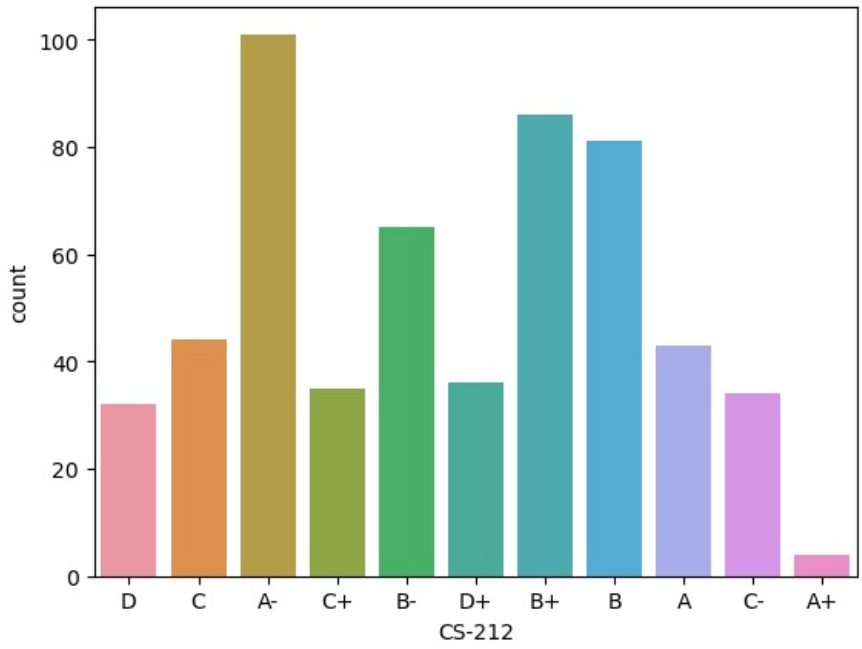
CS-214


count
0
10
20
30
40
50
60
70
D
A-
B
C
B-
C+
A
C-
B+
D+
A+
CS-214

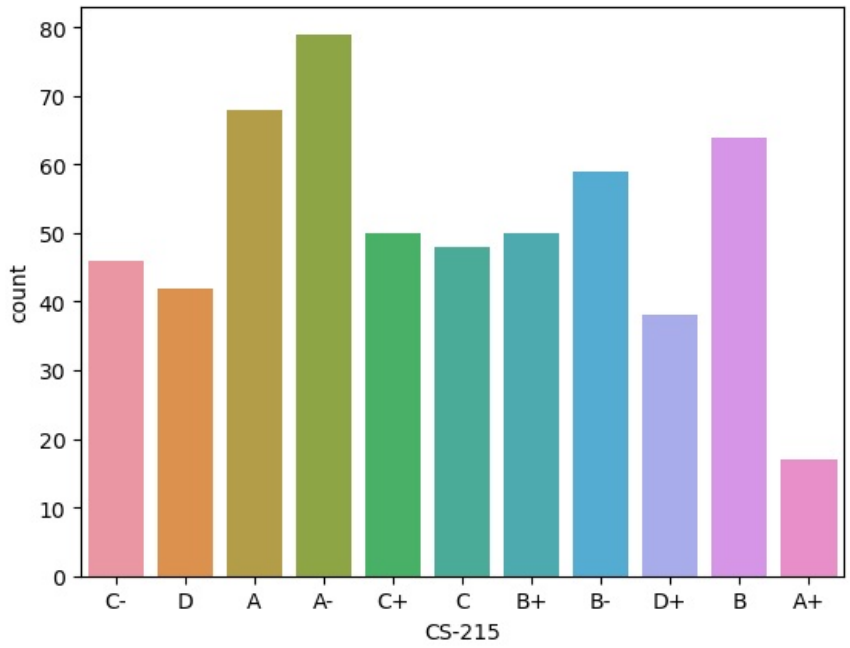
EE-217


140
120
100
80
60
40
20
0
count
A
B-
D+
A-
A+
C
B+
B
C-
D
F
C+
EE-217

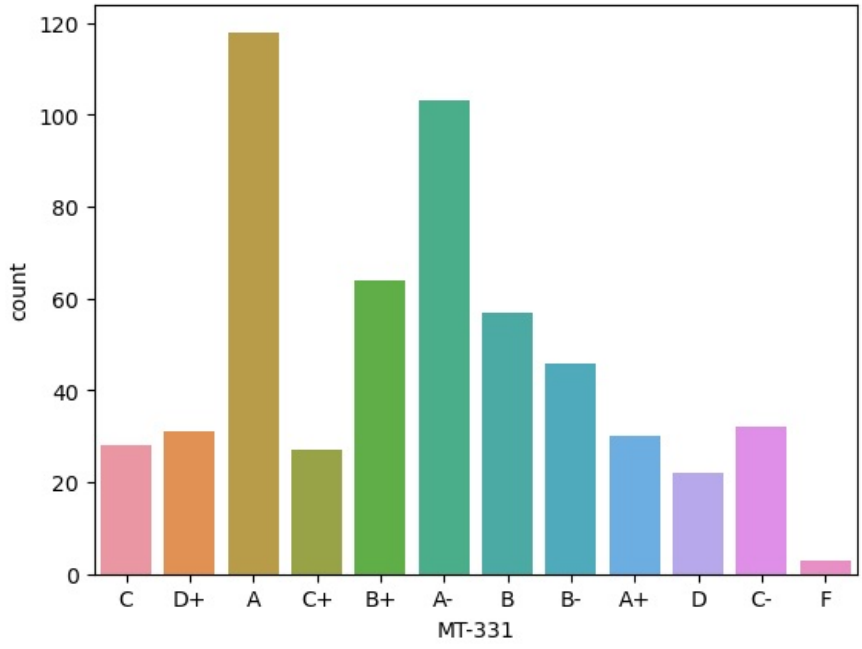
CS-212


100
80
60
40
20
0
count
D
C
A-
C+
B-
D+
B+
B
A
C-
A+
CS-212

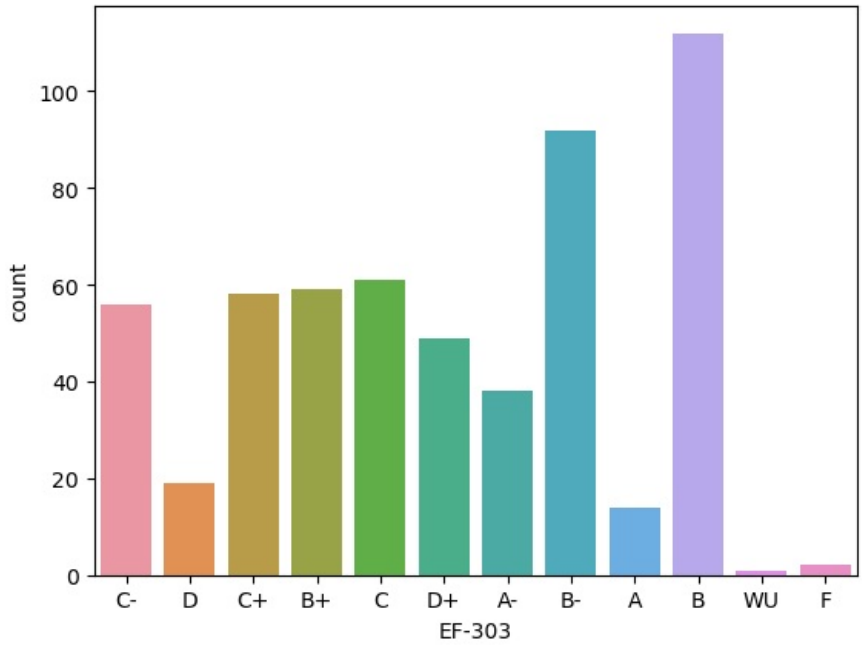
CS-215


80
70
60
50
40
30
20
10
0
count
C-
D
A
A-
C+
C
B+
B-
D+
B
A+
CS-215

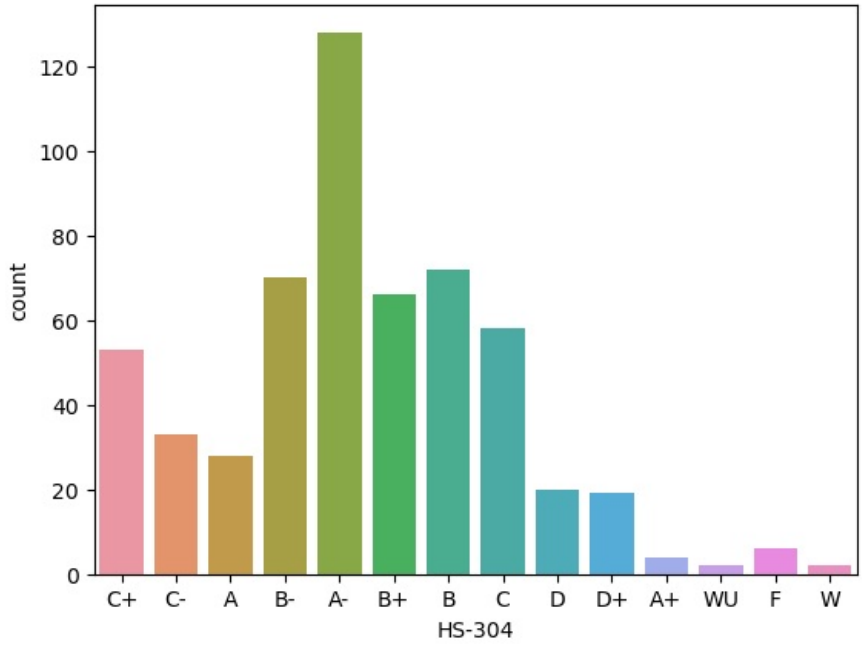
MT-331


120
100
80
60
40
20
0
count
C
D+
A
C+
B+
A-
B
B-
A+
D
C-
F
MT-331

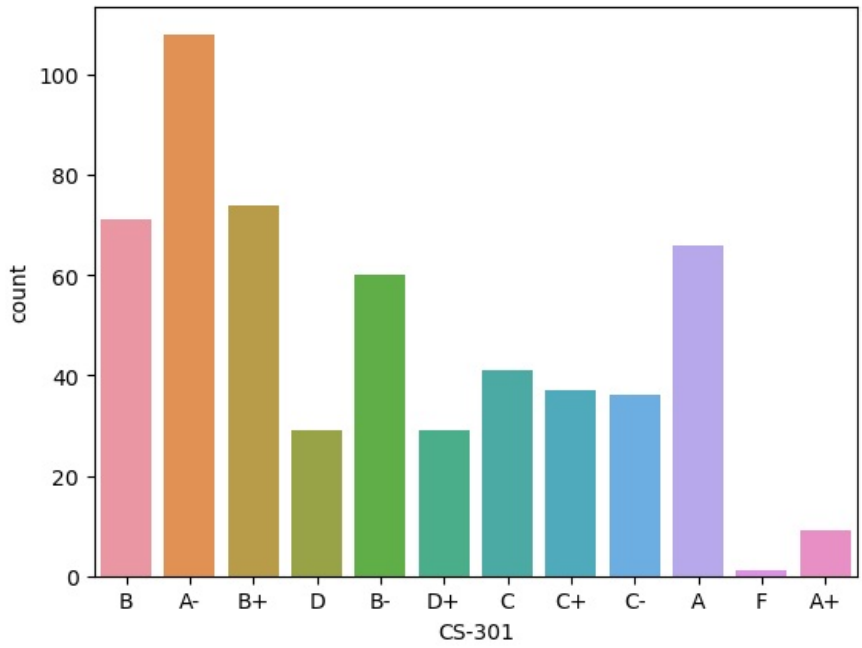
EF-303


count
100
80
60
40
20
0
C-
D
C+
B+
C
D+
A-
B-
A
B
WU
F
EF-303

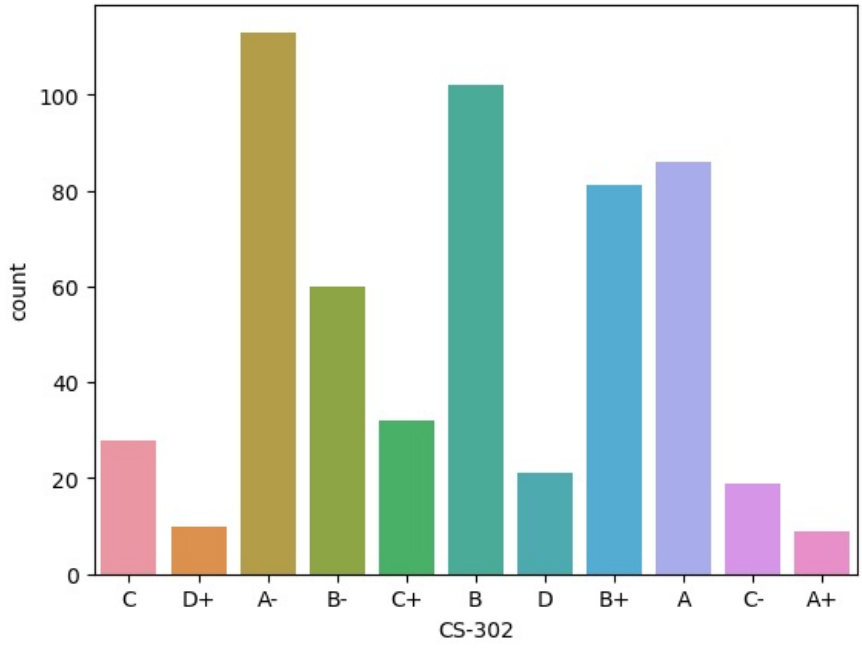
HS-304


count
120
100
80
60
40
20
0
C+
C-
A
B-
A-
B+
B
C
D
D+
A+
WU
F
W
HS-304

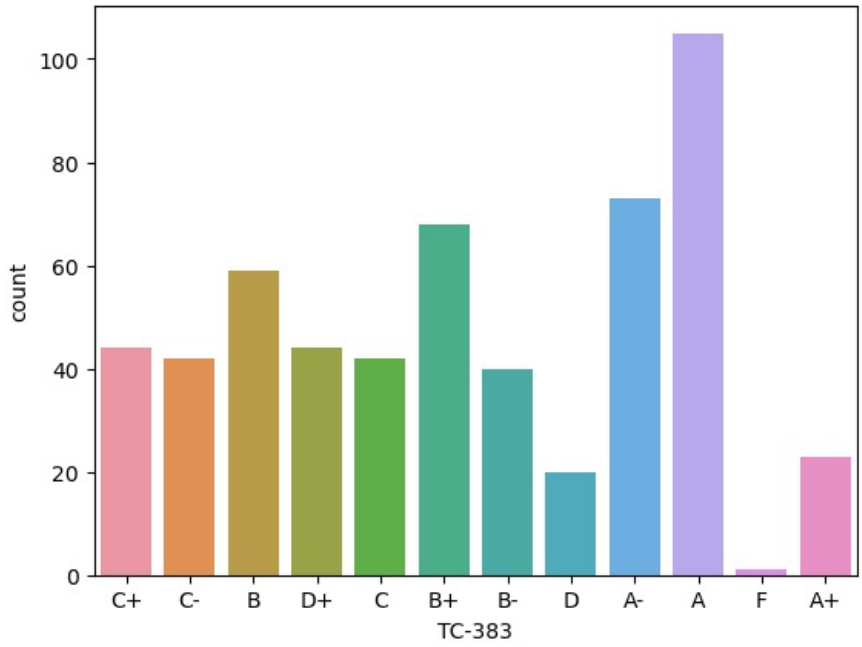
CS-301


100
80
60
40
20
0
count
B
A-
B+
D
B-
D+
C
C+
C-
A
F
A+
CS-301

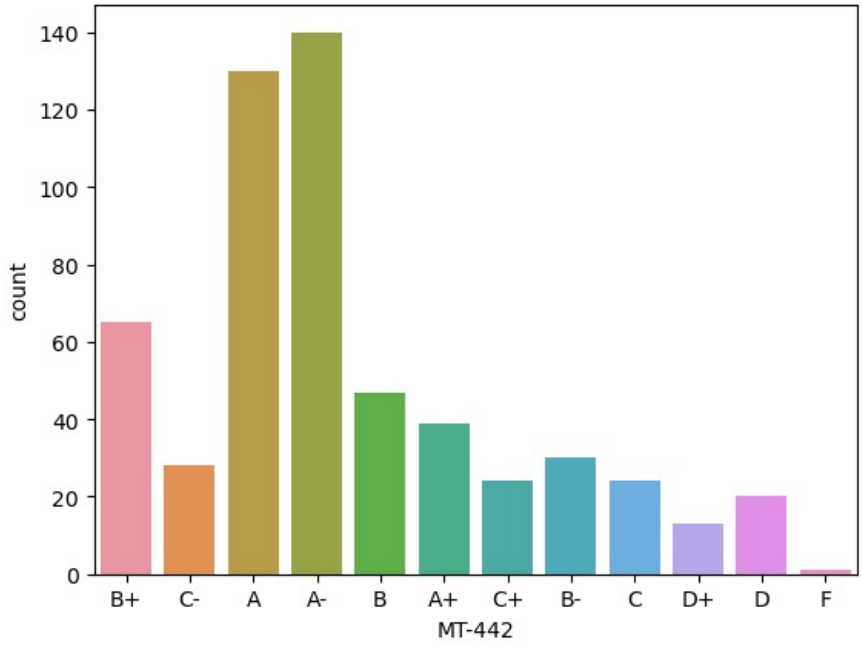
CS-302


100
80
60
40
20
0
count
C
D+
A-
B-
C+
B
D
B+
A
C-
A+
CS-302

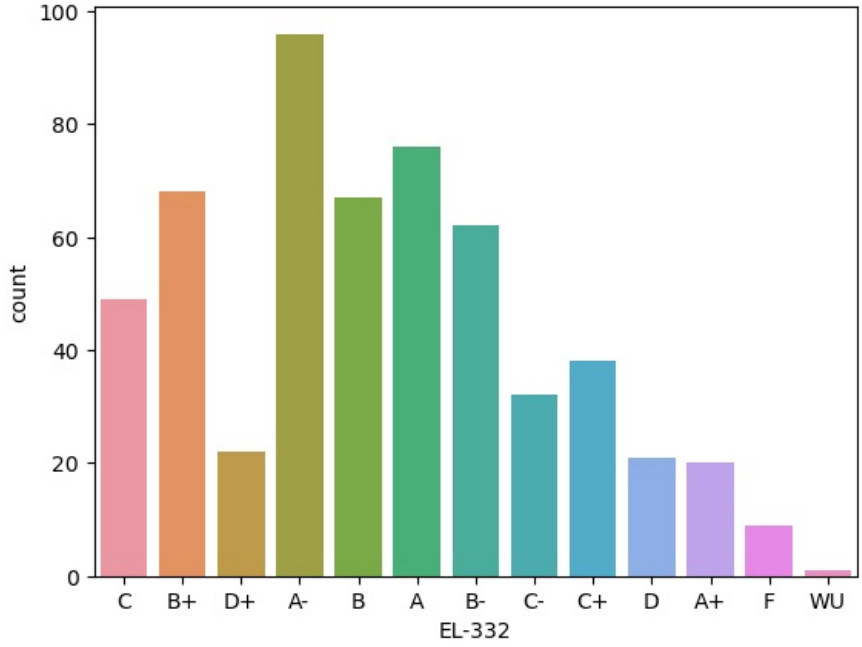
TC-383


count
100
80
60
40
20
0
C+
C-
B
D+
C
B+
B-
D
A-
A
F
A+
TC-383

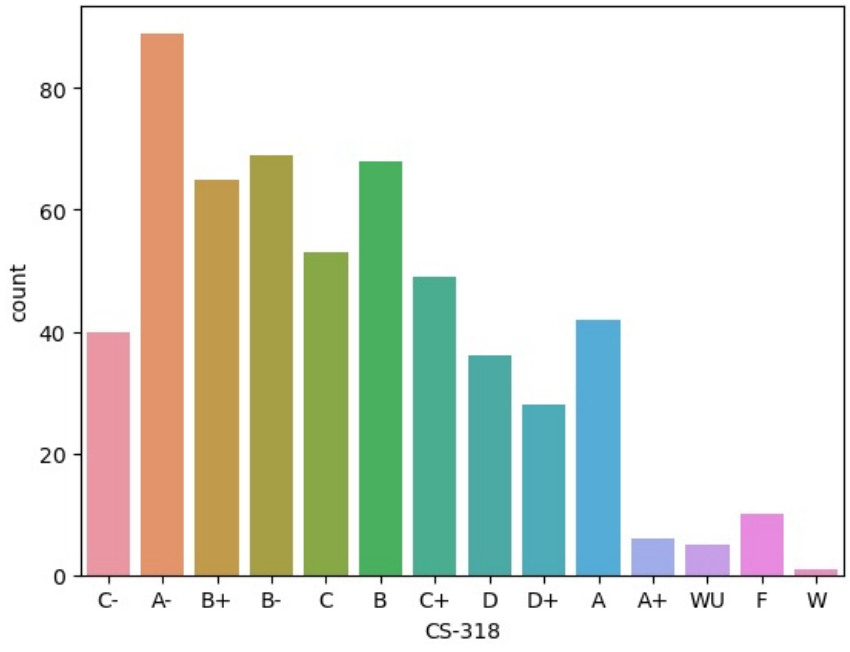
MT-442


count
140
120
100
80
60
40
20
0
B+
C-
A
A-
B
A+
C+
B-
C
D+
D
F
MT-442

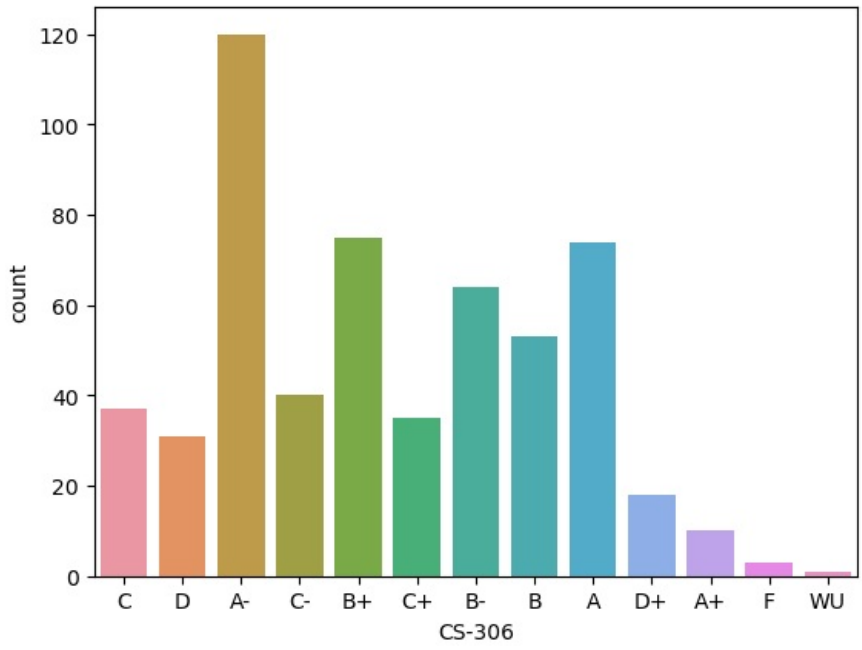
EL-332


100
80
60
40
20
0
count
C
B+
D+
A-
B
A
B-
C-
C+
D
A+
F
WU
EL-332

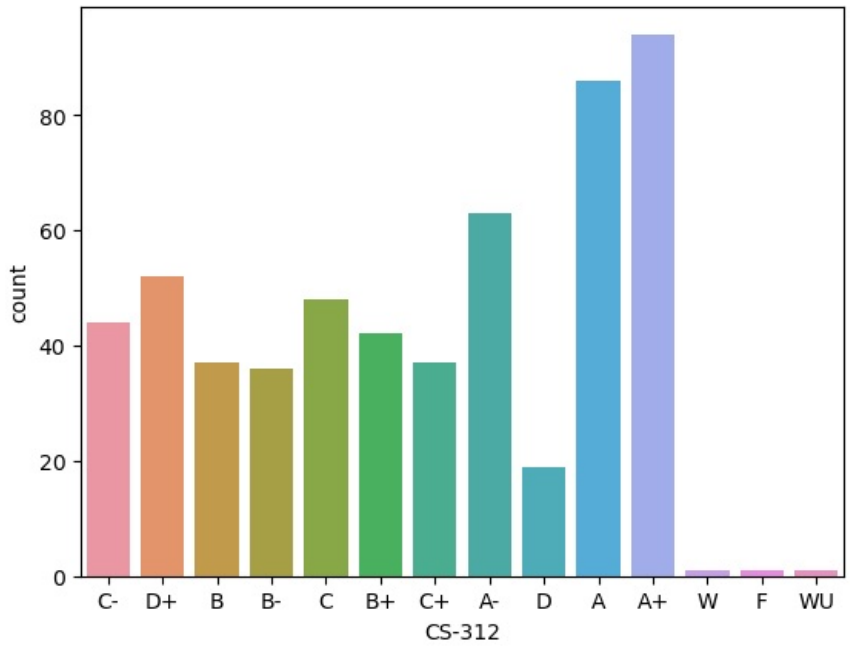
CS-318


80
60
40
20
0
count
C-
A-
B+
B-
C
B
C+
D
D+
A
A+
WU
F
W
CS-318

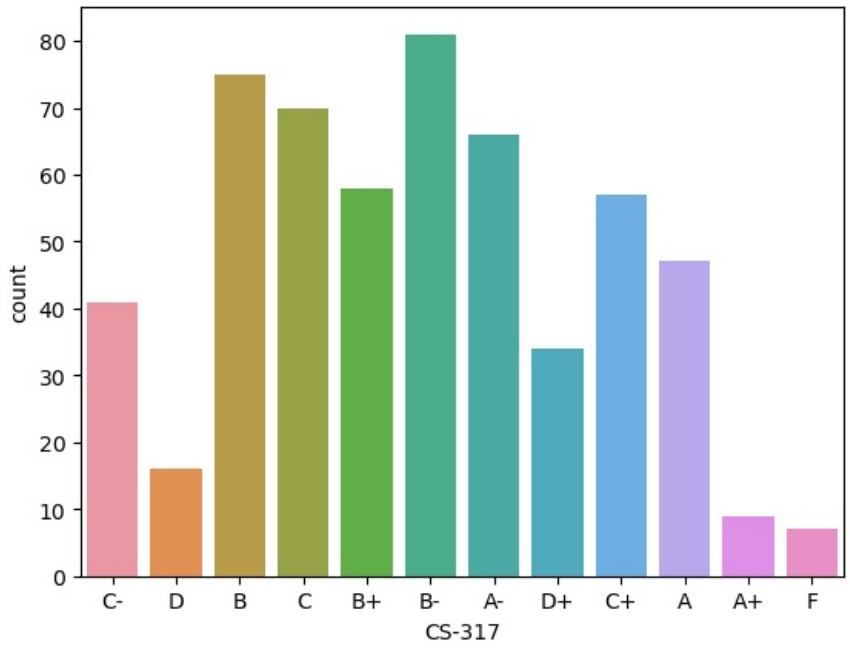
CS-306


120
100
80
60
40
20
0
count
C
D
A-
C-
B+
C+
B-
B
A
D+
A+
F
WU
CS-306

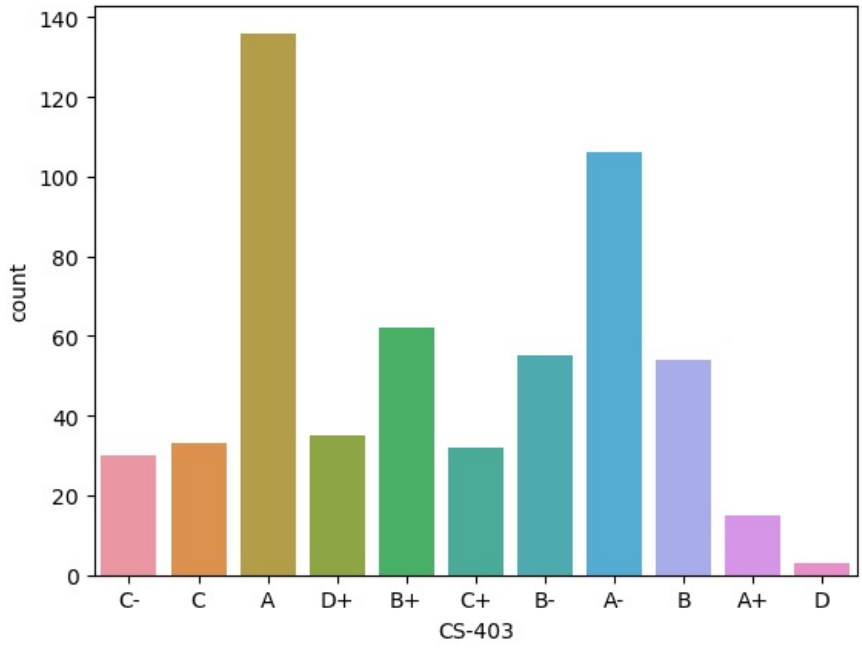
CS-312


80
60
40
20
0
count
C-
D+
B
B-
C
B+
C+
A-
D
A
A+
W
F
WU
CS-312

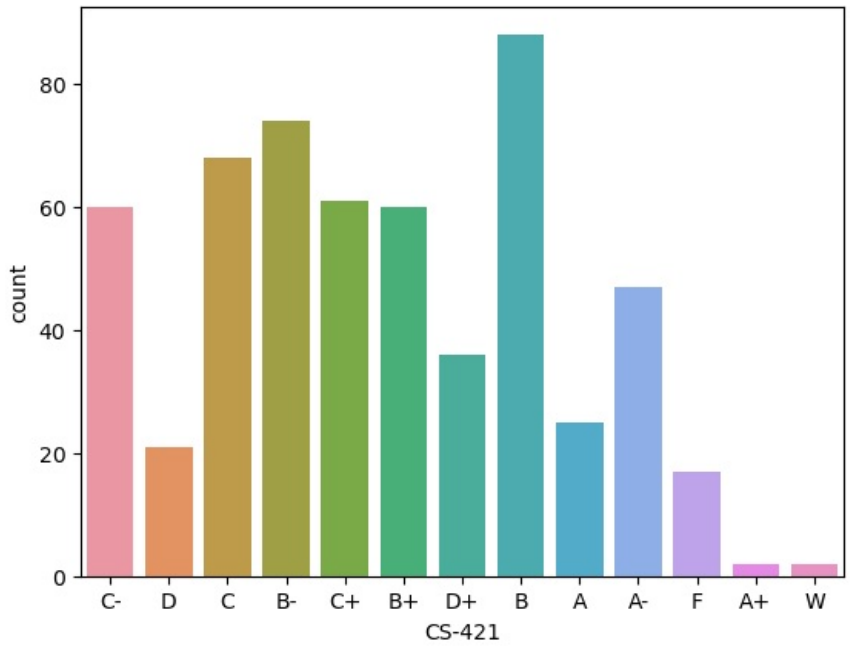
CS-317


count
0
10
20
30
40
50
60
70
80
C-
D
B
C
B+
B-
A-
D+
C+
A
A+
F
CS-317

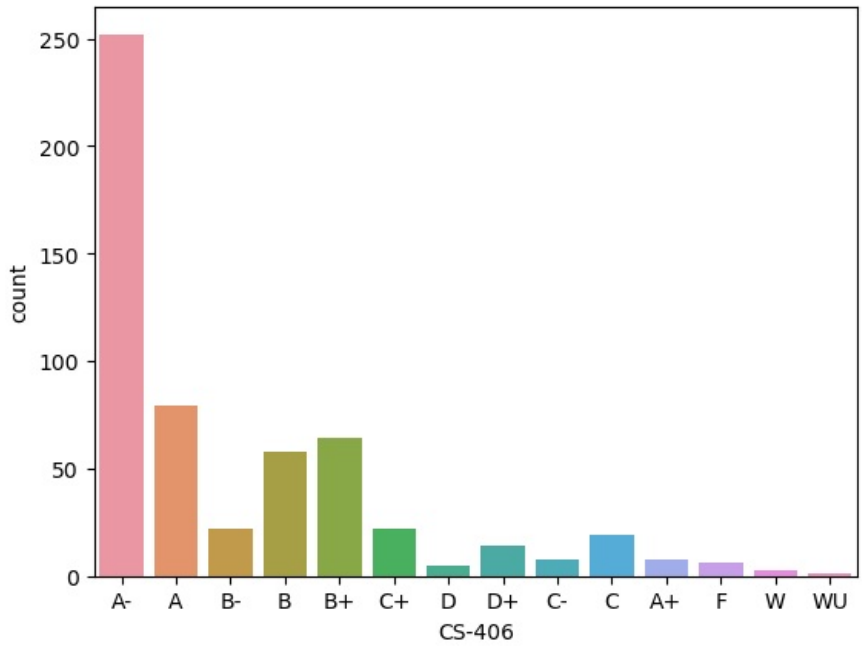
CS-403


count
0
20
40
60
80
100
120
140
C-
C
A
D+
B+
C+
B-
A-
B
A+
D
CS-403

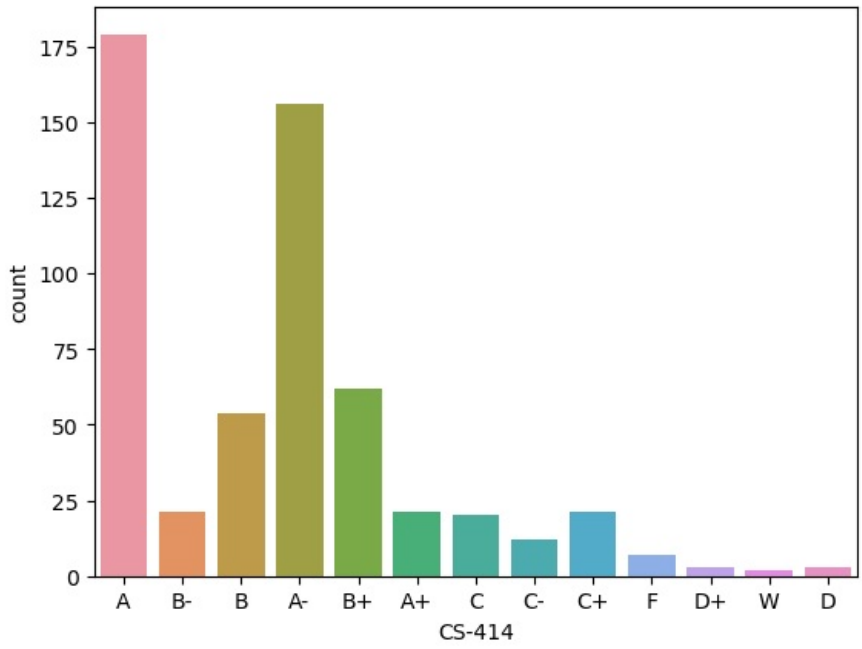
CS-421


count
80
60
40
20
0
C-
D
C
B-
C+
B+
D+
B
A
A-
F
A+
W
CS-421

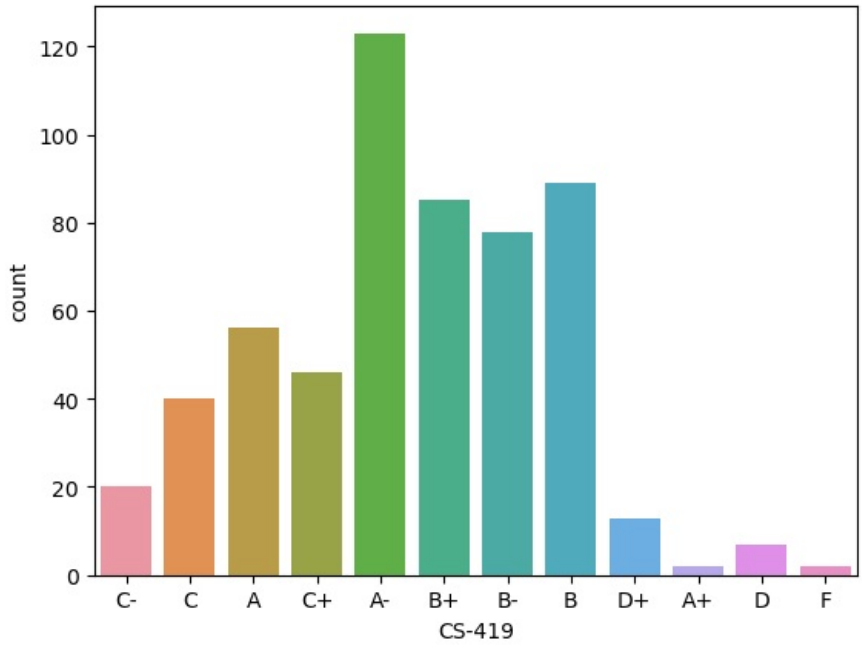
CS-406


count
250
200
150
100
50
0
A-
A
B-
B
B+
C+
D
D+
C-
C
A+
F
W
WU
CS-406

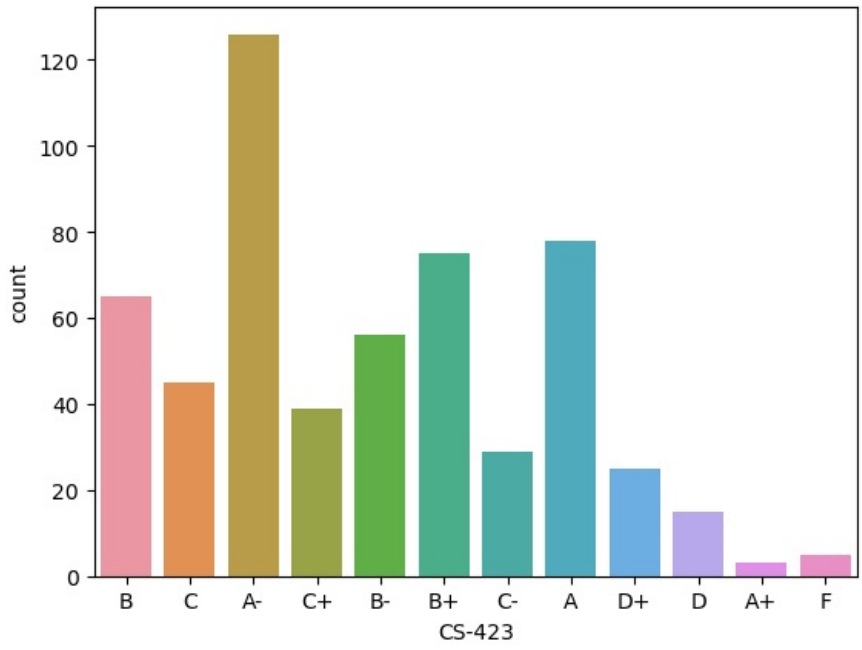
CS-414


175
150
125
100
75
50
25
0
count
A
B-
B
A-
B+
A+
C
C-
C+
F
D+
W
D
CS-414

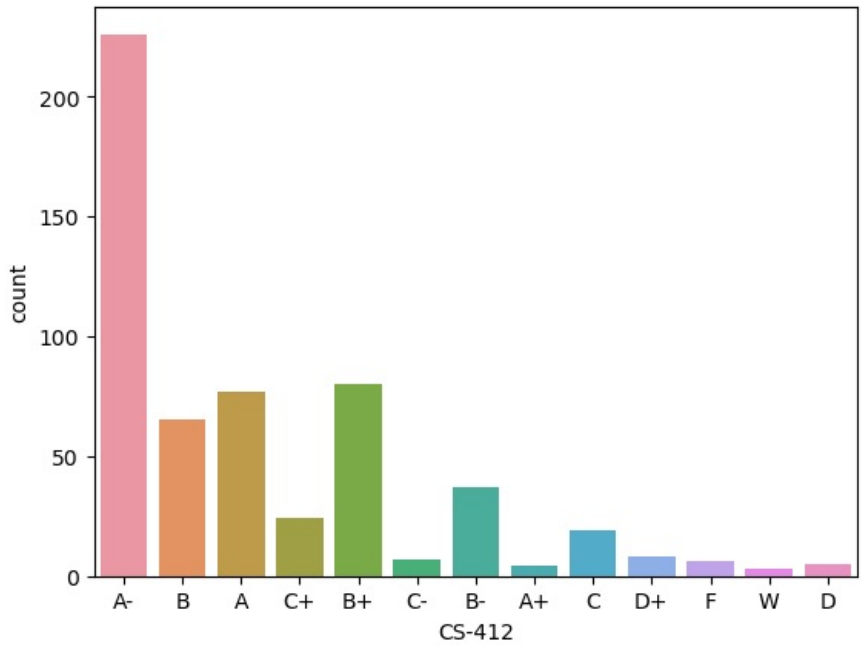
CS-419


120
100
80
60
40
20
0
count
C-
C
A
C+
A-
B+
B-
B
D+
A+
D
F
CS-419

CS-423



CS-412

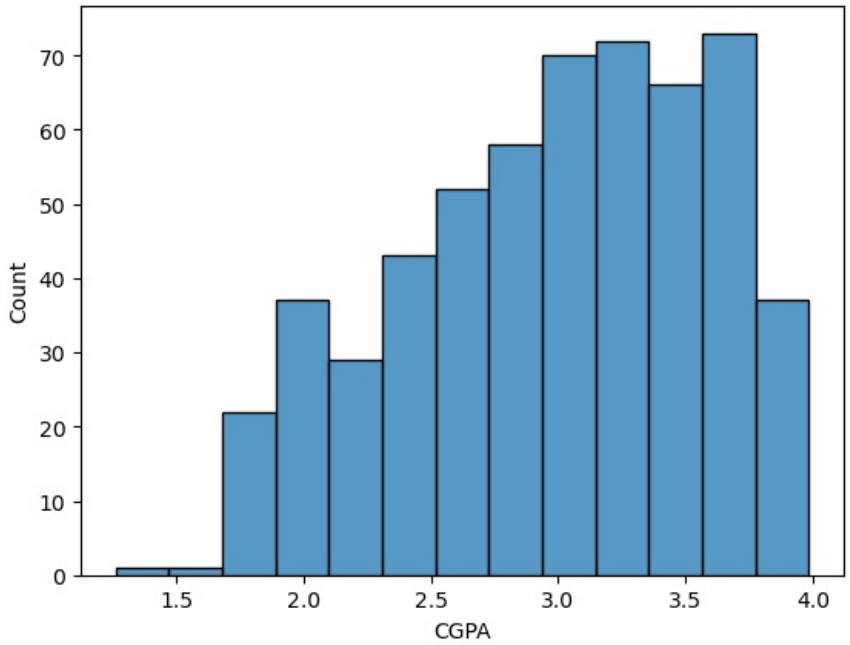


As shown in the count plot, the majority is in the range A-/A/B/B-.

But the majority of students who take the CS-312 receive an A+.

In [59]:
```
sns.histplot(df['CGPA'])
```

Out[59]:
```
<AxesSubplot:xlabel='CGPA', ylabel='Count'>
```



As seen in the graph above, the majority fall between 3 and 3.75 CGPA.

As we proceed with bivariate analysis, we must first change the object data type to encrypt the data.

In [60]:
```
from sklearn.preprocessing import LabelEncoder # importing
```

In [61]:
```
le=LabelEncoder()
for i in df.drop(['CGPA'],axis=1):
    df[i]=le.fit_transform(df[i])
df
```

Out[61]:

	PH-121	HS-101	CY-105	HS-105/12	MT-111	CS-105	CS-106	EL-102	EE-119	ME-107	...	CS-312	CS-317	CS-403	CS-421	CS-406	CS-414	CS-419	CS-423	CS-412	CGPA
0	5	10	8	6	8	10	9	8	5	8	...	8	8	8	8	2	0	8	3	2	2.205
1	0	9	10	9	5	6	9	0	10	9	...	10	9	6	9	2	5	6	6	3	2.008
2	0	3	0	5	4	0	5	4	2	2	...	3	3	0	6	0	0	0	2	0	3.608
3	9	7	10	9	9	2	10	8	9	7	...	10	6	10	8	5	3	7	7	7	1.906
4	2	2	2	4	0	0	2	4	0	2	...	5	4	4	5	2	0	2	2	0	3.448
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
566	3	0	0	2	1	0	2	2	1	4	...	2	2	0	0	0	4	4	3	0	3.798
567	1	0	0	0	0	0	0	2	0	0	...	4	4	0	0	2	3	2	6	2	3.772
568	3	0	2	4	0	0	0	0	0	3	...	2	3	0	4	0	6	4	2	2	3.470
569	0	4	9	0	9	10	5	8	5	8	...	9	3	3	8	9	6	3	5	6	2.193
570	6	9	9	6	6	10	3	7	6	10	...	7	6	5	9	11	8	4	9	8	1.753

561 rows × 42 columns

Now that all of the columns have been transformed, we can see this.

In [62]:
```
df.dtypes
```

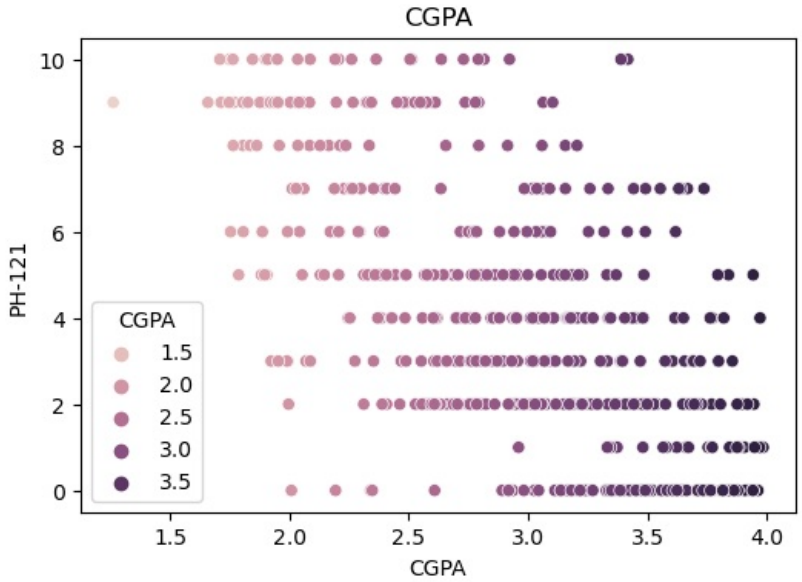
Out[62]:
```
PH-121         int32
HS-101         int32
CY-105         int32
HS-105/12      int32
MT-111         int32
CS-105         int32
CS-106         int32
EL-102         int32
EE-119         int32
ME-107         int32
CS-107         int32
HS-205/20      int32
MT-222         int32
EE-222         int32
MT-224         int32
CS-210         int32
CS-211         int32
CS-203         int32
CS-214         int32
EE-217         int32
CS-212         int32
CS-215         int32
MT-331         int32
EF-303         int32
HS-304         int32
CS-301         int32
CS-302         int32
TC-383         int32
MT-442         int32
EL-332         int32
CS-318         int32
CS-306         int32
CS-312         int32
CS-317         int32
CS-403         int32
CS-421         int32
CS-406         int32
CS-414         int32
CS-419         int32
CS-423         int32
CS-412         int32
CGPA         float64
dtype: object
```

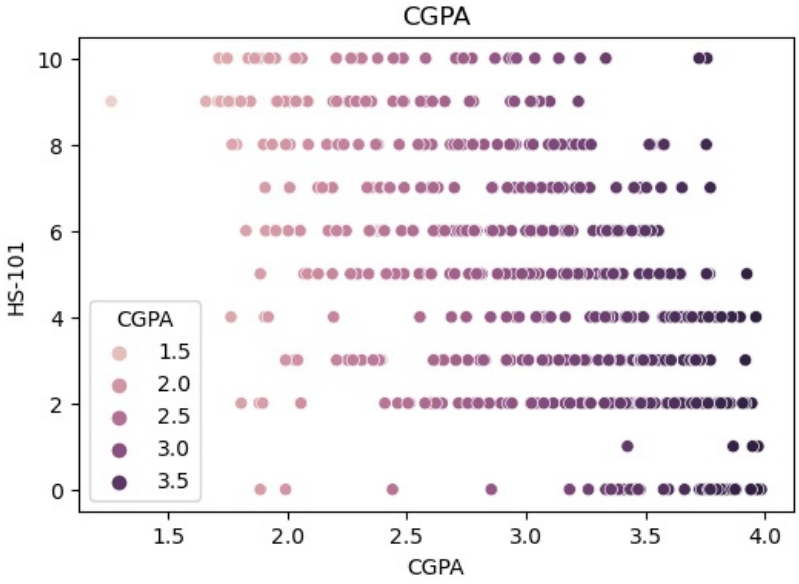
All the columns, with the exception of CGPA, are int32 now, as we can see.

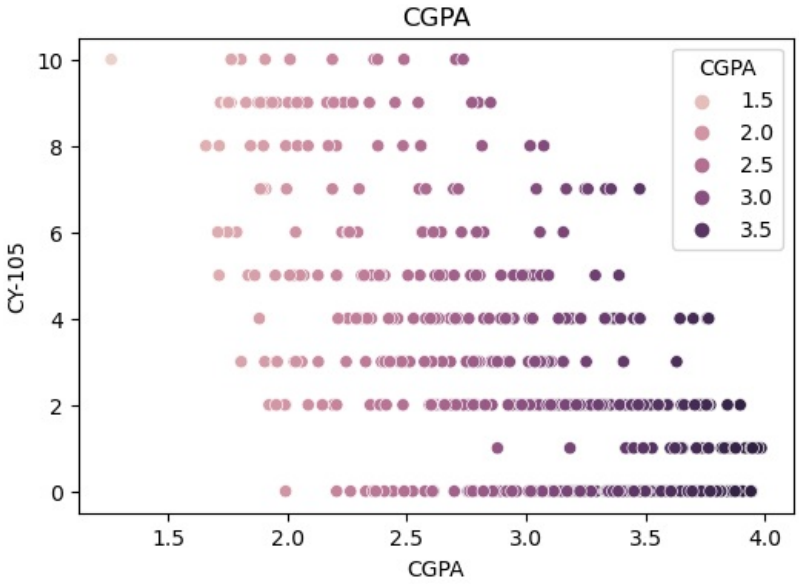
Now, we see that all of the columns have been changed to numeric data types.

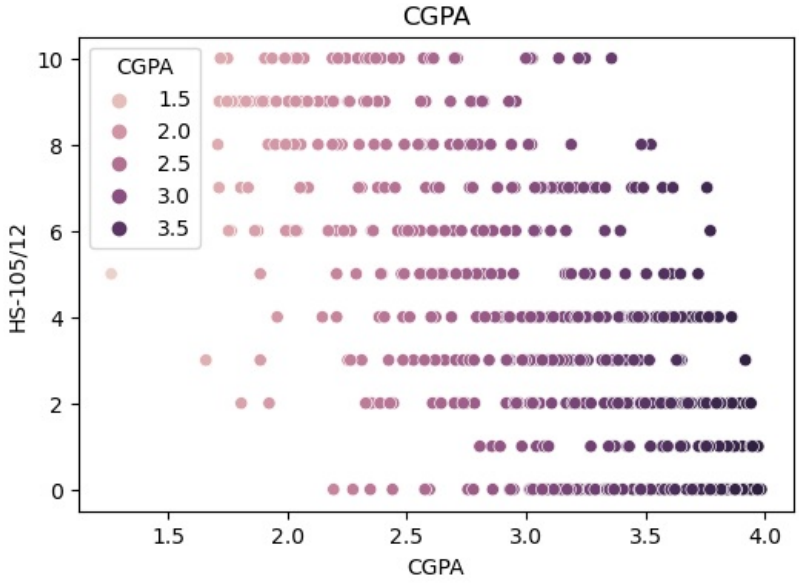
Bivariable Assessment

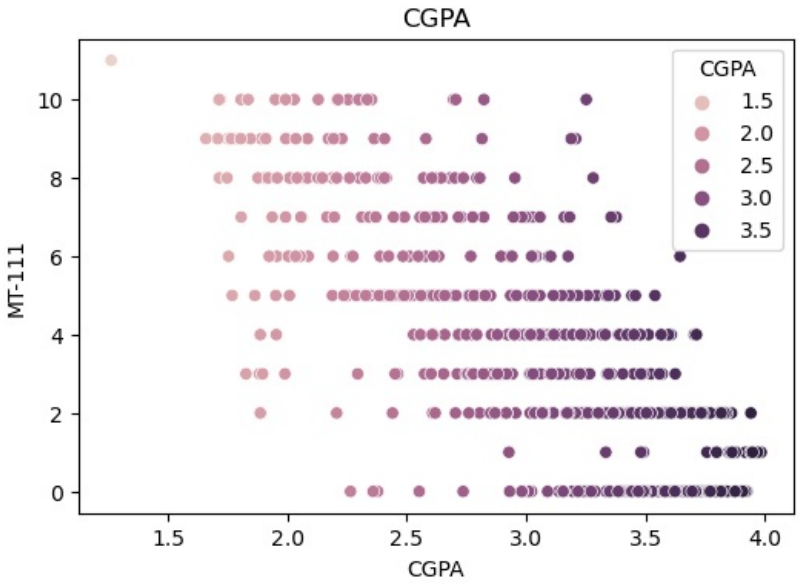
In [63]:
```
# Bivariate Analysis: Co-relation Graph Plot
for col in df.drop(['CGPA'],axis=1):
    plt.figure(figsize=(6,4))
    plt.title('CGPA')
    sns.scatterplot(df['CGPA'],df[col],hue=df['CGPA'])
    plt.show()
```

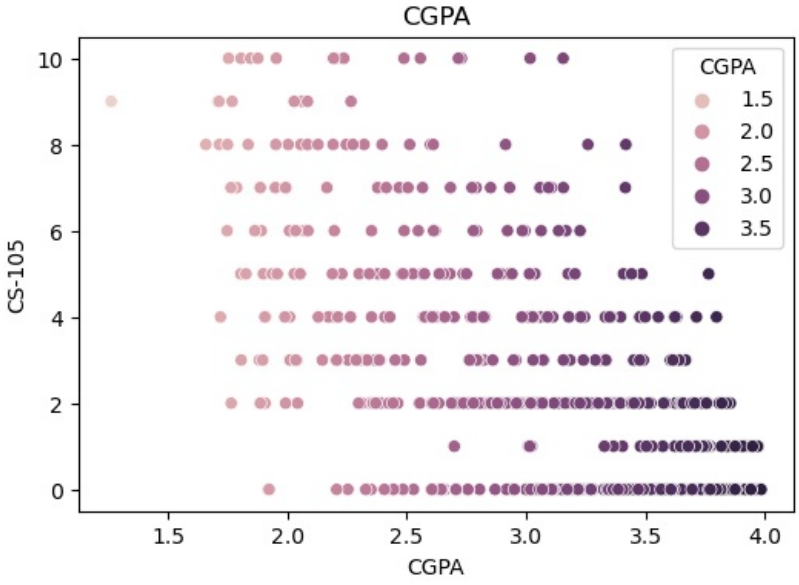


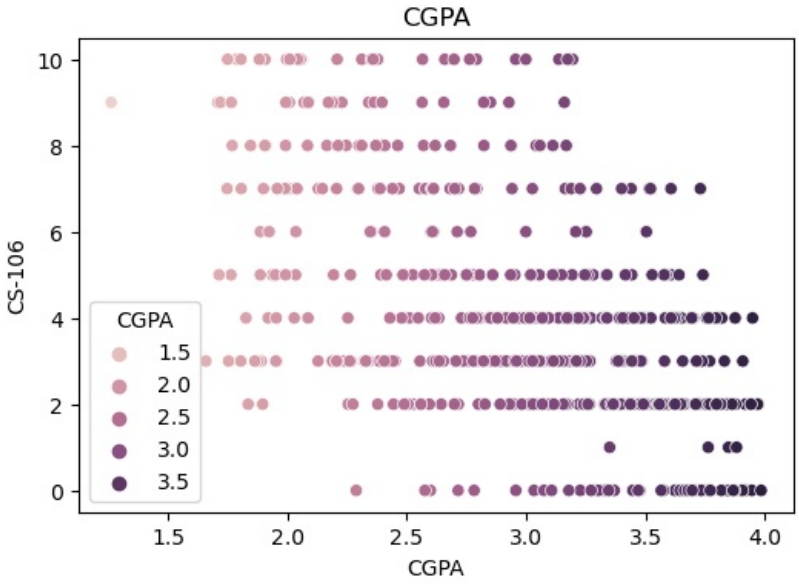

CGPA
HS-101
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

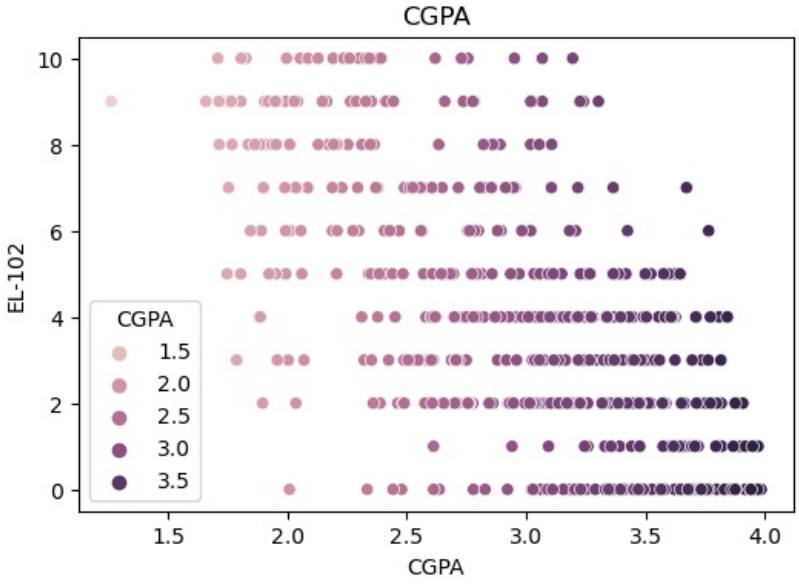

CGPA
CY-105
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

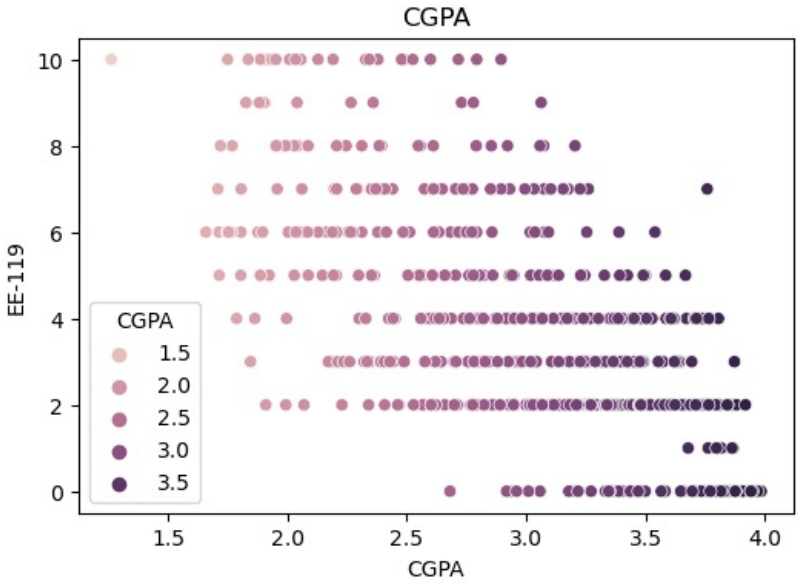

CGPA
HS-105/12
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

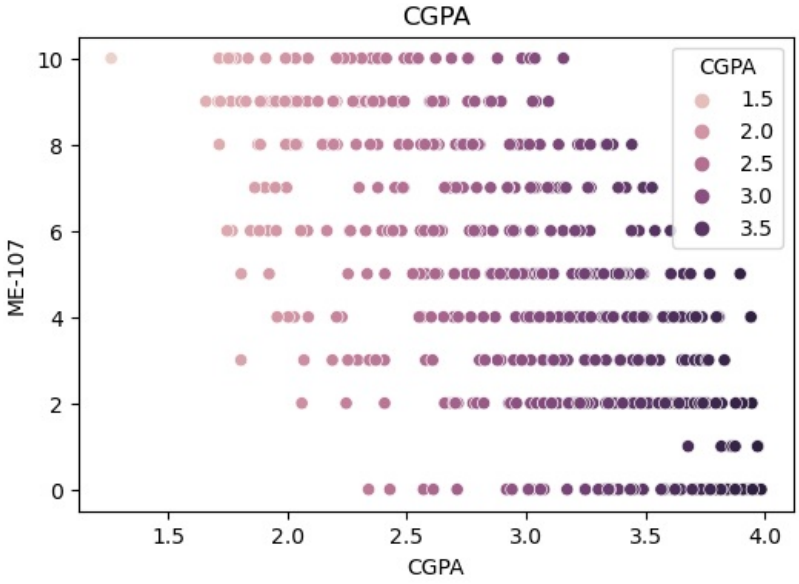

CGPA
MT-111
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

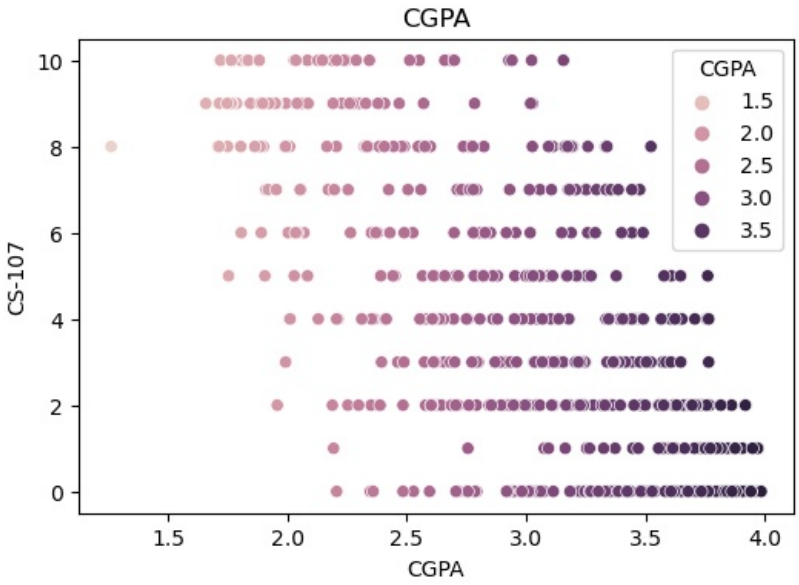

CGPA
CS-105
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

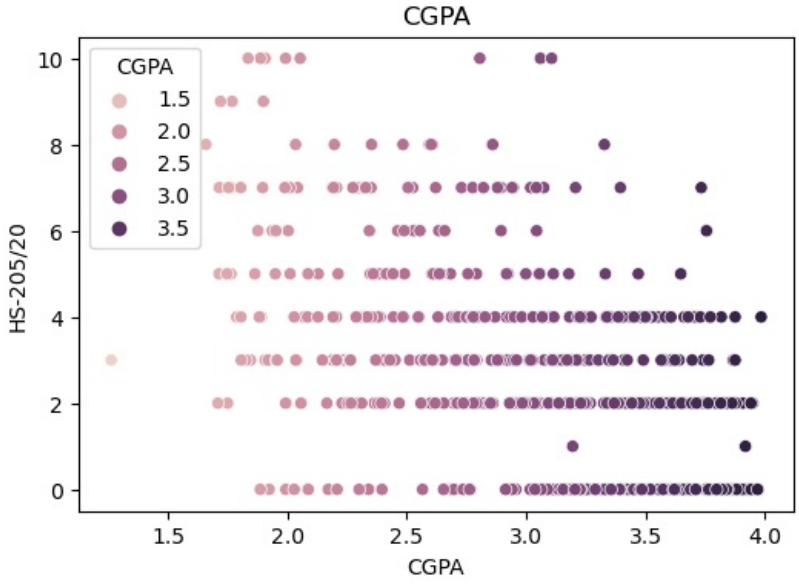

CGPA
CS-106
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

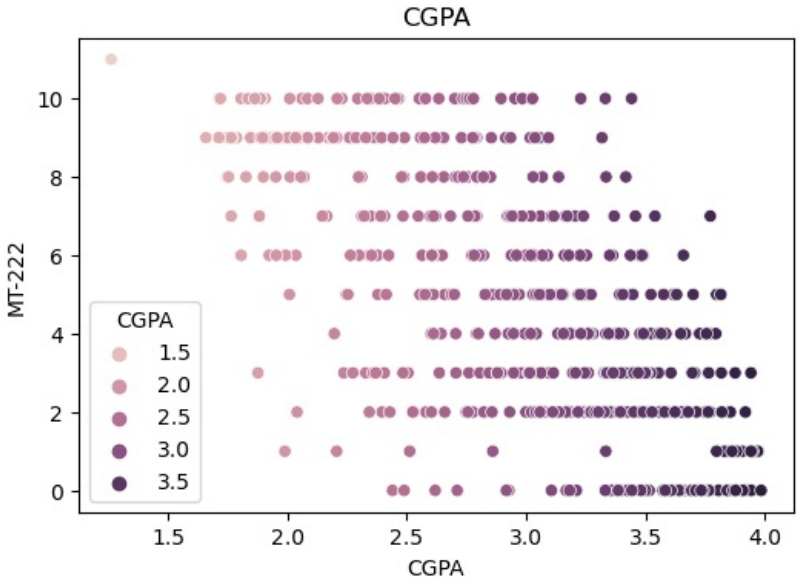

CGPA
EL-102
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

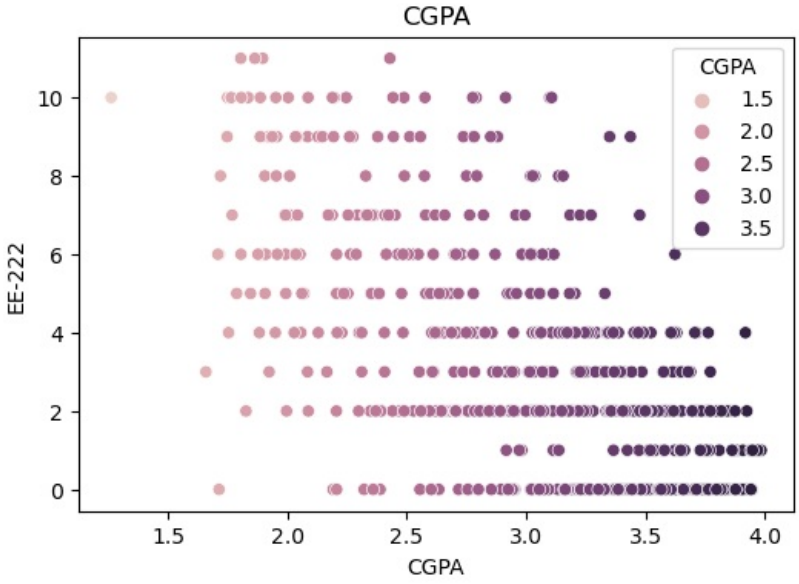

CGPA
EE-119
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

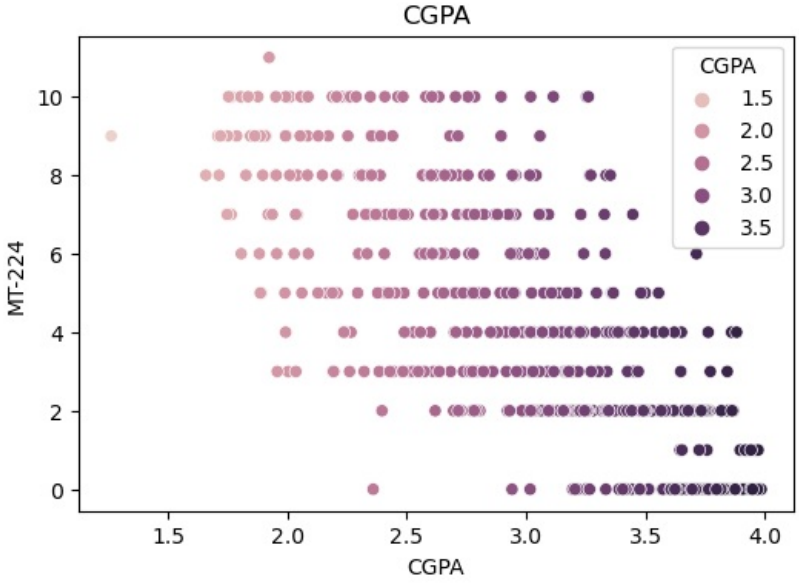

CGPA
ME-107
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

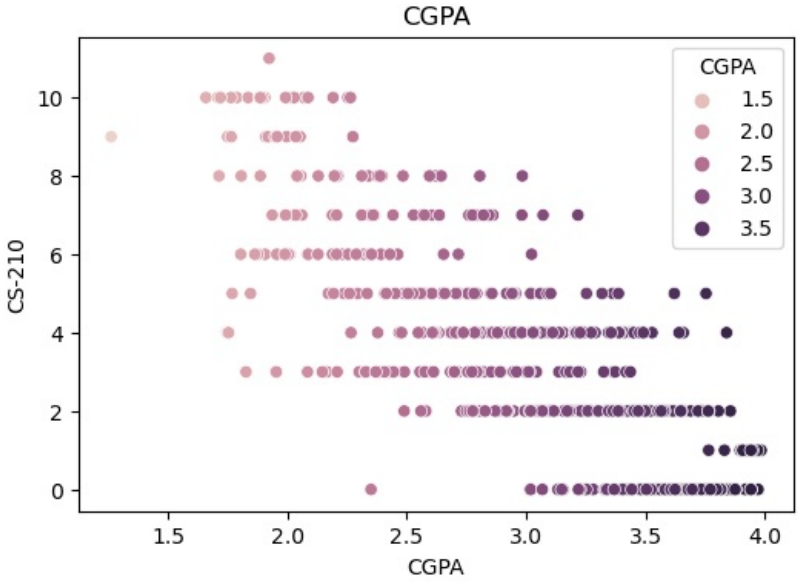

CGPA
CS-107
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

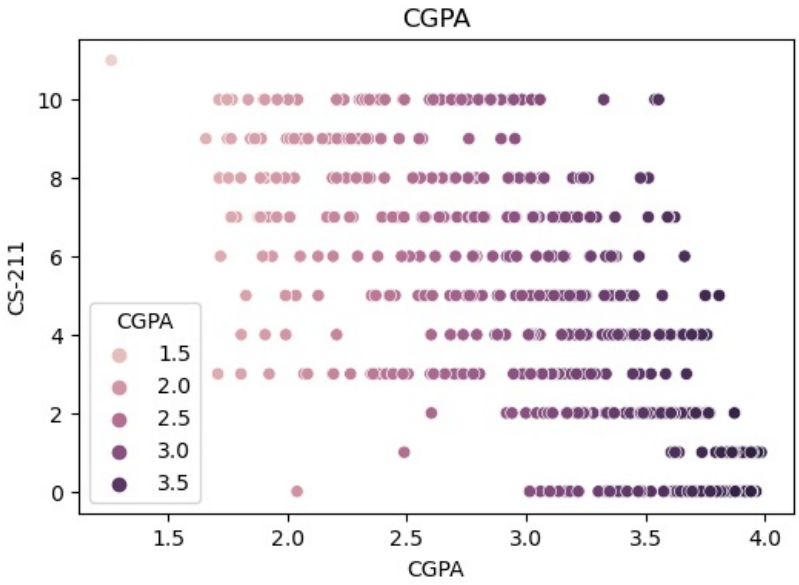

CGPA
HS-205/20
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

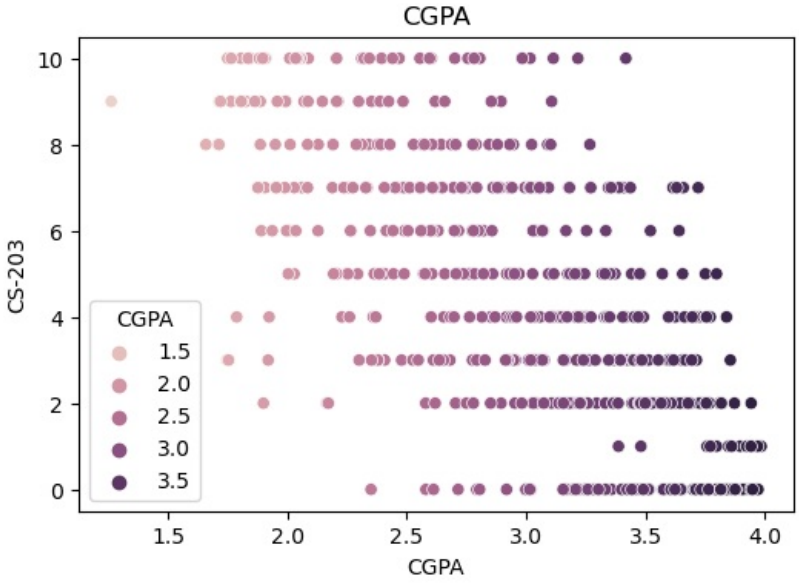

CGPA
MT-222
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

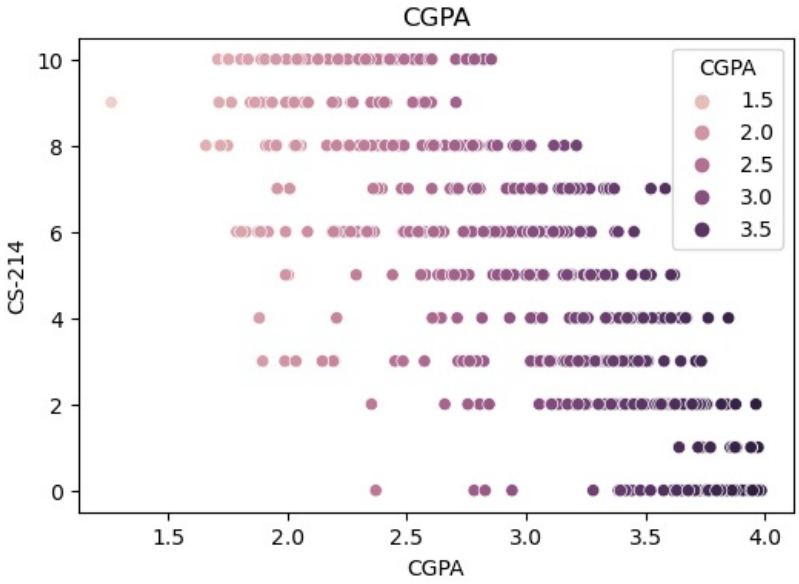

CGPA
CGPA
1.5
2.0
2.5
3.0
3.5
EE-222
CGPA

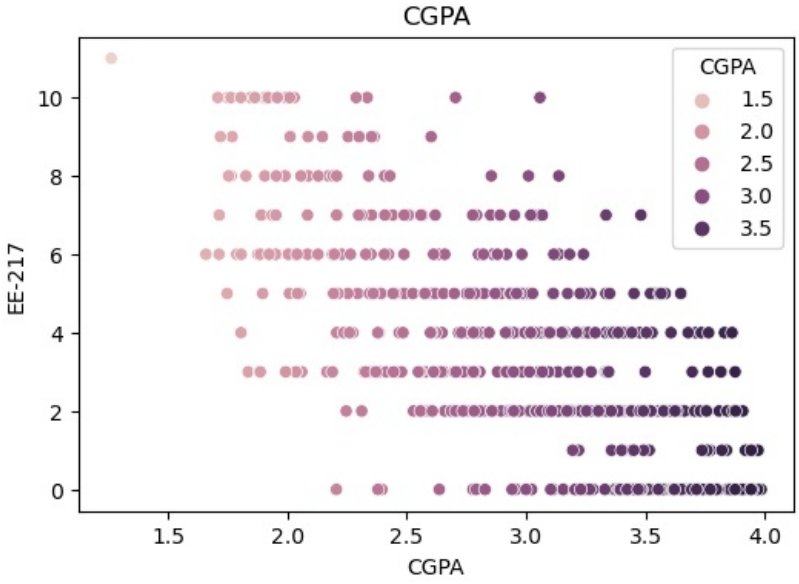

CGPA
CGPA
1.5
2.0
2.5
3.0
3.5
MT-224
CGPA

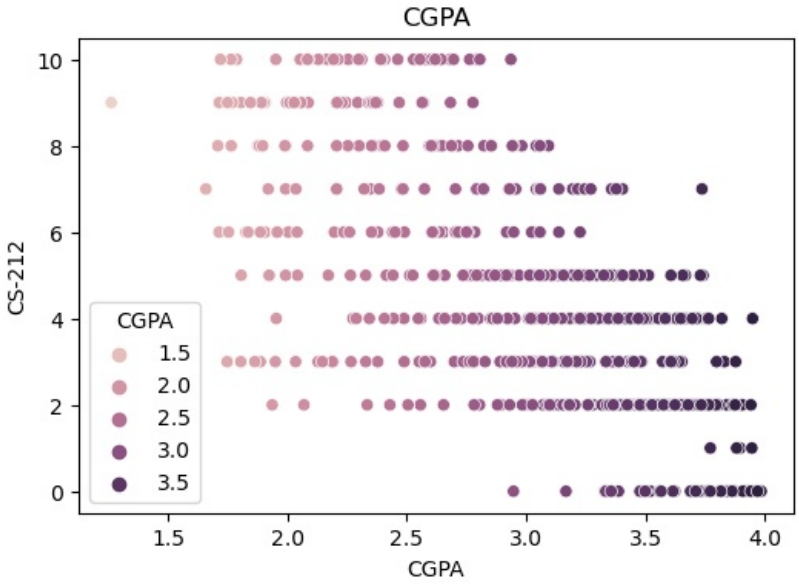

CGPA
CGPA
1.5
2.0
2.5
3.0
3.5
CS-210
CGPA

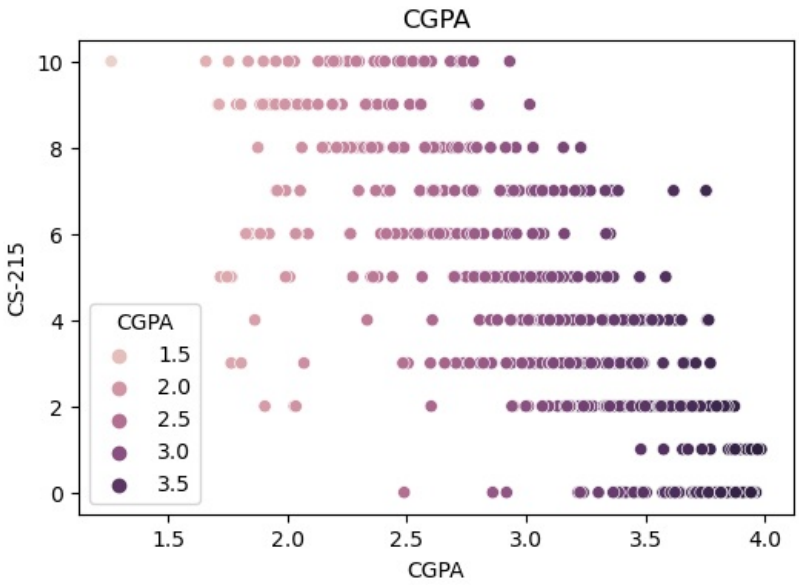

CGPA
CS-211
CGPA
1.5
2.0
2.5
3.0
3.5
CGPA

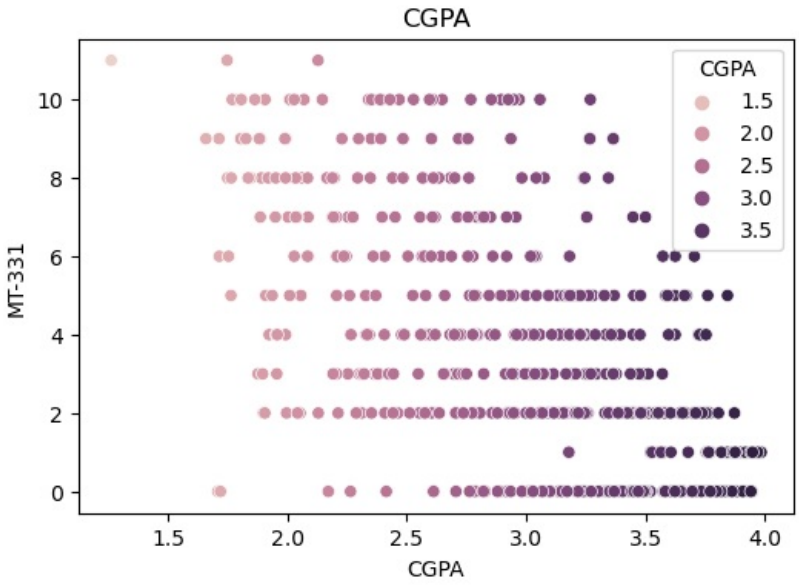

CGPA
CS-203
CGPA
1.5
2.0
2.5
3.0
3.5
CGPA

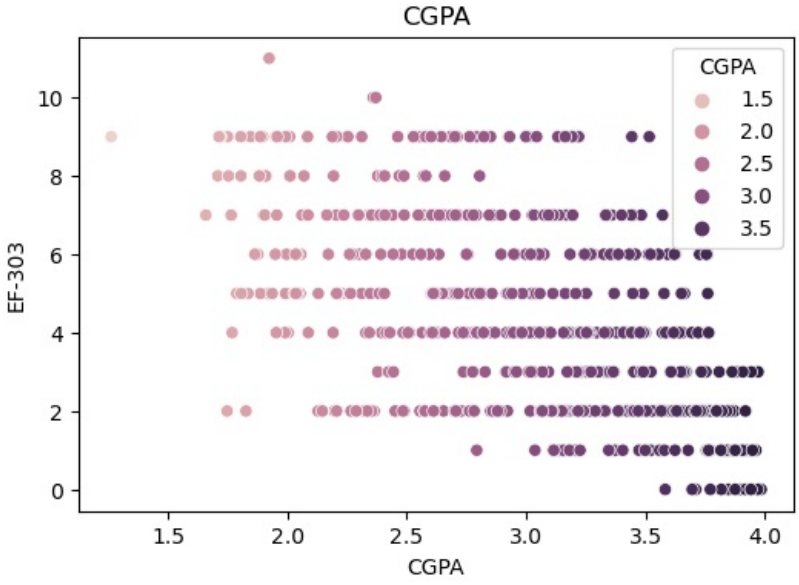

CGPA
CS-214
CGPA
1.5
2.0
2.5
3.0
3.5
CGPA

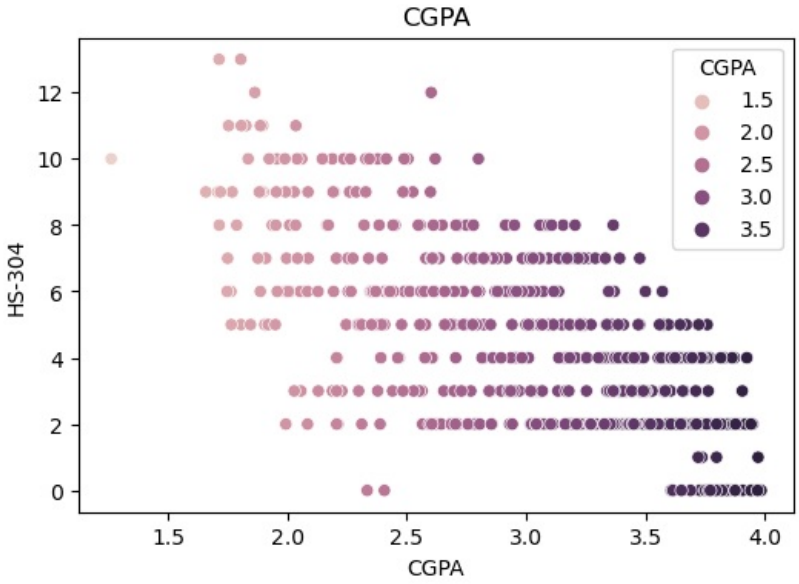

CGPA
CGPA
1.5
2.0
2.5
3.0
3.5
EE-217
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

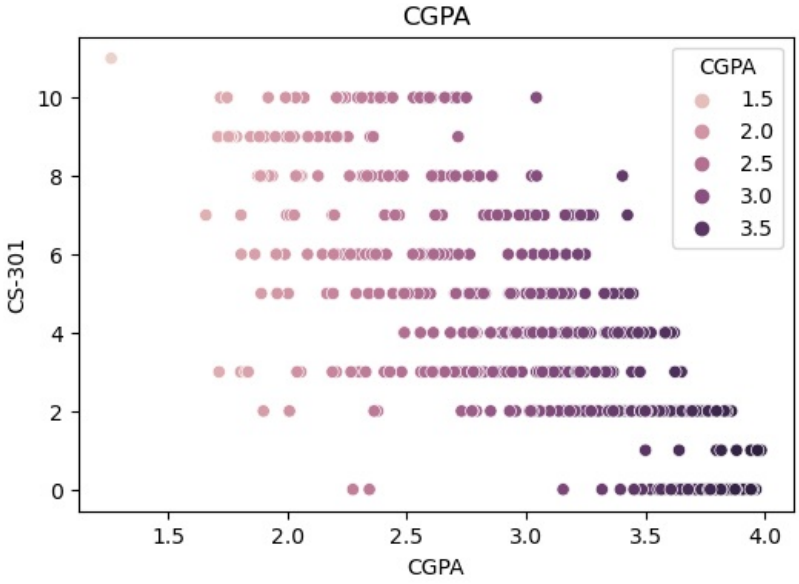

CGPA
CGPA
1.5
2.0
2.5
3.0
3.5
CS-212
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

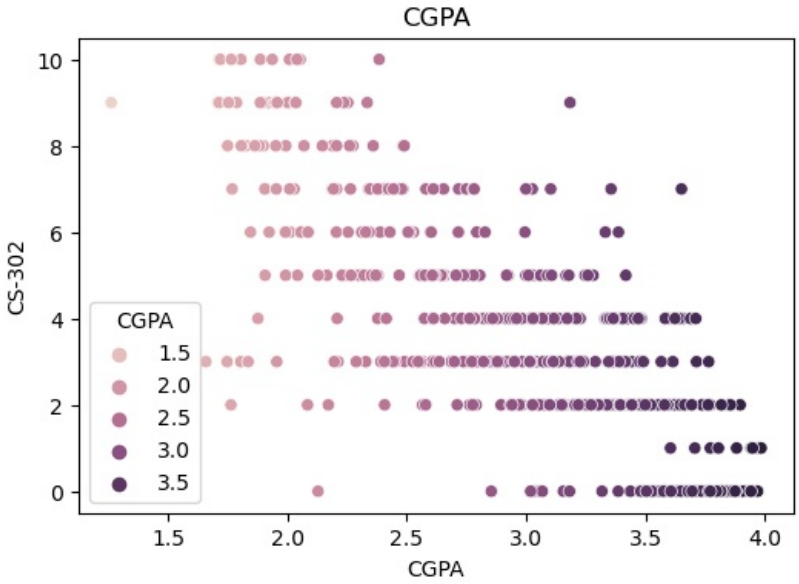

CGPA
CGPA
1.5
2.0
2.5
3.0
3.5
CS-215
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

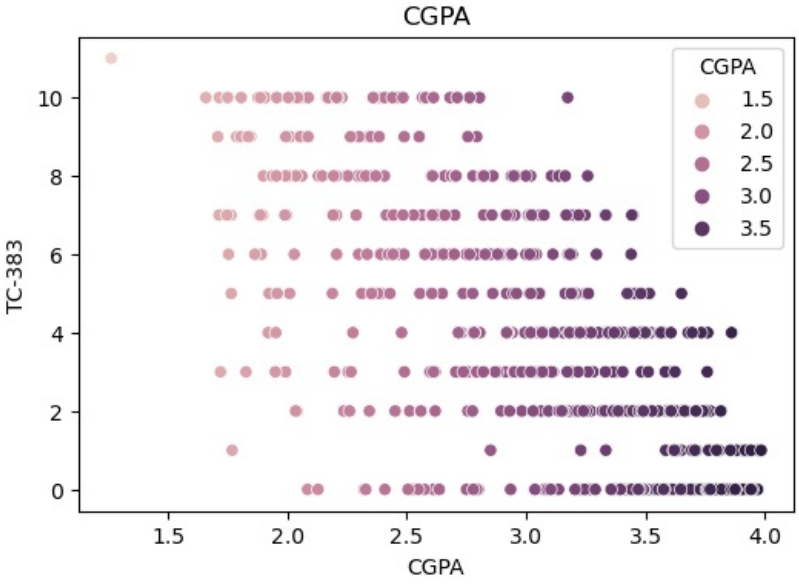

CGPA
CGPA
1.5
2.0
2.5
3.0
3.5
MT-331
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

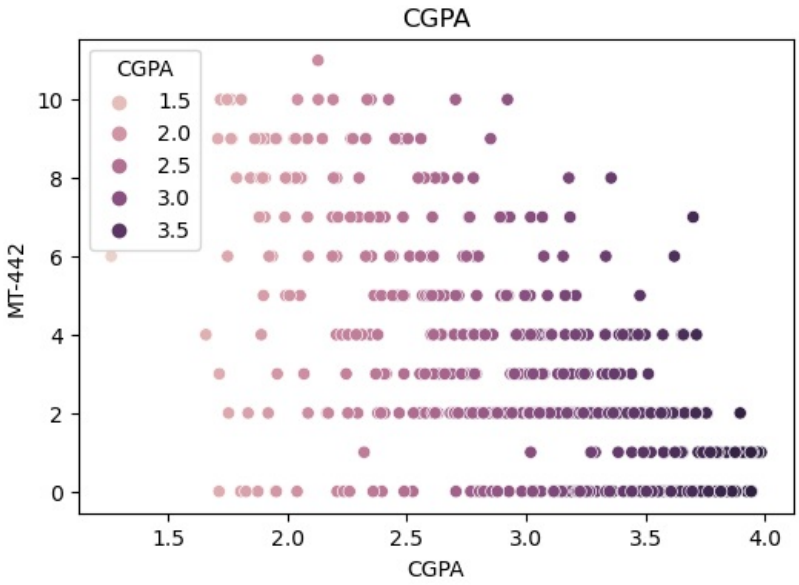

CGPA
CGPA
1.5
2.0
2.5
3.0
3.5
EF-303
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

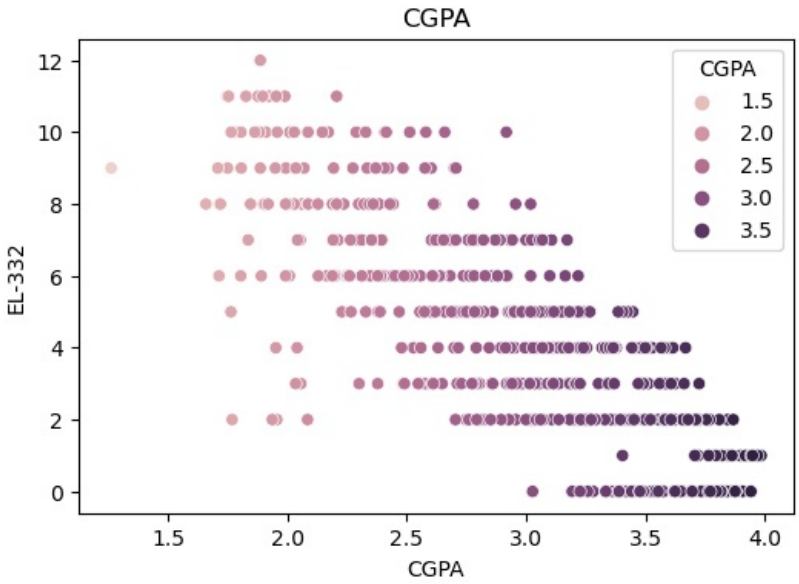

CGPA
CGPA
1.5
2.0
2.5
3.0
3.5
HS-304
0
2
4
6
8
10
12
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

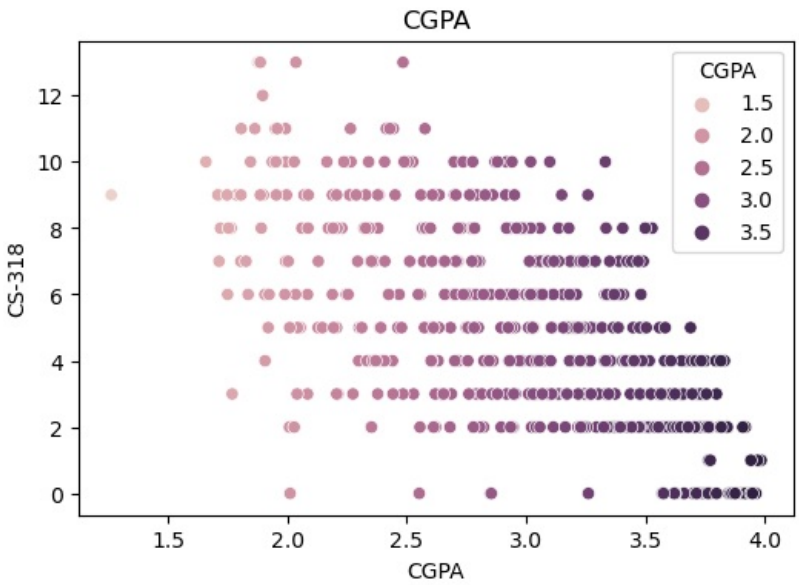

CGPA
CS-301
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

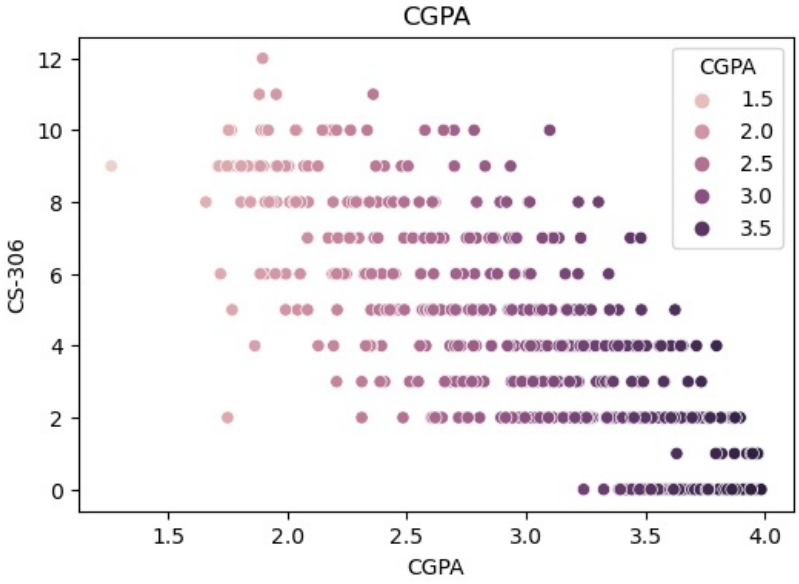

CGPA
CS-302
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

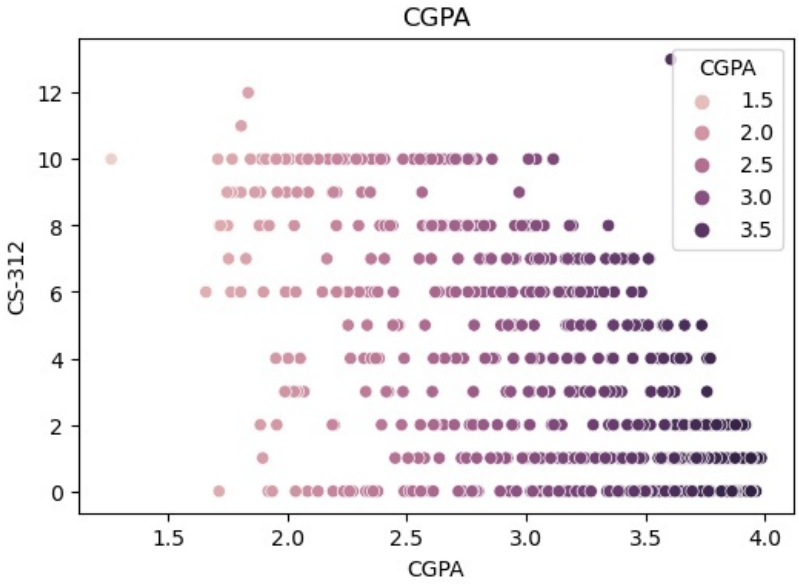

CGPA
TC-383
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

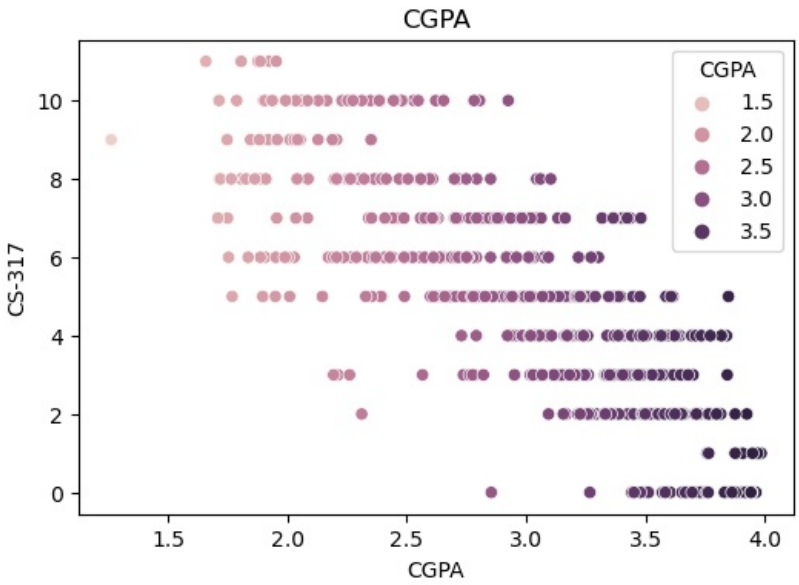

CGPA
CGPA
1.5
2.0
2.5
3.0
3.5
MT-442
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

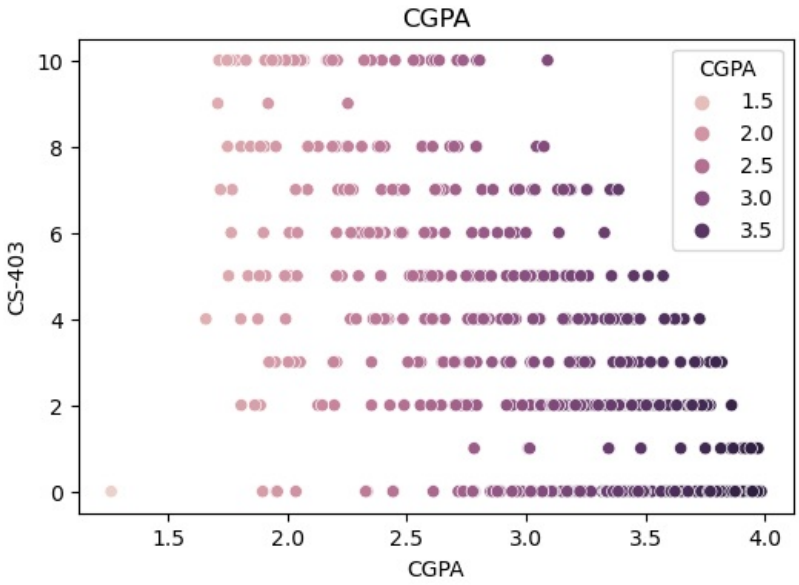

CGPA
CGPA
1.5
2.0
2.5
3.0
3.5
EL-332
0
2
4
6
8
10
12
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

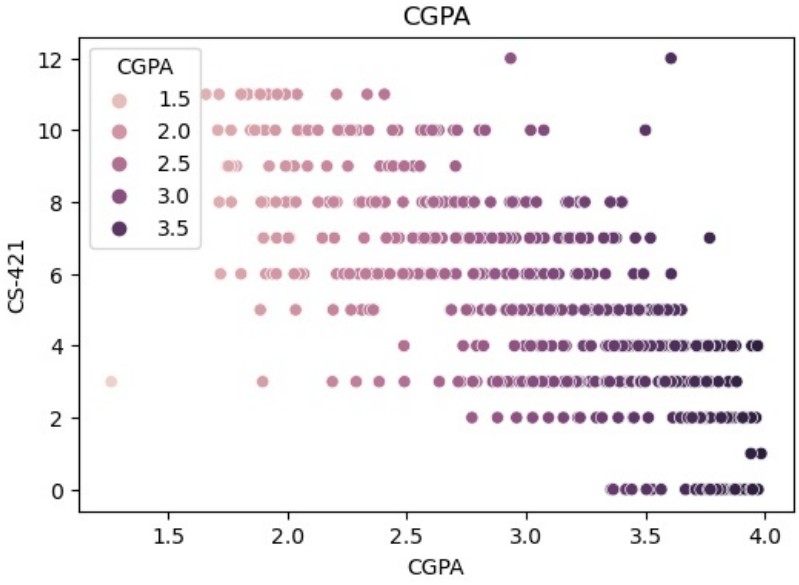

CGPA
CGPA
1.5
2.0
2.5
3.0
3.5
CS-318
0
2
4
6
8
10
12
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

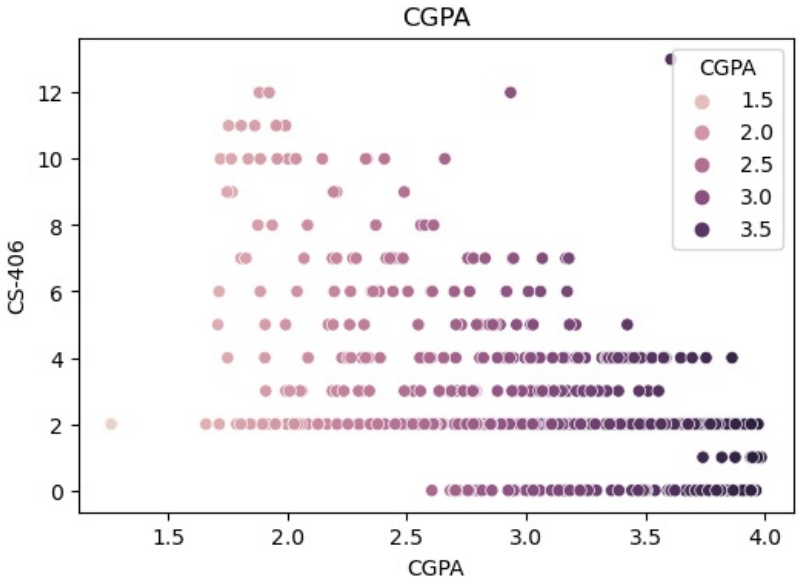

CGPA
CS-306
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
12
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

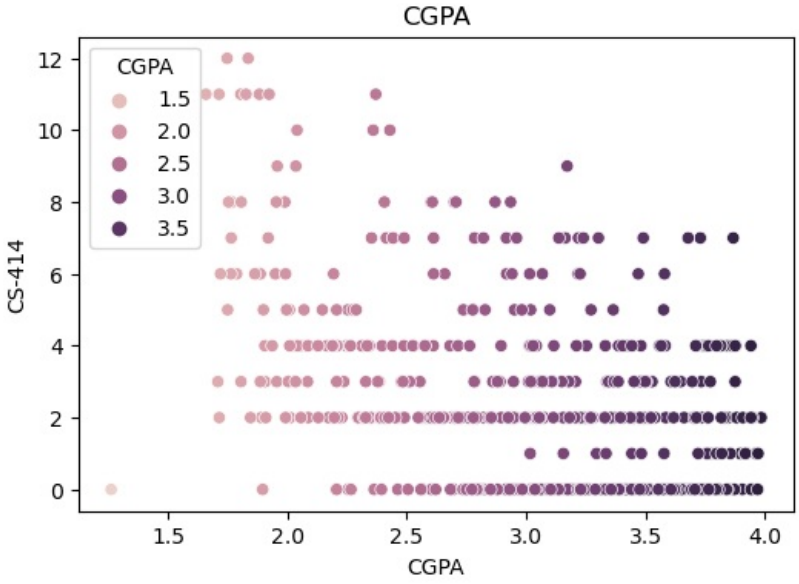

CGPA
CS-312
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
12
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

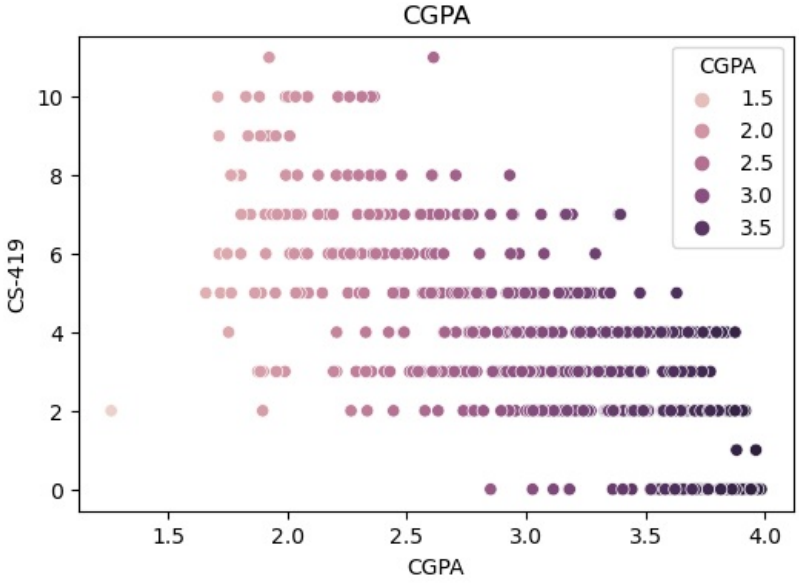

CGPA
CS-317
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

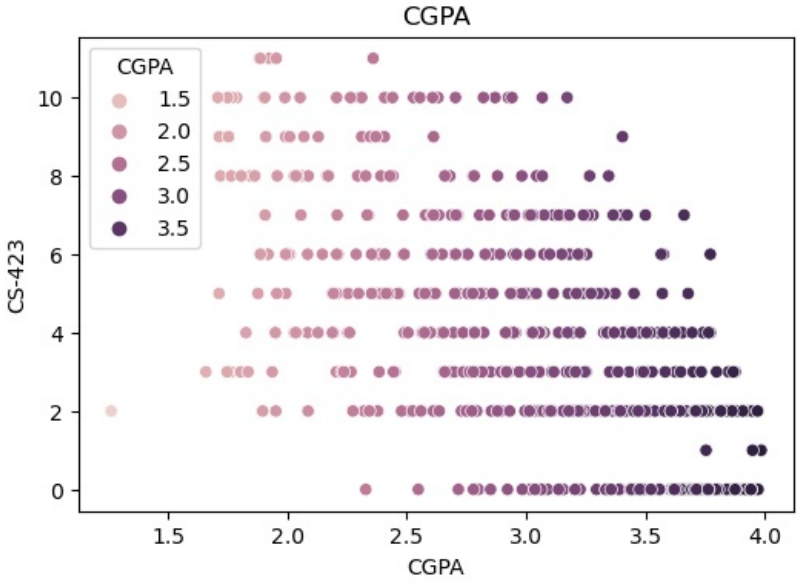

CGPA
CS-403
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
1.5
2.0
2.5
3.0
3.5
4.0
CGPA

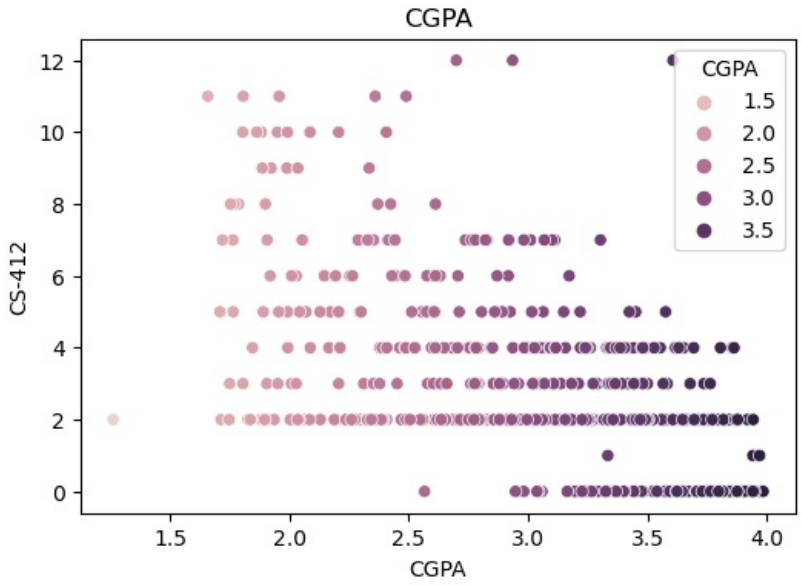

CGPA
CS-421
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
12
1.5
2.0
2.5
3.0
3.5
4.0
CGPA


CGPA
CS-406
CGPA
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
12
1.5
2.0
2.5
3.0
3.5
4.0
CGPA


CGPA
CGPA
1.5
2.0
2.5
3.0
3.5
CS-414
CGPA


CGPA
CGPA
1.5
2.0
2.5
3.0
3.5
CS-419
CGPA

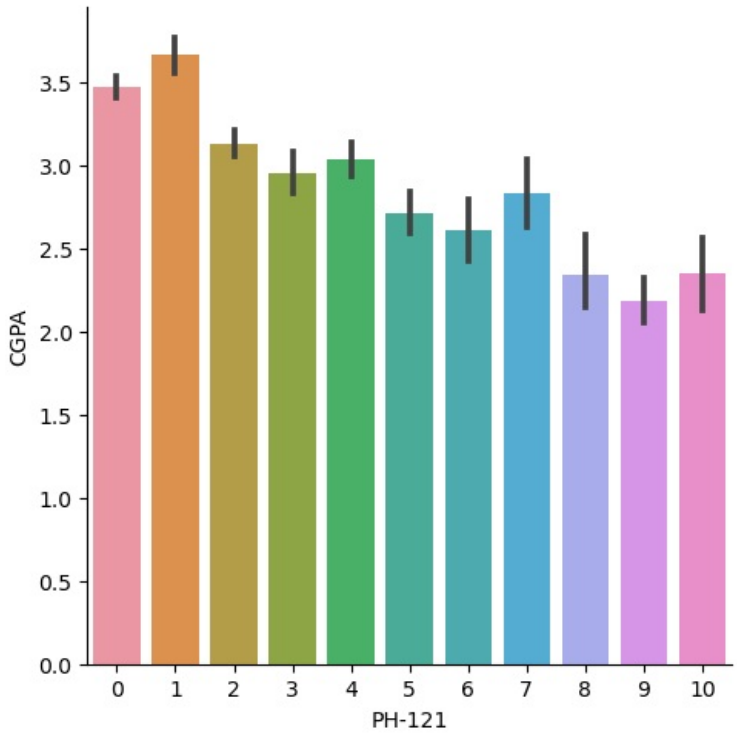

CGPA
CGPA
1.5
2.0
2.5
3.0
3.5
CS-423
CGPA



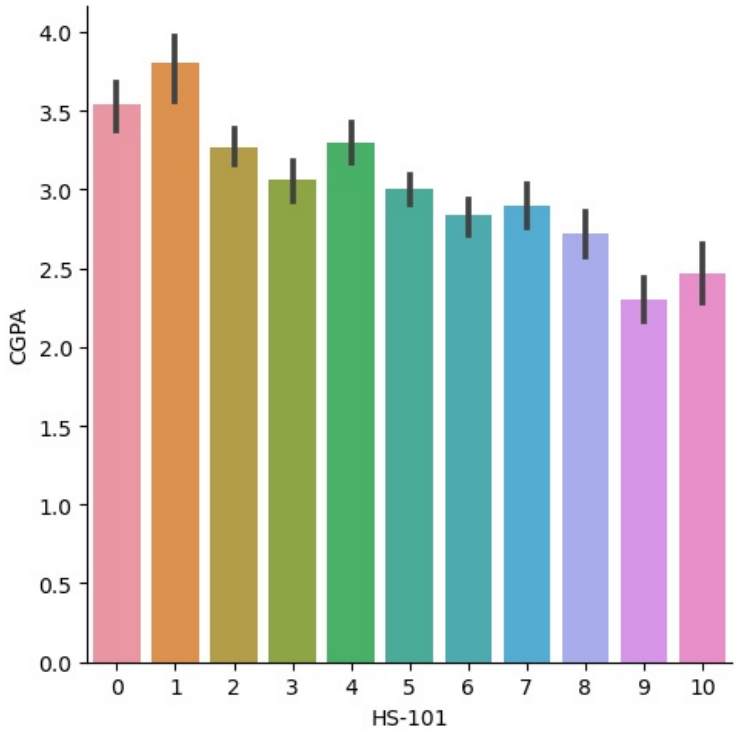
As we can see from these graphs, the majority of students are receiving good grades while maintaining high CGPAs.

In [64]:
```
# Bivariate Analysis: Co-relation Graph Plot
for col in df.drop(['CGPA'],axis=1):
    plt.figure(figsize=(6,4))
    sns.catplot(x=col,y='CGPA',data=df,kind='bar')
    plt.show()
```

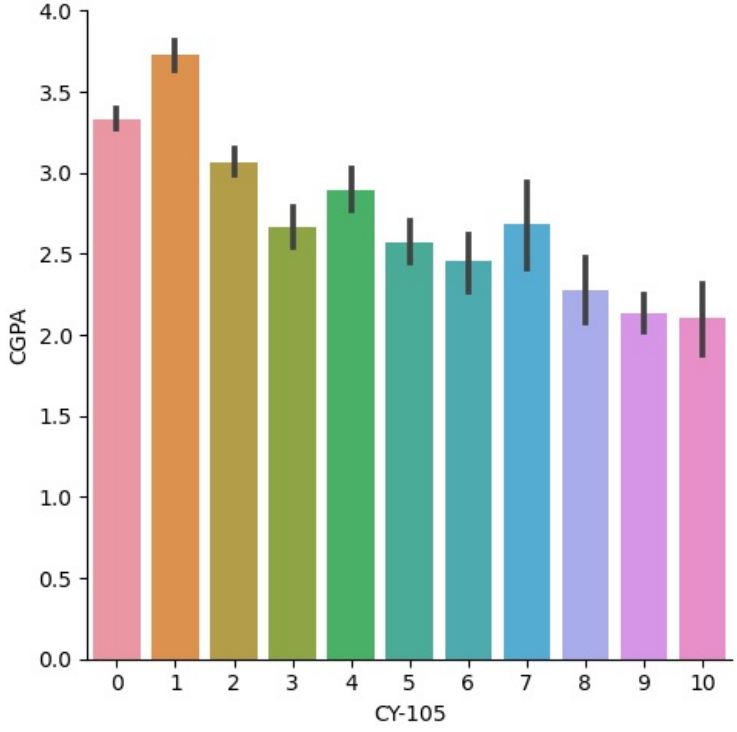
```
<Figure size 600x400 with 0 Axes>
```



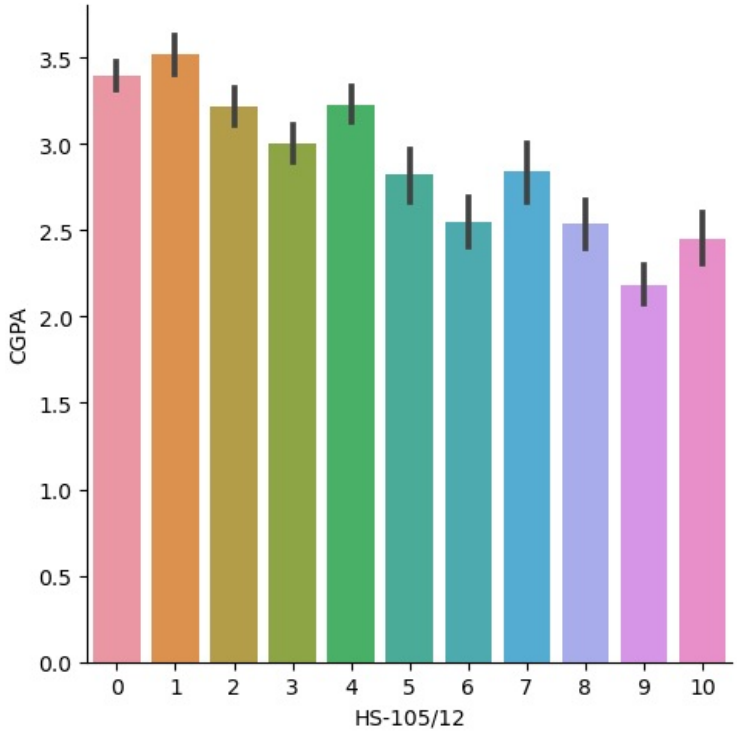

```
<Figure size 600x400 with 0 Axes>
```



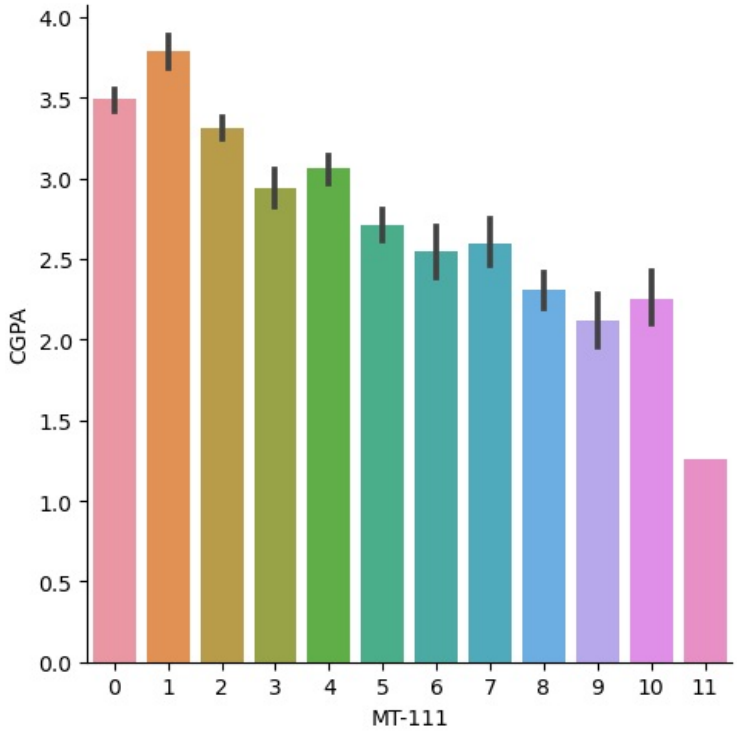
<Figure size 600x400 with 0 Axes>



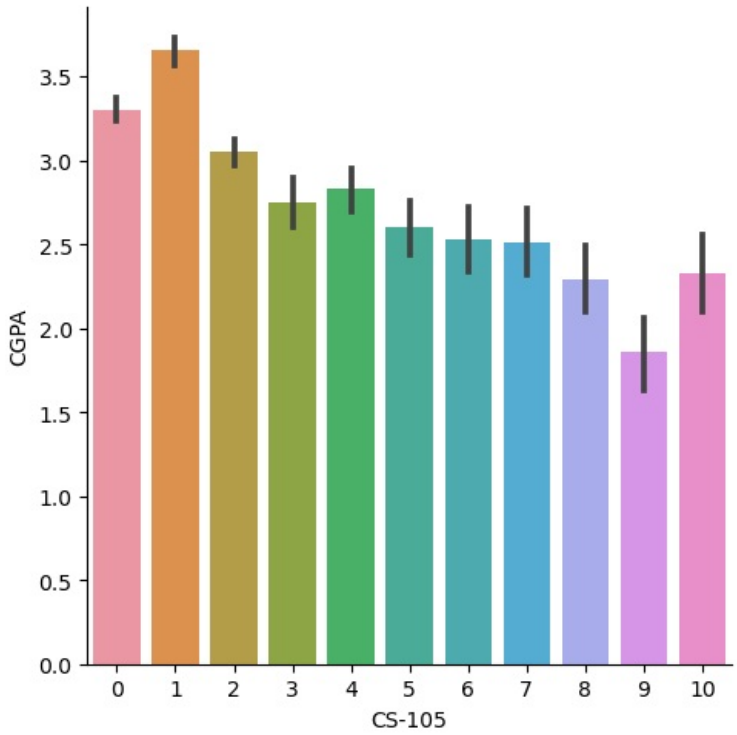
<Figure size 600x400 with 0 Axes>



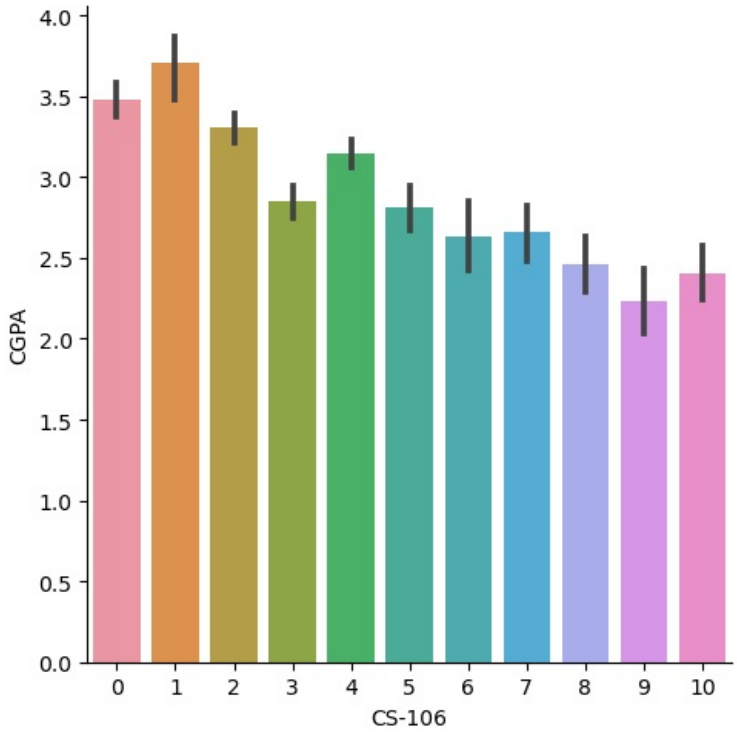
<Figure size 600x400 with 0 Axes>



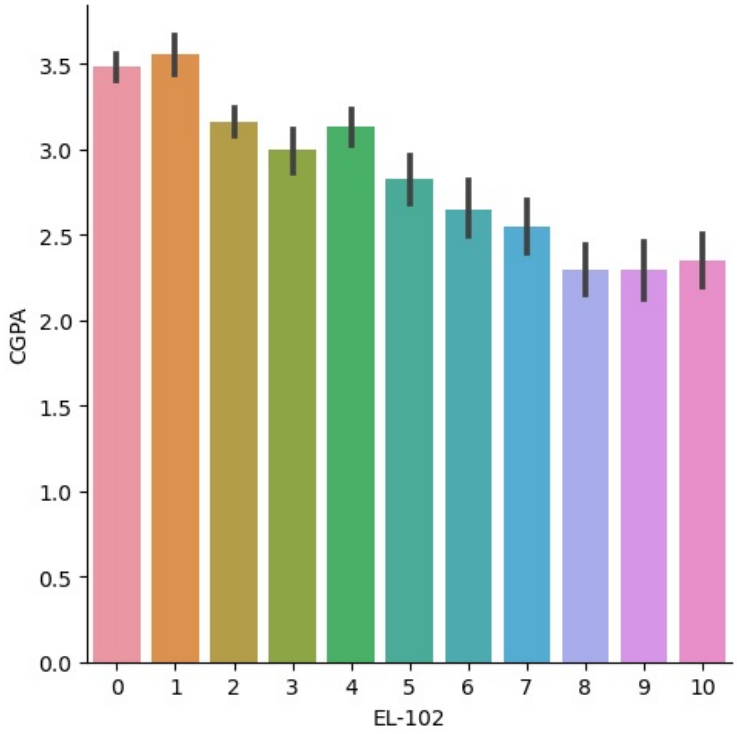
<Figure size 600x400 with 0 Axes>



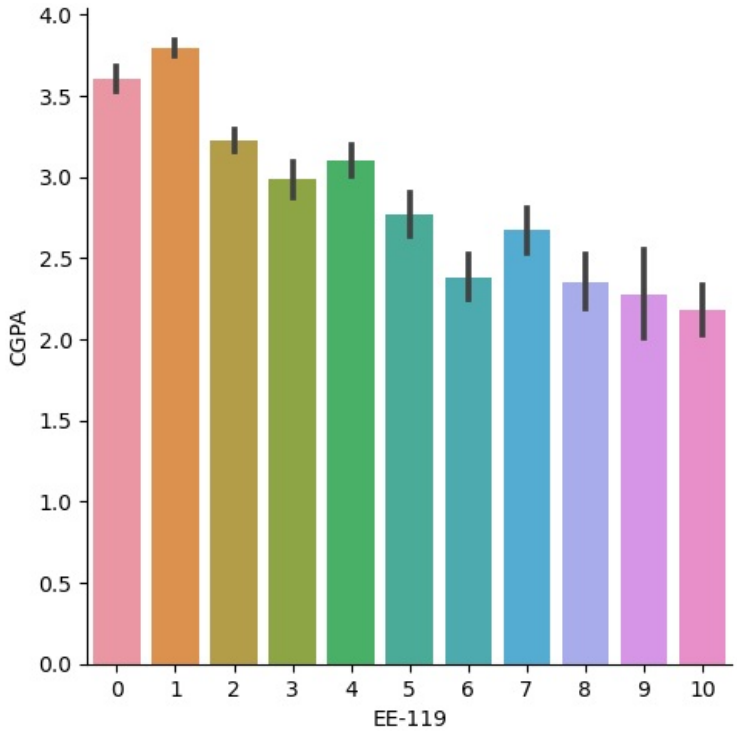
<Figure size 600x400 with 0 Axes>



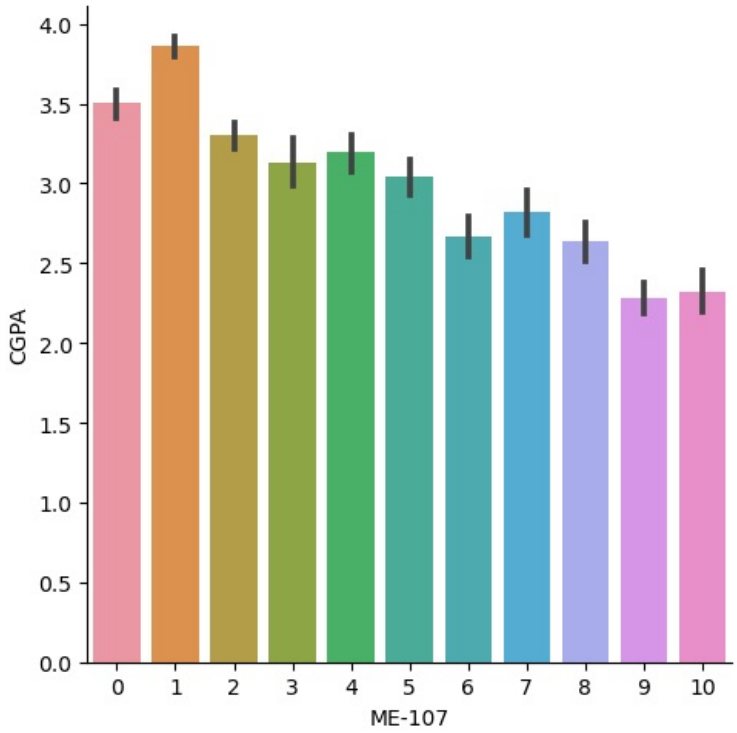
<Figure size 600x400 with 0 Axes>



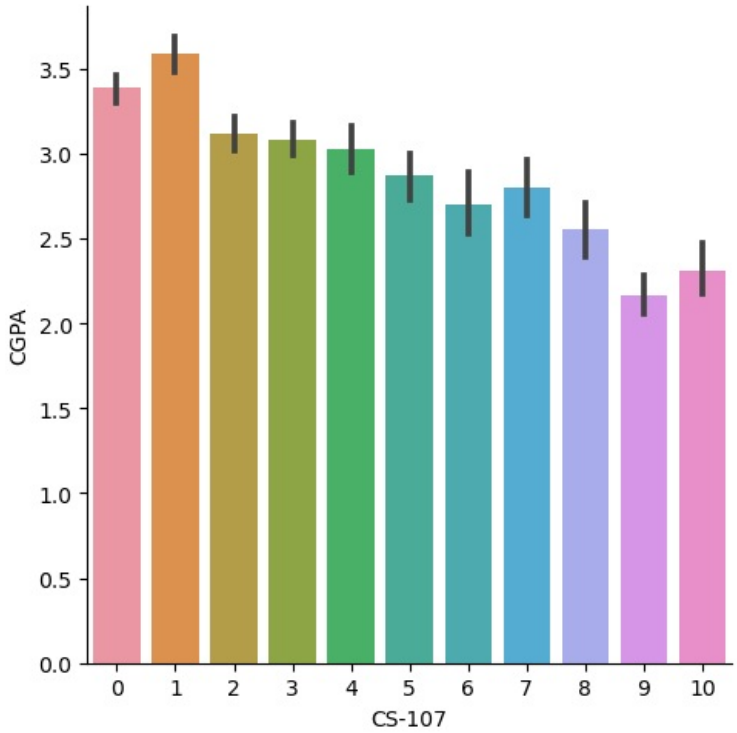
<Figure size 600x400 with 0 Axes>



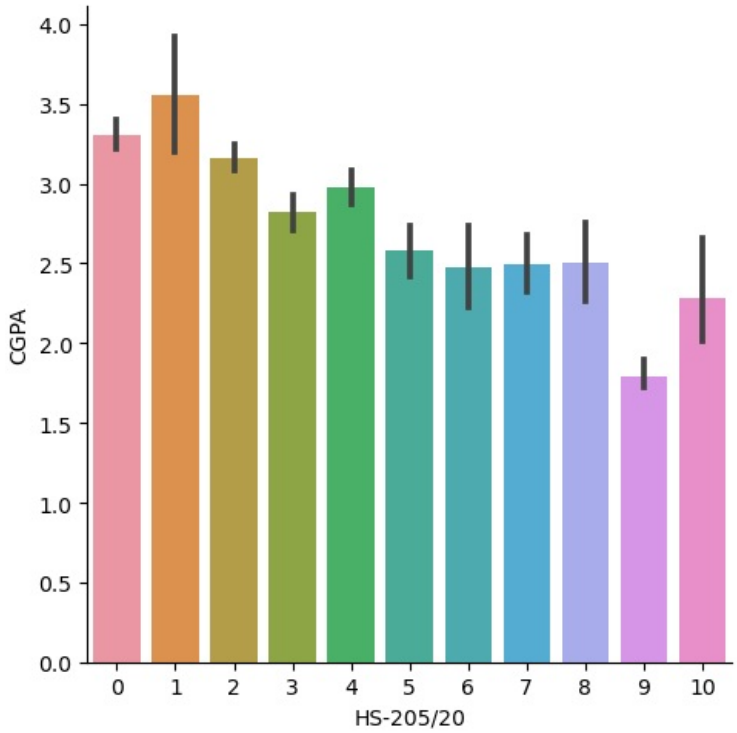
<Figure size 600x400 with 0 Axes>



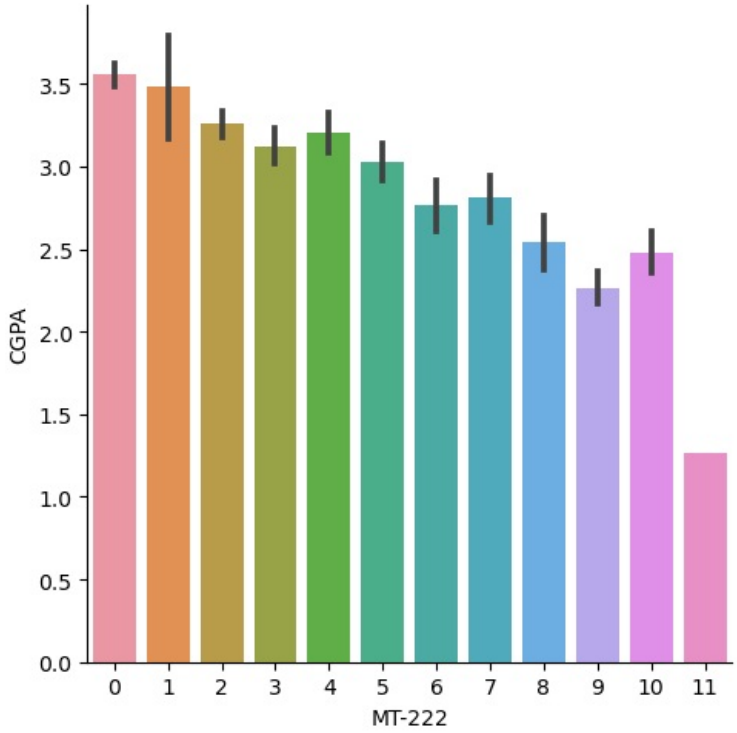
<Figure size 600x400 with 0 Axes>



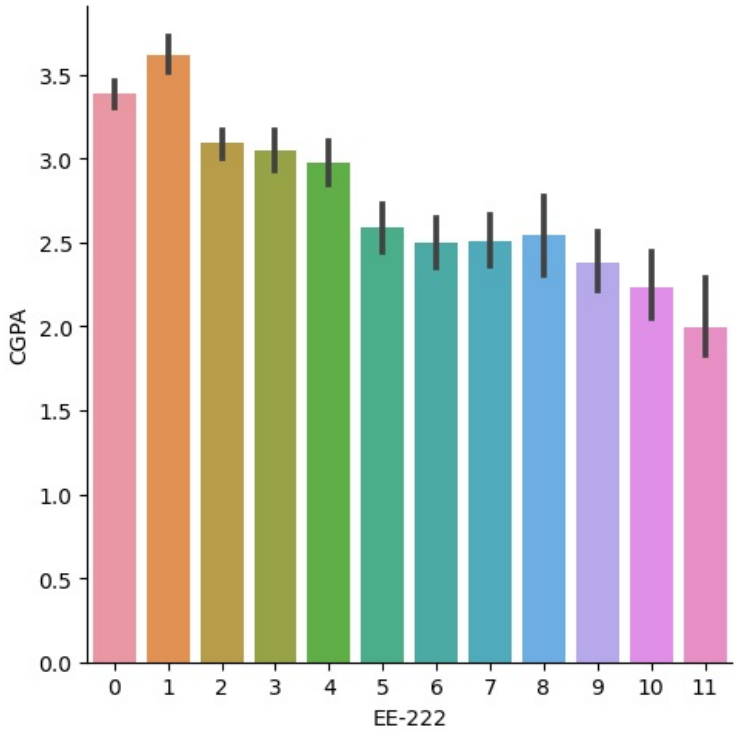
<Figure size 600x400 with 0 Axes>



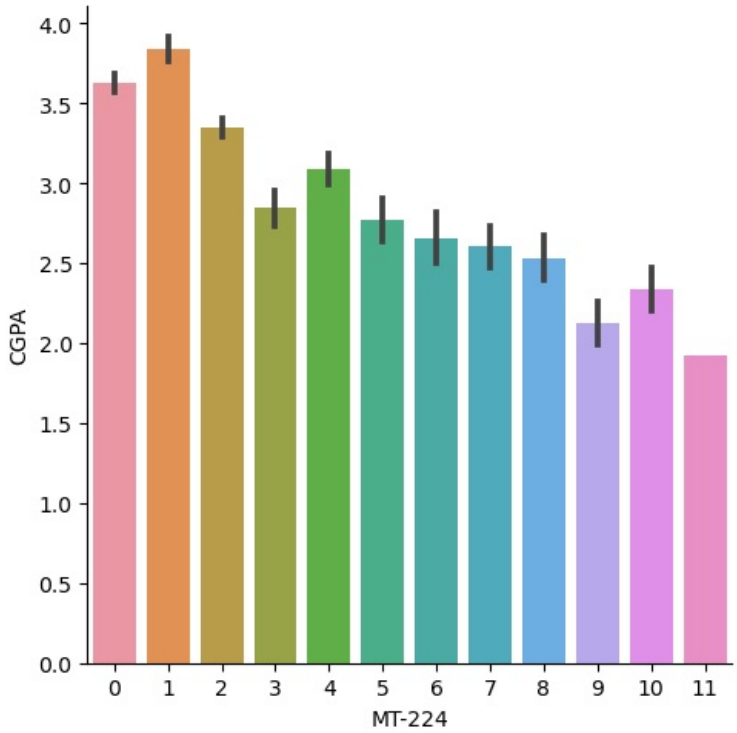
<Figure size 600x400 with 0 Axes>



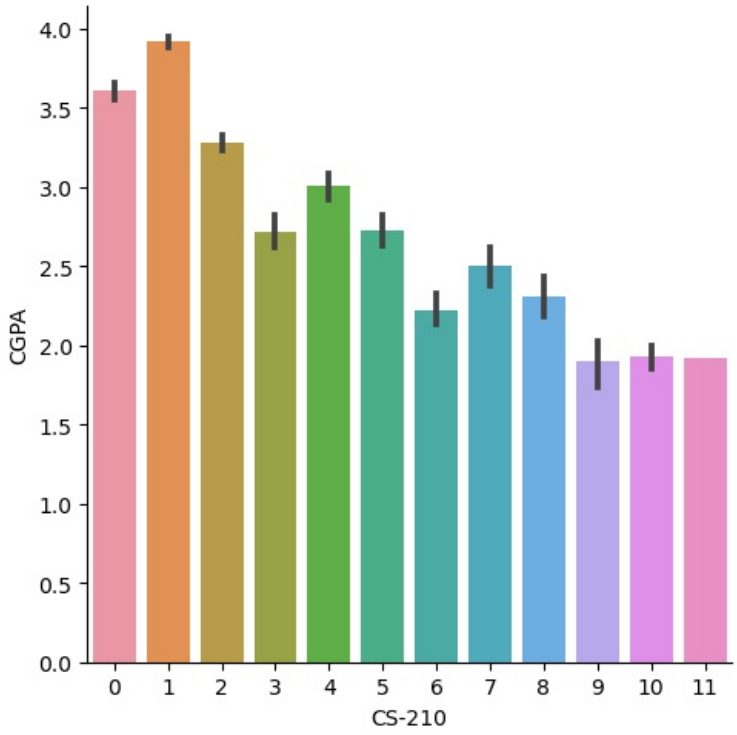
<Figure size 600x400 with 0 Axes>



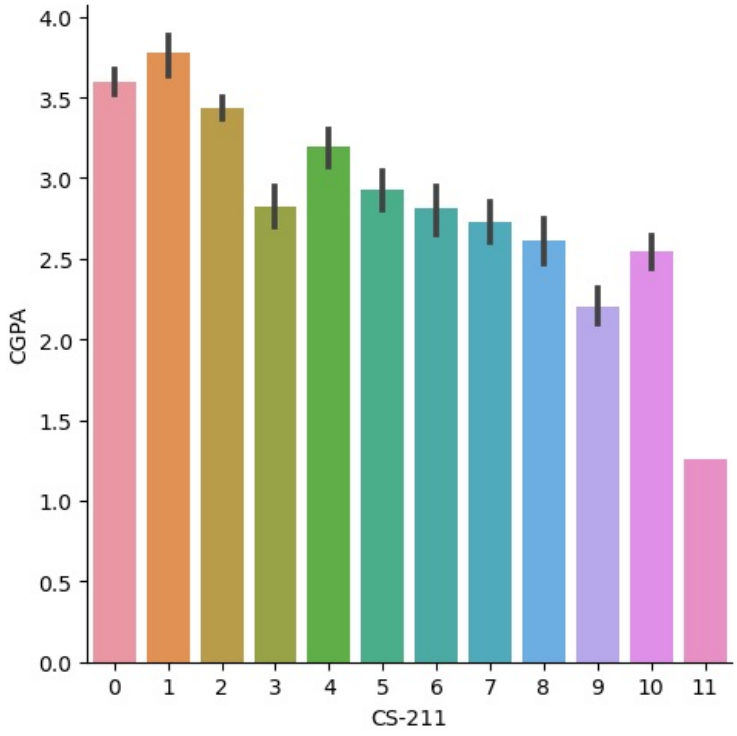
<Figure size 600x400 with 0 Axes>



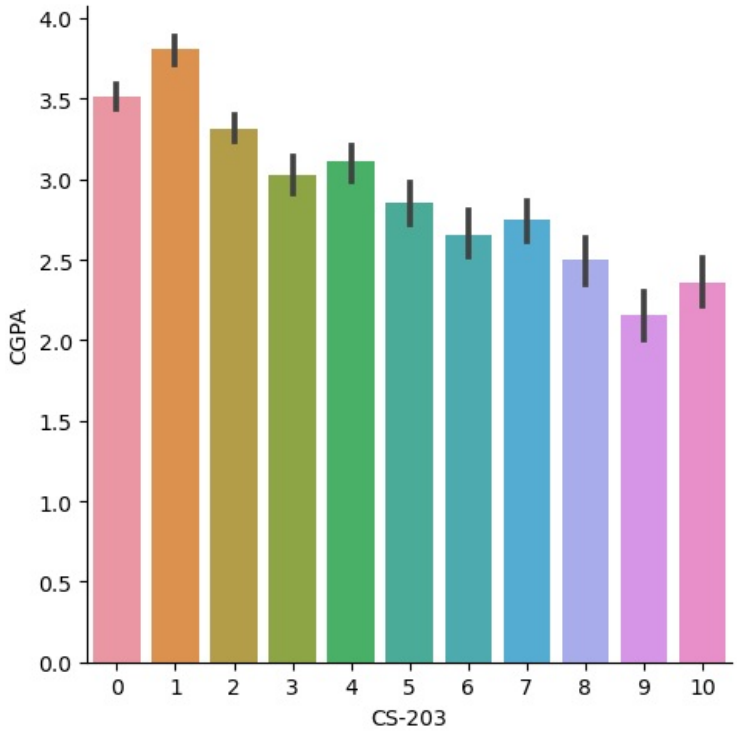
<Figure size 600x400 with 0 Axes>



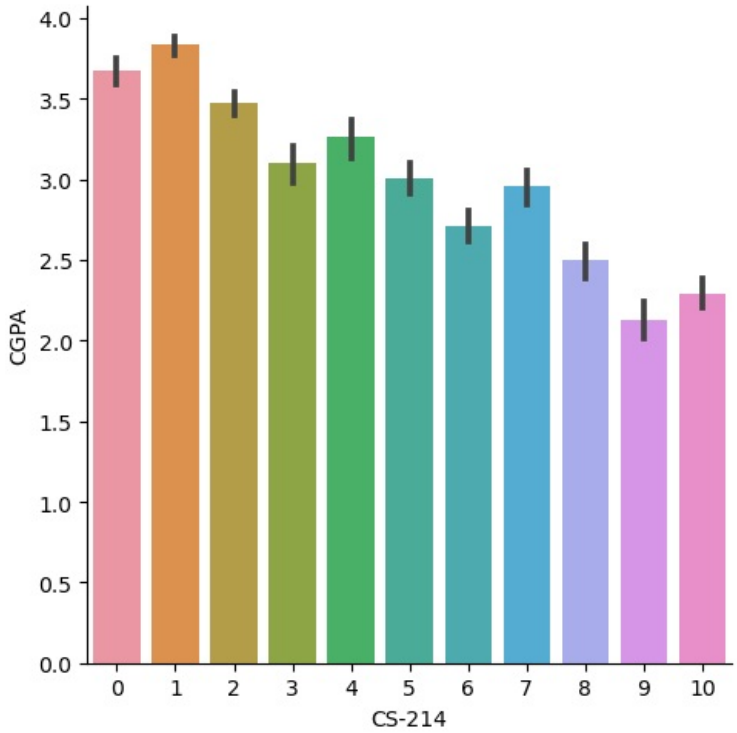
<Figure size 600x400 with 0 Axes>



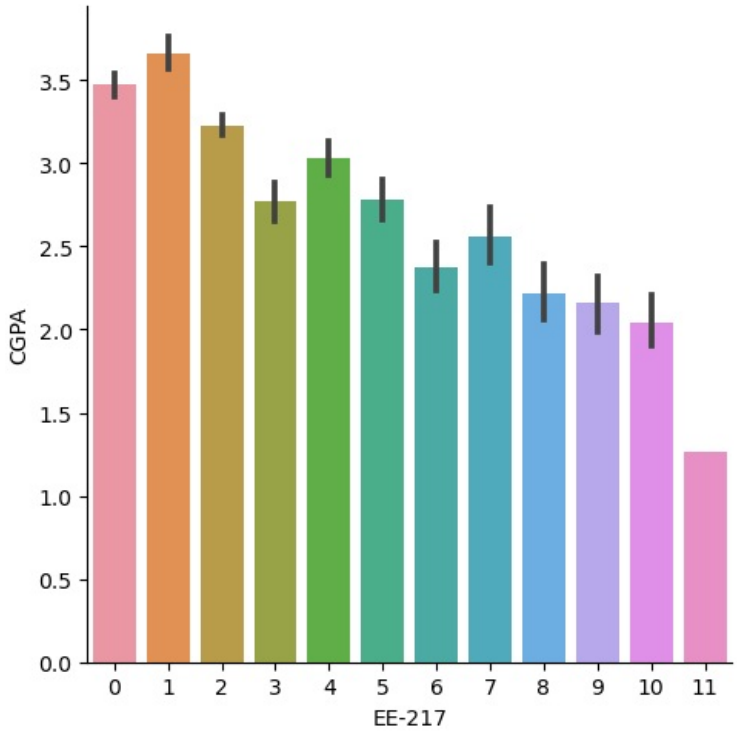
<Figure size 600x400 with 0 Axes>



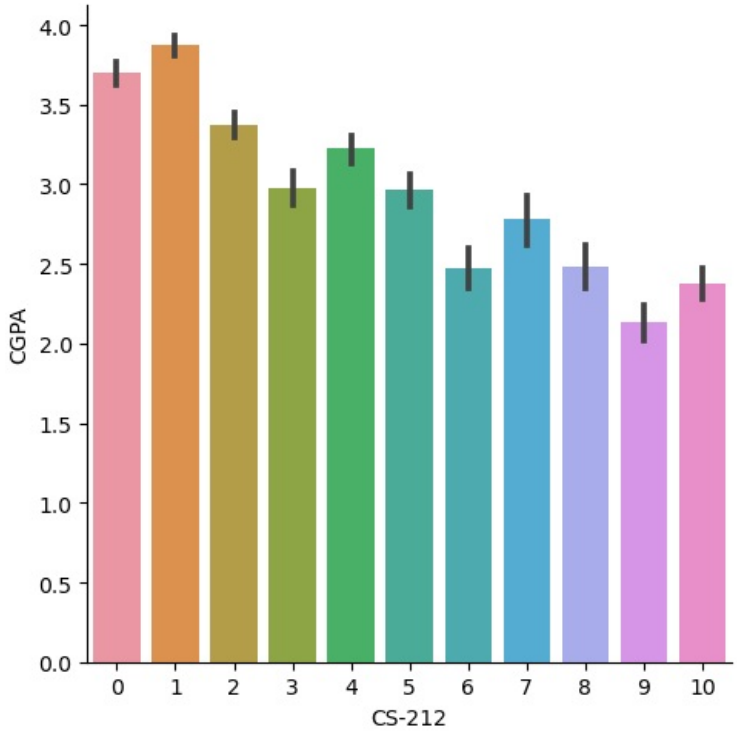
<Figure size 600x400 with 0 Axes>



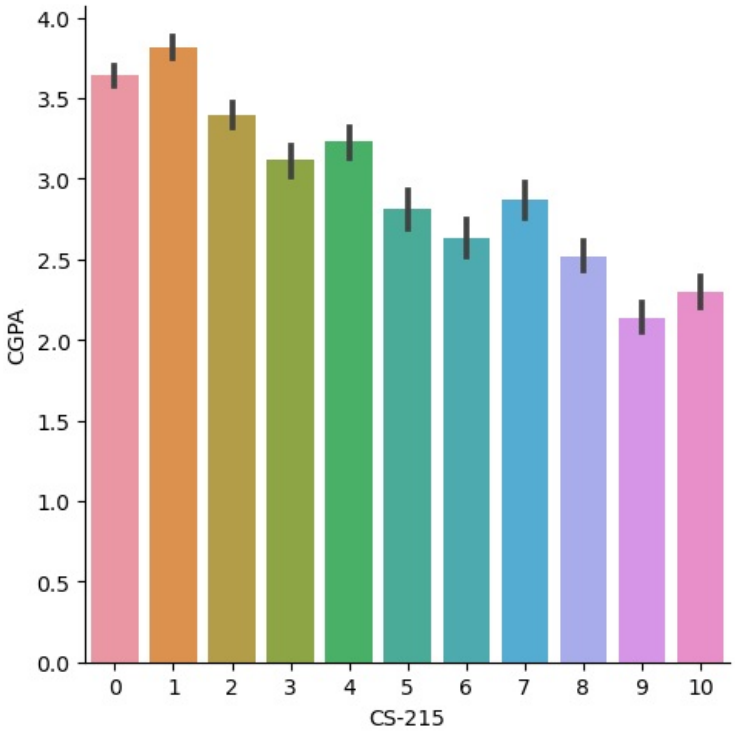
<Figure size 600x400 with 0 Axes>



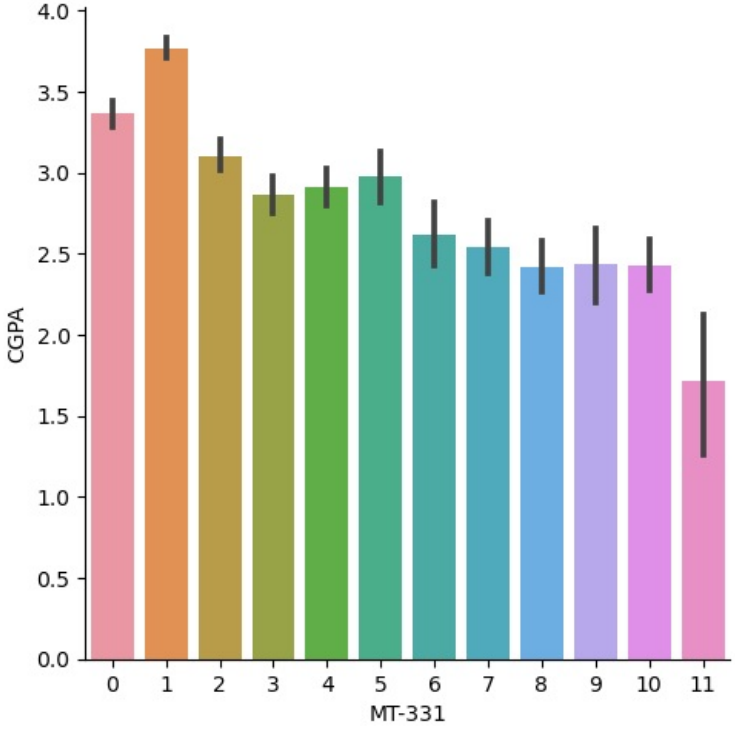
<Figure size 600x400 with 0 Axes>



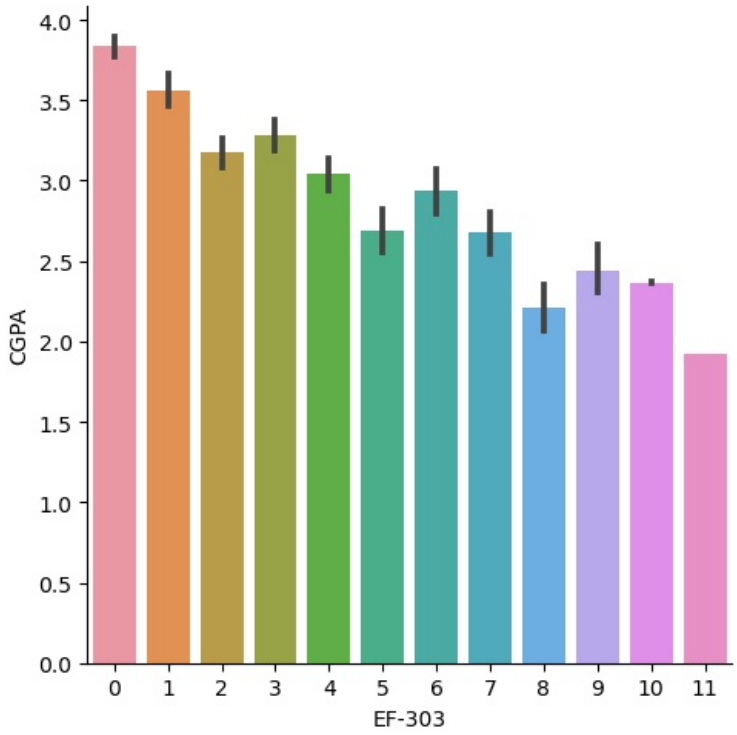
<Figure size 600x400 with 0 Axes>



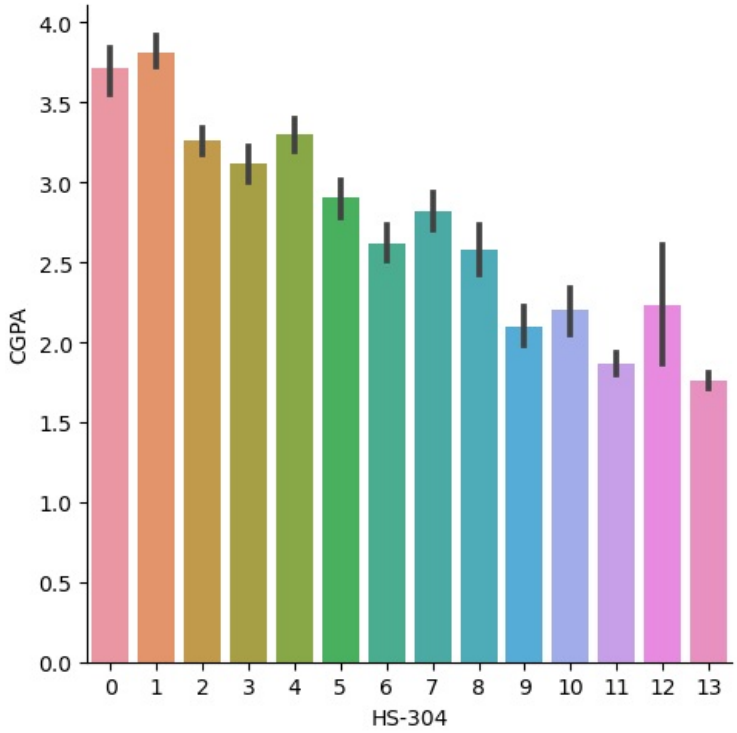
<Figure size 600x400 with 0 Axes>



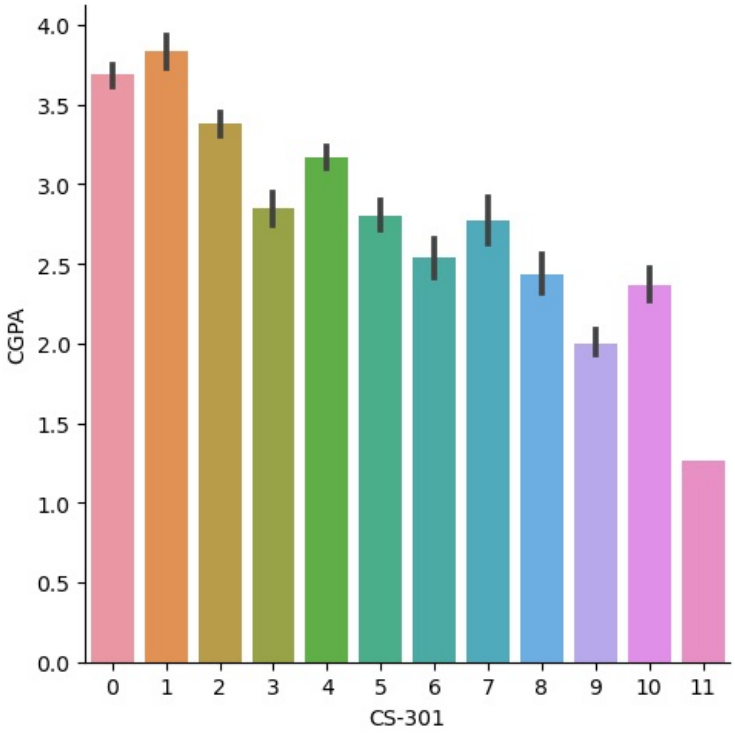
<Figure size 600x400 with 0 Axes>



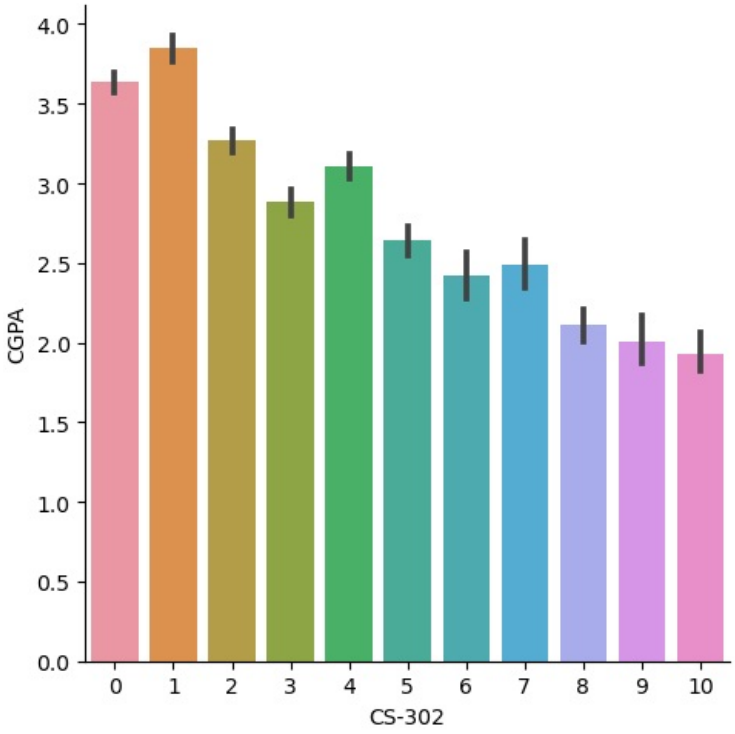
<Figure size 600x400 with 0 Axes>



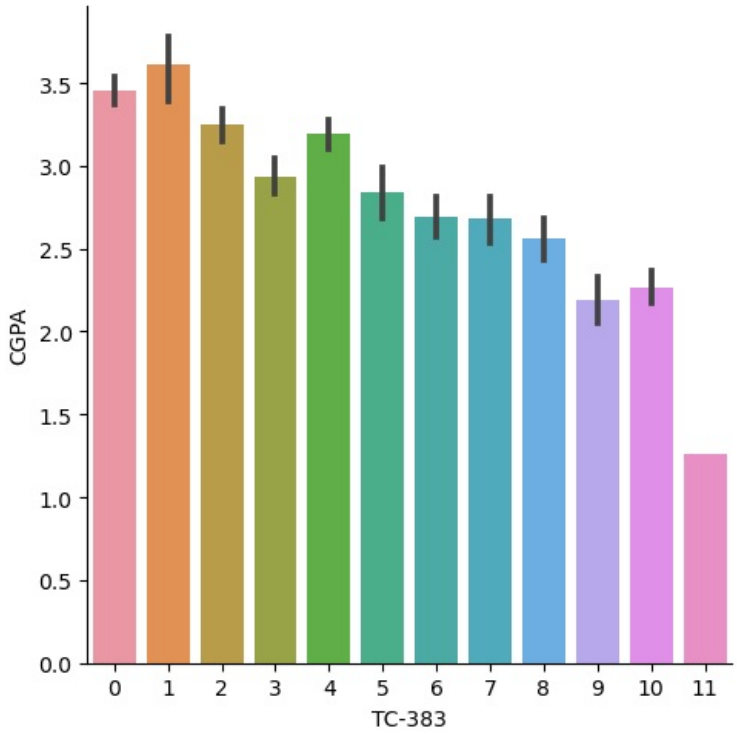
<Figure size 600x400 with 0 Axes>



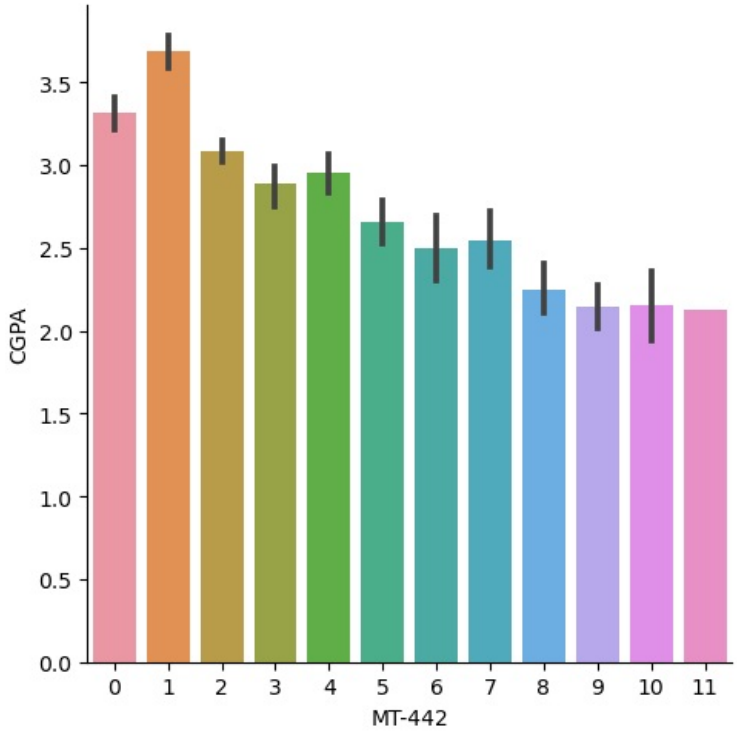
<Figure size 600x400 with 0 Axes>



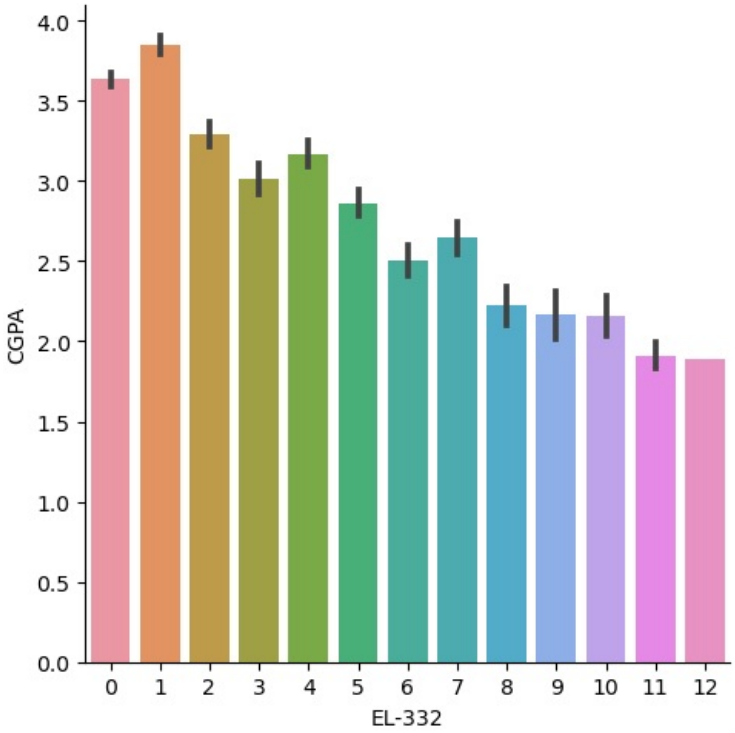
<Figure size 600x400 with 0 Axes>



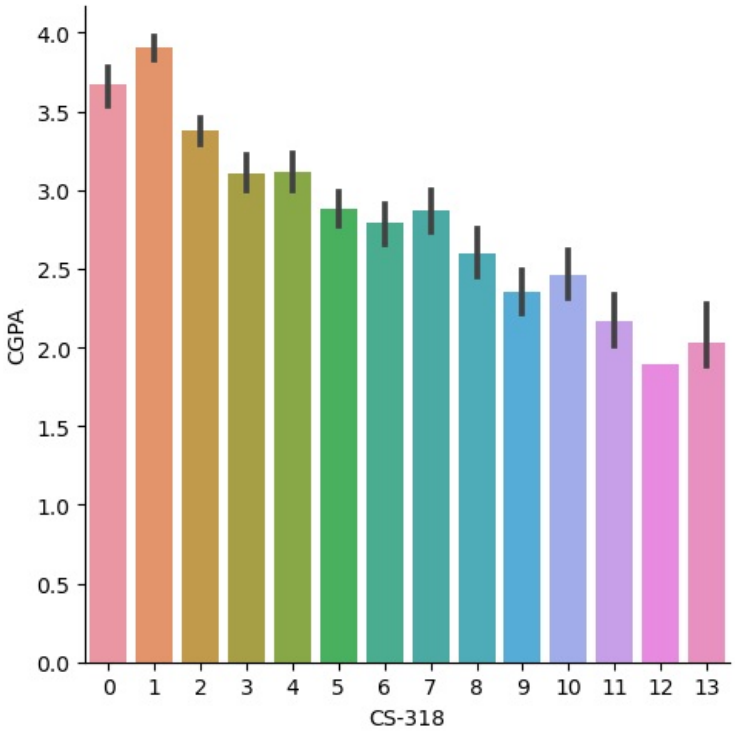
<Figure size 600x400 with 0 Axes>



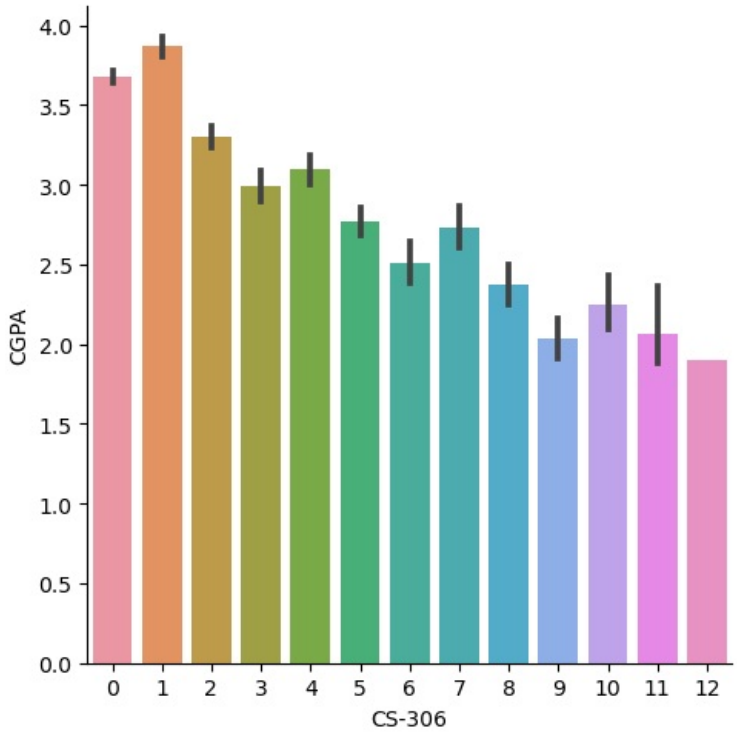
<Figure size 600x400 with 0 Axes>



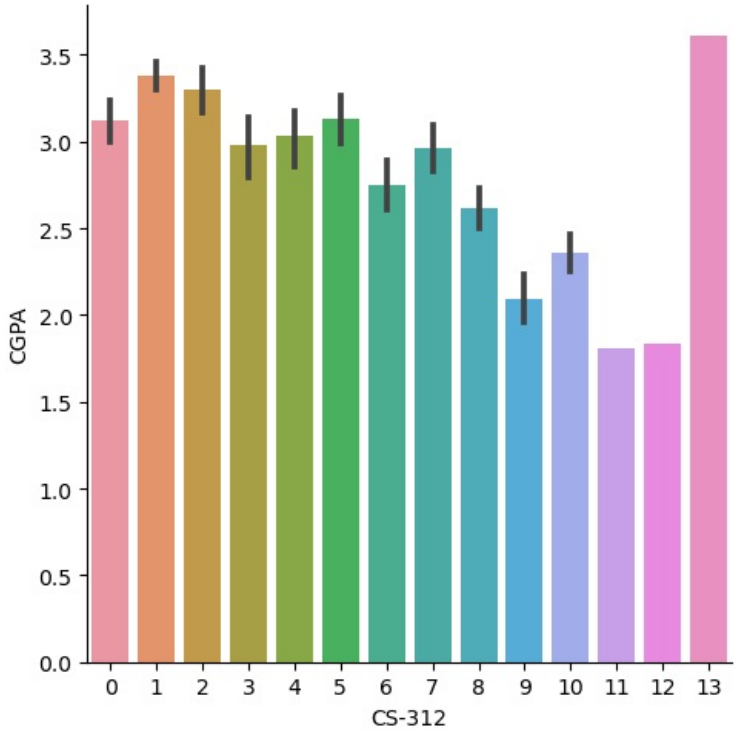
<Figure size 600x400 with 0 Axes>



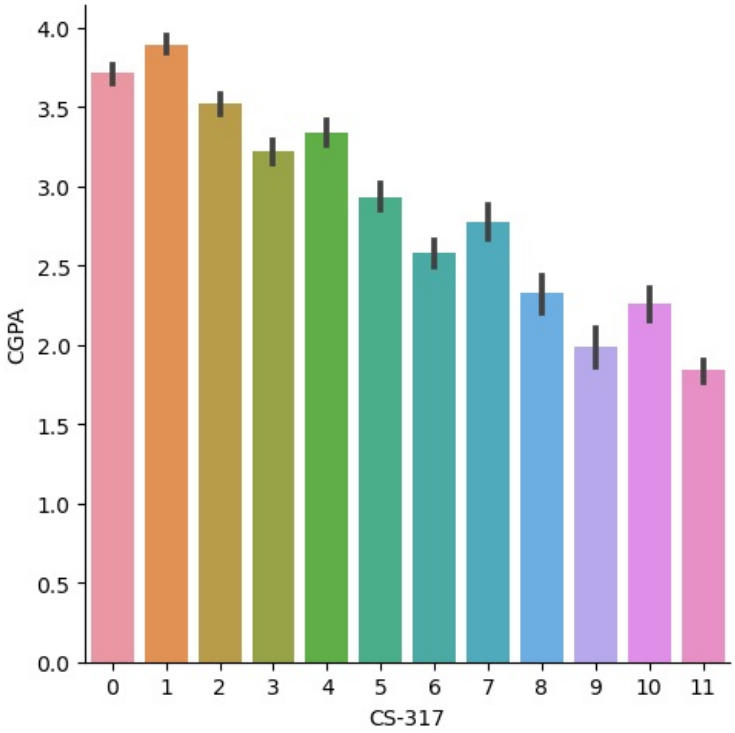
<Figure size 600x400 with 0 Axes>



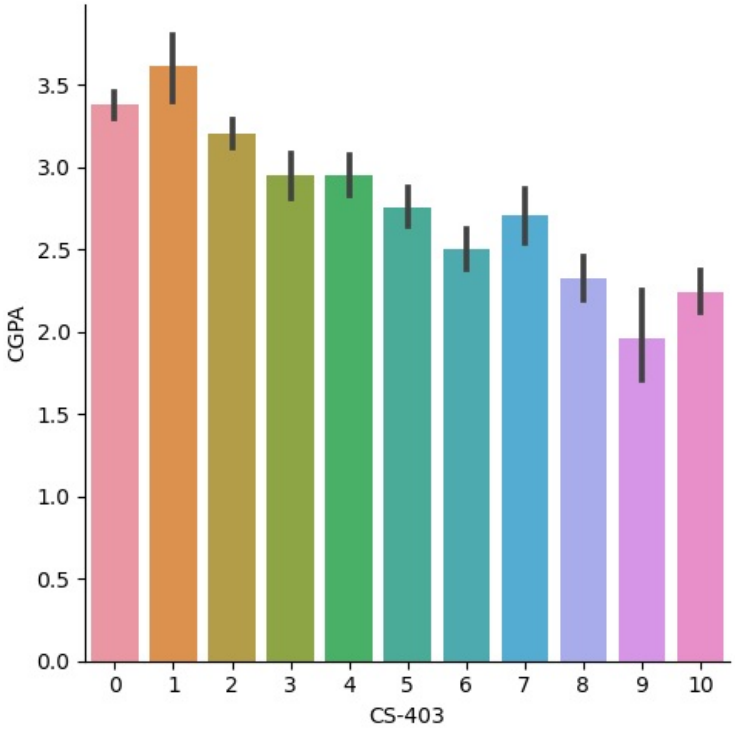
<Figure size 600x400 with 0 Axes>



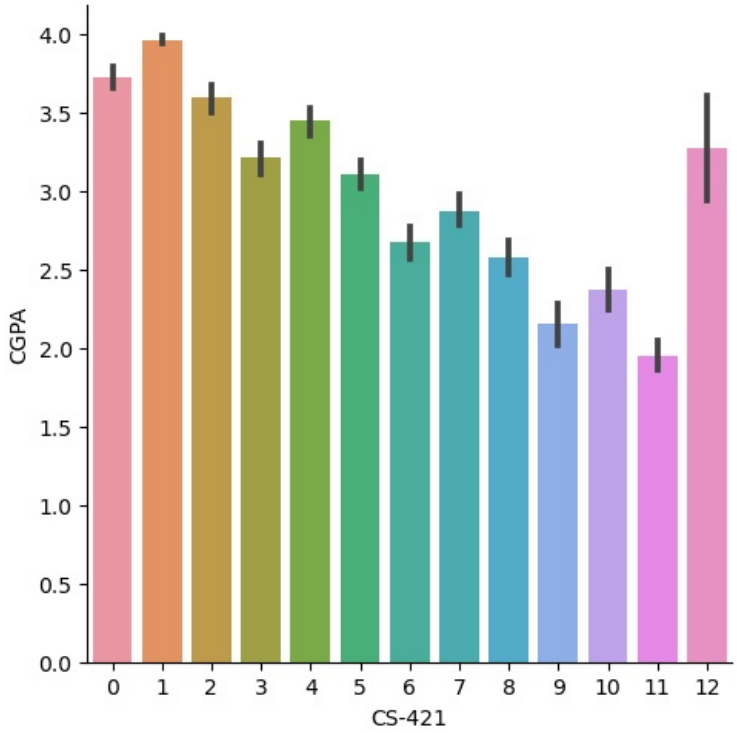
<Figure size 600x400 with 0 Axes>



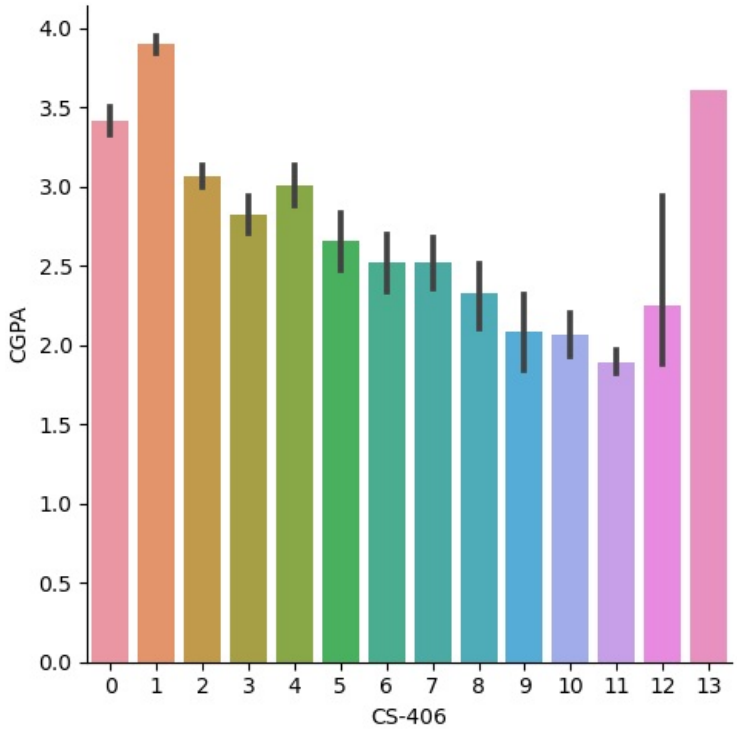
<Figure size 600x400 with 0 Axes>



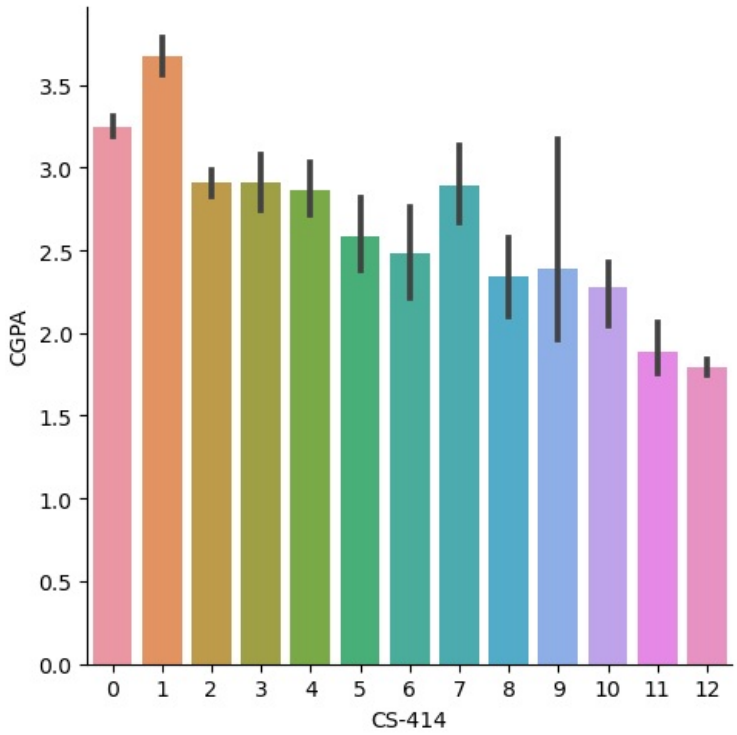
<Figure size 600x400 with 0 Axes>



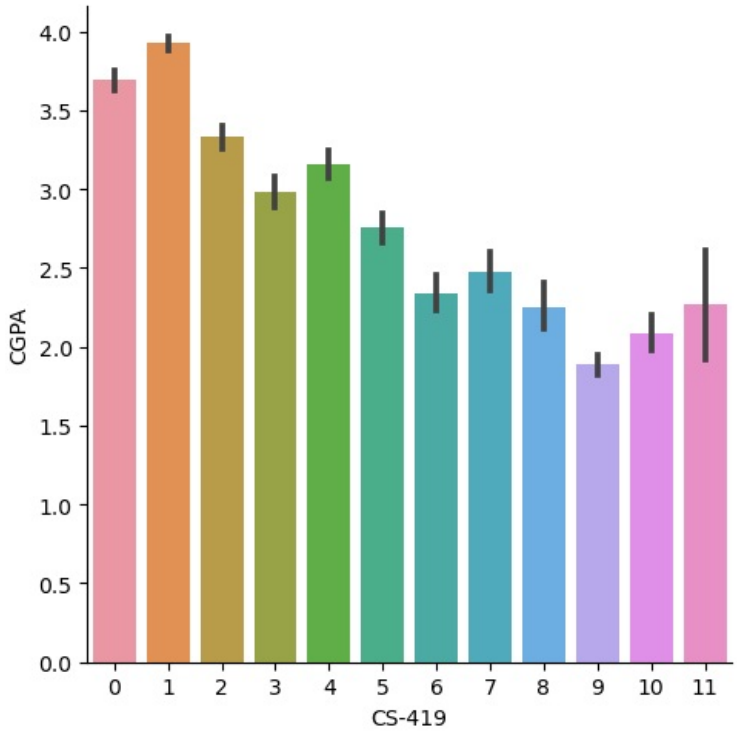
<Figure size 600x400 with 0 Axes>



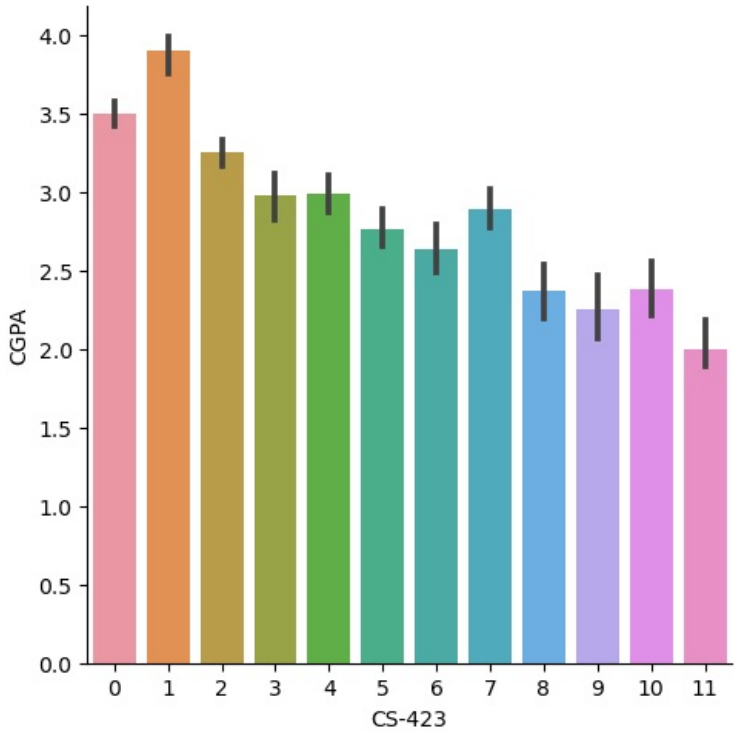
<Figure size 600x400 with 0 Axes>



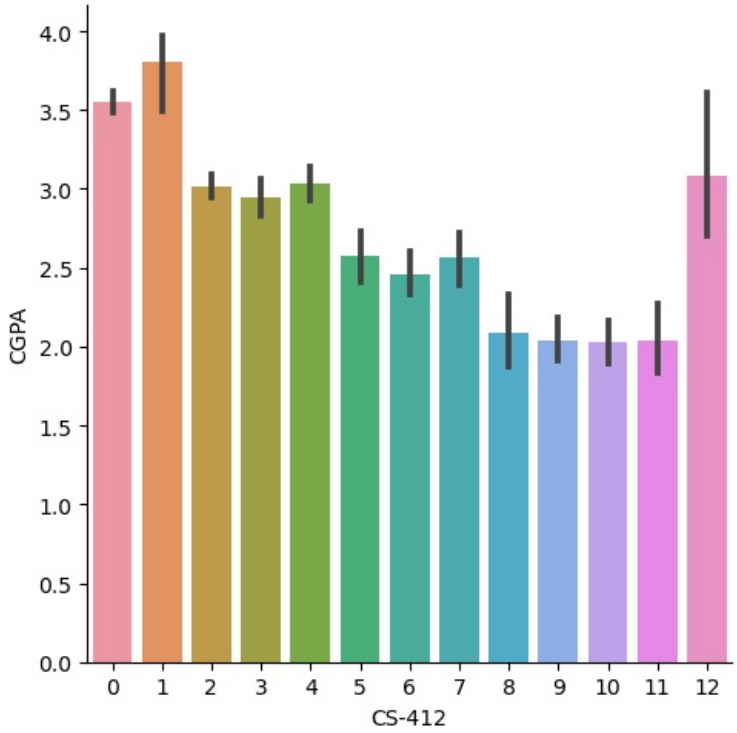
<Figure size 600x400 with 0 Axes>



<Figure size 600x400 with 0 Axes>



<Figure size 600x400 with 0 Axes>



If you receive grades of 0 and 1 in every topic, as shown in each column, your CGPA will be high. If you receive grades of 1, on the other hand, your CGPA will be fantastic.

the average grade and average CGPA are the remaining ones.

After finishing our bivariate analysis, we are going on to our multivariate analysis.

Multivariate research

In [65]:
```
df.corr()
```

Out[65]:

	PH-121	HS-101	CY-105	HS-105/12	MT-111	CS-105	CS-106	EL-102	EE-119	ME-107	...	CS-312	CS-317	
PH-121	1.000000	0.364662	0.542480	0.489461	0.457760	0.590384	0.278479	0.408450	0.580460	0.526575	...	0.065410	0.476340	0
HS-101	0.364662	1.000000	0.318734	0.466245	0.429801	0.383307	0.307744	0.345745	0.303358	0.327945	...	0.228937	0.367613	0
CY-105	0.542480	0.318734	1.000000	0.478800	0.519495	0.568184	0.349782	0.421849	0.484034	0.462803	...	0.298765	0.494913	0
HS-105/12	0.489461	0.466245	0.478800	1.000000	0.475902	0.474835	0.485935	0.341497	0.409673	0.312481	...	0.283881	0.489800	0
MT-111	0.457760	0.429801	0.519495	0.475902	1.000000	0.381212	0.491839	0.543150	0.403776	0.462213	...	0.509412	0.572161	0
CS-105	0.590384	0.383307	0.568184	0.474835	0.381212	1.000000	0.276018	0.345009	0.498891	0.440830	...	0.083275	0.397862	0
CS-106	0.278479	0.307744	0.349782	0.485935	0.491839	0.276018	1.000000	0.316386	0.286380	0.286875	...	0.489331	0.427427	0
EL-102	0.408450	0.345745	0.421849	0.341497	0.543150	0.345009	0.316386	1.000000	0.454893	0.460814	...	0.338204	0.480480	0
EE-119	0.580460	0.303358	0.484034	0.409673	0.403776	0.498891	0.286380	0.454893	1.000000	0.560468	...	0.176641	0.454772	0
ME-107	0.526575	0.327945	0.462803	0.312481	0.462213	0.440830	0.286875	0.460814	0.560468	1.000000	...	0.216019	0.472185	0
CS-107	0.471586	0.502361	0.451199	0.526392	0.468792	0.507680	0.404435	0.419050	0.399198	0.333642	...	0.217646	0.426655	0
HS-205/20	0.289300	0.175735	0.238308	0.165138	0.316970	0.176427	0.112933	0.362221	0.333972	0.387771	...	0.124640	0.392167	0
MT-222	0.562491	0.397886	0.488865	0.461679	0.499918	0.481531	0.375458	0.457086	0.461305	0.484912	...	0.278218	0.541047	0
EE-222	0.551562	0.389994	0.423165	0.390964	0.401936	0.509679	0.212304	0.430263	0.455495	0.400762	...	0.056283	0.466978	0
MT-224	0.352519	0.267045	0.462872	0.400780	0.578365	0.262957	0.409346	0.504926	0.423207	0.479870	...	0.521198	0.614377	0
CS-210	0.518775	0.320298	0.503007	0.465133	0.582871	0.444559	0.438235	0.495737	0.502674	0.552505	...	0.427556	0.657846	0
CS-211	0.330758	0.341461	0.457750	0.351412	0.559808	0.341297	0.372034	0.500261	0.453432	0.486825	...	0.525848	0.488971	0
CS-203	0.467551	0.263625	0.392086	0.286507	0.515142	0.409914	0.295224	0.476247	0.471217	0.603162	...	0.301178	0.496516	0
CS-214	0.410076	0.460450	0.456118	0.405390	0.586454	0.406419	0.430314	0.512849	0.453770	0.540600	...	0.439340	0.596091	0
EE-217	0.427052	0.332679	0.388839	0.347970	0.520679	0.338811	0.332001	0.540070	0.385694	0.456728	...	0.267820	0.491404	0
CS-212	0.355068	0.341184	0.436052	0.465238	0.575886	0.321446	0.558814	0.438183	0.384684	0.460895	...	0.574870	0.543512	0
CS-215	0.493052	0.403469	0.532072	0.462283	0.586355	0.460635	0.396805	0.486480	0.538415	0.569099	...	0.391917	0.580112	0
MT-331	0.224737	0.192493	0.419632	0.285221	0.479254	0.269991	0.310384	0.379440	0.370990	0.356879	...	0.520653	0.465551	0
EF-303	0.233253	0.260642	0.390858	0.417683	0.446447	0.231212	0.470417	0.273927	0.265599	0.260726	...	0.575856	0.482689	0
HS-304	0.465856	0.441408	0.378287	0.376659	0.346363	0.500632	0.160621	0.394740	0.489083	0.429105	...	0.047723	0.471284	0
CS-301	0.335601	0.331199	0.418990	0.445205	0.574229	0.260676	0.514491	0.443139	0.391128	0.431564	...	0.594583	0.634075	0
CS-302	0.482036	0.360221	0.509041	0.444393	0.537211	0.458363	0.385291	0.415193	0.468404	0.466906	...	0.332825	0.647507	0
TC-383	0.237547	0.349655	0.369312	0.441173	0.542332	0.260645	0.544906	0.403689	0.260650	0.295905	...	0.636428	0.567986	0
MT-442	0.469691	0.235440	0.467895	0.337331	0.425729	0.431416	0.261794	0.370634	0.466225	0.456081	...	0.176407	0.506541	0
EL-332	0.455202	0.394341	0.391191	0.374887	0.607271	0.391187	0.385408	0.530719	0.479712	0.522369	...	0.412499	0.607790	0
CS-318	0.487258	0.425836	0.367128	0.400547	0.408317	0.486345	0.268056	0.379361	0.399870	0.394601	...	0.137165	0.526785	0
CS-306	0.510454	0.449423	0.403120	0.510591	0.508714	0.451801	0.366770	0.446901	0.442248	0.422442	...	0.271005	0.673274	0
CS-312	0.065410	0.228937	0.298765	0.283881	0.509412	0.083275	0.489331	0.338204	0.176641	0.216019	...	1.000000	0.462030	0

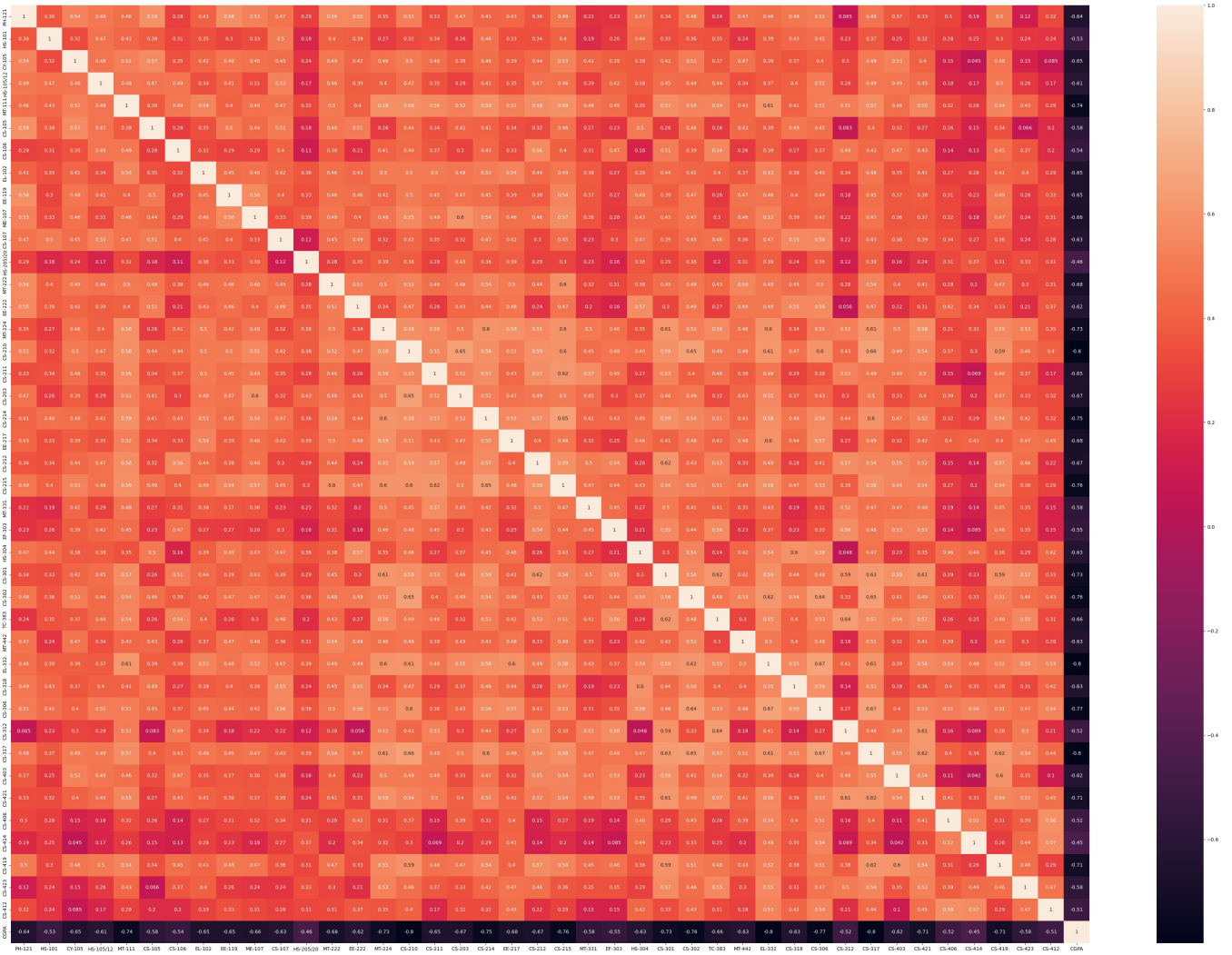
												...		
CS-317	0.476340	0.367613	0.494913	0.489800	0.572161	0.397862	0.427427	0.480480	0.454772	0.472185	...	0.462030	1.000000	
CS-403	0.368667	0.250034	0.525173	0.486041	0.455681	0.319673	0.470442	0.348618	0.372093	0.362915	...	0.487118	0.546391	
CS-421	0.327924	0.321463	0.403245	0.447928	0.548722	0.269655	0.429159	0.406521	0.380161	0.372211	...	0.608301	0.618447	
CS-406	0.301207	0.275355	0.151716	0.181047	0.318325	0.256310	0.141503	0.272197	0.309460	0.315541	...	0.158918	0.401342	
CS-414	0.187699	0.245779	0.044900	0.167917	0.256619	0.153554	0.132402	0.277301	0.232167	0.176587	...	0.089371	0.339997	
CS-419	0.503203	0.301938	0.484092	0.496238	0.536780	0.339665	0.452287	0.414231	0.488164	0.467242	...	0.382856	0.617739	
CS-423	0.119664	0.242951	0.153844	0.256895	0.429186	0.066309	0.365533	0.401338	0.259521	0.239895	...	0.504500	0.537178	
CS-412	0.316964	0.236217	0.085047	0.166924	0.287769	0.198109	0.200543	0.286203	0.333823	0.309134	...	0.212184	0.435420	
CGPA	-0.642891	-0.525553	-0.647309	-0.614840	-0.744468	-0.583631	-0.541780	-0.652520	-0.650335	-0.661765	...	-0.515231	-0.799364	

42 rows × 42 columns

We use a heat map to better visualise what we can't see clearly.

In [66]:
```
plt.figure(figsize=(50,35))
sns.heatmap(df.corr(),annot=True)
```

Out[66]:
```
<AxesSubplot:>
```



As we can see, the target variable has a highly negative correlation with every CGPA column.

Plotting the correlation of the features now.

In [67]:
```
df.drop(['CGPA'],axis=1).corr()
```

Out[67]:

	PH-121	HS-101	CY-105	HS-105/12	MT-111	CS-105	CS-106	EL-102	EE-119	ME-107	...	CS-306	CS-312	CS-317
PH-121	1.000000	0.364662	0.542480	0.489461	0.457760	0.590384	0.278479	0.408450	0.580460	0.526575	...	0.510454	0.065410	0.476340

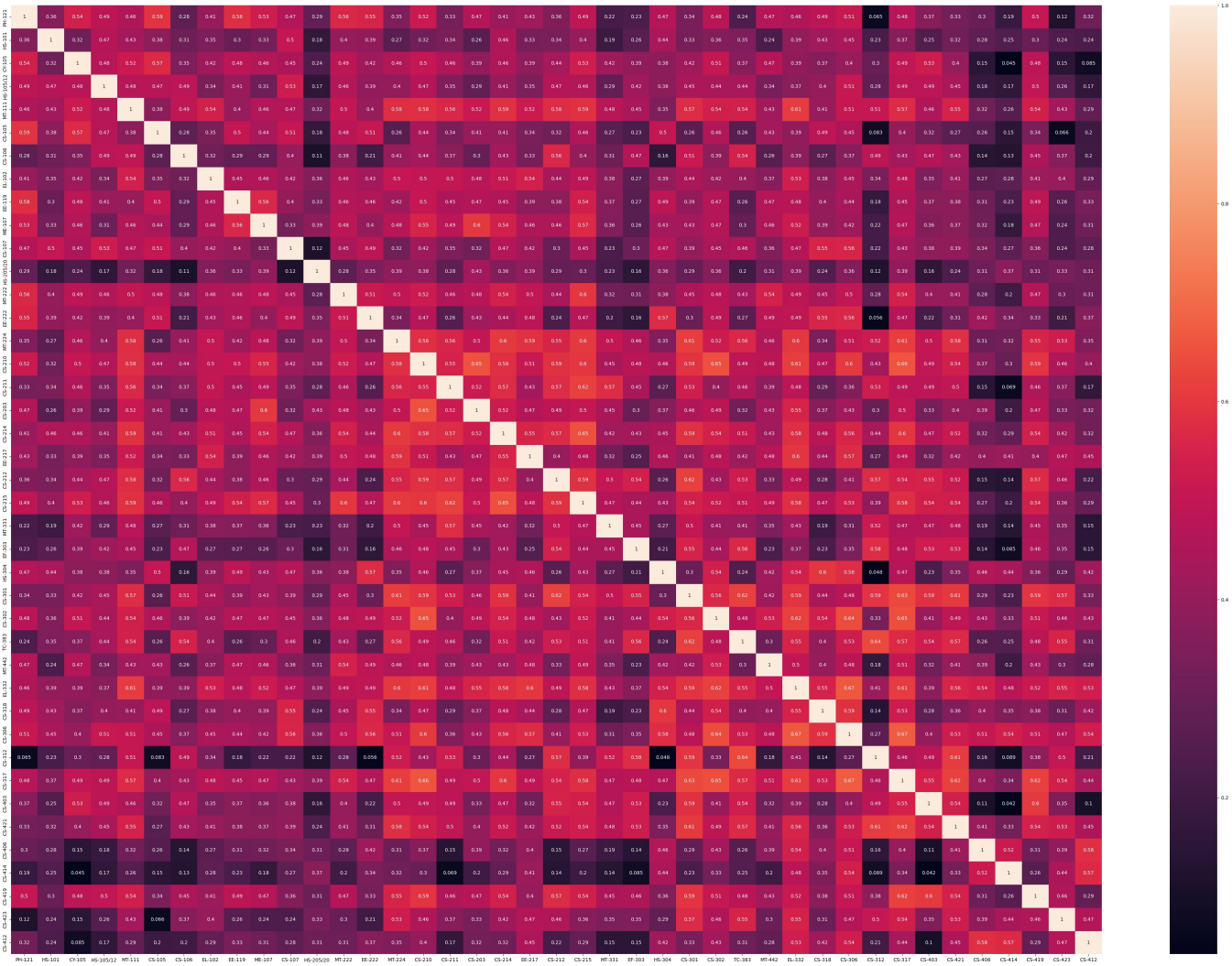
HS-101	0.364662	1.000000	0.318734	0.466245	0.429801	0.383307	0.307744	0.345745	0.303358	0.327945	...	0.449423	0.228937	0.367613
CY-105	0.542480	0.318734	1.000000	0.478800	0.519495	0.568184	0.349782	0.421849	0.484034	0.462803	...	0.403120	0.298765	0.494913
HS-105/12	0.489461	0.466245	0.478800	1.000000	0.475902	0.474835	0.485935	0.341497	0.409673	0.312481	...	0.510591	0.283881	0.489800
MT-111	0.457760	0.429801	0.519495	0.475902	1.000000	0.381212	0.491839	0.543150	0.403776	0.462213	...	0.508714	0.509412	0.572161
CS-105	0.590384	0.383307	0.568184	0.474835	0.381212	1.000000	0.276018	0.345009	0.498891	0.440830	...	0.451801	0.083275	0.397862
CS-106	0.278479	0.307744	0.349782	0.485935	0.491839	0.276018	1.000000	0.316386	0.286380	0.286875	...	0.366770	0.489331	0.427427
EL-102	0.408450	0.345745	0.421849	0.341497	0.543150	0.345009	0.316386	1.000000	0.454893	0.460814	...	0.446901	0.338204	0.480480
EE-119	0.580460	0.303358	0.484034	0.409673	0.403776	0.498891	0.286380	0.454893	1.000000	0.560468	...	0.442248	0.176641	0.454772
ME-107	0.526575	0.327945	0.462803	0.312481	0.462213	0.440830	0.286875	0.460814	0.560468	1.000000	...	0.422442	0.216019	0.472185
CS-107	0.471586	0.502361	0.451199	0.526392	0.468792	0.507680	0.404435	0.419050	0.399198	0.333642	...	0.555985	0.217646	0.426655
HS-205/20	0.289300	0.175735	0.238308	0.165138	0.316970	0.176427	0.112933	0.362221	0.333972	0.387771	...	0.360425	0.124640	0.392167
MT-222	0.562491	0.397886	0.488865	0.461679	0.499918	0.481531	0.375458	0.457086	0.461305	0.484912	...	0.495236	0.278218	0.541047
EE-222	0.551562	0.389994	0.423165	0.390964	0.401936	0.509679	0.212304	0.430263	0.455495	0.400762	...	0.558711	0.056283	0.466978
MT-224	0.352519	0.267045	0.462872	0.400780	0.578365	0.262957	0.409346	0.504926	0.423207	0.479870	...	0.508374	0.521198	0.614377
CS-210	0.518775	0.320298	0.503007	0.465133	0.582871	0.444559	0.438235	0.495737	0.502674	0.552505	...	0.604507	0.427556	0.657846
CS-211	0.330758	0.341461	0.457750	0.351412	0.559808	0.341297	0.372034	0.500261	0.453432	0.486825	...	0.362606	0.525848	0.488971
CS-203	0.467551	0.263625	0.392086	0.286507	0.515142	0.409914	0.295224	0.476247	0.471217	0.603162	...	0.430296	0.301178	0.496516
CS-214	0.410076	0.460450	0.456118	0.405390	0.586454	0.406419	0.430314	0.512849	0.453770	0.540600	...	0.557201	0.439340	0.596091
EE-217	0.427052	0.332679	0.388839	0.347970	0.520679	0.338811	0.332001	0.540070	0.385694	0.456728	...	0.572402	0.267820	0.491404
CS-212	0.355068	0.341184	0.436052	0.465238	0.575886	0.321446	0.558814	0.438183	0.384684	0.460895	...	0.405113	0.574870	0.543512
CS-215	0.493052	0.403469	0.532072	0.462283	0.586355	0.460635	0.396805	0.486480	0.538415	0.569099	...	0.530779	0.391917	0.580112
MT-331	0.224737	0.192493	0.419632	0.285221	0.479254	0.269991	0.310384	0.379440	0.370990	0.356879	...	0.309340	0.520653	0.465551
EF-303	0.233253	0.260642	0.390858	0.417683	0.446447	0.231212	0.470417	0.273927	0.265599	0.260726	...	0.345682	0.575856	0.482689
HS-304	0.465856	0.441408	0.378287	0.376659	0.346363	0.500632	0.160621	0.394740	0.489083	0.429105	...	0.576388	0.047723	0.471284
CS-301	0.335601	0.331199	0.418990	0.445205	0.574229	0.260676	0.514491	0.443139	0.391128	0.431564	...	0.482423	0.594583	0.634075
CS-302	0.482036	0.360221	0.509041	0.444393	0.537211	0.458363	0.385291	0.415193	0.468404	0.466906	...	0.643695	0.332825	0.647507
TC-383	0.237547	0.349655	0.369312	0.441173	0.542332	0.260645	0.544906	0.403689	0.260650	0.295905	...	0.526678	0.636428	0.567986
MT-442	0.469691	0.235440	0.467895	0.337331	0.425729	0.431416	0.261794	0.370634	0.466225	0.456081	...	0.480188	0.176407	0.506541
EL-332	0.455202	0.394341	0.391191	0.374887	0.607271	0.391187	0.385408	0.530719	0.479712	0.522369	...	0.665169	0.412499	0.607790
CS-318	0.487258	0.425836	0.367128	0.400547	0.408317	0.486345	0.268056	0.379361	0.399870	0.394601	...	0.585448	0.137165	0.526785
CS-306	0.510454	0.449423	0.403120	0.510591	0.508714	0.451801	0.366770	0.446901	0.442248	0.422442	...	1.000000	0.271005	0.673274
CS-312	0.065410	0.228937	0.298765	0.283881	0.509412	0.083275	0.489331	0.338204	0.176641	0.216019	...	0.271005	1.000000	0.462030
CS-317	0.476340	0.367613	0.494913	0.489800	0.572161	0.397862	0.427427	0.480480	0.454772	0.472185	...	0.673274	0.462030	1.000000
CS-403	0.368667	0.250034	0.525173	0.486041	0.455681	0.319673	0.470442	0.348618	0.372093	0.362915	...	0.398968	0.487118	0.546391
CS-421	0.327924	0.321463	0.403245	0.447928	0.548722	0.269655	0.429159	0.406521	0.380161	0.372211	...	0.527492	0.608301	0.618447
CS-														

406	0.301207	0.275355	0.151716	0.181047	0.318325	0.256310	0.141503	0.272197	0.309460	0.315541	...	0.509211	0.158918	0.401342
CS-414	0.187699	0.245779	0.044900	0.167917	0.256619	0.153554	0.132402	0.277301	0.232167	0.176587	...	0.536422	0.089371	0.339997
CS-419	0.503203	0.301938	0.484092	0.496238	0.536780	0.339665	0.452287	0.414231	0.488164	0.467242	...	0.505005	0.382856	0.617739
CS-423	0.119664	0.242951	0.153844	0.256895	0.429186	0.066309	0.365533	0.401338	0.259521	0.239895	...	0.467850	0.504500	0.537178
CS-412	0.316964	0.236217	0.085047	0.166924	0.287769	0.198109	0.200543	0.286203	0.333823	0.309134	...	0.535402	0.212184	0.435420

41 rows × 41 columns

In [68]:
```
plt.figure(figsize=(50,35))
sns.heatmap(df.drop(['CGPA'],axis=1).corr(),annot=True)
```

Out[68]:
```
<AxesSubplot:>
```



Now that we have noticed it, we can see that the bulk of the columns are positively correlated with one another. This is because the target variable is our CGPA, and CGPA is based on the subjects' marks.

Looking at the statistics

In [69]:
```
df.describe()
```

Out[69]:

	PH-121	HS-101	CY-105	HS-105/12	MT-111	CS-105	CS-106	EL-102	EE-119	ME-107	...	CS-3
count	561.000000	561.000000	561.000000	561.000000	561.000000	561.000000	561.000000	561.000000	561.000000	561.000000	...	561.0000
mean	3.691622	5.028520	2.795009	4.176471	3.841355	2.766488	4.069519	3.884135	3.855615	4.723708	...	4.1087
std	2.983182	2.777314	2.840846	3.148562	2.919418	2.639809	2.674045	2.960293	2.622466	3.102469	...	3.3568
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0000
25%	2.000000	3.000000	0.000000	2.000000	2.000000	0.000000	2.000000	2.000000	2.000000	2.000000	...	1.0000
50%	3.000000	5.000000	2.000000	4.000000	3.000000	2.000000	4.000000	3.000000	3.000000	4.000000	...	4.0000
75%	6.000000	7.000000	4.000000	7.000000	6.000000	4.000000	5.000000	6.000000	6.000000	7.000000	...	7.0000
max	10.000000	10.000000	10.000000	10.000000	11.000000	10.000000	10.000000	10.000000	10.000000	10.000000	...	13.0000

8 rows × 42 columns

Only the CGPA is observed because the remainder cannot be converted.

Looking for anomalies in the features.

Since our feature data is an object type, there is no need to check for outliers in this case.

Checking the features for skewness.

Since our feature data is an object type, there is no need to reduce skewness in this situation.

Next, verifying the independent variables for multicollinearity.

Since our data is an object datatype, there is no need to test for multicollinearity.

In [70]:
```
x=df.drop(['CGPA'],axis=1)
y=df['CGPA']
```

Our target variable and feature variable are now distinct.

Scaling the input data

In [71]:
```
#We employ a conventional scaler for scaling.
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=sc.fit_transform(x)
x
```

Out[71]:
```
array([[ 0.43897606,  1.79162927,  1.83383298, ...,  1.72077861,
        -0.35709981, -0.45499486],
       [-1.23858231,  1.43124776,  2.53847703, ...,  0.88352206,
         0.71229339, -0.02886445],
       [-1.23858231, -0.73104127, -0.98474319, ..., -1.62824758,
        -0.71356421, -1.30725569],
       ...,
       [-0.23204729, -1.81218579, -0.28009915, ...,  0.04626551,
        -0.71356421, -0.45499486],
       [-1.23858231, -0.37065977,  2.18615501, ..., -0.37236276,
         0.35582899,  1.24952679],
       [ 0.77448773,  1.43124776,  2.18615501, ...,  0.04626551,
         1.78168659,  2.10178761]])
```

Our target variable is a float, hence we should employ a regression model because of this observation.

Since the numbers in our features columns are large, we first train PCA using all of the columns, then we compare the results to decide which method is superior.

In [72]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from sklearn.model_selection import train_test_split
```

In [73]:
```
for i in range(0,1000):
    x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=i,test_size=0.25)
    lr.fit(x_train,y_train)
    pred_train=lr.predict(x_train)
    pred_test=lr.predict(x_test)
```

```
    if round(r2_score(y_train,pred_train)*100,1)==round(r2_score(y_test,pred_test)*100,1):
        print('At random state',i,'The model perform very well')
        print('Random State = ',i)
        print("Training r2_score is = ",r2_score(y_train,pred_train))
        print("Test r2_score is = ",r2_score(y_test,pred_test))
        print('\n')
```

```
At random state 4 The model perform very well
Random State =  4
Training r2_score is =  0.9880886688047826
Test r2_score is =  0.9875486350534771


At random state 7 The model perform very well
Random State =  7
Training r2_score is =  0.9880267075717799
Test r2_score is =  0.9884458695944688


At random state 8 The model perform very well
Random State =  8
Training r2_score is =  0.9882596748775893
Test r2_score is =  0.9876482227675912


At random state 22 The model perform very well
Random State =  22
Training r2_score is =  0.9878559950221795
Test r2_score is =  0.9884270261054219


At random state 44 The model perform very well
Random State =  44
Training r2_score is =  0.9879994281294348
Test r2_score is =  0.9884085612942367


At random state 57 The model perform very well
Random State =  57
Training r2_score is =  0.9880572552351253
Test r2_score is =  0.9880889949392024


At random state 61 The model perform very well
Random State =  61
Training r2_score is =  0.9875963295966113
Test r2_score is =  0.9876296172162224


At random state 62 The model perform very well
Random State =  62
Training r2_score is =  0.9882272204911627
Test r2_score is =  0.987776115956785


At random state 63 The model perform very well
Random State =  63
Training r2_score is =  0.988083605556211
Test r2_score is =  0.988416755934873


At random state 84 The model perform very well
Random State =  84
Training r2_score is =  0.9881971351271192
Test r2_score is =  0.9881871231271262


At random state 88 The model perform very well
Random State =  88
Training r2_score is =  0.9881061820213096
Test r2_score is =  0.987924185657569


At random state 116 The model perform very well
Random State =  116
Training r2_score is =  0.9883385484108236
Test r2_score is =  0.9875646395519527


At random state 139 The model perform very well
Random State =  139
Training r2_score is =  0.9882528586556556
Test r2_score is =  0.9880882987187225


At random state 152 The model perform very well
Random State =  152
Training r2_score is =  0.9881124940674608
Test r2_score is =  0.9877316113804933
```

```
At random state 160 The model perform very well
Random State =  160
Training r2_score is =  0.9882669204496463
Test r2_score is =  0.9875062372946514


At random state 162 The model perform very well
Random State =  162
Training r2_score is =  0.9879734744375678
Test r2_score is =  0.9881385783484214


At random state 163 The model perform very well
Random State =  163
Training r2_score is =  0.9879848123670563
Test r2_score is =  0.9879984730178111


At random state 167 The model perform very well
Random State =  167
Training r2_score is =  0.9879928039978703
Test r2_score is =  0.9880019086171875


At random state 168 The model perform very well
Random State =  168
Training r2_score is =  0.9880461512464912
Test r2_score is =  0.9884738798203754


At random state 172 The model perform very well
Random State =  172
Training r2_score is =  0.9880295451849749
Test r2_score is =  0.98839516690752


At random state 174 The model perform very well
Random State =  174
Training r2_score is =  0.9880638645141621
Test r2_score is =  0.9884487338965234


At random state 185 The model perform very well
Random State =  185
Training r2_score is =  0.988142308405874
Test r2_score is =  0.9881491967304736


At random state 187 The model perform very well
Random State =  187
Training r2_score is =  0.9880254864144116
Test r2_score is =  0.9883177098316911


At random state 189 The model perform very well
Random State =  189
Training r2_score is =  0.9880216425658249
Test r2_score is =  0.9876041509130504


At random state 196 The model perform very well
Random State =  196
Training r2_score is =  0.987911780331297
Test r2_score is =  0.9881988222967348


At random state 197 The model perform very well
Random State =  197
Training r2_score is =  0.9883922836530843
Test r2_score is =  0.9876463893721315


At random state 223 The model perform very well
Random State =  223
Training r2_score is =  0.9881586084859283
Test r2_score is =  0.9876643084913487


At random state 227 The model perform very well
Random State =  227
Training r2_score is =  0.9878985390900992
Test r2_score is =  0.9884541127873262


At random state 233 The model perform very well
Random State =  233
Training r2_score is =  0.9879635039595438
```

```
Test r2_score is =  0.9879342167331756


At random state 234 The model perform very well
Random State =  234
Training r2_score is =  0.9881585096466217
Test r2_score is =  0.9877775862971828


At random state 268 The model perform very well
Random State =  268
Training r2_score is =  0.9880725988363767
Test r2_score is =  0.9883214734978968


At random state 274 The model perform very well
Random State =  274
Training r2_score is =  0.9881409096501046
Test r2_score is =  0.9878425186845585


At random state 276 The model perform very well
Random State =  276
Training r2_score is =  0.9882429679752373
Test r2_score is =  0.9877854909575261


At random state 277 The model perform very well
Random State =  277
Training r2_score is =  0.9880768836492468
Test r2_score is =  0.9883686441256028


At random state 280 The model perform very well
Random State =  280
Training r2_score is =  0.9877346740376194
Test r2_score is =  0.9883902595488442


At random state 294 The model perform very well
Random State =  294
Training r2_score is =  0.9881919924058475
Test r2_score is =  0.9875481474550583


At random state 295 The model perform very well
Random State =  295
Training r2_score is =  0.9880920845859734
Test r2_score is =  0.9880839033619746


At random state 299 The model perform very well
Random State =  299
Training r2_score is =  0.9881166862615863
Test r2_score is =  0.9882909849919025


At random state 305 The model perform very well
Random State =  305
Training r2_score is =  0.9880082056482313
Test r2_score is =  0.9879762689336613


At random state 306 The model perform very well
Random State =  306
Training r2_score is =  0.9878923519523868
Test r2_score is =  0.9882626411640262


At random state 322 The model perform very well
Random State =  322
Training r2_score is =  0.988137754391364
Test r2_score is =  0.9883415771739279


At random state 336 The model perform very well
Random State =  336
Training r2_score is =  0.9880051492092163
Test r2_score is =  0.9883454067305655


At random state 348 The model perform very well
Random State =  348
Training r2_score is =  0.9881205973801983
Test r2_score is =  0.9884080311608954


At random state 363 The model perform very well
Random State =  363
```

```
Training r2_score is =  0.9878346442193239
Test r2_score is =  0.9882589320876362


At random state 365 The model perform very well
Random State =  365
Training r2_score is =  0.9884375577211797
Test r2_score is =  0.9875175911723856


At random state 371 The model perform very well
Random State =  371
Training r2_score is =  0.9879308553805175
Test r2_score is =  0.9882094282729179


At random state 380 The model perform very well
Random State =  380
Training r2_score is =  0.9883017462753316
Test r2_score is =  0.9876095731715901


At random state 385 The model perform very well
Random State =  385
Training r2_score is =  0.9880637257061817
Test r2_score is =  0.9875048163954668


At random state 390 The model perform very well
Random State =  390
Training r2_score is =  0.9881047730489813
Test r2_score is =  0.9881701500775302


At random state 391 The model perform very well
Random State =  391
Training r2_score is =  0.9882260955820588
Test r2_score is =  0.9877948093340074


At random state 395 The model perform very well
Random State =  395
Training r2_score is =  0.9882444097457415
Test r2_score is =  0.9877666953847243


At random state 396 The model perform very well
Random State =  396
Training r2_score is =  0.988037675839731
Test r2_score is =  0.9876648350507857


At random state 409 The model perform very well
Random State =  409
Training r2_score is =  0.9881045170335225
Test r2_score is =  0.9877258563068201


At random state 424 The model perform very well
Random State =  424
Training r2_score is =  0.9881743094009853
Test r2_score is =  0.9877894485464804


At random state 427 The model perform very well
Random State =  427
Training r2_score is =  0.9880351156585955
Test r2_score is =  0.9880787233306292


At random state 444 The model perform very well
Random State =  444
Training r2_score is =  0.9881265116642771
Test r2_score is =  0.9878649260965994


At random state 450 The model perform very well
Random State =  450
Training r2_score is =  0.9880890214916914
Test r2_score is =  0.9883019936072317


At random state 451 The model perform very well
Random State =  451
Training r2_score is =  0.988124914820715
Test r2_score is =  0.9875107372246089


At random state 467 The model perform very well
```

```
Random State =  467
Training r2_score is =  0.9881420097548065
Test r2_score is =  0.9880021770053657


At random state 482 The model perform very well
Random State =  482
Training r2_score is =  0.9880295449204554
Test r2_score is =  0.9875167012812617


At random state 498 The model perform very well
Random State =  498
Training r2_score is =  0.9881965272930049
Test r2_score is =  0.9877281377587683


At random state 499 The model perform very well
Random State =  499
Training r2_score is =  0.9882147772967605
Test r2_score is =  0.9879627690846441


At random state 520 The model perform very well
Random State =  520
Training r2_score is =  0.9879879042286253
Test r2_score is =  0.9881590950223372


At random state 529 The model perform very well
Random State =  529
Training r2_score is =  0.9881006189725269
Test r2_score is =  0.9880145637374969


At random state 534 The model perform very well
Random State =  534
Training r2_score is =  0.9882590476261192
Test r2_score is =  0.987728496490425


At random state 559 The model perform very well
Random State =  559
Training r2_score is =  0.9881088389126511
Test r2_score is =  0.9882199771629595


At random state 565 The model perform very well
Random State =  565
Training r2_score is =  0.9879084620095526
Test r2_score is =  0.9880598380477815


At random state 568 The model perform very well
Random State =  568
Training r2_score is =  0.9882027386861105
Test r2_score is =  0.9877436062571336


At random state 572 The model perform very well
Random State =  572
Training r2_score is =  0.9883286916566111
Test r2_score is =  0.9876356105903299


At random state 578 The model perform very well
Random State =  578
Training r2_score is =  0.9879313038778741
Test r2_score is =  0.9884634445469755


At random state 580 The model perform very well
Random State =  580
Training r2_score is =  0.988235512818232
Test r2_score is =  0.9876941390999711


At random state 588 The model perform very well
Random State =  588
Training r2_score is =  0.9882547349164325
Test r2_score is =  0.9882573074319585


At random state 589 The model perform very well
Random State =  589
Training r2_score is =  0.9880835701191241
Test r2_score is =  0.9883615165293687
```

```
At random state 592 The model perform very well
Random State =  592
Training r2_score is =  0.9879838788680885
Test r2_score is =  0.9881380363522059


At random state 595 The model perform very well
Random State =  595
Training r2_score is =  0.9882850753738787
Test r2_score is =  0.9877821600897594


At random state 598 The model perform very well
Random State =  598
Training r2_score is =  0.9877666808453225
Test r2_score is =  0.9883771072920188


At random state 607 The model perform very well
Random State =  607
Training r2_score is =  0.9880120642622202
Test r2_score is =  0.9881182463496542


At random state 616 The model perform very well
Random State =  616
Training r2_score is =  0.9880413467853557
Test r2_score is =  0.9881997599261398


At random state 646 The model perform very well
Random State =  646
Training r2_score is =  0.9880240347149547
Test r2_score is =  0.9882714324983343


At random state 656 The model perform very well
Random State =  656
Training r2_score is =  0.9882033370013213
Test r2_score is =  0.9878482346620268


At random state 662 The model perform very well
Random State =  662
Training r2_score is =  0.9880446782276174
Test r2_score is =  0.9875208471553039


At random state 670 The model perform very well
Random State =  670
Training r2_score is =  0.9882917020295822
Test r2_score is =  0.9875391880700787


At random state 679 The model perform very well
Random State =  679
Training r2_score is =  0.9877227023109957
Test r2_score is =  0.9884156690412025


At random state 687 The model perform very well
Random State =  687
Training r2_score is =  0.9882775636390121
Test r2_score is =  0.9880267252849895


At random state 690 The model perform very well
Random State =  690
Training r2_score is =  0.988106175542924
Test r2_score is =  0.9879428050989583


At random state 702 The model perform very well
Random State =  702
Training r2_score is =  0.988408283881161
Test r2_score is =  0.9875157580097584


At random state 717 The model perform very well
Random State =  717
Training r2_score is =  0.9881578211546077
Test r2_score is =  0.988195976916042


At random state 725 The model perform very well
Random State =  725
Training r2_score is =  0.9883210343794425
Test r2_score is =  0.9879246444928081
```

```
At random state 727 The model perform very well
Random State =  727
Training r2_score is =  0.9881248290058732
Test r2_score is =  0.9875579658329909


At random state 732 The model perform very well
Random State =  732
Training r2_score is =  0.9880305531483753
Test r2_score is =  0.9879334888298476


At random state 757 The model perform very well
Random State =  757
Training r2_score is =  0.9877959157967504
Test r2_score is =  0.9879018707001017


At random state 765 The model perform very well
Random State =  765
Training r2_score is =  0.9883068311893054
Test r2_score is =  0.9876299625560417


At random state 766 The model perform very well
Random State =  766
Training r2_score is =  0.9880078818682997
Test r2_score is =  0.9877616332334981


At random state 767 The model perform very well
Random State =  767
Training r2_score is =  0.987939388083962
Test r2_score is =  0.9883054780623788


At random state 786 The model perform very well
Random State =  786
Training r2_score is =  0.9882278153984311
Test r2_score is =  0.987933725516552


At random state 793 The model perform very well
Random State =  793
Training r2_score is =  0.9877944051129218
Test r2_score is =  0.9879469446336216


At random state 794 The model perform very well
Random State =  794
Training r2_score is =  0.988256731962379
Test r2_score is =  0.9876392145815899


At random state 798 The model perform very well
Random State =  798
Training r2_score is =  0.9880145322164453
Test r2_score is =  0.9881794264126452


At random state 799 The model perform very well
Random State =  799
Training r2_score is =  0.9883795947239843
Test r2_score is =  0.9875454359904231


At random state 812 The model perform very well
Random State =  812
Training r2_score is =  0.9879269479341973
Test r2_score is =  0.9881282985672181


At random state 836 The model perform very well
Random State =  836
Training r2_score is =  0.9881192284332181
Test r2_score is =  0.988188750567609


At random state 839 The model perform very well
Random State =  839
Training r2_score is =  0.9878539327257138
Test r2_score is =  0.9884374705465259


At random state 841 The model perform very well
Random State =  841
Training r2_score is =  0.9883554712356969
Test r2_score is =  0.9876091517808185
```

```
At random state 856 The model perform very well
Random State =  856
Training r2_score is =  0.9881512139263186
Test r2_score is =  0.9875632495171422


At random state 862 The model perform very well
Random State =  862
Training r2_score is =  0.9880985693356811
Test r2_score is =  0.988021308631508


At random state 870 The model perform very well
Random State =  870
Training r2_score is =  0.9879206878529406
Test r2_score is =  0.9877126926836223


At random state 876 The model perform very well
Random State =  876
Training r2_score is =  0.9880233248533387
Test r2_score is =  0.9878533170147501


At random state 880 The model perform very well
Random State =  880
Training r2_score is =  0.9882378848153679
Test r2_score is =  0.9877641868490975


At random state 883 The model perform very well
Random State =  883
Training r2_score is =  0.9878955887490851
Test r2_score is =  0.9879733272760934


At random state 884 The model perform very well
Random State =  884
Training r2_score is =  0.9878782391943173
Test r2_score is =  0.9883288756926584


At random state 891 The model perform very well
Random State =  891
Training r2_score is =  0.9880106604724375
Test r2_score is =  0.988259857495587


At random state 903 The model perform very well
Random State =  903
Training r2_score is =  0.9881418114654222
Test r2_score is =  0.988393082185126


At random state 909 The model perform very well
Random State =  909
Training r2_score is =  0.9880430745620662
Test r2_score is =  0.9880731645802563


At random state 915 The model perform very well
Random State =  915
Training r2_score is =  0.9881364698522731
Test r2_score is =  0.9876829874750434


At random state 925 The model perform very well
Random State =  925
Training r2_score is =  0.9880679857401647
Test r2_score is =  0.9880747311599867


At random state 931 The model perform very well
Random State =  931
Training r2_score is =  0.9880161638864593
Test r2_score is =  0.9875028131907402


At random state 935 The model perform very well
Random State =  935
Training r2_score is =  0.9882065237442235
Test r2_score is =  0.9878913662295042


At random state 959 The model perform very well
Random State =  959
Training r2_score is =  0.9882585673283092
```

```
Test r2_score is =  0.9876916708223591


At random state 979 The model perform very well
Random State =  979
Training r2_score is =  0.9882923944877128
Test r2_score is =  0.98761761022409


At random state 983 The model perform very well
Random State =  983
Training r2_score is =  0.9882044234422563
Test r2_score is =  0.9875824927982426


At random state 999 The model perform very well
Random State =  999
Training r2_score is =  0.98776048003398
Test r2_score is =  0.9884691963413345
```

As we can see from the random state, 348 has the least variation, therefore we use it.

In [74]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=348,test_size=0.25)
```

In [75]:
```
lr.fit(x_train,y_train)
pred_test_lr=lr.predict(x_test)
print("r2_score is = ",r2_score(y_test,pred_test_lr)*100)
print('mean_absolute_error = ',mean_absolute_error(y_test,pred_test_lr))
print('mean_squared_error = ',mean_squared_error(y_test,pred_test_lr))
print('root_mean_squared_error = ',np.sqrt(mean_squared_error(y_test,pred_test_lr)))
```

```
r2_score is =  98.84080311608953
mean_absolute_error =  0.04552735127218016
mean_squared_error =  0.0036605825633282117
root_mean_squared_error =  0.060502748394830885
```

The r2 score for linear regression is 98.84%, and there are extremely few errors, such as mean_absolute_error = 0.046, mean_squared_error = 0.0037, and root_mean_squared_error = 0.061.

Checking the Cross Val score now.

In [76]:
```
from sklearn.model_selection import cross_val_score
pred_lr_ac=r2_score(y_test,pred_test_lr)
for i in range(2,15):
    lsscore=cross_val_score(lr,x,y,cv=i)
    lsc=lsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',lsc*100)
    print('Accuracy score is =',pred_lr_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 96.4004199482534
Accuracy score is = 98.84080311608953


At cv = 3
Cross validation score is = 98.1720313277595
Accuracy score is = 98.84080311608953


At cv = 4
Cross validation score is = 98.14876935050859
Accuracy score is = 98.84080311608953


At cv = 5
Cross validation score is = 98.45947286446608
Accuracy score is = 98.84080311608953


At cv = 6
Cross validation score is = 98.46575505256924
Accuracy score is = 98.84080311608953


At cv = 7
Cross validation score is = 98.45799404138565
Accuracy score is = 98.84080311608953


At cv = 8
Cross validation score is = 98.47801905664768
Accuracy score is = 98.84080311608953


At cv = 9
Cross validation score is = 98.51173543346303
Accuracy score is = 98.84080311608953


At cv = 10
Cross validation score is = 98.5451534902641
Accuracy score is = 98.84080311608953


At cv = 11
Cross validation score is = 98.56602072518996
Accuracy score is = 98.84080311608953


At cv = 12
Cross validation score is = 98.49999486751314
Accuracy score is = 98.84080311608953


At cv = 13
Cross validation score is = 98.51901688637538
Accuracy score is = 98.84080311608953


At cv = 14
Cross validation score is = 98.54912659193799
Accuracy score is = 98.84080311608953
```

We chose cv=11 because, as we can see, it produces the least difference between the Cross Validation score and the r2_score.

In [77]:
```
lsscore_selected=cross_val_score(lr,x,y,cv=11)
print('Cross validation score =',lsscore_selected.mean()*100,'\n','r2_score =',pred_lr_ac*100)
```

```
Cross validation score = 98.56602072518996
 r2_score = 98.84080311608953
```

In [78]:
```
from sklearn.ensemble import GradientBoostingRegressor,AdaBoostRegressor,RandomForestRegressor,ExtraTreesRegres
from sklearn.linear_model import Lasso,Ridge
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
```

In [79]:
```
gb=GradientBoostingRegressor()
ada=AdaBoostRegressor()
rfc=RandomForestRegressor()
svc=SVR()
dtc=DecisionTreeRegressor()
knn=KNeighborsRegressor()
```

In [80]:
```
# GradientBoostingRegression
```

```
gb.fit(x_train,y_train)
pred_test_gb=gb.predict(x_test)
print("r2_score is = ",r2_score(y_test,pred_test_gb)*100)
print('mean_absolute_error = ',mean_absolute_error(y_test,pred_test_gb))
print('mean_squared_error = ',mean_squared_error(y_test,pred_test_gb))
print('root_mean_squared_error = ',np.sqrt(mean_squared_error(y_test,pred_test_gb)))
```

```
r2_score is =  96.0687054157111
mean_absolute_error =  0.08511982997274144
mean_squared_error =  0.012414481617659552
root_mean_squared_error =  0.11142029266547253
```

In [81]:
```
pred_gb_ac=r2_score(y_test,pred_test_gb)
for i in range(2,15):
    lsscore=cross_val_score(gb,x,y,cv=i)
    lsc=lsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',lsc*100)
    print('Accuracy score is =',pred_gb_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 92.83586253102301
Accuracy score is = 96.0687054157111


At cv = 3
Cross validation score is = 95.99222434940773
Accuracy score is = 96.0687054157111


At cv = 4
Cross validation score is = 94.94538893328853
Accuracy score is = 96.0687054157111


At cv = 5
Cross validation score is = 96.10982603069782
Accuracy score is = 96.0687054157111


At cv = 6
Cross validation score is = 96.39619128742403
Accuracy score is = 96.0687054157111


At cv = 7
Cross validation score is = 96.06349097648679
Accuracy score is = 96.0687054157111


At cv = 8
Cross validation score is = 96.06485679285021
Accuracy score is = 96.0687054157111


At cv = 9
Cross validation score is = 96.53363182203127
Accuracy score is = 96.0687054157111


At cv = 10
Cross validation score is = 96.46632350309119
Accuracy score is = 96.0687054157111


At cv = 11
Cross validation score is = 96.6156676304685
Accuracy score is = 96.0687054157111


At cv = 12
Cross validation score is = 96.59792758885973
Accuracy score is = 96.0687054157111


At cv = 13
Cross validation score is = 96.50551375818911
Accuracy score is = 96.0687054157111


At cv = 14
Cross validation score is = 96.62356870604617
Accuracy score is = 96.0687054157111
```

In [82]:
```
lsscore_selected=cross_val_score(gb,x,y,cv=8)
print('Cross validation score =',lsscore_selected.mean()*100,'\n','r2_score =',pred_gb_ac*100)
```

```
Cross validation score = 96.04752724140636
 r2_score = 96.0687054157111
```

In [83]:
```
# AdaBoostRegression
ada.fit(x_train,y_train)
pred_test_ada=ada.predict(x_test)
print("r2_score is = ",r2_score(y_test,pred_test_ada)*100)
print('mean_absolute_error = ',mean_absolute_error(y_test,pred_test_ada))
print('mean_squared_error = ',mean_squared_error(y_test,pred_test_ada))
print('root_mean_squared_error = ',np.sqrt(mean_squared_error(y_test,pred_test_ada)))
```

```
r2_score is =  92.89560369963623
mean_absolute_error =  0.12289381220379654
mean_squared_error =  0.02243469559058433
root_mean_squared_error =  0.14978216045505663
```

In [84]:
```
pred_ada_ac=r2_score(y_test,pred_test_ada)
for i in range(2,15):
    lsscore=cross_val_score(ada,x,y,cv=i)
    lsc=lsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',lsc*100)
    print('Accuracy score is =',pred_ada_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 91.42866850611813
Accuracy score is = 92.89560369963623


At cv = 3
Cross validation score is = 90.41711104615845
Accuracy score is = 92.89560369963623


At cv = 4
Cross validation score is = 89.8809288490288
Accuracy score is = 92.89560369963623


At cv = 5
Cross validation score is = 91.99756485653327
Accuracy score is = 92.89560369963623


At cv = 6
Cross validation score is = 91.80059422096284
Accuracy score is = 92.89560369963623


At cv = 7
Cross validation score is = 91.51507275185492
Accuracy score is = 92.89560369963623


At cv = 8
Cross validation score is = 91.43606544952031
Accuracy score is = 92.89560369963623


At cv = 9
Cross validation score is = 91.71983946995483
Accuracy score is = 92.89560369963623


At cv = 10
Cross validation score is = 91.87133039103689
Accuracy score is = 92.89560369963623


At cv = 11
Cross validation score is = 91.97639641453371
Accuracy score is = 92.89560369963623


At cv = 12
Cross validation score is = 91.7633187515889
Accuracy score is = 92.89560369963623


At cv = 13
Cross validation score is = 91.97059473539386
Accuracy score is = 92.89560369963623


At cv = 14
Cross validation score is = 92.39843435305683
Accuracy score is = 92.89560369963623
```

We chose cv=14 because, as we can see, it produces the least difference between the Cross Validation score and the r2_score.

In [85]:
```
lsscore_selected=cross_val_score(ada,x,y,cv=14)
print('Cross validation score =',lsscore_selected.mean()*100,'\n','r2_score =',pred_ada_ac*100)
```

```
Cross validation score = 92.09727152663459
 r2_score = 92.89560369963623
```

In [86]:
```
# RandomForestRegression
rfc.fit(x_train,y_train)
pred_test_rfc=rfc.predict(x_test)
print("r2_score is = ",r2_score(y_test,pred_test_rfc)*100)
print('mean_absolute_error = ',mean_absolute_error(y_test,pred_test_rfc))
print('mean_squared_error = ',mean_squared_error(y_test,pred_test_rfc))
print('root_mean_squared_error = ',np.sqrt(mean_squared_error(y_test,pred_test_rfc)))
```

```
r2_score is =  95.46174317917429
mean_absolute_error =  0.09513035460992918
mean_squared_error =  0.014331183957446801
root_mean_squared_error =  0.1197129231012542
```

In [87]:
```
pred_rfc_ac=r2_score(y_test,pred_test_rfc)
for i in range(2,15):
```

```
    lsscore=cross_val_score(rfc,x,y,cv=i)
    lsc=lsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',lsc*100)
    print('Accuracy score is =',pred_rfc_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 92.22699591391594
Accuracy score is = 95.46174317917429


At cv = 3
Cross validation score is = 93.3476590246476
Accuracy score is = 95.46174317917429


At cv = 4
Cross validation score is = 93.07802465625296
Accuracy score is = 95.46174317917429


At cv = 5
Cross validation score is = 94.66704015782972
Accuracy score is = 95.46174317917429


At cv = 6
Cross validation score is = 94.61665809715969
Accuracy score is = 95.46174317917429


At cv = 7
Cross validation score is = 94.26373442607925
Accuracy score is = 95.46174317917429


At cv = 8
Cross validation score is = 94.15724970434138
Accuracy score is = 95.46174317917429


At cv = 9
Cross validation score is = 94.64095859428414
Accuracy score is = 95.46174317917429


At cv = 10
Cross validation score is = 94.5714786986337
Accuracy score is = 95.46174317917429


At cv = 11
Cross validation score is = 94.67823578667378
Accuracy score is = 95.46174317917429


At cv = 12
Cross validation score is = 94.7842542547409
Accuracy score is = 95.46174317917429


At cv = 13
Cross validation score is = 94.61865847944821
Accuracy score is = 95.46174317917429


At cv = 14
Cross validation score is = 95.06240759080414
Accuracy score is = 95.46174317917429
```

As we can see, when we use cv=14, the difference between the Cross Validation score and the r2_score is the least.

In [88]:
```
lsscore_selected=cross_val_score(rfc,x,y,cv=14)
print('Cross validation score =',lsscore_selected.mean()*100,'\n','r2_score =',pred_rfc_ac*100)
```

```
Cross validation score = 94.96880169707215
 r2_score = 95.46174317917429
```

In [89]:
```
# SVR
svc.fit(x_train,y_train)
pred_test_svc=svc.predict(x_test)
print("r2_score is = ",r2_score(y_test,pred_test_svc)*100)
print('mean_absolute_error = ',mean_absolute_error(y_test,pred_test_svc))
print('mean_squared_error = ',mean_squared_error(y_test,pred_test_svc))
print('root_mean_squared_error = ',np.sqrt(mean_squared_error(y_test,pred_test_svc)))
```

```
r2_score is =  95.72381215265717
mean_absolute_error =  0.07514614841483142
mean_squared_error =  0.013503606582079357
root_mean_squared_error =  0.11620501960792984
```

As we can see, the r2_score for SVR is 95.724%, and the total quantity of errors is very low, with mean_absolute_error = 0.075, mean_squared_error = 0.013, and root_mean_squared_error = 0.116.

In [90]:
```
pred_svc_ac=r2_score(y_test,pred_test_svc)
for i in range(2,15):
    lsscore=cross_val_score(svc,x,y,cv=i)
    lsc=lsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',lsc*100)
    print('Accuracy score is =',pred_svc_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 92.28466618825199
Accuracy score is = 95.72381215265717


At cv = 3
Cross validation score is = 94.40014676658612
Accuracy score is = 95.72381215265717


At cv = 4
Cross validation score is = 94.15049889152397
Accuracy score is = 95.72381215265717


At cv = 5
Cross validation score is = 95.6862131634741
Accuracy score is = 95.72381215265717


At cv = 6
Cross validation score is = 95.94748056455416
Accuracy score is = 95.72381215265717


At cv = 7
Cross validation score is = 96.01578705181868
Accuracy score is = 95.72381215265717


At cv = 8
Cross validation score is = 96.12589601415547
Accuracy score is = 95.72381215265717


At cv = 9
Cross validation score is = 96.26734641948337
Accuracy score is = 95.72381215265717


At cv = 10
Cross validation score is = 96.2544709993988
Accuracy score is = 95.72381215265717


At cv = 11
Cross validation score is = 96.45667156837943
Accuracy score is = 95.72381215265717


At cv = 12
Cross validation score is = 96.14833853376675
Accuracy score is = 95.72381215265717


At cv = 13
Cross validation score is = 96.27608419159812
Accuracy score is = 95.72381215265717


At cv = 14
Cross validation score is = 96.44623107431185
Accuracy score is = 95.72381215265717
```

We select cv=5 because, as we can see, it produces the least difference between the Cross Validation score and the r2_score.

In [91]:
```
lsscore_selected=cross_val_score(svc,x,y,cv=5)
print('Cross validation score =',lsscore_selected.mean()*100,'\n','r2_score =',pred_svc_ac*100)
```

```
Cross validation score = 95.6862131634741
 r2_score = 95.72381215265717
```

In [92]:
```
# DecisionTreeRegression
dtc.fit(x_train,y_train)
pred_test_dtc=dtc.predict(x_test)
print("r2_score is = ",r2_score(y_test,pred_test_dtc)*100)
print('mean_absolute_error = ',mean_absolute_error(y_test,pred_test_dtc))
print('mean_squared_error = ',mean_squared_error(y_test,pred_test_dtc))
print('root_mean_squared_error = ',np.sqrt(mean_squared_error(y_test,pred_test_dtc)))
```

```
r2_score is =  78.60196201847216
mean_absolute_error =  0.1808368794326241
mean_squared_error =  0.06757202836879433
root_mean_squared_error =  0.2599462028358836
```

According to our observations, the DecisionTreeRegressor yields an R2 score of 82.37% and exceptionally low levels of error, including mean_absolute_error = 0.17, mean_squared_error = 0.056, and root_mean_squared_error = 0.23.

In [93]:
```
pred_dtc_ac=r2_score(y_test,pred_test_dtc)
for i in range(2,15):
    lsscore=cross_val_score(dtc,x,y,cv=i)
    lsc=lsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',lsc*100)
    print('Accuracy score is =',pred_dtc_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 61.25102781927092
Accuracy score is = 78.60196201847216


At cv = 3
Cross validation score is = 77.83500457048838
Accuracy score is = 78.60196201847216


At cv = 4
Cross validation score is = 74.82199575716845
Accuracy score is = 78.60196201847216


At cv = 5
Cross validation score is = 81.65284675134224
Accuracy score is = 78.60196201847216


At cv = 6
Cross validation score is = 79.65901694961411
Accuracy score is = 78.60196201847216


At cv = 7
Cross validation score is = 81.62336695010943
Accuracy score is = 78.60196201847216


At cv = 8
Cross validation score is = 81.15461380943397
Accuracy score is = 78.60196201847216


At cv = 9
Cross validation score is = 82.08714185130636
Accuracy score is = 78.60196201847216


At cv = 10
Cross validation score is = 81.58109169611883
Accuracy score is = 78.60196201847216


At cv = 11
Cross validation score is = 83.87403476030215
Accuracy score is = 78.60196201847216


At cv = 12
Cross validation score is = 81.65288029743488
Accuracy score is = 78.60196201847216


At cv = 13
Cross validation score is = 82.59300717935673
Accuracy score is = 78.60196201847216


At cv = 14
Cross validation score is = 83.52940010543085
Accuracy score is = 78.60196201847216
```

We select cv=5 because, as we can see, it produces the least difference between the Cross Validation score and the r2_score.

In [94]:
```
lsscore_selected=cross_val_score(dtc,x,y,cv=5)
print('Cross validation score =',lsscore_selected.mean()*100,'\n','r2_score =',pred_dtc_ac*100)
```

```
Cross validation score = 82.29983476042507
 r2_score = 78.60196201847216
```

In [95]:
```
# KNeighborsRegression
knn.fit(x_train,y_train)
pred_test_knn=knn.predict(x_test)
print("r2_score is = ",r2_score(y_test,pred_test_knn)*100)
print('mean_absolute_error = ',mean_absolute_error(y_test,pred_test_knn))
print('mean_squared_error = ',mean_squared_error(y_test,pred_test_knn))
print('root_mean_squared_error = ',np.sqrt(mean_squared_error(y_test,pred_test_knn)))
```

```
r2_score is =  95.80668054699328
mean_absolute_error =  0.08756737588652481
mean_squared_error =  0.013241919716312054
root_mean_squared_error =  0.11507354047004921
```

According to what we can see, the KNeighborsRegressor has a r2 score of 95.80% and relatively little error, with mean_absolute_error =

0.087, mean_squared_error = 0.013, and root_mean_squared_error = 0.11.

In [96]:
```
pred_knn_ac=r2_score(y_test,pred_test_knn)
for i in range(2,15):
    lsscore=cross_val_score(knn,x,y,cv=i)
    lsc=lsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',lsc*100)
    print('Accuracy score is =',pred_knn_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 93.8658645155117
Accuracy score is = 95.80668054699328


At cv = 3
Cross validation score is = 95.33683324301187
Accuracy score is = 95.80668054699328


At cv = 4
Cross validation score is = 94.7979456598048
Accuracy score is = 95.80668054699328


At cv = 5
Cross validation score is = 95.75099108998597
Accuracy score is = 95.80668054699328


At cv = 6
Cross validation score is = 95.865765972429
Accuracy score is = 95.80668054699328


At cv = 7
Cross validation score is = 95.66429589744907
Accuracy score is = 95.80668054699328


At cv = 8
Cross validation score is = 95.85972510419968
Accuracy score is = 95.80668054699328


At cv = 9
Cross validation score is = 96.03626046604445
Accuracy score is = 95.80668054699328


At cv = 10
Cross validation score is = 95.96591043493615
Accuracy score is = 95.80668054699328


At cv = 11
Cross validation score is = 96.20226238413477
Accuracy score is = 95.80668054699328


At cv = 12
Cross validation score is = 95.81936985804825
Accuracy score is = 95.80668054699328


At cv = 13
Cross validation score is = 95.92534753134743
Accuracy score is = 95.80668054699328


At cv = 14
Cross validation score is = 95.76617745420496
Accuracy score is = 95.80668054699328
```

We select cv=12 because, as we can see, it produces the least difference between the Cross Validation score and the r2_score.

In [97]:
```
lsscore_selected=cross_val_score(knn,x,y,cv=12)
print('Cross validation score =',lsscore_selected.mean()*100,'\n','r2_score =',pred_knn_ac*100)
```

```
Cross validation score = 95.81936985804825
 r2_score = 95.80668054699328
```

Reviewing all the models.

model -- r2 -- cross -- val -- difference

Linear: 98.84, 98.57, and 0.27 RFR: 95.81, 94.90, and 0.91; DTC: 82.37, 82.47; and 0.1; SVR: 95.723, 95.68, and 0.04 ABR - 93.027 - 92.60 - 0.427 KNN - 95.80 - 95.82 - 0.02 GBR - 96.08 - 96.069 - 0.011**

As a result of our observation that the linear model has a significantly high difference but a high r2_score, we apply the hyperparameter to this model.

Enhanced Parameter

Linear Regression

In [98]:
```
grid_paramlr={

    'fit_intercept':[True,False],
    'copy_X' : [True,False],
    'positive' :[True,False]
}
```

In [99]:
```
#gridsearch cv use
from sklearn.model_selection import GridSearchCV
```

In [100…
```
gd_lr=GridSearchCV(estimator=lr,n_jobs=-1,
                   param_grid=grid_paramlr,
                   scoring='accuracy',
                   cv=11)
```

In [101…
```
#right now The parameter in our model is fitted.
gd_lr.fit(x,y)
```

Out[101]:
```
GridSearchCV(cv=11, estimator=LinearRegression(), n_jobs=-1,
             param_grid={'copy_X': [True, False],
                         'fit_intercept': [True, False],
                         'positive': [True, False]},
             scoring='accuracy')
```

Now that all of the hyperparameters have been applied to our model, checking the optimal parameter.

In [102…
```
best_parameters=gd_lr.best_params_
print(best_parameters)
```

```
{'copy_X': True, 'fit_intercept': True, 'positive': True}
```

Checking the cross-val-score and r2_score now.

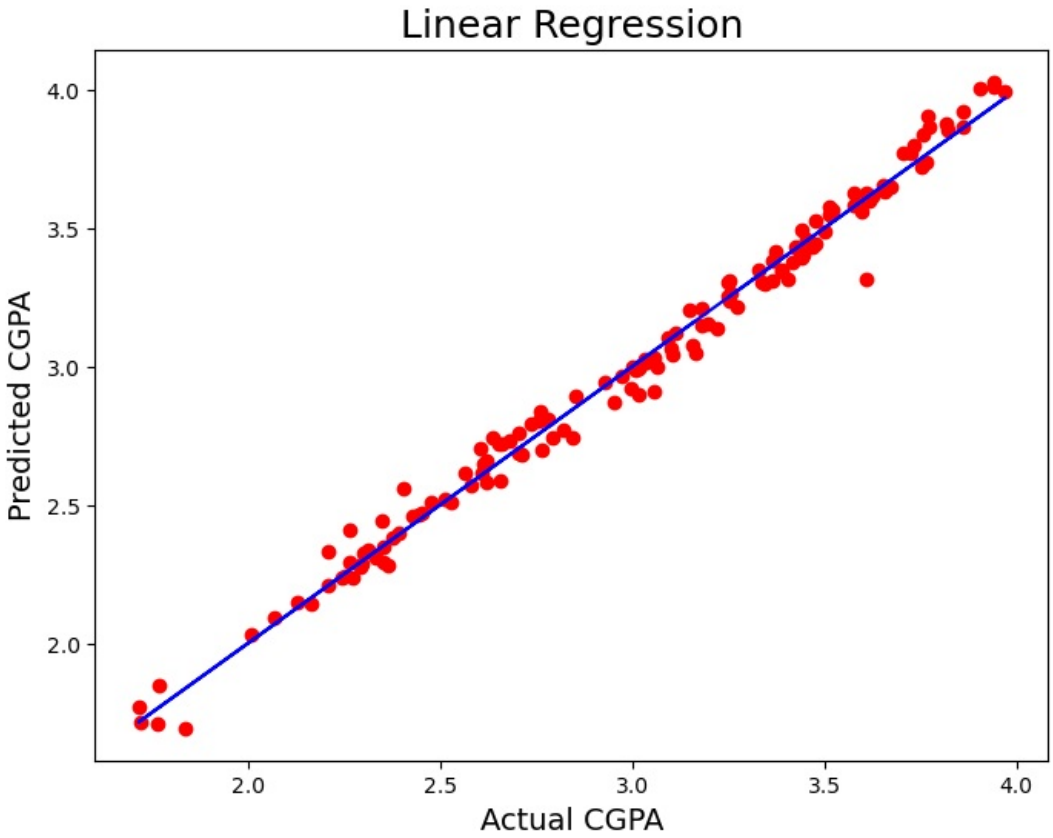
In [103…
```
lrh=LinearRegression(copy_X=True,fit_intercept=True)
```

In [104…
```
# Linear Regression
lrh.fit(x_train,y_train)
pred_test_lrh=lrh.predict(x_test)
print("r2_score is = ",r2_score(y_test,pred_test_lrh)*100)
print('mean_absolute_error = ',mean_absolute_error(y_test,pred_test_lrh))
print('mean_squared_error = ',mean_squared_error(y_test,pred_test_lrh))
print('root_mean_squared_error = ',np.sqrt(mean_squared_error(y_test,pred_test_lrh)))
```

```
r2_score is =  98.84080311608953
mean_absolute_error =  0.04552735127218016
mean_squared_error =  0.0036605825633282117
root_mean_squared_error =  0.060502748394830885
```

The output is the same as previously, so there is no need to check the cross-val score.

In [105…
```
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
plt.scatter(x=y_test,y=pred_test_lrh,color='r')
plt.plot(y_test,y_test,color='b')
plt.xlabel('Actual CGPA',fontsize=14)
plt.ylabel('Predicted CGPA',fontsize=14)
plt.title('Linear Regression',fontsize=18)
plt.savefig('lrh.png')
plt.show()
```



Applying PCA now, evaluating the new r2_score, then creating our model and saving it.

In [106…
```
x.shape
```

Out[106]:
```
(561, 41)
```

In [107…
```
y.shape
```

Out[107]:
```
(561,)
```

In [108…
```
from sklearn.decomposition import PCA
```

In [109…
```
mz=[10,20,30,35,33]
for i in mz:
    pca = PCA(i)
    x_pca = pca.fit_transform(x)
    x_train_pca, x_test_pca, y_train, y_test = train_test_split(x_pca, y, test_size=0.25, random_state=348)
    lr.fit(x_train_pca,y_train)
    pred_test_lrp=lr.predict(x_test_pca)
    print('n_components =',i)
    print("r2_score is = ",r2_score(y_test,pred_test_lrp)*100)
    print('mean_absolute_error = ',mean_absolute_error(y_test,pred_test_lrp))
    print('mean_squared_error = ',mean_squared_error(y_test,pred_test_lrp))
    print('root_mean_squared_error = ',np.sqrt(mean_squared_error(y_test,pred_test_lrp)),'\n')
```

```
n_components = 10
r2_score is =  98.81118906128594
mean_absolute_error =  0.045020065851818225
mean_squared_error =  0.003754099630314983
root_mean_squared_error =  0.06127070776737431

n_components = 20
r2_score is =  98.81143671028234
mean_absolute_error =  0.04642839575092093
mean_squared_error =  0.0037533175892220344
root_mean_squared_error =  0.06126432558367091

n_components = 30
r2_score is =  98.9090520511125
mean_absolute_error =  0.04493132752004172
mean_squared_error =  0.003445061917113278
root_mean_squared_error =  0.05869464981677017

n_components = 35
r2_score is =  98.85615626084063
mean_absolute_error =  0.0457743116535098
mean_squared_error =  0.003612099467187954
root_mean_squared_error =  0.060100744314758314

n_components = 33
r2_score is =  98.86157045663163
mean_absolute_error =  0.045664663133275295
mean_squared_error =  0.00359500218976942
root_mean_squared_error =  0.059958337116446284
```

Since the r2_score is high at n_components=30 and is 98.91298765418273, we will proceed with this.

In [110…
```
pca = PCA(n_components=30)
x_pca = pca.fit_transform(x)
x_train_pca, x_test_pca, y_train, y_test = train_test_split(x_pca, y, test_size=0.25, random_state=348)
```

In [111…
```
lrp=LinearRegression()
gbp=GradientBoostingRegressor()
adap=AdaBoostRegressor()
rfcp=RandomForestRegressor()
svcp=SVR()
dtcp=DecisionTreeRegressor()
knnp=KNeighborsRegressor()
```

In [112…
```
lrp.fit(x_train_pca,y_train)
pred_test_lrp=lrp.predict(x_test_pca)
print("r2_score is = ",r2_score(y_test,pred_test_lrp)*100)
print('mean_absolute_error = ',mean_absolute_error(y_test,pred_test_lrp))
print('mean_squared_error = ',mean_squared_error(y_test,pred_test_lrp))
print('root_mean_squared_error = ',np.sqrt(mean_squared_error(y_test,pred_test_lrp)),'\n')
```

```
r2_score is =  98.90928388188416
mean_absolute_error =  0.044917237651672985
mean_squared_error =  0.003444329827774313
root_mean_squared_error =  0.058688413062326986
```

In [113…
```
pred_lrp_ac=r2_score(y_test,pred_test_lrp)
for i in range(2,15):
    lsscore=cross_val_score(lrp,x_pca,y,cv=i)
    lsc=lsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',lsc*100)
    print('Accuracy score is =',pred_lrp_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 96.28736030970806
Accuracy score is = 98.90928388188416


At cv = 3
Cross validation score is = 98.18693538532166
Accuracy score is = 98.90928388188416


At cv = 4
Cross validation score is = 98.11836228123455
Accuracy score is = 98.90928388188416


At cv = 5
Cross validation score is = 98.48676210475318
Accuracy score is = 98.90928388188416


At cv = 6
Cross validation score is = 98.48090772909546
Accuracy score is = 98.90928388188416


At cv = 7
Cross validation score is = 98.50106528044533
Accuracy score is = 98.90928388188416


At cv = 8
Cross validation score is = 98.49654947067104
Accuracy score is = 98.90928388188416


At cv = 9
Cross validation score is = 98.54806050934047
Accuracy score is = 98.90928388188416


At cv = 10
Cross validation score is = 98.58082109543048
Accuracy score is = 98.90928388188416


At cv = 11
Cross validation score is = 98.58779726235485
Accuracy score is = 98.90928388188416


At cv = 12
Cross validation score is = 98.53126142959346
Accuracy score is = 98.90928388188416


At cv = 13
Cross validation score is = 98.55199362673838
Accuracy score is = 98.90928388188416


At cv = 14
Cross validation score is = 98.57807535165604
Accuracy score is = 98.90928388188416
```

We chose cv=11 because, as we can see, it produces the least difference between the Cross Validation score and the r2_score.

In [114…
```
# GradientBoostingRegressor
gbp.fit(x_train_pca,y_train)
pred_test_gbp=gbp.predict(x_test_pca)
print("r2_score is = ",r2_score(y_test,pred_test_gbp)*100)
print('mean_absolute_error = ',mean_absolute_error(y_test,pred_test_gbp))
print('mean_squared_error = ',mean_squared_error(y_test,pred_test_gbp))
print('root_mean_squared_error = ',np.sqrt(mean_squared_error(y_test,pred_test_gbp)))
```

```
r2_score is =  98.73608508513192
mean_absolute_error =  0.04570547840925493
mean_squared_error =  0.003991267543170806
root_mean_squared_error =  0.06317647935086923
```

In [115…
```
pred_gbp_ac=r2_score(y_test,pred_test_gbp)
for i in range(2,15):
    lsscore=cross_val_score(gbp,x_pca,y,cv=i)
    lsc=lsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',lsc*100)
    print('Accuracy score is =',pred_gbp_ac*100)
```

```
    print('\n')
```

```
At cv = 2
Cross validation score is = 97.60159178366558
Accuracy score is = 98.73608508513192


At cv = 3
Cross validation score is = 98.39704954365797
Accuracy score is = 98.73608508513192


At cv = 4
Cross validation score is = 98.18954630675226
Accuracy score is = 98.73608508513192


At cv = 5
Cross validation score is = 98.54396119531377
Accuracy score is = 98.73608508513192


At cv = 6
Cross validation score is = 98.65165606731304
Accuracy score is = 98.73608508513192


At cv = 7
Cross validation score is = 98.58543036392648
Accuracy score is = 98.73608508513192


At cv = 8
Cross validation score is = 98.59681779951931
Accuracy score is = 98.73608508513192


At cv = 9
Cross validation score is = 98.65806815821601
Accuracy score is = 98.73608508513192


At cv = 10
Cross validation score is = 98.6127303852367
Accuracy score is = 98.73608508513192


At cv = 11
Cross validation score is = 98.65231728966062
Accuracy score is = 98.73608508513192


At cv = 12
Cross validation score is = 98.69417102510641
Accuracy score is = 98.73608508513192


At cv = 13
Cross validation score is = 98.58552219261706
Accuracy score is = 98.73608508513192


At cv = 14
Cross validation score is = 98.61062746678975
Accuracy score is = 98.73608508513192
```

We select cv=12 because, as we can see, it produces the least difference between the Cross Validation score and the r2_score.

In [116…
```
# AdaBoostRegression
adap.fit(x_train_pca,y_train)
pred_test_adap=adap.predict(x_test_pca)
print("r2_score is = ",r2_score(y_test,pred_test_adap)*100)
print('mean_absolute_error = ',mean_absolute_error(y_test,pred_test_adap))
print('mean_squared_error = ',mean_squared_error(y_test,pred_test_adap))
print('root_mean_squared_error = ',np.sqrt(mean_squared_error(y_test,pred_test_adap)))
```

```
r2_score is =  98.29744240672952
mean_absolute_error =  0.0562551242713348
mean_squared_error =  0.005376440124617686
root_mean_squared_error =  0.07332421240366435
```

In [117…
```
pred_adap_ac=r2_score(y_test,pred_test_adap)
for i in range(2,15):
    lsscore=cross_val_score(adap,x_pca,y,cv=i)
    lsc=lsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',lsc*100)
```

```
    print('Accuracy score is =',pred_adap_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 97.14961879220756
Accuracy score is = 98.29744240672952


At cv = 3
Cross validation score is = 98.16227709967968
Accuracy score is = 98.29744240672952


At cv = 4
Cross validation score is = 98.02315587956011
Accuracy score is = 98.29744240672952


At cv = 5
Cross validation score is = 98.15122241854961
Accuracy score is = 98.29744240672952


At cv = 6
Cross validation score is = 98.08604501225578
Accuracy score is = 98.29744240672952


At cv = 7
Cross validation score is = 98.10883544607924
Accuracy score is = 98.29744240672952


At cv = 8
Cross validation score is = 98.1451641720994
Accuracy score is = 98.29744240672952


At cv = 9
Cross validation score is = 98.2040990021741
Accuracy score is = 98.29744240672952


At cv = 10
Cross validation score is = 98.21047087357817
Accuracy score is = 98.29744240672952


At cv = 11
Cross validation score is = 98.18646908124002
Accuracy score is = 98.29744240672952


At cv = 12
Cross validation score is = 98.19006727431224
Accuracy score is = 98.29744240672952


At cv = 13
Cross validation score is = 98.09687780333891
Accuracy score is = 98.29744240672952


At cv = 14
Cross validation score is = 98.20788198491854
Accuracy score is = 98.29744240672952
```

We chose cv=11 because, as we can see, it produces the least difference between the Cross Validation score and the r2_score.

In [118…
```
# RandomForestRegression
rfcp.fit(x_train_pca,y_train)
pred_test_rfcp=rfcp.predict(x_test_pca)
print("r2_score is = ",r2_score(y_test,pred_test_rfcp)*100)
print('mean_absolute_error = ',mean_absolute_error(y_test,pred_test_rfcp))
print('mean_squared_error = ',mean_squared_error(y_test,pred_test_rfcp))
print('root_mean_squared_error = ',np.sqrt(mean_squared_error(y_test,pred_test_rfcp)))
```

```
r2_score is =  98.55092844720583
mean_absolute_error =  0.04969361702127671
mean_squared_error =  0.0045759664581560425
root_mean_squared_error =  0.06764589017934529
```

In [119…
```
pred_rfcp_ac=r2_score(y_test,pred_test_rfcp)
for i in range(2,15):
    lsscore=cross_val_score(rfcp,x_pca,y,cv=i)
    lsc=lsscore.mean()
    print('At cv =',i)
```

```
    print('Cross validation score is =',lsc*100)
    print('Accuracy score is =',pred_rfcp_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 97.6859625599806
Accuracy score is = 98.55092844720583


At cv = 3
Cross validation score is = 98.36793398465042
Accuracy score is = 98.55092844720583


At cv = 4
Cross validation score is = 98.32276685206095
Accuracy score is = 98.55092844720583


At cv = 5
Cross validation score is = 98.48297603169239
Accuracy score is = 98.55092844720583


At cv = 6
Cross validation score is = 98.51749773557692
Accuracy score is = 98.55092844720583


At cv = 7
Cross validation score is = 98.48967225202989
Accuracy score is = 98.55092844720583


At cv = 8
Cross validation score is = 98.51054356528677
Accuracy score is = 98.55092844720583


At cv = 9
Cross validation score is = 98.4575574414931
Accuracy score is = 98.55092844720583


At cv = 10
Cross validation score is = 98.60327126007797
Accuracy score is = 98.55092844720583


At cv = 11
Cross validation score is = 98.52686337208628
Accuracy score is = 98.55092844720583


At cv = 12
Cross validation score is = 98.52598745613757
Accuracy score is = 98.55092844720583


At cv = 13
Cross validation score is = 98.49832909799441
Accuracy score is = 98.55092844720583


At cv = 14
Cross validation score is = 98.4794330261216
Accuracy score is = 98.55092844720583
```

We select cv=6 because, as we can see, it produces the least variation between the Cross Validation score and the r2_score.

In [120…
```
# DecisionTreeRegression
dtcp.fit(x_train_pca,y_train)
pred_test_dtcp=dtcp.predict(x_test_pca)
print("r2_score is = ",r2_score(y_test,pred_test_dtcp)*100)
print('mean_absolute_error = ',mean_absolute_error(y_test,pred_test_dtcp))
print('mean_squared_error = ',mean_squared_error(y_test,pred_test_dtcp))
print('root_mean_squared_error = ',np.sqrt(mean_squared_error(y_test,pred_test_dtcp)))
```

```
r2_score is =  97.6628110398202
mean_absolute_error =  0.06416312056737586
mean_squared_error =  0.007380517730496451
root_mean_squared_error =  0.08590939964900948
```

In [121…
```
pred_dtcp_ac=r2_score(y_test,pred_test_dtcp)
for i in range(2,15):
    lsscore=cross_val_score(dtcp,x_pca,y,cv=i)
    lsc=lsscore.mean()
```

```
    print('At cv =',i)
    print('Cross validation score is =',lsc*100)
    print('Accuracy score is =',pred_dtcp_ac*100)
    print('\n')
```

```
At cv = 2
Cross validation score is = 96.75677555740724
Accuracy score is = 97.6628110398202


At cv = 3
Cross validation score is = 96.07416756155853
Accuracy score is = 97.6628110398202


At cv = 4
Cross validation score is = 97.33346104759548
Accuracy score is = 97.6628110398202


At cv = 5
Cross validation score is = 96.85209801782933
Accuracy score is = 97.6628110398202


At cv = 6
Cross validation score is = 97.49717166065524
Accuracy score is = 97.6628110398202


At cv = 7
Cross validation score is = 97.2498710956102
Accuracy score is = 97.6628110398202


At cv = 8
Cross validation score is = 97.29473971831959
Accuracy score is = 97.6628110398202


At cv = 9
Cross validation score is = 97.37384529748648
Accuracy score is = 97.6628110398202


At cv = 10
Cross validation score is = 97.38646081861327
Accuracy score is = 97.6628110398202


At cv = 11
Cross validation score is = 97.3771121672863
Accuracy score is = 97.6628110398202


At cv = 12
Cross validation score is = 97.4515678337474
Accuracy score is = 97.6628110398202


At cv = 13
Cross validation score is = 97.40945525090781
Accuracy score is = 97.6628110398202


At cv = 14
Cross validation score is = 97.0670407826379
Accuracy score is = 97.6628110398202
```

We chose cv=2 because, as we can see, it produces the least difference between the Cross Validation score and the r2_score.

In [122…
```
# KNeighborsRegression
knnp.fit(x_train_pca,y_train)
pred_test_knnp=knnp.predict(x_test_pca)
print("r2_score is = ",r2_score(y_test,pred_test_knnp)*100)
print('mean_absolute_error = ',mean_absolute_error(y_test,pred_test_knnp))
print('mean_squared_error = ',mean_squared_error(y_test,pred_test_knnp))
print('root_mean_squared_error = ',np.sqrt(mean_squared_error(y_test,pred_test_knnp)))
```

```
r2_score is =  96.41799102293767
mean_absolute_error =  0.07805531914893618
mean_squared_error =  0.011311486241134755
root_mean_squared_error =  0.10635547113869956
```

In [123…
```
pred_knnp_ac=r2_score(y_test,pred_test_knnp)
for i in range(2,15):
    lsscore=cross_val_score(knnp,x_pca,y,cv=i)
```

```
    lsc=lsscore.mean()
    print('At cv =',i)
    print('Cross validation score is =',lsc*100)
    print('Accuracy score is =',pred_knnp_ac*100)
    print('\n')
```

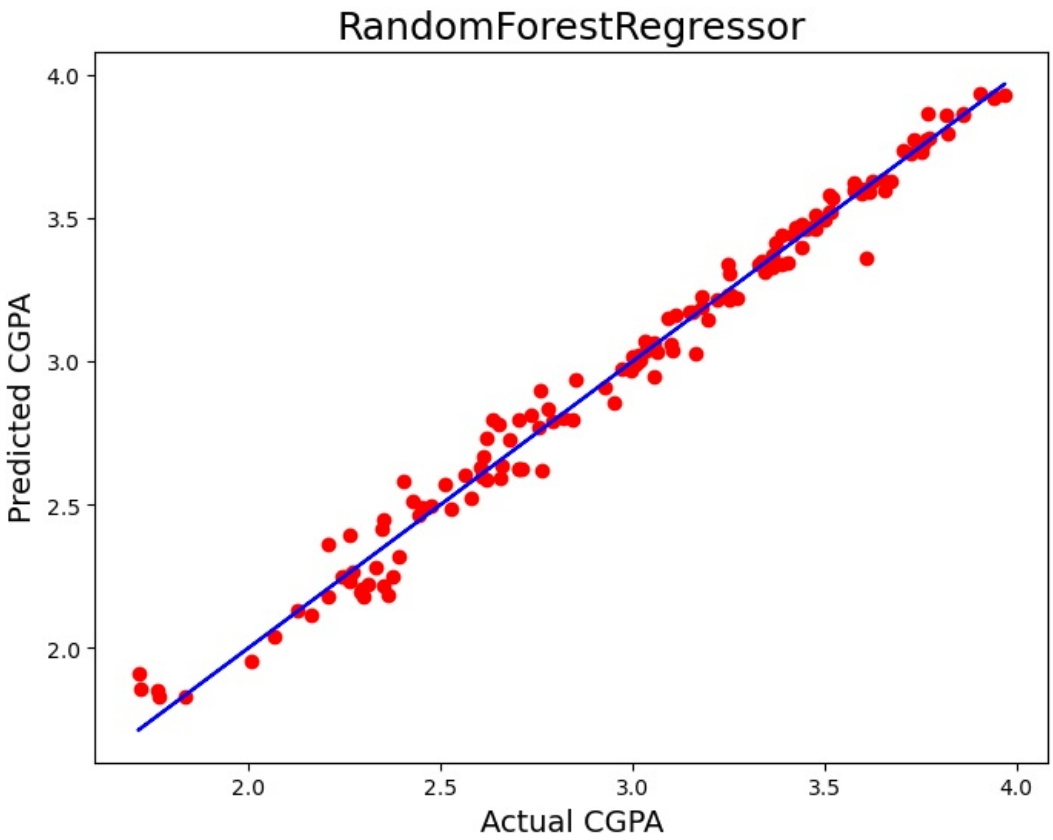
```
At cv = 2
Cross validation score is = 94.36453804156915
Accuracy score is = 96.41799102293767


At cv = 3
Cross validation score is = 95.8662673816197
Accuracy score is = 96.41799102293767


At cv = 4
Cross validation score is = 95.26668702381032
Accuracy score is = 96.41799102293767


At cv = 5
Cross validation score is = 96.2456518443714
Accuracy score is = 96.41799102293767


At cv = 6
Cross validation score is = 96.18671320385513
Accuracy score is = 96.41799102293767


At cv = 7
Cross validation score is = 96.10864114309527
Accuracy score is = 96.41799102293767


At cv = 8
Cross validation score is = 96.42482065849195
Accuracy score is = 96.41799102293767


At cv = 9
Cross validation score is = 96.5911633779187
Accuracy score is = 96.41799102293767


At cv = 10
Cross validation score is = 96.52667748861431
Accuracy score is = 96.41799102293767


At cv = 11
Cross validation score is = 96.56019442984085
Accuracy score is = 96.41799102293767


At cv = 12
Cross validation score is = 96.43971316018579
Accuracy score is = 96.41799102293767


At cv = 13
Cross validation score is = 96.53367349881432
Accuracy score is = 96.41799102293767


At cv = 14
Cross validation score is = 96.52137920284892
Accuracy score is = 96.41799102293767
```

We chose cv=13 because, as we can see, it produces the least variation between the Cross Validation score and the r2_score.

Analysing every model.. model -- r2-score -- cross - val -- difference lrp - 98.90 - 98.56 - 0.32 RFRp - 98.63 - 98.57 - 2.06 DTCp - 96.90 - 96.87 - 0.19 SVRp - 95.83 - 95.90 - 0.06 KNNp - 96.45 - 96.45 - 0.07 GBRp - 98.78 - 96.72 - 0.03 ABRp - 98.41 - 98.22 - 0.03 R2_score and difference observations of all the approaches led us to choose RandomForestRegressor.

In [124…

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
plt.scatter(x=y_test,y=pred_test_rfcp,color='r')
plt.plot(y_test,y_test,color='b')
plt.xlabel('Actual CGPA',fontsize=14)
plt.ylabel('Predicted CGPA',fontsize=14)
plt.title('RandomForestRegressor',fontsize=18)
plt.savefig('rfh.png')
plt.show()
```



We have now saved two models, one of which is a linear predictor and the other a PCA random forest predictor. Each is excellent and quite accurate.

The model is saved

In [125…
```
import joblib
joblib.dump(rfcp,'RandomForestRegressorGrades.obj')
```

Out[125]:
```
['RandomForestRegressorGrades.obj']
```

In [126…
```
joblib.dump(lrh,'LinearRegressionGrades.obj')
```

Out[126]:
```
['LinearRegressionGrades.obj']
```

In [ ]: