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

In [2]:
```
df = pd.read_csv ("grades.csv")
df.head(10)
```

Out[2]:

| | 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 |
| **5** | CS-97006 | A | B- | B | D+ | C+ | A- | C+ | B | B+ | ... | C | B- | C+ | C+ | A- | A- | B+ | B- | B | 3.026 |
| **6** | CS-97007 | B- | C- | A- | D | A- | B | A | B+ | A- | ... | B- | C | B+ | B- | A | A- | B+ | B+ | A | 2.957 |
| **7** | CS-97008 | B+ | B+ | C+ | C | C | A- | C- | A- | B | ... | D+ | B+ | B+ | C | A | B+ | A- | A- | A- | 3.043 |
| **8** | CS-97009 | A- | C | A- | D+ | B | A- | A- | A- | B+ | ... | B+ | B | B- | C+ | A | A- | B+ | A- | A | 3.358 |
| **9** | CS-97010 | A | B | C+ | B | B | A- | C | B- | A- | ... | C+ | B- | B+ | B- | A | A | A- | B | A | 3.247 |

10 rows × 43 columns

In [4]:
```
df.shape
# here in the given data set there are 571 rows and 43 columns are present.
```

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

In [6]:
```
df.columns
# here following we can see the columns names which are present in the dataset.
```

Out[6]:
```
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')
```

In [8]:
```
df.columns.unique()
# here we can find that the total columns and the unique columns are same , there is no such any difference inthe total col
#  and unique columns.
```

Out[8]:
```
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')
```

In [10]:
```
df.columns.nunique()
# the total number of unique columns are 43 , same as number of total number of columns.
```

Out[10]:
```
43
```

In [12]:
```
df.dtypes
# here we can see that out of all 43 columns only one column 'CGPA' datatype is ' float64', rest of the columns are 'object
```

Out[12]:
```
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 [14]:
```
df.info()
#  here we can see that
# 1) total number for columns present : 43
# 2) total number of rows presnet : 571
# 3) total "data types present in data set" : 2 (i.e "object & float64")
#  out of which   42 columns of - object
#                 1 column of - float64
# 4)NULL VALUES are may present in some of the columns in dataset.
#    because there is a difference between the total count and present values (may be whitespaces) are present.
#  5) So we have to check 'null values' & 'whitespaces' ni our dataset.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 571 entries, 0 to 570
Data columns (total 43 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Seat No.   571 non-null    object
 1   PH-121     571 non-null    object
 2   HS-101     571 non-null    object
 3   CY-105     570 non-null    object
 4   HS-105/12  570 non-null    object
 5   MT-111     569 non-null    object
 6   CS-105     571 non-null    object
 7   CS-106     569 non-null    object
 8   EL-102     569 non-null    object
 9   EE-119     569 non-null    object
 10  ME-107     569 non-null    object
 11  CS-107     569 non-null    object
 12  HS-205/20  566 non-null    object
 13  MT-222     566 non-null    object
 14  EE-222     564 non-null    object
 15  MT-224     564 non-null    object
 16  CS-210     564 non-null    object
 17  CS-211     566 non-null    object
 18  CS-203     566 non-null    object
 19  CS-214     565 non-null    object
 20  EE-217     565 non-null    object
 21  CS-212     565 non-null    object
 22  CS-215     565 non-null    object
 23  MT-331     562 non-null    object
 24  EF-303     561 non-null    object
 25  HS-304     561 non-null    object
 26  CS-301     561 non-null    object
 27  CS-302     561 non-null    object
 28  TC-383     561 non-null    object
 29  MT-442     561 non-null    object
 30  EL-332     562 non-null    object
 31  CS-318     562 non-null    object
 32  CS-306     562 non-null    object
 33  CS-312     561 non-null    object
 34  CS-317     559 non-null    object
 35  CS-403     559 non-null    object
 36  CS-421     559 non-null    object
 37  CS-406     486 non-null    object
 38  CS-414     558 non-null    object
 39  CS-419     558 non-null    object
 40  CS-423     557 non-null    object
 41  CS-412     492 non-null    object
 42  CGPA       571 non-null    float64
dtypes: float64(1), object(42)
memory usage: 191.9+ KB
```

In [ ]:
```
```

CHECKING NULL VALUES

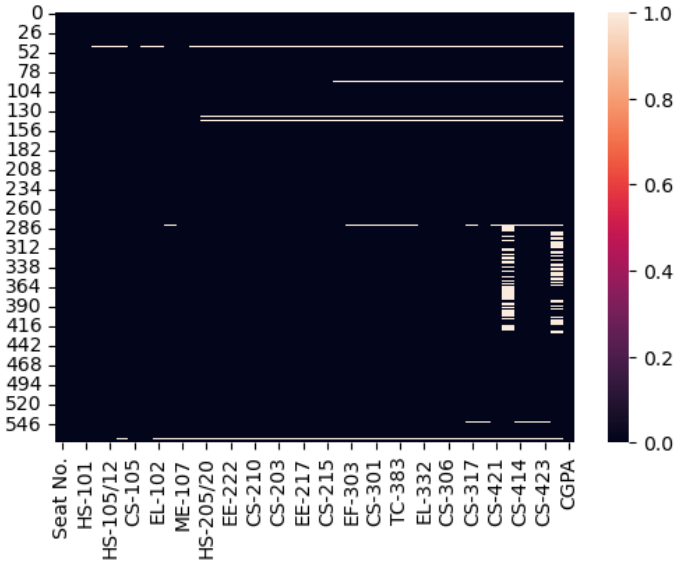
==========

In [18]:
```
df.isnull().sum()
#  here we can clearly see the presence of null values .
#  so first we have to remove null values.
```

Out[18]:
```
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
```

In [19]:
```
plt.figure(figsize=(6,4))
sns.heatmap(df.isnull())
# Here we can also check null values with the help of Heatmap
#  here in the heatmap we can clearly see the presence of null vlaues in the given dataset.
#  here we can also observe that , there is white line of null values thoughout the columns, we can say that some rows may
#  deleted from the data set.
# now we have to replace the null values fro the dataset.
```

Out[19]: <AxesSubplot:>




In [20]:
```
from sklearn.impute import SimpleImputer
#  to replace null values we have to import simpleimputer, first
```

In [21]:
```
imp = SimpleImputer(strategy="most_frequent")
#  as we can see the values of the independent columns are in 'object' datatype. (they are grades)
#  so we can't replace the null values with 'mean' or 'median'.
#  we can replace the null values with the 'most-frequent' values present in the columns.
#  so the null values can be replaced by the most - frequent values present in the column.
```

In [22]:
```
df.isnull().sum()
```

Out[22]:
```
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
```

In [ ]:
```

```

REMOVING NULL VALUES

==========

In [25]:
```
df['CY-105']= imp.fit_transform(df['CY-105'].values.reshape(-1,1))
```

In [27]:
```
df['HS-105/12']= imp.fit_transform(df['HS-105/12'].values.reshape(-1,1))
```

In [28]:
```
df['MT-111']= imp.fit_transform(df['MT-111'].values.reshape(-1,1))
```

In [29]:
```
df['CS-106']= imp.fit_transform(df['CS-106'].values.reshape(-1,1))
```

In [30]:
```
df['EL-102']= imp.fit_transform(df['EL-102'].values.reshape(-1,1))
```

In [31]:
```
df['EE-119']= imp.fit_transform(df['EE-119'].values.reshape(-1,1))
```

In [32]:
```
df['ME-107']= imp.fit_transform(df['ME-107'].values.reshape(-1,1))
```

In [33]:
```
df['CS-107']= imp.fit_transform(df['CS-107'].values.reshape(-1,1))
```

In [35]:
```
df['HS-205/20']= imp.fit_transform(df['HS-205/20'].values.reshape(-1,1))
```

In [36]:
```
df['MT-222']= imp.fit_transform(df['MT-222'].values.reshape(-1,1))
```

In [37]:
```
df['MT-224']= imp.fit_transform(df['MT-224'].values.reshape(-1,1))
```

```
In [38]: df['CS-210']= imp.fit_transform(df['CS-210'].values.reshape(-1,1))
In [39]: df['CS-211']= imp.fit_transform(df['CS-211'].values.reshape(-1,1))
In [40]: df['CS-203']= imp.fit_transform(df['CS-203'].values.reshape(-1,1))
In [41]: df['CS-214']= imp.fit_transform(df['CS-214'].values.reshape(-1,1))
In [42]: df['EE-217']= imp.fit_transform(df['EE-217'].values.reshape(-1,1))
In [43]: df['CS-212']= imp.fit_transform(df['CS-212'].values.reshape(-1,1))
In [44]: df['MT-442']= imp.fit_transform(df['MT-442'].values.reshape(-1,1))
In [45]: df['EL-332']= imp.fit_transform(df['EL-332'].values.reshape(-1,1))
In [46]: df['CS-318']= imp.fit_transform(df['CS-318'].values.reshape(-1,1))
In [47]: df['CS-306']= imp.fit_transform(df['CS-306'].values.reshape(-1,1))
In [48]: df['CS-312']= imp.fit_transform(df['CS-312'].values.reshape(-1,1))
In [49]: df['CS-317']= imp.fit_transform(df['CS-317'].values.reshape(-1,1))
In [50]: df['CS-403']= imp.fit_transform(df['CS-403'].values.reshape(-1,1))
In [51]: df['CS-421']= imp.fit_transform(df['CS-421'].values.reshape(-1,1))
In [52]: df['CS-406']= imp.fit_transform(df['CS-406'].values.reshape(-1,1))
In [53]: df['CS-414']= imp.fit_transform(df['CS-414'].values.reshape(-1,1))
In [54]: df['CS-419']= imp.fit_transform(df['CS-419'].values.reshape(-1,1))
In [55]: df['CS-423']= imp.fit_transform(df['CS-423'].values.reshape(-1,1))
In [56]: df['CS-412']= imp.fit_transform(df['CS-412'].values.reshape(-1,1))
In [57]: df['CS-215']= imp.fit_transform(df['CS-215'].values.reshape(-1,1))
In [58]: df['MT-331']= imp.fit_transform(df['MT-331'].values.reshape(-1,1))
In [59]: df['EF-303']= imp.fit_transform(df['EF-303'].values.reshape(-1,1))
In [60]: df['HS-304']= imp.fit_transform(df['HS-304'].values.reshape(-1,1))
In [61]: df['CS-301']= imp.fit_transform(df['CS-301'].values.reshape(-1,1))
In [62]: df['CS-302']= imp.fit_transform(df['CS-302'].values.reshape(-1,1))
In [63]: df['TC-383']= imp.fit_transform(df['TC-383'].values.reshape(-1,1))
In [65]: df['EE-222']= imp.fit_transform(df['EE-222'].values.reshape(-1,1))
```

In [66]:
```
df.isnull().sum()
#  here above we can successfully removed the null values from our dataset.
```

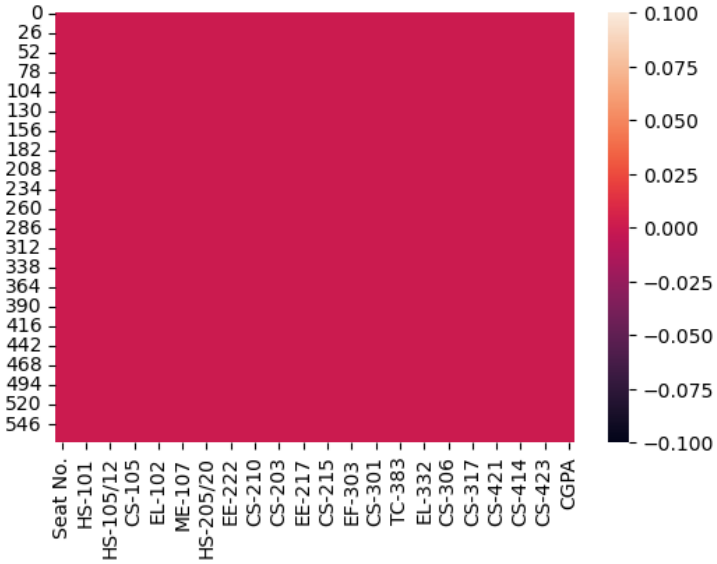
Out[66]:
```
Seat No.     0
PH-121       0
HS-101       0
CY-105       0
HS-105/12    0
MT-111       0
CS-105       0
CS-106       0
EL-102       0
EE-119       0
ME-107       0
CS-107       0
HS-205/20    0
MT-222       0
EE-222       0
MT-224       0
CS-210       0
CS-211       0
CS-203       0
CS-214       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
```

In [67]:
```
df.info()
# here also we can see that we have successfully remove all the null present in the dataset.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 571 entries, 0 to 570
Data columns (total 43 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Seat No.   571 non-null    object 
 1   PH-121     571 non-null    object 
 2   HS-101     571 non-null    object 
 3   CY-105     571 non-null    object 
 4   HS-105/12  571 non-null    object 
 5   MT-111     571 non-null    object 
 6   CS-105     571 non-null    object 
 7   CS-106     571 non-null    object 
 8   EL-102     571 non-null    object 
 9   EE-119     571 non-null    object 
 10  ME-107     571 non-null    object 
 11  CS-107     571 non-null    object 
 12  HS-205/20  571 non-null    object 
 13  MT-222     571 non-null    object 
 14  EE-222     571 non-null    object 
 15  MT-224     571 non-null    object 
 16  CS-210     571 non-null    object 
 17  CS-211     571 non-null    object 
 18  CS-203     571 non-null    object 
 19  CS-214     571 non-null    object 
 20  EE-217     571 non-null    object 
 21  CS-212     571 non-null    object 
 22  CS-215     571 non-null    object 
 23  MT-331     571 non-null    object 
 24  EF-303     571 non-null    object 
 25  HS-304     571 non-null    object 
 26  CS-301     571 non-null    object 
 27  CS-302     571 non-null    object 
 28  TC-383     571 non-null    object 
 29  MT-442     571 non-null    object 
 30  EL-332     571 non-null    object 
 31  CS-318     571 non-null    object 
 32  CS-306     571 non-null    object 
 33  CS-312     571 non-null    object 
 34  CS-317     571 non-null    object 
 35  CS-403     571 non-null    object 
 36  CS-421     571 non-null    object 
 37  CS-406     571 non-null    object 
 38  CS-414     571 non-null    object 
 39  CS-419     571 non-null    object 
 40  CS-423     571 non-null    object 
 41  CS-412     571 non-null    object 
 42  CGPA       571 non-null    float64
dtypes: float64(1), object(42)
memory usage: 191.9+ KB
```

In [69]:
```
plt.figure(figsize=(6,4))
sns.heatmap(df.isnull())
#  successful removal of null values.
```

Out[69]: <AxesSubplot:>




In [ ]:

CHECKING UNIQUE VALUES PRENSENT IN DATASET & UNIVARIATE ANALYSIS

==========

In [72]:
```
df.nunique()
# here in the following we can find the 'NUMBER OF UNIQUE VLAUES PRESENT IN ALL COLUMNS'
```

Out[72]:
```
Seat No.     571
PH-121        13
HS-101        12
CY-105        13
HS-105/12     13
MT-111        13
CS-105        11
CS-106        13
EL-102        13
EE-119        12
ME-107        13
CS-107        14
HS-205/20     13
MT-222        14
EE-222        13
MT-224        14
CS-210        14
CS-211        14
CS-203        13
CS-214        13
EE-217        13
CS-212        12
CS-215        14
MT-331        12
EF-303        12
HS-304        14
CS-301        12
CS-302        11
TC-383        12
MT-442        12
EL-332        13
CS-318        14
CS-306        13
CS-312        14
CS-317        12
CS-403        11
CS-421        13
CS-406        14
CS-414        13
CS-419        12
CS-423        12
CS-412        13
CGPA         491
dtype: int64
```

In [75]:
```
df['PH-121'].unique()
```

Out[75]:
```
array(['B-', 'A', 'D', 'A-', 'B+', 'B', 'C+', 'C', 'C-', 'D+', 'WU', 'A+',
       'F'], dtype=object)
```

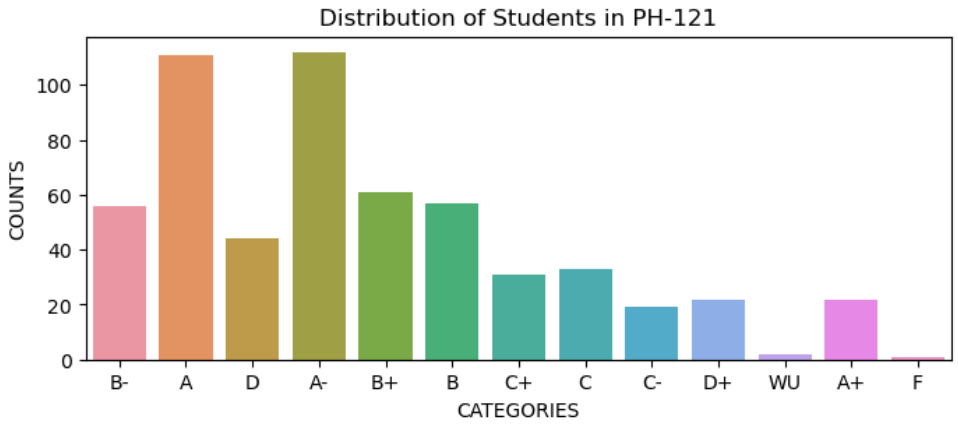
In [78]:
```
df['PH-121'].value_counts()
# here we can find that majority of the students find's, 'A', 'A-' grades..
#  similarly we can also find same for all other columns.
```

Out[78]:
```
A-    112
A     111
B+     61
B      57
B-     56
D      44
C      33
C+     31
D+     22
A+     22
C-     19
WU      2
F       1
Name: PH-121, dtype: int64
```

In [98]:
```
plt.figure (figsize = (8,3), facecolor = "white")
plt.title('Distribution of Students in PH-121')
sns.countplot(x='PH-121', data = df)
plt.xlabel('CATEGORIES', fontsize=10)
# plt.xticks(rotation=30, ha = 'right')
plt.ylabel('COUNTS')
# plt.yticks(rotation=30, ha = 'right')

# Here we can see that the most of counts are present in category 'A' & 'A-'.
# as above also we find the same in numerical form.
#  similarly we can also find same for all other columns.
```

Out[98]:
```
Text(0, 0.5, 'COUNTS')
```




In [79]:
```
df['HS-101'].value_counts()
#  here also we can see that most of the students got 'A-' 'B-' & 'C' grades.
#  we are not going to check this for all the columns , but if want to find the same then we can also do the same.
```

Out[79]:
```
A-    82
B-    78
C     68
B     63
B+    59
C-    50
C+    47
D     45
A     38
D+    36
A+     4
F      1
Name: HS-101, dtype: int64
```

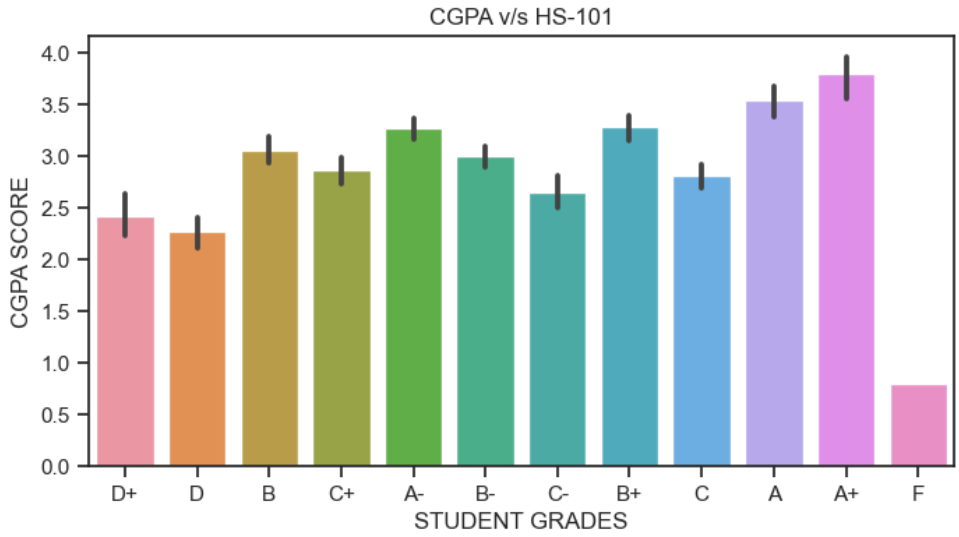
In [ ]:

BIVARIATE ANALYSIS

==========

In [120]:
```
plt.figure (figsize = (8,4), facecolor = "white")
plt.title('CGPA v/s HS-101')
sns.barplot (x= 'HS-101', y = 'CGPA', data= df )
# plt.xticks(rotation=30, ha = 'right')
plt.xlabel('STUDENT GRADES')
plt.ylabel('CGPA SCORE')
# plt.legend(loc= 'center', fontsize=6)
plt.show()

# Here we can see that the CGPA SCORE of 'A' & 'A+' grade studentd is higher as compared to others.
```

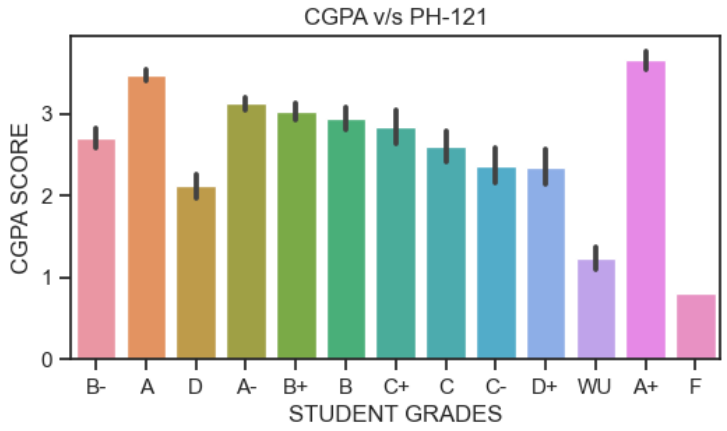



In [132]:
```
df['PH-121'].value_counts()
```

Out[132]:
```
A-    112
A     111
B+     61
B      57
B-     56
D      44
C      33
C+     31
D+     22
A+     22
C-     19
WU      2
F       1
Name: PH-121, dtype: int64
```

In [131]:
```
plt.figure (figsize = (6,3), facecolor = "white")
plt.title('CGPA v/s PH-121')
sns.barplot (x= 'PH-121', y = 'CGPA', data= df )
# plt.xticks(rotation=30, ha = 'right')
plt.xlabel('STUDENT GRADES')
plt.ylabel('CGPA SCORE')
# plt.legend(loc= 'center', fontsize=6)
plt.show()

# Here also we can see that the CGPA SCORE of 'A' & 'A+' grade studentd is higher and then it is decreasing
# as  according to grades.
# from this we can say say that the grades are positively interconnected with the CGPA SCORE.
```




In [ ]:

In [138]:
```
df.head()
```

Out[138]:

| | 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 |

5 rows × 43 columns

In [140]:
```
# here as we can see all independent columns are 'object' & only dependent / Target column is 'float64'
# therefor first we have to change all independent columns from 'objet' to 'float / int'
#  for this we can use ENCODING TECHNIQUES
```

In [ ]:

ENCODING TECHNIQUES ====>>>>>>>>>

In [142]:
```
# here we are having 43-2 = 41 categorical columns whose datatype is "object", so first we have to encode them .
```

In [143]:
```
# here in our all categorical columns , we are having categories more then two.
# so here we are apllying "LABEL ENCODER" for all the categorical columns:-
#  for which we have to import some libraries
```

In [144]:
```
from sklearn.preprocessing import LabelEncoder
```

In [145]:
```
le = LabelEncoder()
```

In [146]:
```
df.columns
```

Out[146]:
```
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')
```

In [153]:
```
df["MT-331"] = le.fit_transform(df["MT-331"])
df["EF-303"] = le.fit_transform(df["EF-303"])
df["HS-304"] = le.fit_transform(df["HS-304"])
df["CS-301"] = le.fit_transform(df["CS-301"])
df["CS-302"] = le.fit_transform(df["CS-302"])
df["TC-383"] = le.fit_transform(df["TC-383"])
df["MT-442"] = le.fit_transform(df["MT-442"])
df["EL-332"] = le.fit_transform(df["EL-332"])
df["CS-318"] = le.fit_transform(df["CS-318"])
df["CS-306"] = le.fit_transform(df["CS-306"])
df["CS-312"] = le.fit_transform(df["CS-312"])
df["CS-317"] = le.fit_transform(df["CS-317"])
df["CS-403"] = le.fit_transform(df["CS-403"])
df["CS-421"] = le.fit_transform(df["CS-421"])
df["CS-406"] = le.fit_transform(df["CS-406"])
df["CS-414"] = le.fit_transform(df["CS-419"])
df["CS-423"] = le.fit_transform(df["CS-423"])
df["CS-412"] = le.fit_transform(df["CS-412"])
df["PH-121"] = le.fit_transform(df["PH-121"])
df["HS-101"] = le.fit_transform(df["HS-101"])
df["CY-105"] = le.fit_transform(df["CY-105"])
df["HS-105/12"] = le.fit_transform(df["HS-105/12"])
df["MT-111"] = le.fit_transform(df["MT-111"])
df["CS-105"] = le.fit_transform(df["CS-105"])
df["CS-106"] = le.fit_transform(df["CS-106"])
df["EL-102"] = le.fit_transform(df["EL-102"])
df["EE-119"] = le.fit_transform(df["EE-119"])
df["ME-107"] = le.fit_transform(df["ME-107"])
df["CS-107"] = le.fit_transform(df["CS-107"])
df["HS-205/20"] = le.fit_transform(df["HS-205/20"])
df["MT-222"] = le.fit_transform(df["MT-222"])
df["EE-222"] = le.fit_transform(df["EE-222"])
df["MT-224"] = le.fit_transform(df["MT-224"])
df["CS-210"] = le.fit_transform(df["CS-210"])
df["CS-211"] = le.fit_transform(df["CS-211"])
df["CS-203"] = le.fit_transform(df["CS-203"])
df["CS-214"] = le.fit_transform(df["CS-214"])
df["EE-217"] = le.fit_transform(df["EE-217"])
df["CS-212"] = le.fit_transform(df["CS-212"])
df["CS-215"] = le.fit_transform(df["CS-215"])
df["CS-419"] = le.fit_transform(df["CS-419"])
```

In [155]:
```
df.head()
```

Out[155]:

| | 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 | 5 | 10 | 8 | 6 | 8 | 10 | 9 | 8 | 5 | ... | 8 | 8 | 8 | 8 | 2 | 8 | 8 | 3 | 2 | 2.205 |
| **1** | CS-97002 | 0 | 9 | 10 | 9 | 5 | 6 | 9 | 0 | 10 | ... | 10 | 9 | 6 | 9 | 2 | 6 | 6 | 6 | 3 | 2.008 |
| **2** | CS-97003 | 0 | 3 | 0 | 5 | 4 | 0 | 5 | 4 | 2 | ... | 3 | 3 | 0 | 6 | 0 | 0 | 0 | 2 | 0 | 3.608 |
| **3** | CS-97004 | 9 | 7 | 10 | 9 | 9 | 2 | 10 | 8 | 9 | ... | 10 | 6 | 10 | 8 | 5 | 7 | 7 | 7 | 7 | 1.906 |
| **4** | CS-97005 | 2 | 2 | 2 | 4 | 0 | 0 | 2 | 4 | 0 | ... | 5 | 4 | 4 | 5 | 2 | 2 | 2 | 2 | 0 | 3.448 |

5 rows × 43 columns

In [158]:
```
df.dtypes
```

Out[158]:
```
Seat No.      object
PH-121         int64
HS-101         int64
CY-105         int64
HS-105/12      int64
MT-111         int64
CS-105         int64
CS-106         int64
EL-102         int64
EE-119         int64
ME-107         int64
CS-107         int64
HS-205/20      int64
MT-222         int64
EE-222         int64
MT-224         int64
CS-210         int64
CS-211         int64
CS-203         int64
CS-214         int64
EE-217         int64
CS-212         int64
CS-215         int64
MT-331         int64
EF-303         int64
HS-304         int64
CS-301         int64
CS-302         int64
TC-383         int64
MT-442         int64
EL-332         int64
CS-318         int64
CS-306         int64
CS-312         int64
CS-317         int64
CS-403         int64
CS-421         int64
CS-406         int64
CS-414         int32
CS-419         int32
CS-423         int64
CS-412         int64
CGPA         float64
dtype: object
```

In [159]:
```
#  here above we can see that our all 'object' columns are converted into 'int64'
#  and they are encoded succesfully.
```

CHECKING FOR OUTLIERS

==========

In [161]:
```
df.describe()
```

Out[161]:

| | PH-121 | HS-101 | CY-105 | HS-105/12 | MT-111 | CS-105 | CS-106 | EL-102 | EE-119 | ME-107 | ... | CS-312 | CS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 571.000000 | 571.000000 | 571.000000 | 571.000000 | 571.000000 | 571.000000 | 571.000000 | 571.000000 | 571.000000 | 571.000000 | ... | 571.000000 | 571.00 |
| **mean** | 3.781086 | 5.071804 | 2.898424 | 4.241681 | 3.896673 | 2.838879 | 4.122592 | 3.959720 | 3.886165 | 4.779335 | ... | 4.071804 | 4.83 |
| **std** | 3.046895 | 2.785317 | 2.964737 | 3.200507 | 2.988546 | 2.696709 | 2.727192 | 3.031436 | 2.657528 | 3.146202 | ... | 3.362345 | 2.74 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.00 |
| **25%** | 2.000000 | 3.000000 | 0.000000 | 2.000000 | 2.000000 | 0.000000 | 2.000000 | 2.000000 | 2.000000 | 2.000000 | ... | 1.000000 | 3.00 |
| **50%** | 3.000000 | 5.000000 | 2.000000 | 4.000000 | 4.000000 | 2.000000 | 4.000000 | 3.000000 | 3.000000 | 5.000000 | ... | 3.000000 | 5.00 |
| **75%** | 6.000000 | 7.000000 | 4.500000 | 7.000000 | 6.000000 | 4.000000 | 5.000000 | 6.000000 | 6.000000 | 8.000000 | ... | 7.000000 | 7.00 |
| **max** | 12.000000 | 11.000000 | 12.000000 | 12.000000 | 12.000000 | 10.000000 | 12.000000 | 12.000000 | 11.000000 | 12.000000 | ... | 13.000000 | 11.00 |

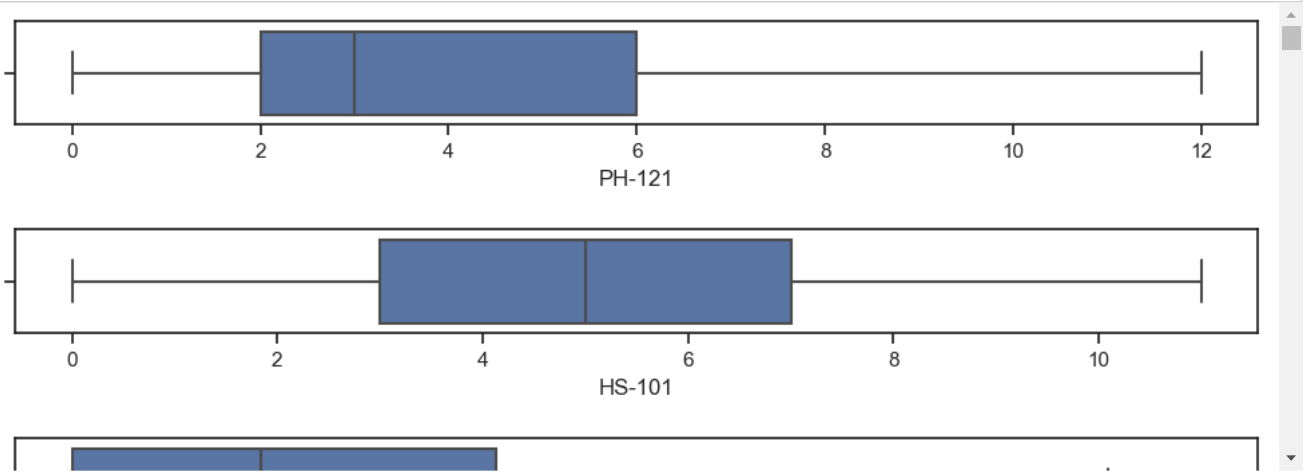
8 rows × 42 columns

In [162]:
```
df.columns
```

Out[162]:
```
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')
```

In [168]:
```
for i in df.columns[1:42]:
    plt.figure (figsize = (12,1), facecolor = "white")
    sns.boxplot(x=i,data=df)
    plt.show()

#     here below we can find the outliers for all the cloumns by using boxplot.
#  and we are found outliers in :
#  CY-105, CS-106, HS-205/20, EE-222, CS-210, EE-217, HS-304, CS-302, MT-442, CS-406, CS-414, CS-419, CS-412
# so out of 41 columns we found OUTLIERS IN 13 COLUMNS , now we have to remove those outliers from out dataset.
```




In [ ]:
```

```

REMOVING IRRELEVANT COLUMNS

==========

In [176]:
```
df.head(1)
```

Out[176]:

| | 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 | 5 | 10 | 8 | 6 | 8 | 10 | 9 | 8 | 5 | ... | 8 | 8 | 8 | 8 | 2 | 8 | 8 | 3 | 2 | 2.205 |

1 rows × 43 columns

In [177]:
```
# here we can see that out of all the above columns, ' seat no.' is not a relevant for our preduction,
#  so we need to drop this columns.
```

In [178]:
```
df.drop(['Seat No.'], axis = 1, inplace = True)
```

In [180]:
```
df.head(1)
#  now here we are succefully rmeoved 'seat no.' column from our dataset
```

Out[180]:

| | 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 | 8 | 8 | 3 | 2 | 2.205 |

1 rows × 42 columns

In [ ]:
```

```

Removing Of OutLiers by applyin Z-Score Method

==========

In [ ]:
```
#  we can not remove outliers from out TARGET COLUMN, so first we have to seprate target column first.
# For this first we need to identify the ZSCORE VALUES, for which we have to import some libraries.
```

In [173]:
```
from scipy.stats import zscore
```

In [181]:
```
z = np.abs(zscore(df))
z.head(5)

# by applying 'abs' (absolute method), we are getting all the entries whose z-score value is positive side
#  Ideally we can call the OUTLIERS whos ZSCORE VALUE is LESS THEN 3 AND MORE THEN 3
# so we have to remove all the data whose ZSCORE >3 & <3
#  below here we apllying "abs" i.e absolute method it returns us the all zscore values greater then 3
#  so we just need to remove lesserr then 3 zscore values.
```

Out[181]:

| | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.400402 | 1.770900 | 1.722261 | 0.549869 | 1.374222 | 2.657832 | 1.790004 | 1.333963 | 0.419492 | 1.024565 | ... | 1.169315 | 1.155824 | 1.533685 | 0.958939 | 0.39 |
| **1** | 1.242052 | 1.411559 | 2.397448 | 1.488043 | 0.369509 | 1.173242 | 1.790004 | 1.307365 | 2.302589 | 1.342687 | ... | 1.764660 | 1.521056 | 0.862110 | 1.325686 | 0.39 |
| **2** | 1.242052 | 0.744483 | 0.978490 | 0.237145 | 0.034605 | 1.053643 | 0.322008 | 0.013299 | 0.710366 | 0.884168 | ... | 0.319046 | 0.670339 | 1.152616 | 0.225444 | 1.21 |
| **3** | 1.714365 | 0.692879 | 2.397448 | 1.488043 | 1.709126 | 0.311348 | 2.157003 | 1.333963 | 1.925970 | 0.706443 | ... | 1.764660 | 0.425358 | 2.205260 | 0.958939 | 0.81 |
| **4** | 0.585070 | 1.103823 | 0.303302 | 0.075580 | 1.305012 | 1.053643 | 0.778989 | 0.013299 | 1.463605 | 0.884168 | ... | 0.276298 | 0.305107 | 0.190534 | 0.141304 | 0.39 |

5 rows × 42 columns

In [182]:
```
threshold = 3
print(np.where(z>3))
```

```
(array([ 60,  60,  60,  91, 137, 143, 143, 281, 281, 281, 288, 288, 340,
       352, 387, 432, 446, 447, 453, 453, 454, 454, 454, 509, 513, 516,
       516, 521, 522, 522, 527, 543, 543, 550, 563, 563, 565, 570],
      dtype=int64), array([ 2, 11, 19, 15,  2,  2, 41, 11, 15, 41, 11, 41, 24, 24, 40, 40, 36,
       36, 36, 40, 36, 37, 38, 36, 40, 36, 40, 36, 36, 40, 40, 36, 40, 40,
       37, 38, 41, 36], dtype=int64))
```

In [ ]:
```
# here above we found 35 those values whose z-score is more then > 3
# i.e means we are having 35 outlier still present in our dataset, and we have to remove those outliers
```

In [183]:
```
df_new = df[(z<3).all(axis=1)]
df_new.shape
df_new
```

Out[183]:

| | 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 | 8 | 8 | 3 | 2 | 2.205 |
| **1** | 0 | 9 | 10 | 9 | 5 | 6 | 9 | 0 | 10 | 9 | ... | 10 | 9 | 6 | 9 | 2 | 6 | 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 | 7 | 7 | 7 | 7 | 1.906 |
| **4** | 2 | 2 | 2 | 4 | 0 | 0 | 2 | 4 | 0 | 2 | ... | 5 | 4 | 4 | 5 | 2 | 2 | 2 | 2 | 0 | 3.448 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **564** | 2 | 9 | 0 | 6 | 4 | 4 | 6 | 2 | 5 | 9 | ... | 3 | 7 | 4 | 10 | 6 | 3 | 3 | 6 | 5 | 2.607 |
| **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 | 2 | 2 | 6 | 2 | 3.772 |
| **568** | 3 | 0 | 2 | 4 | 0 | 0 | 0 | 0 | 0 | 3 | ... | 2 | 3 | 0 | 4 | 0 | 4 | 4 | 2 | 2 | 3.470 |
| **569** | 0 | 4 | 9 | 0 | 9 | 10 | 5 | 8 | 5 | 8 | ... | 9 | 3 | 3 | 8 | 9 | 3 | 3 | 5 | 6 | 2.193 |

546 rows × 42 columns

In [184]:
```
df_new.shape
```

Out[184]: (546, 42)

In [185]:
```
df.shape
```

Out[185]: (571, 42)

In [188]:
```
# here you can see our rows are reduced from 571-546, that means 25 Outliers are removed from our dataset.
```

In [ ]:
```
```

CHECKING REMOVAL OF OUTLIERS BY BOXPLOT (COMPARING 'df' & 'df_new')

==========

In [190]:
```
df_new.columns
```

Out[190]:
```
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')
```

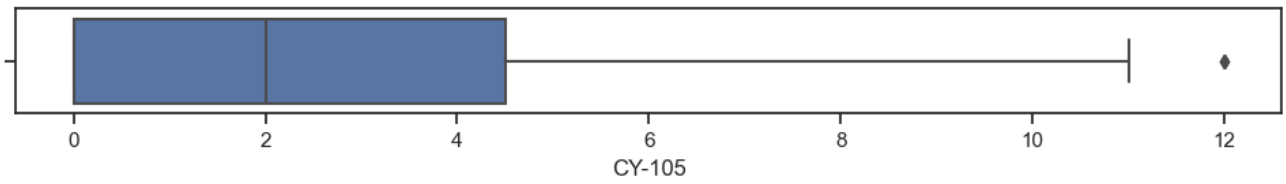
In [191]:
```
# Here as above we was founded outliers in the below 13 columns,
#  so now we are comparing those 13 columns, before removing & after removing of outliers.
#  CY-105, CS-106, HS-205/20, EE-222, CS-210, EE-217, HS-304, CS-302, MT-442, CS-406, CS-414, CS-419, CS-412
```

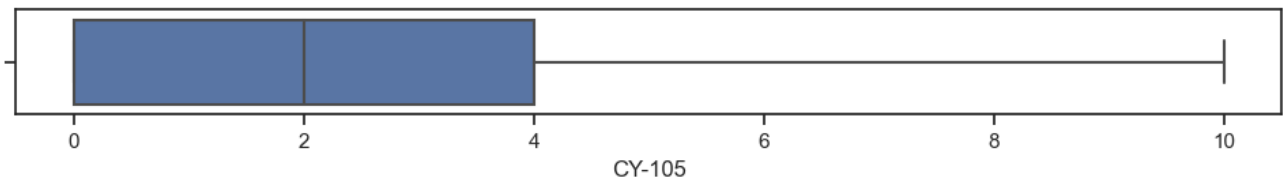
In [197]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='CY-105',data=df)
plt.show()

# it is the EARLIER (df dataset) PRESENCE OF OUTLIERS

plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='CY-105',data=df_new)
plt.show()

# outliers are succesfully removed.
# it is the EARLIER (df dataset) PRESENCE OF OUTLIERS
```



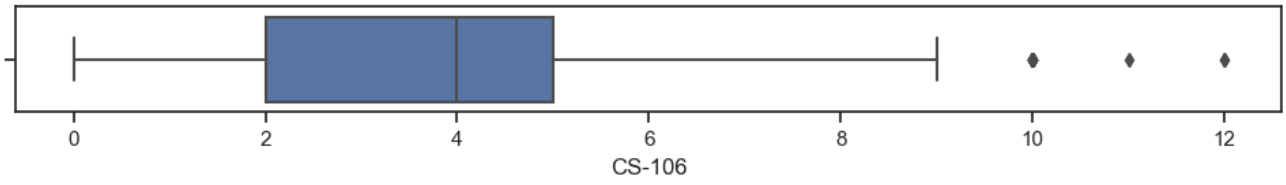



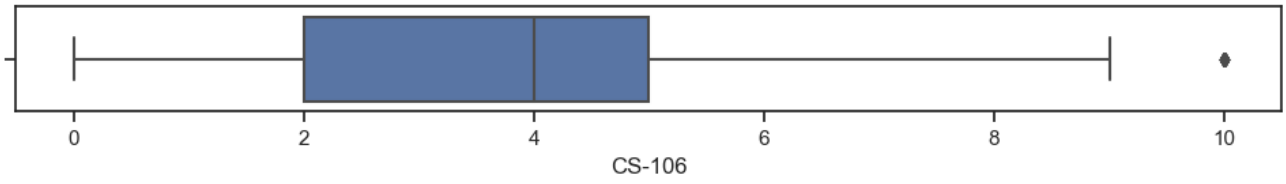
In [196]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='CS-106',data=df)
plt.show()

# it is the EARLIER (df dataset) PRESENCE OF OUTLIERS

plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='CS-106',data=df_new)
plt.show()

# outliers are succesfully removed.
```



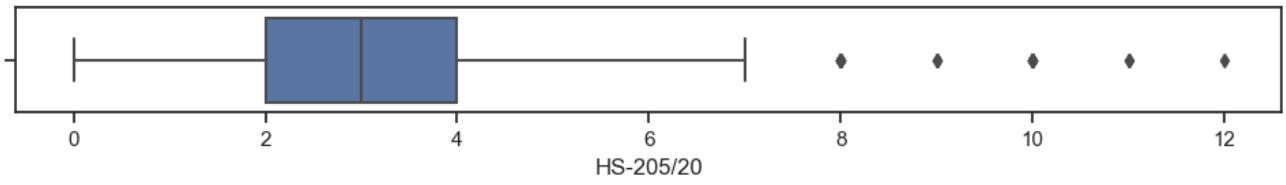



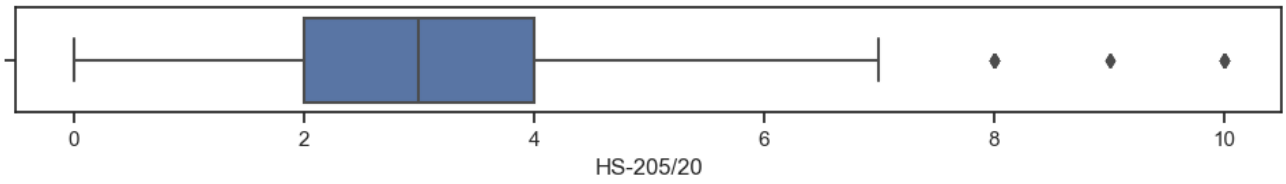

In [198]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='HS-205/20',data=df)
plt.show()

# it is the EARLIER (df dataset) PRESENCE OF OUTLIERS

plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='HS-205/20',data=df_new)
plt.show()

# outliers are succesfully removed.
```



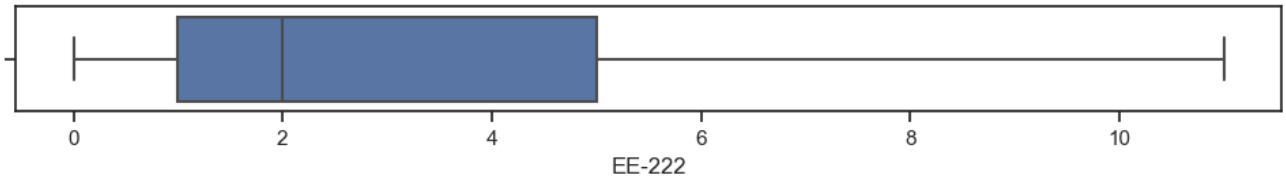



In [199]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='EE-222',data=df)
plt.show()

# it is the EARLIER (df dataset) PRESENCE OF OUTLIERS

plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='EE-222',data=df_new)
plt.show()

# outliers are succesfully removed.
```

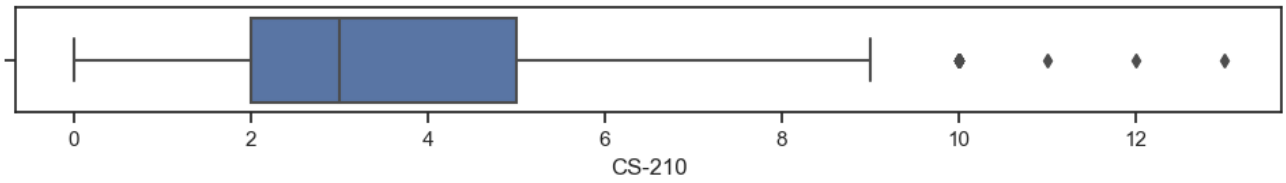



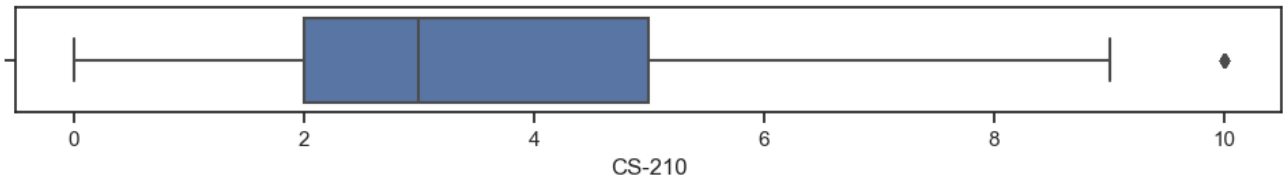

In [200]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='CS-210',data=df)
plt.show()

# it is the EARLIER (df dataset) PRESENCE OF OUTLIERS

plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='CS-210',data=df_new)
plt.show()

# outliers are succesfully removed.
```



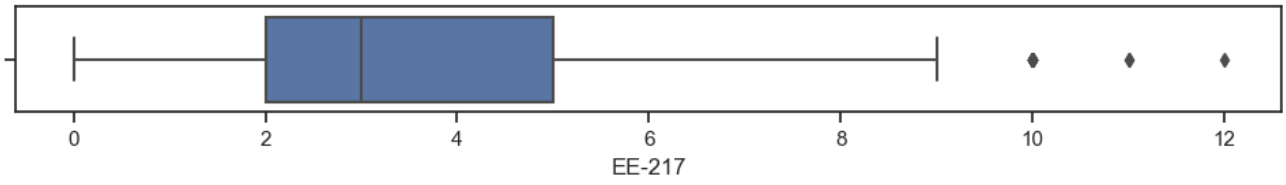



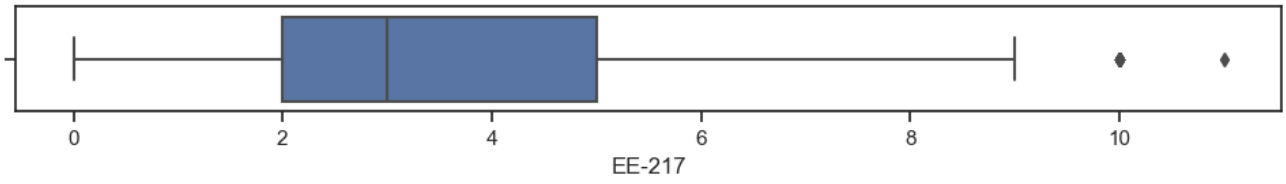
In [201]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='EE-217',data=df)
plt.show()

# it is the EARLIER (df dataset) PRESENCE OF OUTLIERS

plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='EE-217',data=df_new)
plt.show()

# outliers are succesfully removed.
```



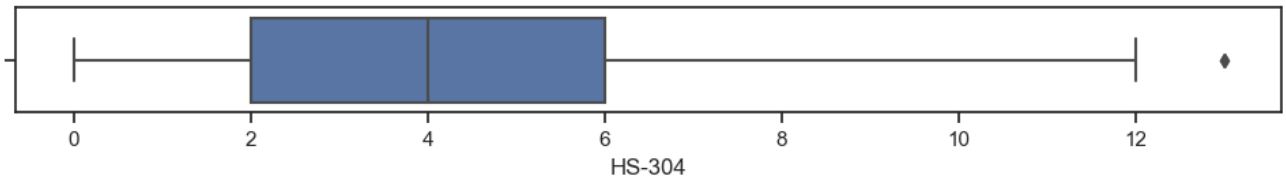



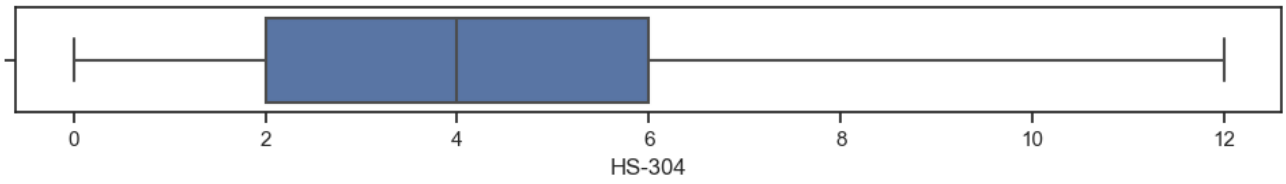

In [202]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='HS-304',data=df)
plt.show()

# it is the EARLIER (df dataset) PRESENCE OF OUTLIERS

plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='HS-304',data=df_new)
plt.show()

# outliers are succesfully removed.
```



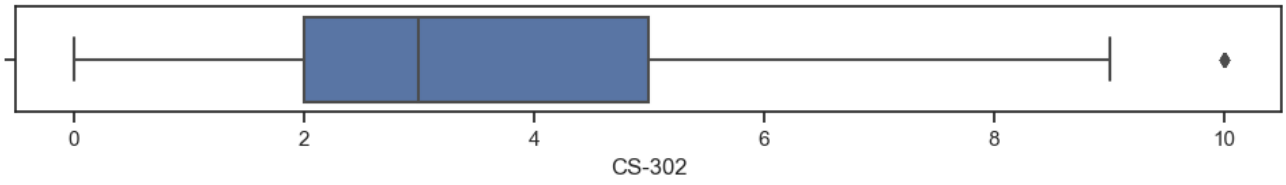



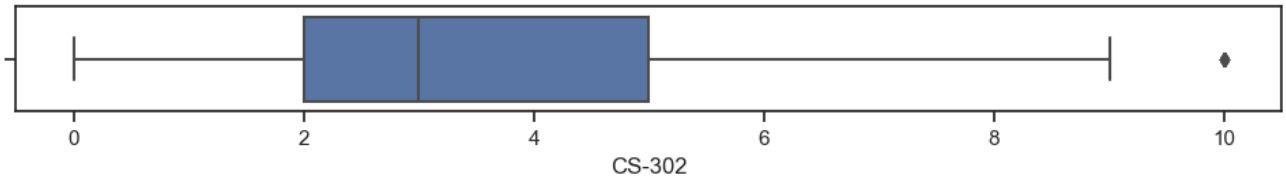
In [203]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='CS-302',data=df)
plt.show()

# it is the EARLIER (df dataset) PRESENCE OF OUTLIERS

plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='CS-302',data=df_new)
plt.show()

# here the outliers are very nearby position so may they can't removed.
```



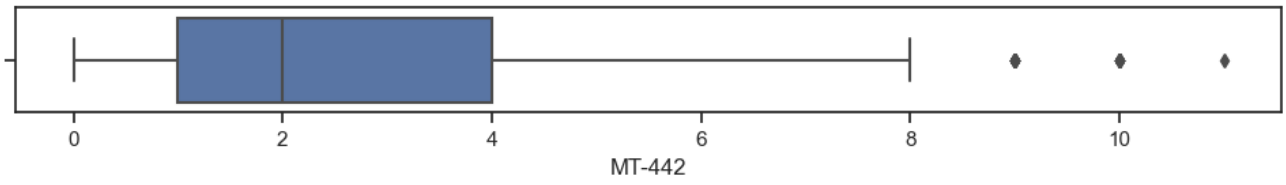



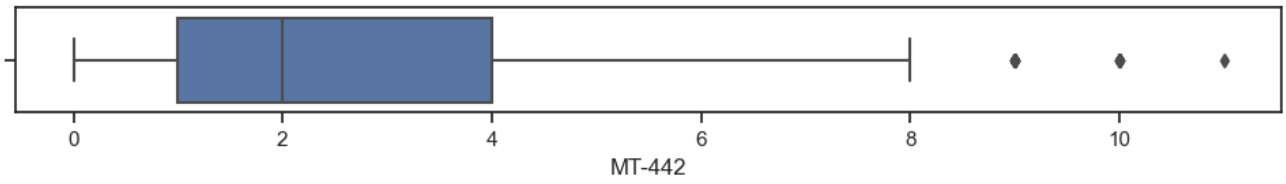

In [204]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='MT-442',data=df)
plt.show()

# it is the EARLIER (df dataset) PRESENCE OF OUTLIERS

plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='MT-442',data=df_new)
plt.show()

# here the outliers are very nearby position so may they can't removed.
```



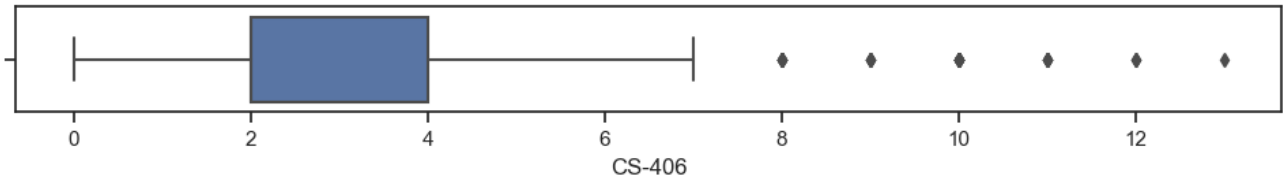



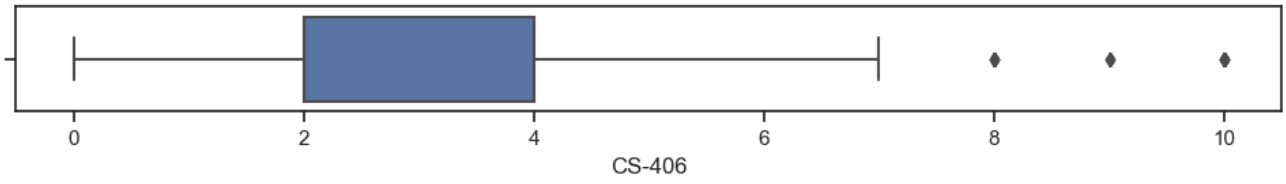
In [205]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='CS-406',data=df)
plt.show()

# it is the EARLIER (df dataset) PRESENCE OF OUTLIERS

plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='CS-406',data=df_new)
plt.show()

# outliers are succesfully removed.
```



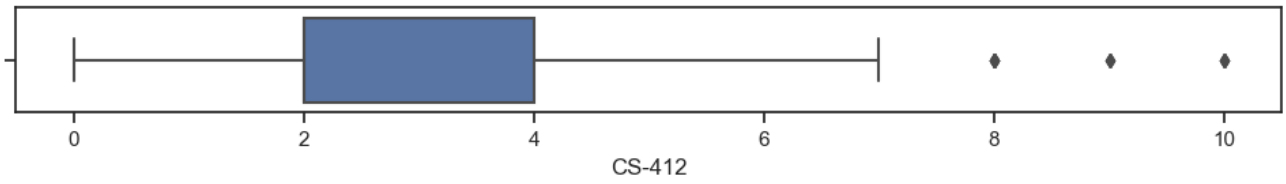




In [206]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='CS-412',data=df)
plt.show()

# it is the EARLIER (df dataset) PRESENCE OF OUTLIERS

plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='CS-412',data=df_new)
plt.show()

# outliers are succesfully removed.
```

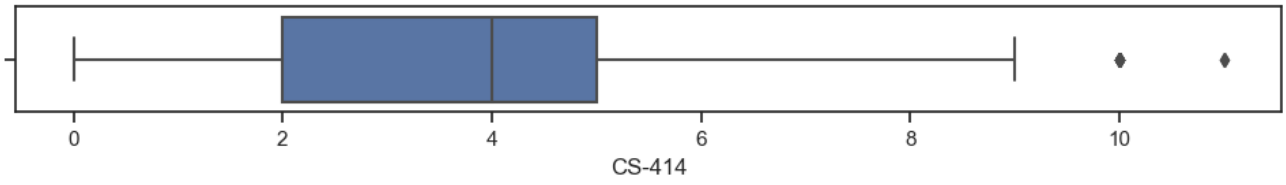



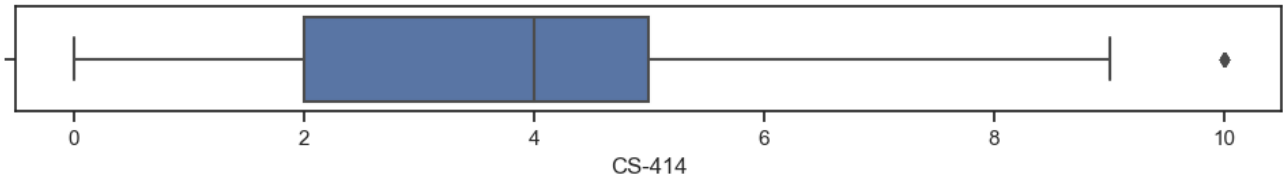
In [207]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='CS-414',data=df)
plt.show()

# it is the EARLIER (df dataset) PRESENCE OF OUTLIERS

plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='CS-414',data=df_new)
plt.show()

# outliers are succesfully removed.
```



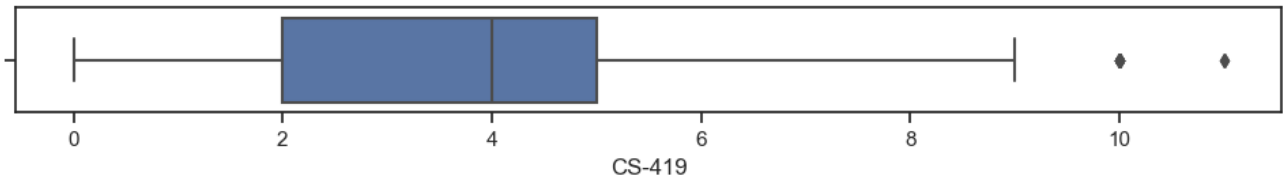



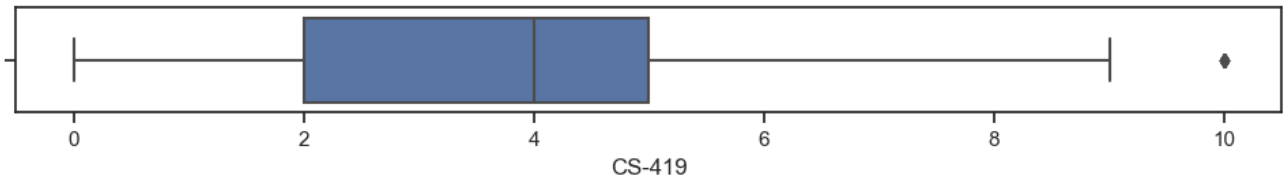

In [208]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='CS-419',data=df)
plt.show()

# it is the EARLIER (df dataset) PRESENCE OF OUTLIERS

plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='CS-419',data=df_new)
plt.show()

# outliers are succesfully removed.
```




In [209]:
```
df_new.head(5)
```

Out[209]:

| | 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 | 8 | 8 | 3 | 2 | 2.205 |
| **1** | 0 | 9 | 10 | 9 | 5 | 6 | 9 | 0 | 10 | 9 | ... | 10 | 9 | 6 | 9 | 2 | 6 | 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 | 7 | 7 | 7 | 7 | 1.906 |
| **4** | 2 | 2 | 2 | 4 | 0 | 0 | 2 | 4 | 0 | 2 | ... | 5 | 4 | 4 | 5 | 2 | 2 | 2 | 2 | 0 | 3.448 |

5 rows × 42 columns

In [ ]:
```
```

CHECKING SKEWNESS

==========>>>

In [211]:
```
# the skewness shows the distribution of data, if the data is widely skewed that means it is not good for our model.
#  ideal range of skewness is ( -0.5 to +0.5)
# We can't remove skewness from our Target Column
```

In [213]:
```
df_new.skew()
#  here we can't see skewness in our dataset.
```

Out[213]:
```
PH-121       0.567643
HS-101       0.067677
CY-105       0.949907
HS-105/12    0.332294
MT-111       0.464443
CS-105       0.959824
CS-106       0.619676
EL-102       0.509105
EE-119       0.606672
ME-107       0.119195
CS-107       0.412356
HS-205/20    0.818763
MT-222       0.188242
EE-222       0.739285
MT-224       0.417702
CS-210       0.667224
CS-211       0.086421
CS-203       0.278097
CS-214       0.030814
EE-217       0.723954
CS-212       0.408541
CS-215       0.139372
MT-331       0.649765
EF-303       0.357767
HS-304       0.451754
CS-301       0.424303
CS-302       0.614168
TC-383       0.357496
MT-442       0.871425
EL-332       0.450124
CS-318       0.333968
CS-306       0.417563
CS-312       0.431200
CS-317       0.130602
CS-403       0.598576
CS-421       0.117699
CS-406       1.328723
CS-414       0.452447
CS-419       0.452447
CS-423       0.495446
CS-412       0.906402
CGPA        -0.379481
dtype: float64
```

In [ ]:
```

```

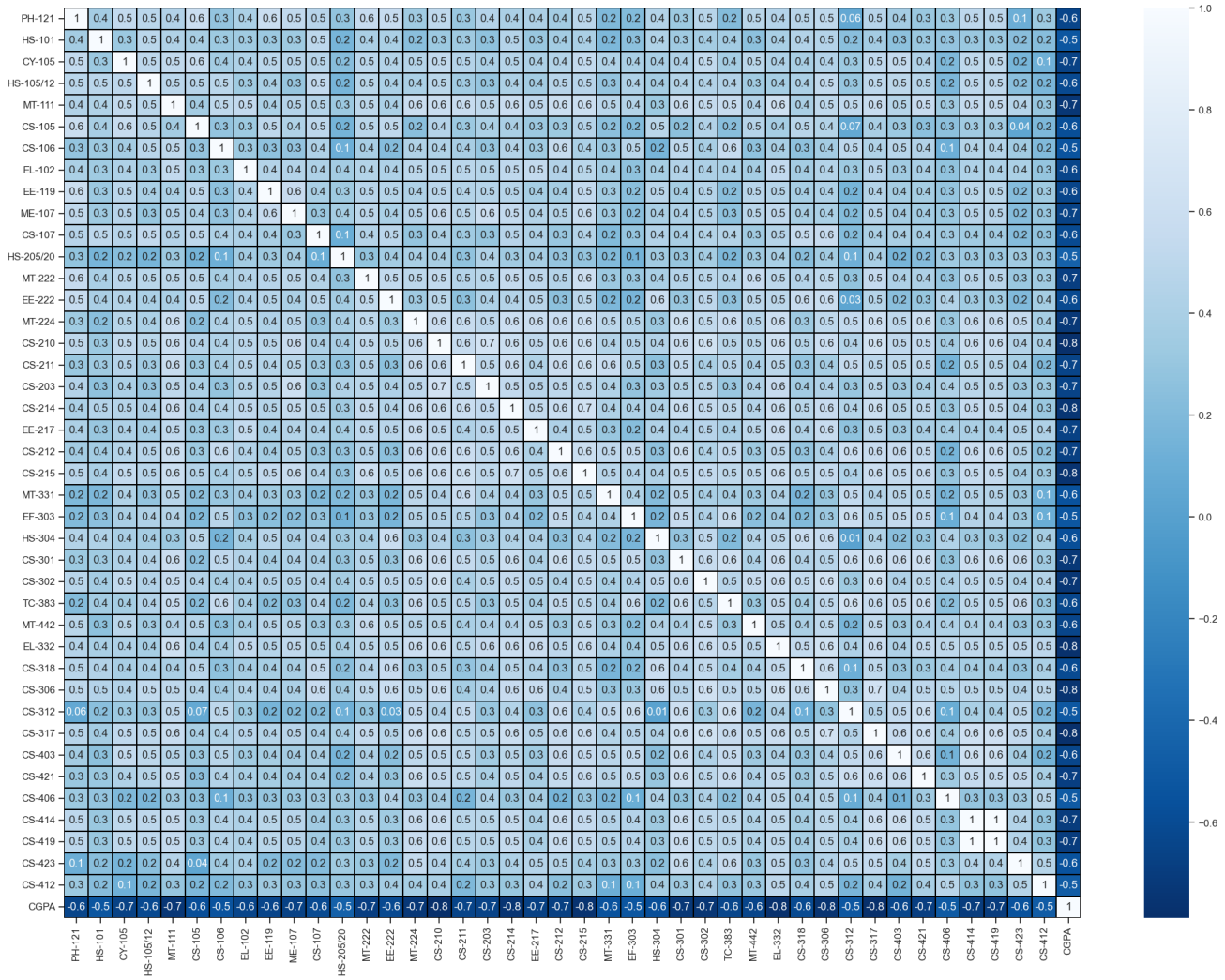
CHECKING CORRELATION (GRAPHICALLY)

==========

In [215]:
```
cor = df_new.corr()
```

In [218]:
```
plt.figure (figsize = (25,18), facecolor = "white")
sns.heatmap(df_new.corr(),linewidth=0.1,fmt="0.1g",linecolor="black",annot=True,cmap="Blues_r")
plt.yticks(rotation=0);
plt.show()

# here due to huge number of column it is difficult to identifie in this way, so we have to go for another way.
```

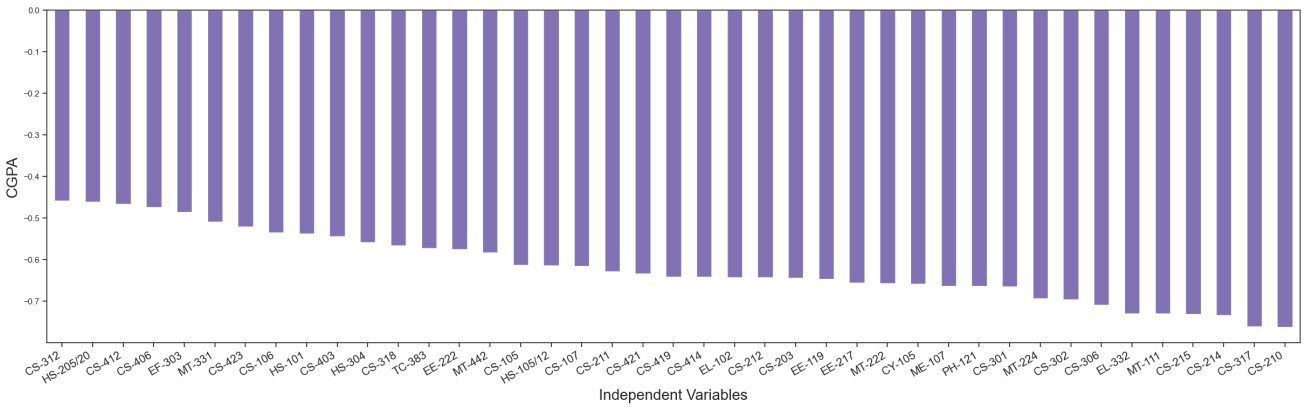


In [220]:
```
cor['CGPA'].sort_values(ascending=False)
# here we can see in the correltion of all independent vaules with Target Column = 'CGPA'
#  there no such any POITIVE correation with target column.
# even all columns are showing NEGATIVE CORRELATION with the TARGET COLUMN.
```

Out[220]:
```
CGPA         1.000000
HS-205/20   -0.451955
CS-412      -0.481427
CS-406      -0.494620
CS-312      -0.516041
EF-303      -0.523902
HS-101      -0.525742
CS-106      -0.543536
CS-423      -0.552054
MT-331      -0.556123
CS-105      -0.579571
HS-304      -0.581825
CS-318      -0.610487
CS-107      -0.612013
HS-105/12   -0.614845
EE-222      -0.615017
CS-403      -0.637139
TC-383      -0.637715
EL-102      -0.641966
PH-121      -0.642060
MT-442      -0.643022
EE-119      -0.646466
CS-211      -0.650517
CS-203      -0.655440
ME-107      -0.663497
CY-105      -0.664564
EE-217      -0.669594
CS-212      -0.679448
CS-421      -0.684063
MT-222      -0.684081
CS-414      -0.701111
CS-419      -0.701111
MT-224      -0.724888
CS-301      -0.726766
MT-111      -0.737637
CS-302      -0.743127
CS-214      -0.752014
CS-306      -0.755660
CS-215      -0.762717
EL-332      -0.776895
CS-317      -0.784281
CS-210      -0.787616
Name: CGPA, dtype: float64
```

In [221]:
```
plt.figure(figsize=(30,8))
df.corr()['CGPA'].sort_values(ascending=False).drop(['CGPA']).plot(kind='bar',color="m")
plt.xlabel('Independent Variables',fontsize=20)
plt.xticks(rotation=30,ha='right',fontsize=15)
plt.ylabel('CGPA',fontsize =20)
plt.title=("Correlation with CGPA")
plt.show()

# here we can see that  neagativly correlated with the TARGET COLUMN.
```




In [ ]:
```
```

DIVIDING DATA INTO INDEPENDENT & TARGET VARIABLE

==========

In [223]:
```
df_new.columns
```

Out[223]:
```
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 [224]:
```
x = df_new[['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']]
```

In [225]:
```
y = df_new[['CGPA']]
```

In [226]:
```
x.shape
```

Out[226]:
```
(546, 41)
```

In [227]:
```
y.shape
```

Out[227]:
```
(546, 1)
```

In [ ]:
```

```

APPLYING SCALING TECHNIQUES

==========>

In [ ]:
```
# here we need to apply scaling techniques on our dataset,by scaling techniques we normalise the values.
#  we can't apply SCALING TECHNIQUES on TARGET VARIABLE
#  to aplly scaling techinuque we need to import some libraries first.
```

In [229]:
```
from sklearn.preprocessing import StandardScaler
```

In [230]:
```
st = StandardScaler()
```

In [231]:
```
x = st.fit_transform(x)
x
```

Out[231]:
```
array([[ 0.44983947,  1.83764119,  1.81838796, ...,  1.79064736,
         -0.32455924, -0.42352816],
       [-1.22555175,  1.47158789,  2.51929853, ...,  0.93639357,
          0.7738458 ,  0.08027658],
       [-1.22555175, -0.7247319 , -0.98525433, ..., -1.62636778,
         -0.69069426, -1.43113764],
       ...,
       [-0.8904735 , -1.82289179, -0.98525433, ..., -0.772114  ,
          0.7738458 , -0.42352816],
       [-0.22031701, -1.82289179, -0.28434376, ...,  0.08213979,
         -0.69069426, -0.42352816],
       [-1.22555175, -0.3586786 ,  2.16884324, ..., -0.34498711,
          0.40771079,  1.5916908 ]])
```

In [232]:
```
xf = pd.DataFrame(data=x)
print(xf)

# here we get our dataset (xf) after applying SCALING TECHING (STANDARD SCALER)
```

```
           0         1         2         3         4         5         6   \
0    0.449839  1.837641  1.818388  0.593890  1.457448  2.801383  1.860067
1   -1.225552  1.471588  2.519299  1.547608  0.423985  1.261821  1.860067
2   -1.225552 -0.724732 -0.985254  0.275984  0.079497 -1.047523  0.362137
3    1.790152  0.739481  2.519299  1.547608  1.801936 -0.277742  2.234549
4   -0.555395 -1.090785 -0.284344 -0.041922 -1.298454 -1.047523 -0.761310
..        ...       ...       ...       ...       ...       ...       ...
541 -0.555395  1.471588 -0.985254  0.593890  0.079497  0.492040  0.736619
542 -0.220317 -1.822892 -0.985254 -0.677734 -0.953966 -1.047523 -0.761310
543 -0.890474 -1.822892 -0.985254 -1.313546 -1.298454 -1.047523 -1.510275
544 -0.220317 -1.822892 -0.284344 -0.041922 -1.298454 -1.047523 -1.510275
545 -1.225552 -0.358679  2.168843 -1.313546  1.801936  2.801383  0.362137

           7         8         9   ...        31        32        33  \
0    1.401321  0.460722  1.073142  ...  0.727333  1.190669  1.218911
1   -1.296330  2.375137  1.394908  ...  1.812370  1.791784  1.590091
2    0.052495 -0.687927 -0.857453  ... -0.719384 -0.312117 -0.636988
3    1.401321  1.992254  0.751376  ...  1.450691  1.791784  0.476551
4    0.052495 -1.453693 -0.857453  ... -0.719384  0.288997 -0.265808
..        ...       ...       ...  ...       ...       ...       ...
541 -0.621917  0.460722  1.394908  ...  1.450691 -0.312117  0.847731
542 -0.621917 -1.070810 -0.213921  ...  0.003974 -0.612674 -1.008168
543 -0.621917 -1.453693 -1.500985  ... -0.719384 -0.011560 -0.265808
544 -1.296330 -1.453693 -0.535687  ...  0.003974 -0.612674 -0.636988
545  1.401321  0.460722  1.073142  ...  0.003974  1.491226 -0.636988

           34        35        36        37        38        39        40
0    1.510984  1.034288 -0.365876  1.790647  1.790647 -0.324559 -0.423528
1    0.842428  1.413806 -0.365876  0.936394  0.936394  0.773846  0.080277
2   -1.163237  0.275254 -1.286468 -1.626368 -1.626368 -0.690694 -1.431138
3    2.179539  1.034288  1.015011  1.363520  1.363520  1.139981  2.095496
4    0.173873 -0.104263 -0.365876 -0.772114 -0.772114 -0.690694 -1.431138
..        ...       ...       ...       ...       ...       ...       ...
541  0.173873  1.793323  1.475307 -0.344987 -0.344987  0.773846  1.087886
542 -1.163237 -2.001849 -1.286468  0.082140  0.082140 -0.324559 -1.431138
543 -1.163237 -2.001849 -0.365876 -0.772114 -0.772114  0.773846 -0.423528
544 -1.163237 -0.483780 -1.286468  0.082140  0.082140 -0.690694 -0.423528
545 -0.160404  1.034288  2.856194 -0.344987 -0.344987  0.407711  1.591691

[546 rows x 41 columns]
```

In [233]:
```
xf.columns
```

Out[233]:
```
RangeIndex(start=0, stop=41, step=1)
```

In [234]:
```
df_new.columns
```

Out[234]:
```
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 [235]:
```
colum = ['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']
```

In [236]:
```
xf.columns=colum
```

In [237]:
```
xf.head(2)
```

Out[237]:

| | 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 | CS-403 | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.449839 | 1.837641 | 1.818388 | 0.593890 | 1.457448 | 2.801383 | 1.860067 | 1.401321 | 0.460722 | 1.073142 | ... | 0.727333 | 1.190669 | 1.218911 | 1.510984 | 1.0 |
| **1** | -1.225552 | 1.471588 | 2.519299 | 1.547608 | 0.423985 | 1.261821 | 1.860067 | -1.296330 | 2.375137 | 1.394908 | ... | 1.812370 | 1.791784 | 1.590091 | 0.842428 | 1.4 |

2 rows × 41 columns

In [238]:
```
# similarly for target column.
```

In [239]:
```
yf=y
```

In [240]:
```
yf.head(2)
```

Out[240]:

| | CGPA |
|---|---|
| 0 | 2.205 |
| 1 | 2.008 |

FINDING MULTICOLINEARITY

==========

In [ ]:
```
# We have to find the multicollinearity between the features and to remove it we can use VIF (VARIANCE INFLATION FACTOR)
# we can not apply VIF on the TARGET COLUMN
# for apllyin VIF we have to import some libraries as follows
```

In [243]:
```
import statsmodels.api as sm
from scipy import stats
from statsmodels .stats.outliers_influence import variance_inflation_factor
```

In [244]:
```
# here we are making "def function" for calculating VIF
def calc_vif(xf):
    vif = pd.DataFrame()
    vif["FETURES"] = xf.columns
    vif["VIF FACTOR"] = [variance_inflation_factor(xf.values,i) for i in range (xf.shape[1])]
    return (vif)
```

In [245]:
```
xf.shape
```

Out[245]: (546, 41)

In [246]:
```
yf.shape
```

Out[246]: (546, 1)

In [249]:
```
calc_vif(xf)
# here we didn't find MULTICOLINEARITY between the independent Columns.
```

Out[249]:

| | FETURES | VIF FACTOR |
|---|---|---|
| 0 | PH-121 | 2.736378 |
| 1 | HS-101 | 1.844670 |
| 2 | CY-105 | 2.496045 |
| 3 | HS-105/12 | 2.379461 |
| 4 | MT-111 | 2.659041 |
| 5 | CS-105 | 2.339163 |
| 6 | CS-106 | 1.972390 |
| 7 | EL-102 | 1.917454 |
| 8 | EE-119 | 2.198283 |
| 9 | ME-107 | 2.315175 |
| 10 | CS-107 | 2.392629 |
| 11 | HS-205/20 | 1.581450 |
| 12 | MT-222 | 2.305743 |
| 13 | EE-222 | 2.393286 |
| 14 | MT-224 | 2.933404 |
| 15 | CS-210 | 3.146274 |
| 16 | CS-211 | 2.563989 |
| 17 | CS-203 | 2.550046 |
| 18 | CS-214 | 2.720520 |
| 19 | EE-217 | 2.364243 |
| 20 | CS-212 | 2.875367 |
| 21 | CS-215 | 2.988891 |
| 22 | MT-331 | 2.040982 |
| 23 | EF-303 | 2.110097 |
| 24 | HS-304 | 2.349188 |
| 25 | CS-301 | 3.105948 |
| 26 | CS-302 | 2.753152 |
| 27 | TC-383 | 2.999340 |
| 28 | MT-442 | 2.169373 |
| 29 | EL-332 | 3.086319 |
| 30 | CS-318 | 2.330000 |
| 31 | CS-306 | 3.200799 |
| 32 | CS-312 | 3.522723 |
| 33 | CS-317 | 3.139042 |
| 34 | CS-403 | 2.568177 |
| 35 | CS-421 | 2.673457 |
| 36 | CS-406 | 1.908708 |
| 37 | CS-414 | inf |
| 38 | CS-419 | inf |
| 39 | CS-423 | 2.493603 |
| 40 | CS-412 | 1.976911 |

In [ ]:
```
# here we can see that the highest VIF values are 14.95 & 10.43 for 'magnesium' & 'calcium'
#  we can drop 'magnesium' & 'calcium' column
#  but before droping those column, we need to chek the correlation of the column with the "TARGET COLUMN"
```

========== NOW WE NEED TO APPLY ML MODELS
==========

APPLYING TRAIN TEST SPLIT==========

In [260]:
```
# here we can see that out Target Column - 'CGPA' is NOT CATEGORICAL column.
# NOW HERE WE CAN SEE THAT OUR TARGET/LABEL COLUMN IN NOT A CATEGORICAL DATA, IT IS HAVING FLOATING DATA,
# AND WHEN WE ARE HAVING "Y" (TARGET) IN DECIMAL FORM THEN WE CAN APPLY "REGRESSION MODEL",
# SO HERE WE CAN APPLY REGRESSION MODEL ON OUR DATASET TO PREDICT, "HAPPINESS SCORE".
```

In [259]:
```
from sklearn.model_selection import train_test_split
```

In [261]:
```
from sklearn.linear_model import LinearRegression
```

In [262]:
```
lr = LinearRegression()
```

In [263]:
```
 x_train,x_test,y_train,y_test = train_test_split(xf,yf,test_size=0.20,random_state=42)
```

In [264]:
```
lr.fit(x_train,y_train)
y_pred = lr.predict(x_test)
y_test.head(),y_pred[0:4]
```

Out[264]:
```
(      CGPA
 322  1.886
 78   3.193
 368  3.437
 92   1.890
 510  3.240,
 array([[2.07098101],
        [3.22821075],
        [3.42701625],
        [1.90546095]]))
```

In [265]:
```
#  here above we can see the similarity between "actual values" and "predicted values"
```

In [266]:
```
from sklearn.metrics import mean_squared_error
```

In [267]:
```
mean_squared_error (y_test,y_pred)
```

Out[267]:
```
0.005078610684100649
```

In [268]:
```
# as we can see the mean squared error is very low that means our model working very good.
```

In [269]:
```
from sklearn.metrics import r2_score
```

In [270]:
```
r2_score(y_test,y_pred)
```

Out[270]:
```
0.9820481196097671
```

In [271]:
```
# r2 score is also very high.
```

In [272]:
```
xf.shape
```

Out[272]:
```
(546, 41)
```

In [274]:
```
def pred_func(r):
    r= r.reshape(1,41)
    rt = lr.predict(r)
    print(rt)

# making 'def' function to predict CGPA SCORE  of any STUDENT
```

In [278]:
```
r= np.array([0.449839,1.837641,1.818388,0.593890,1.457448,2.801383,1.860067,1.401321,0.460722,1.073142,0.727333,1.190669,1
pred_func(r)

# here we are giving values to the model, and the model is predicting the CGPA SCORE of the given values of a student.
```

```
[[2.67576193e+11]]
```

In [ ]:
```

```

SAVING THE MODEL

==========

In [281]:
```
import pickle
```

In [282]:
```
file_name = 'CGPA SCORE.pkl'
pickle.dump(lr,open(file_name,'wb'))
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: