# Student Grades Prediction

## Project Description

The dataset contains grades scored by students throughout their university tenure in various courses and their CGPA calculated based on their grades Columns Description- total 43 columns -Seat No : The enrolled number of candidate that took the exams

CGPA : The cumulative GPA based on the four year total grade progress of each candidate . CGPA is a Final Marks -- provided to student.

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

**Predict** - CGPA of a student based on different grades in four years.

## Importing Libraries

In [1]:
```
# Analyse and Manipulate Data
import numpy as np
import pandas as pd

# Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Preprocessor
from sklearn.preprocessing import StandardScaler

#Model Evaluation
from sklearn.model_selection import train_test_split, cross_val_score, GridSear

# Regressor
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor,
from xgboost import XGBRegressor

#Metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Model saving
import pickle

# Prevent WARNINGS!
import warnings
warnings.filterwarnings("ignore")
```

## Import & Analyse Data

In [2]:
```
df=pd.read_csv("E:\\dataset\\dataset4-main\\Grades.csv")
df
```

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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | CS-97001 | B- | D+ | C- | C | C- | D+ | D | C- | B- | ... | C- | C- | C- | C- | A- | A |
| **1** | CS-97002 | A | D | D+ | D | B- | C | D | A | D+ | ... | D+ | D | C | D | A- | B- |
| **2** | CS-97003 | A | B | A | B- | B+ | A | B- | B+ | A- | ... | B | B | A | C | A | A |
| **3** | CS-97004 | D | C+ | D+ | D | D | A- | D+ | C- | D | ... | D+ | C | D+ | C- | B- | B |
| **4** | CS-97005 | A- | A- | A- | B+ | A | A | A- | B+ | A | ... | B- | B+ | B+ | B- | A- | A |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **566** | CS-97567 | B | A | A | A- | A+ | A | A- | A- | A+ | ... | A- | A- | A | A | A | B+ |
| **567** | CS-97568 | A+ | A | A | A | A | A | A | A- | A | ... | B+ | B+ | A | A | A- | B |
| **568** | CS-97569 | B | A | A- | B+ | A | A | A | A | A | ... | A- | B | A | B+ | A | C |
| **569** | CS-97570 | A | B+ | D | A | D | D+ | B- | C- | B- | ... | D | B | B | C- | D | C |
| **570** | CS-97571 | C | D | D | C | C | D+ | B | C+ | C | ... | C+ | C | B- | D | F | C- |

571 rows × 43 columns

## Data Inspection

In [3]:
```
df.head()
```

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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | CS-97001 | B- | D+ | C- | C | C- | D+ | D | C- | B- | ... | C- | C- | C- | C- | A- | A |
| **1** | CS-97002 | A | D | D+ | D | B- | C | D | A | D+ | ... | D+ | D | C | D | A- | B- |
| **2** | CS-97003 | A | B | A | B- | B+ | A | B- | B+ | A- | ... | B | B | A | C | A | A |
| **3** | CS-97004 | D | C+ | D+ | D | D | A- | D+ | C- | D | ... | D+ | C | D+ | C- | B- | B |
| **4** | CS-97005 | A- | A- | A- | B+ | A | A | A- | B+ | A | ... | B- | B+ | B+ | B- | A- | A |

5 rows × 43 columns

In [4]:
```
df.tail()
```

Out[4]:

| | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **566** | CS-97567 | B | A | A | A- | A+ | A | A- | A- | A+ | ... | A- | A- | A | A | A | B+ |
| **567** | CS-97568 | A+ | A | A | A | A | A | A | A- | A | ... | B+ | B+ | A | A | A- | B |
| **568** | CS-97569 | B | A | A- | B+ | A | A | A | A | A | ... | A- | B | A | B+ | A | C |
| **569** | CS-97570 | A | B+ | D | A | D | D+ | B- | C- | B- | ... | D | B | B | C- | D | C |
| **570** | CS-97571 | C | D | D | C | C | D+ | B | C+ | C | ... | C+ | C | B- | D | F | C- |

5 rows × 43 columns

In [5]:
```
# Dimension of data

print("Dataset contain {0} rows & {1} columns".format(df.shape[0],df.shape[1]))
```

```
Dataset contain 571 rows & 43 columns
```

In [6]:
```
# Name of the columns
print("Columns/Variables we have in our dataset are:\n\n",df.columns)
```

```
Columns/Variables we have in our dataset are:

 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-205/2
0',
       '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 [7]:
```
#Data info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 571 entries, 0 to 570
Data columns (total 43 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Seat No.   571 non-null    object 
 1   PH-121     571 non-null    object 
 2   HS-101     571 non-null    object 
 3   CY-105     570 non-null    object 
 4   HS-105/12  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 [8]:
```
# Datatypes of the columns
df.dtypes
```

Out[8]:
```
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 [9]:
```
#renaming two columns for our convenience
df.rename(columns={'HS-105/12': 'HS-105', 'HS-205/20': 'HS-205'},inplace=True)
```

## Observations:

- We can clearly see that in the dataset there are 43 columns and 571 rows.

- Renamed two columns for our convenience
- We have two types of datatypes(object and float) present in out dataset.
- As we can clearly see that all the columns in our dataset has object values except CGPA column which is our target/label.
- Memory usage: 191.9+ KB

## Duplicate & Missing Values

In [10]:
```
#Checking for Duplicate Values
print("We have {} duplicated values in our dataframe".format(df.duplicated().su
```

```
We have 0 duplicated values in our dataframe
```

In [11]:
```
#checking for missing Values

#Checking Null Values
df.isnull().sum().sum()
```

Out[11]:
```
425
```

In [12]:
```
df.isnull().sum()
```

Out[12]:
```
Seat No.     0
PH-121       0
HS-101       0
CY-105       1
HS-105       1
MT-111       2
CS-105       0
CS-106       2
EL-102       2
EE-119       2
ME-107       2
CS-107       2
HS-205       5
MT-222       5
EE-222       7
MT-224       7
CS-210       7
CS-211       5
CS-203       5
CS-214       6
EE-217       6
CS-212       6
CS-215       6
MT-331       9
EF-303      10
HS-304      10
CS-301      10
CS-302      10
TC-383      10
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
```

#### Handling Missing Values

In [13]:
```python
def replace_null_with_mode(df):
    # Find columns with null values
    columns_with_null = df.columns[df.isnull().any()]

    # Iterate over columns with null values
    for column in columns_with_null:
        # Calculate mode
        mode_value = df[column].mode()[0]

        # Replace null values with mode
        df[column].fillna(mode_value, inplace=True)

    return df
```

In [14]:
```python
df= replace_null_with_mode(df)
df
```

Out[14]:

| | Seat No. | PH-121 | HS-101 | CY-105 | HS-105 | MT-111 | CS-105 | CS-106 | EL-102 | EE-119 | ... | CS-312 | CS-317 | CS-403 | CS-421 | CS-406 | CS-414 | CS-41 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | CS-97001 | B- | D+ | C- | C | C- | D+ | D | C- | B- | ... | C- | C- | C- | C- | A- | A | C |
| **1** | CS-97002 | A | D | D+ | D | B- | C | D | A | D+ | ... | D+ | D | C | D | A- | B- | |
| **2** | CS-97003 | A | B | A | B- | B+ | A | B- | B+ | A- | ... | B | B | A | C | A | A | |
| **3** | CS-97004 | D | C+ | D+ | D | D | A- | D+ | C- | D | ... | D+ | C | D+ | C- | B- | B | C |
| **4** | CS-97005 | A- | A- | A- | B+ | A | A | A- | B+ | A | ... | B- | B+ | B+ | B- | A- | A | A |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **566** | CS-97567 | B | A | A | A- | A+ | A | A- | A- | A+ | ... | A- | A- | A | A | A | B+ | B |
| **567** | CS-97568 | A+ | A | A | A | A | A | A | A- | A | ... | B+ | B+ | A | A | A- | B | A |
| **568** | CS-97569 | B | A | A- | B+ | A | A | A | A | A | ... | A- | B | A | B+ | A | C | B |
| **569** | CS-97570 | A | B+ | D | A | D | D+ | B- | C- | B- | ... | D | B | B | C- | D | C | |
| **570** | CS-97571 | C | D | D | C | C | D+ | B | C+ | C | ... | C+ | C | B- | D | F | C- | B |

571 rows × 43 columns

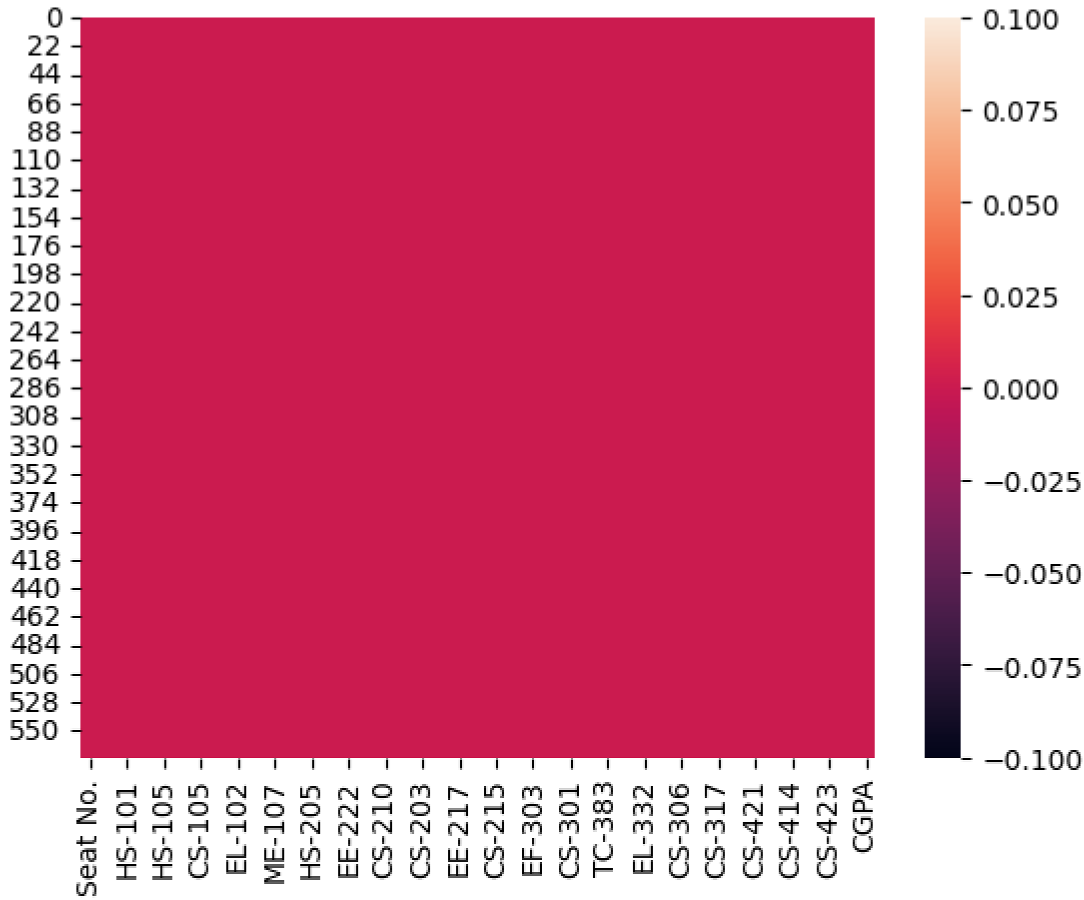
In [15]:
```
#Checking Null values
df.isnull().sum()
```

Out[15]:
```
Seat No.    0
PH-121      0
HS-101      0
CY-105      0
HS-105      0
MT-111      0
CS-105      0
CS-106      0
EL-102      0
EE-119      0
ME-107      0
CS-107      0
HS-205      0
MT-222      0
EE-222      0
MT-224      0
CS-210      0
CS-211      0
CS-203      0
CS-214      0
EE-217      0
CS-212      0
CS-215      0
MT-331      0
EF-303      0
HS-304      0
CS-301      0
CS-302      0
TC-383      0
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 [16]:

```
#Visualizing null values
sns.heatmap(df.isnull())
```

Out[16]: <Axes: >




### Observations:

- **Duplicate Values** Our dataset does not contain any duplicate values.
- **Missing Values:** There are a total of 425 missing values distributed across various columns. And we have filled those missing values with mode values.So now we dont have any missing values in our dataframe.

In [17]:

```
# checking value counts
for i in df.columns:
    print(df[i].value_counts())
```

```
CS-97001    1
CS-97384    1
CS-97378    1
CS-97379    1
CS-97380    1
           ..
CS-97185    1
CS-97184    1
CS-97183    1
CS-97182    1
CS-97571    1
Name: Seat No., Length: 571, dtype: int64
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
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
A     178
A-    120
B+     50
B      49
B-     42
D      31
A+     31
C      19
C+     17
C-     16
D+     14
WU      3
F       1
Name: CY-105, dtype: int64
A     97
A-    75
B+    70
B     57
```

```
D      45
C      41
B-     40
C+     39
C-     36
D+     34
A+     34
WU      2
F       1
Name: HS-105, dtype: int64
A-     107
A      100
B-      70
B+      62
B       55
C-      39
C+      33
C       30
D       26
A+      23
D+      21
WU       3
F        2
Name: MT-111, dtype: int64
A      151
A-     134
B+      60
B       51
A+      43
B-      38
C+      23
C       22
C-      22
D+      15
D       12
Name: CS-105, dtype: int64
A-     118
B+     101
B       96
A       56
B-      54
C+      41
D+      29
C-      27
D       24
C       18
A+       4
WU       2
F        1
Name: CS-106, dtype: int64
A-     107
A       92
B+      69
B       59
B-      53
D       38
C+      35
```

```
C-      32
C       30
A+      29
D+      23
WU       3
F        1
Name: EL-102, dtype: int64
A-     139
B+      83
B       77
A       68
B-      48
C       48
C+      38
D+      26
C-      26
D       11
A+       6
WU       1
Name: EE-119, dtype: int64
A-     81
A      77
B+     68
D      56
B-     56
B      50
C      49
C-     48
C+     37
D+     37
A+      8
WU      2
F       2
Name: ME-107, dtype: int64
A      107
A-      81
B+      57
B       55
C-      49
B-      43
A+      42
D       38
C+      34
D+      31
C       30
WU       2
I        1
F        1
Name: CS-107, dtype: int64
A-     155
A      118
B       97
B+      89
B-      36
C+      33
C       15
C-      11
```

```
D+      9
D       3
A+      2
F       2
WU      1
Name: HS-205, dtype: int64
A-    91
A     80
D     66
B     61
B-    52
D+    46
C     43
B+    42
C+    39
C-    30
A+    16
F      3
WU     1
W      1
Name: MT-222, dtype: int64
A     129
A-    121
B+     65
B      53
C      39
B-     35
A+     32
C+     29
D      25
D+     21
C-     16
F       4
W       2
Name: EE-222, dtype: int64
A-    127
A      80
B+     65
B      57
B-     49
C-     43
C+     39
D+     37
D      31
C      30
A+     10
WU      1
W       1
F       1
Name: MT-224, dtype: int64
A-    140
A     101
B+     84
B      59
B-     58
C      30
C+     27
```

```
C-      24
D+      21
D       12
A+      12
WU       1
W        1
F        1
Name: CS-210, dtype: int64
A-    73
A     67
B-    60
D+    56
B+    56
B     56
C+    55
C-    50
C     39
D     33
A+    21
F      3
WU     1
W      1
Name: CS-211, dtype: int64
A-    93
A     81
B     66
C+    62
B+    59
B-    53
D+    39
C     35
C-    35
D     30
A+    15
F      2
I      1
Name: CS-203, dtype: int64
C     82
A-    73
B     63
C-    57
B-    56
A     56
D+    47
C+    46
B+    45
D     31
A+    12
F      2
I      1
Name: CS-214, dtype: int64
A-    143
A      97
B+     70
B      63
B-     57
C      36
```

```
C+     29
A+     22
C-     20
D+     19
D      12
F       2
WU      1
Name: EE-217, dtype: int64
A-    107
B+     86
B      81
B-     65
C      44
A      43
D+     36
C+     35
C-     35
D      33
A+      4
WU      2
Name: CS-212, dtype: int64
A-    85
A     68
B     64
B-    59
C+    50
B+    50
C     48
C-    47
D     42
D+    38
A+    17
WU     1
W      1
F      1
Name: CS-215, dtype: int64
A     127
A-    103
B+     64
B      57
B-     46
C-     32
D+     31
A+     30
C      28
C+     27
D      22
F       4
Name: MT-331, dtype: int64
B     122
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
A-     138
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
A-     118
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
A-     123
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
A      115
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
A-    150
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
A-    105
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
A-    98
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
A-    129
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
```

```
A+    103
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
B-    91
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
A     145
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
B     98
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
A-    262
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
A      189
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
A-     133
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
A-     136
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
A-     236
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
3.019    5
3.058    3
2.793    3
3.443    3
2.206    3
        ..
2.555    1
2.042    1
2.634    1
2.053    1
1.753    1
Name: CGPA, Length: 491, dtype: int64
```

In [18]:

```
#Removing some inconsistent grades
for i in df.columns:
    df.drop(df[(df.loc[:,i]=='WU')| (df.loc[:,i]=='W')].index,inplace=True)
df.reset_index(drop=True,inplace=True)
```

## Observations:

- We have checked the unique values in our dataframe for each column, found some inconsistent grades.
- We have removed those inconsistent grades from the DataFrame.

## Statistical Summary

In [19]:
```
df.describe(include="object").T
```

Out[19]:

| | count | unique | top | freq |
|---|---|---|---|---|
| **Seat No.** | 547 | 547 | CS-97001 | 1 |
| **PH-121** | 547 | 12 | A- | 112 |
| **HS-101** | 547 | 12 | A- | 80 |
| **CY-105** | 547 | 11 | A | 174 |
| **HS-105** | 547 | 11 | A | 95 |
| **MT-111** | 547 | 12 | A- | 105 |
| **CS-105** | 547 | 11 | A | 148 |
| **CS-106** | 547 | 11 | A- | 113 |
| **EL-102** | 547 | 11 | A- | 103 |
| **EE-119** | 547 | 11 | A- | 134 |
| **ME-107** | 547 | 11 | A- | 79 |
| **CS-107** | 547 | 11 | A | 105 |
| **HS-205** | 547 | 11 | A- | 151 |
| **MT-222** | 547 | 12 | A- | 87 |
| **EE-222** | 547 | 12 | A- | 121 |
| **MT-224** | 547 | 11 | A- | 121 |
| **CS-210** | 547 | 11 | A- | 134 |
| **CS-211** | 547 | 12 | A- | 67 |
| **CS-203** | 547 | 11 | A- | 89 |
| **CS-214** | 547 | 11 | C | 75 |
| **EE-217** | 547 | 12 | A- | 138 |
| **CS-212** | 547 | 11 | A- | 102 |
| **CS-215** | 547 | 11 | A- | 78 |
| **MT-331** | 547 | 12 | A | 118 |
| **EF-303** | 547 | 11 | B | 111 |
| **HS-304** | 547 | 12 | A- | 130 |
| **CS-301** | 547 | 12 | A- | 110 |
| **CS-302** | 547 | 11 | A- | 114 |
| **TC-383** | 547 | 12 | A | 106 |
| **MT-442** | 547 | 12 | A- | 141 |
| **EL-332** | 547 | 12 | A- | 96 |
| **CS-318** | 547 | 12 | A- | 89 |
| **CS-306** | 547 | 12 | A- | 120 |
| **CS-312** | 547 | 12 | A+ | 94 |
| **CS-317** | 547 | 12 | B- | 81 |
| **CS-403** | 547 | 11 | A | 134 |

| | count | unique | top | freq |
|---|---|---|---|---|
| **CS-421** | 547 | 12 | B | 89 |
| **CS-406** | 547 | 12 | A- | 252 |
| **CS-414** | 547 | 12 | A | 179 |
| **CS-419** | 547 | 12 | A- | 123 |
| **CS-423** | 547 | 12 | A- | 126 |
| **CS-412** | 547 | 12 | A- | 223 |

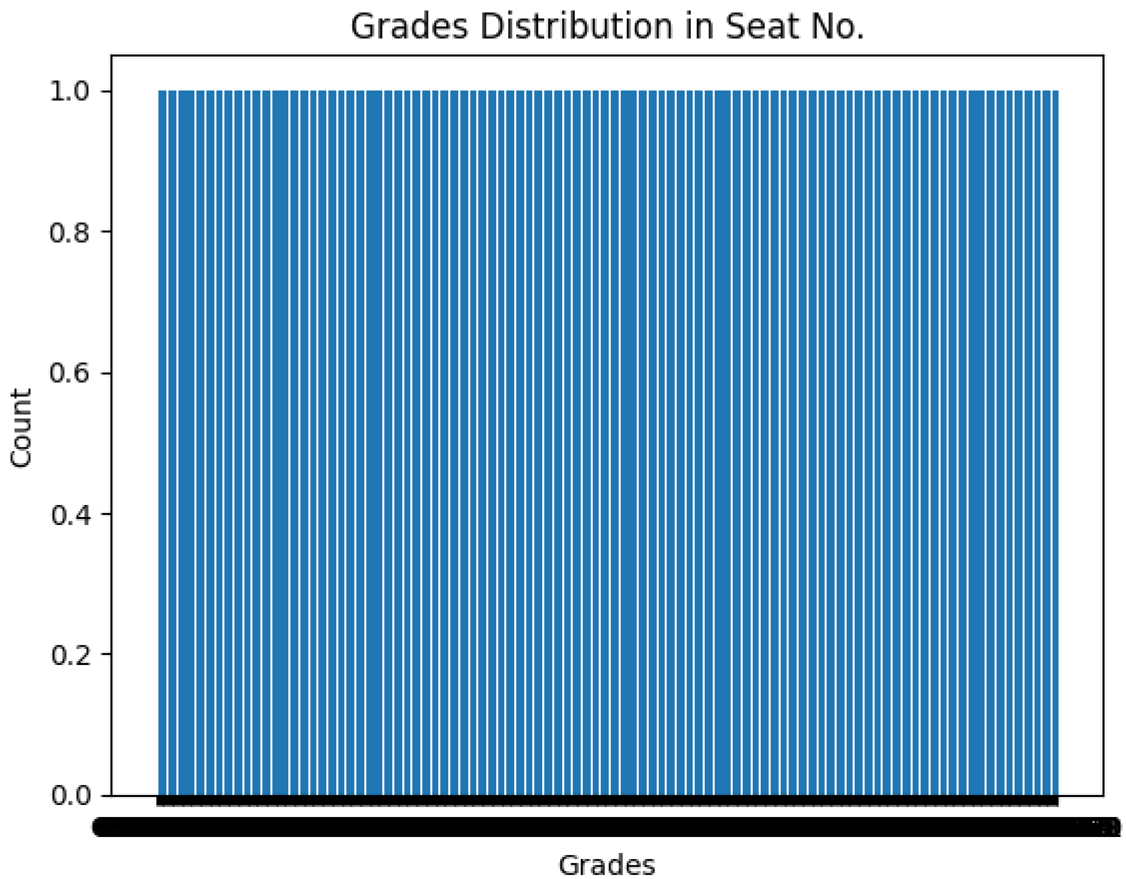
## Observations:

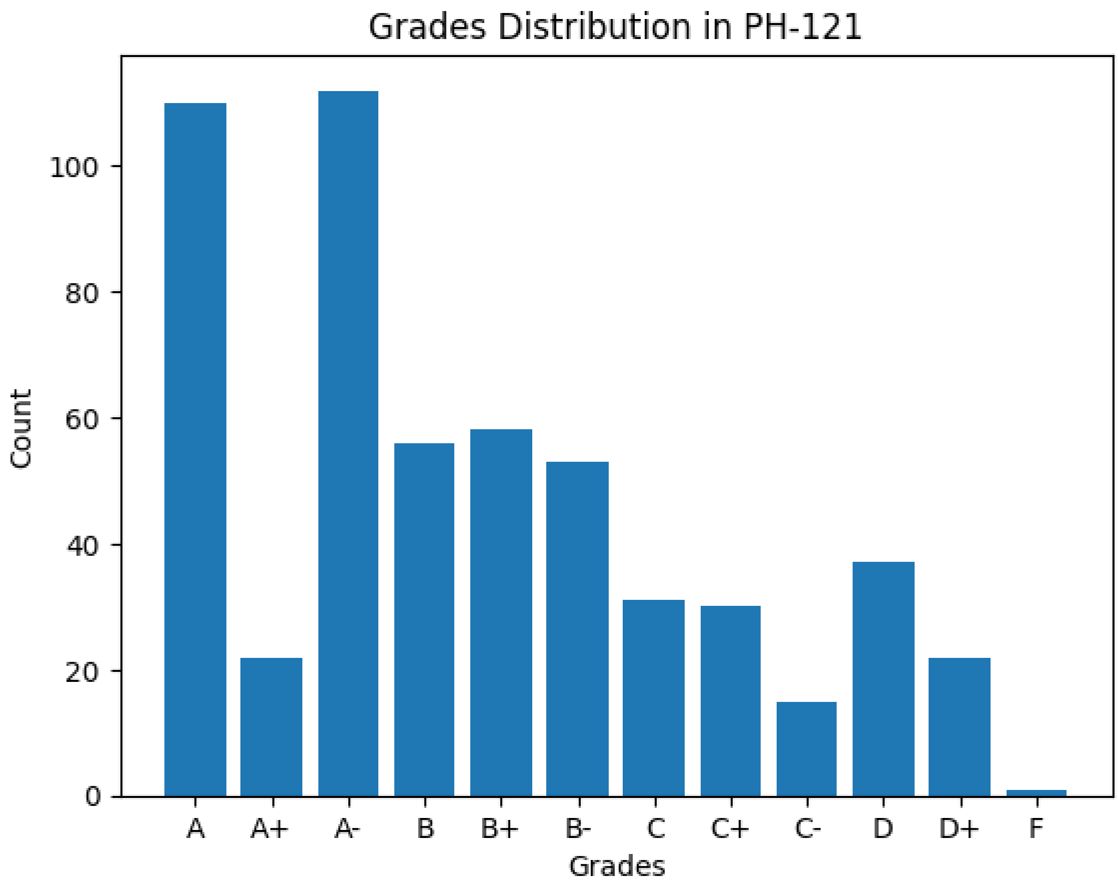
- "Seat No.": This column contains 547 unique values, indicating that there are 547 distinct seat numbers in the dataset. The most frequent seat number is "CS-97001," which appears once.
- "PH-121": There are 12 unique values in this column. The most common value is "A-" with a frequency of 112.
- "HS-101": Similar to "PH-121," there are 12 unique values, and "A-" is the most frequent value with a frequency of 80.
- "CY-105" and "HS-105": These columns both have 11 unique values, and "A" is the most common value in both, with frequencies of 174 and 95, respectively.
- "MT-111," "CS-105," "CS-106," "EL-102," "EE-119," "ME-107," "CS-107," "HS-205," "MT-222," "EE-222," "MT-224," "CS-210," "CS-211," "CS-203," "CS-214," "EE-217," "CS-212," and "CS-215": Each of these columns has 11 or 12 unique values, and "A-" is the most frequent value in most of them.
- "MT-331" and "HS-304": These columns both have 12 unique values, and "A" is the most common value in both, with frequencies of 118 and 130, respectively.
- "EF-303," "CS-301," and "CS-302": These columns have 11 or 12 unique values, and "A-" is the most common value in each of them.
- "TC-383," "MT-442," "EL-332," "CS-318," "CS-306," "CS-312," "CS-317," "CS-403," "CS-421," "CS-406," "CS-414," "CS-419," and "CS-423": Each of these columns has 11 or 12 unique values, and various grades are the most common values in them.

Overall, it appears that the dataset contains information related to courses, seat numbers, and grades. The grades are categorized with varying frequencies in different columns.

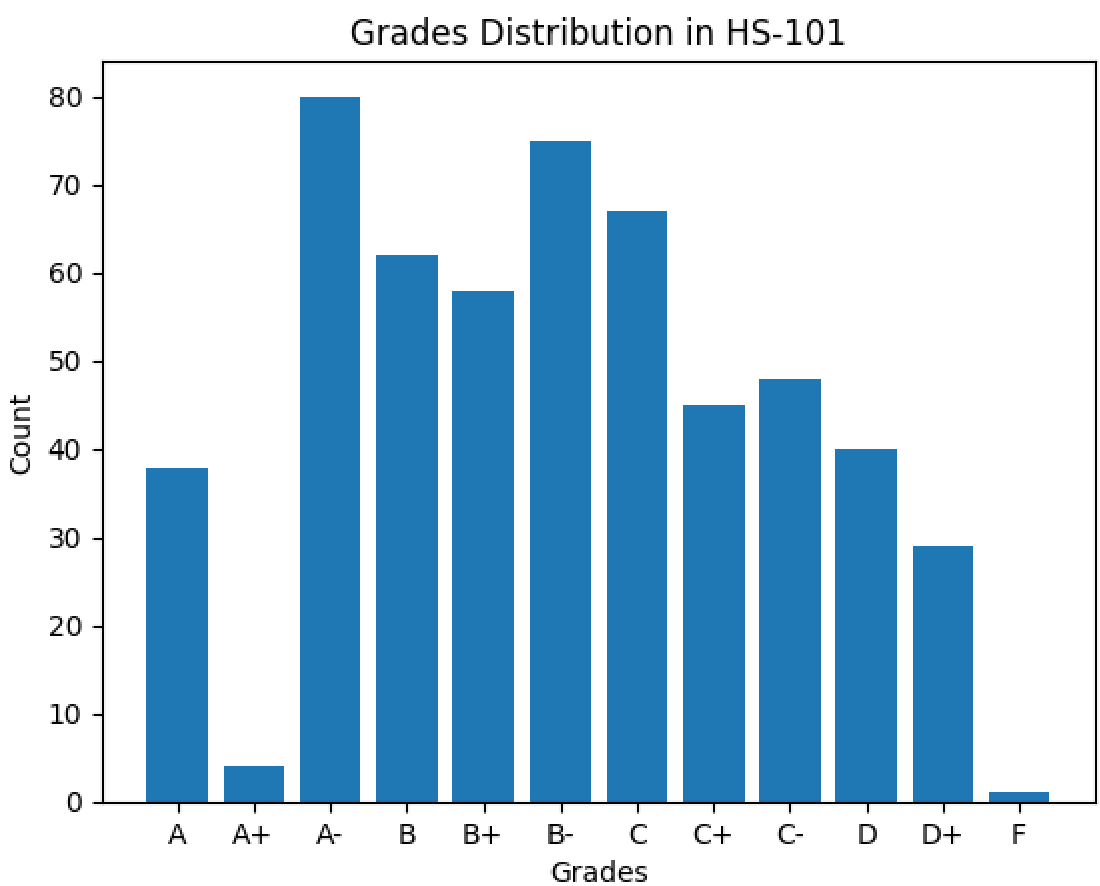
## Visiualizing it

In [20]:
```
# Bar Chart for grades distribution in each subject
subjects = df.columns
for subject in subjects:
    grades_count = df[subject].value_counts().sort_index()
    plt.bar(grades_count.index, grades_count.values)
    plt.xlabel('Grades')
    plt.ylabel('Count')
    plt.title(f'Grades Distribution in {subject}')
    plt.show()
```

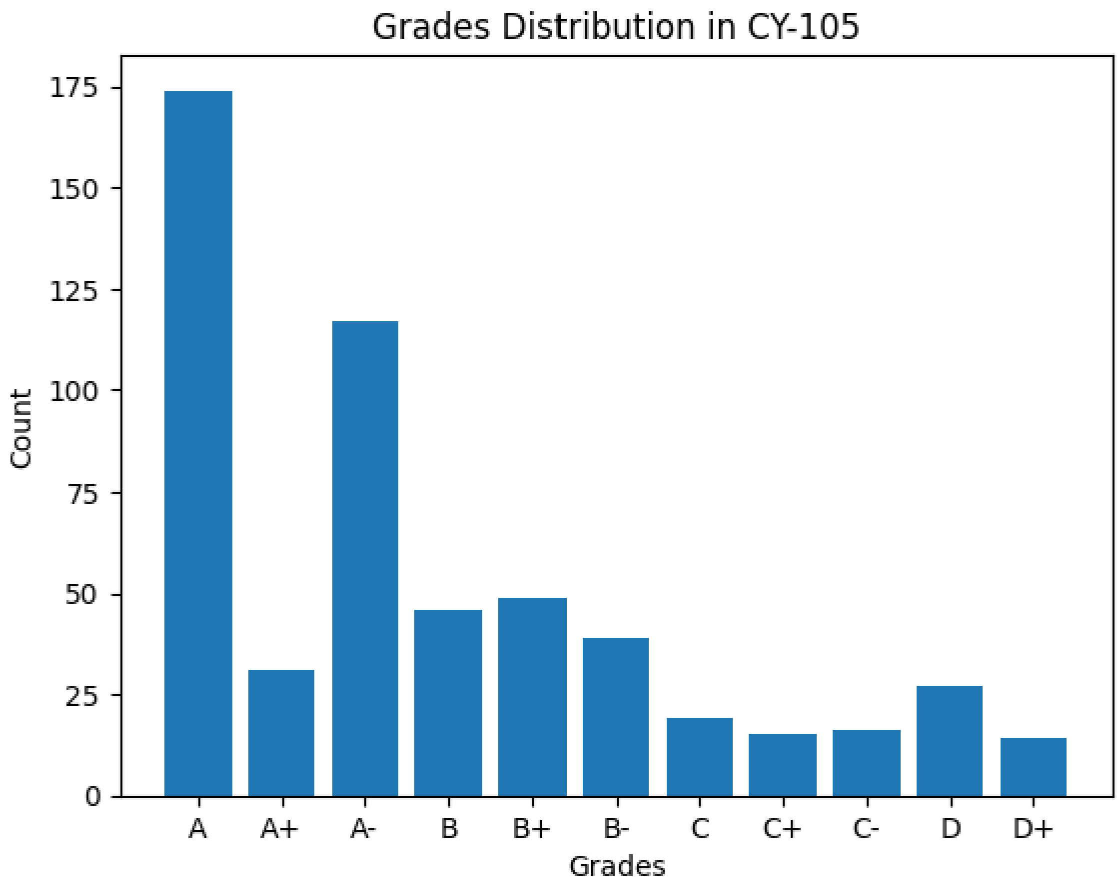


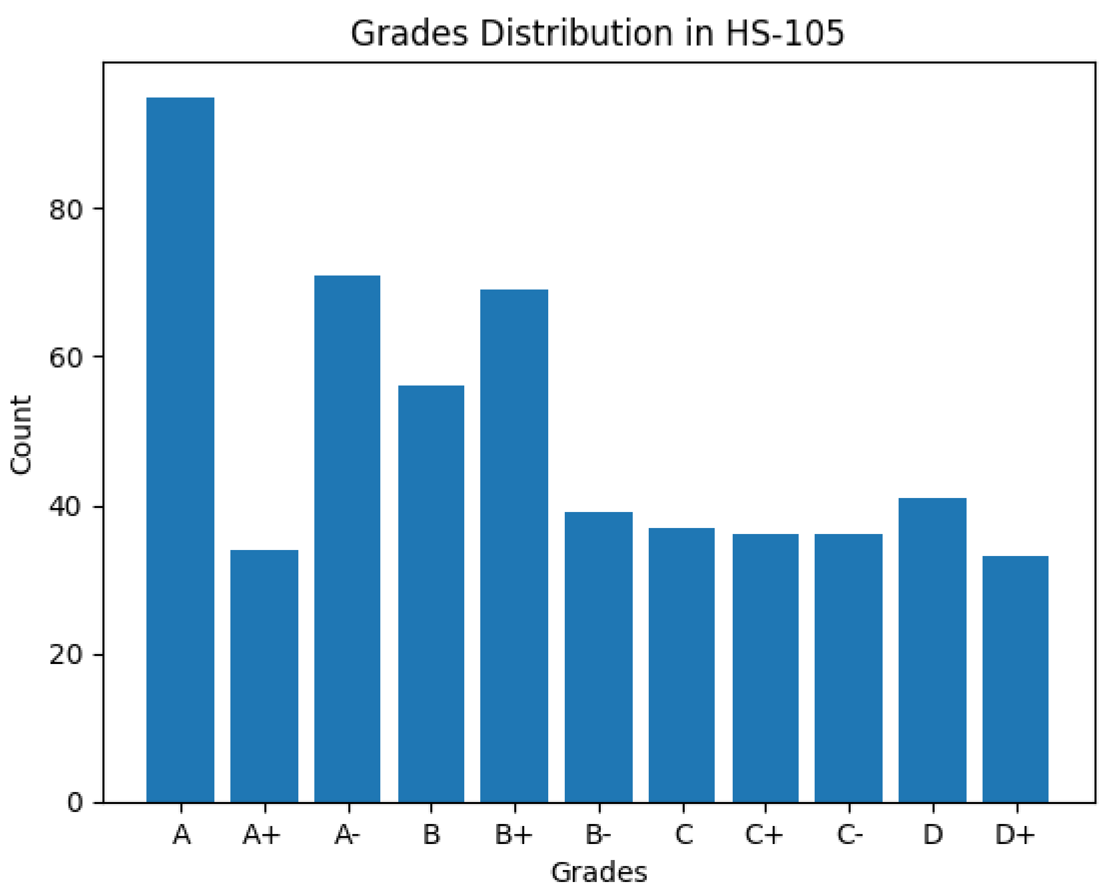


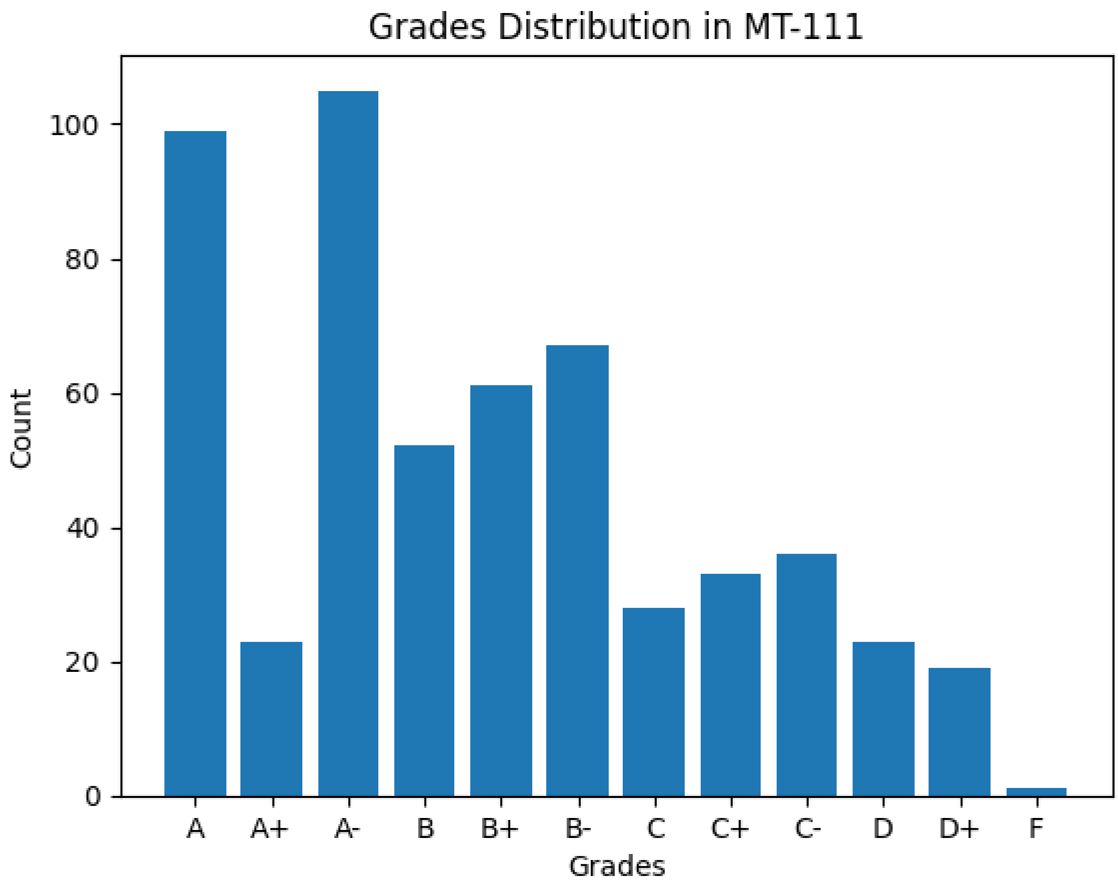
Grades Distribution in PH-121
Count
0
20
40
60
80
100
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F
Grades

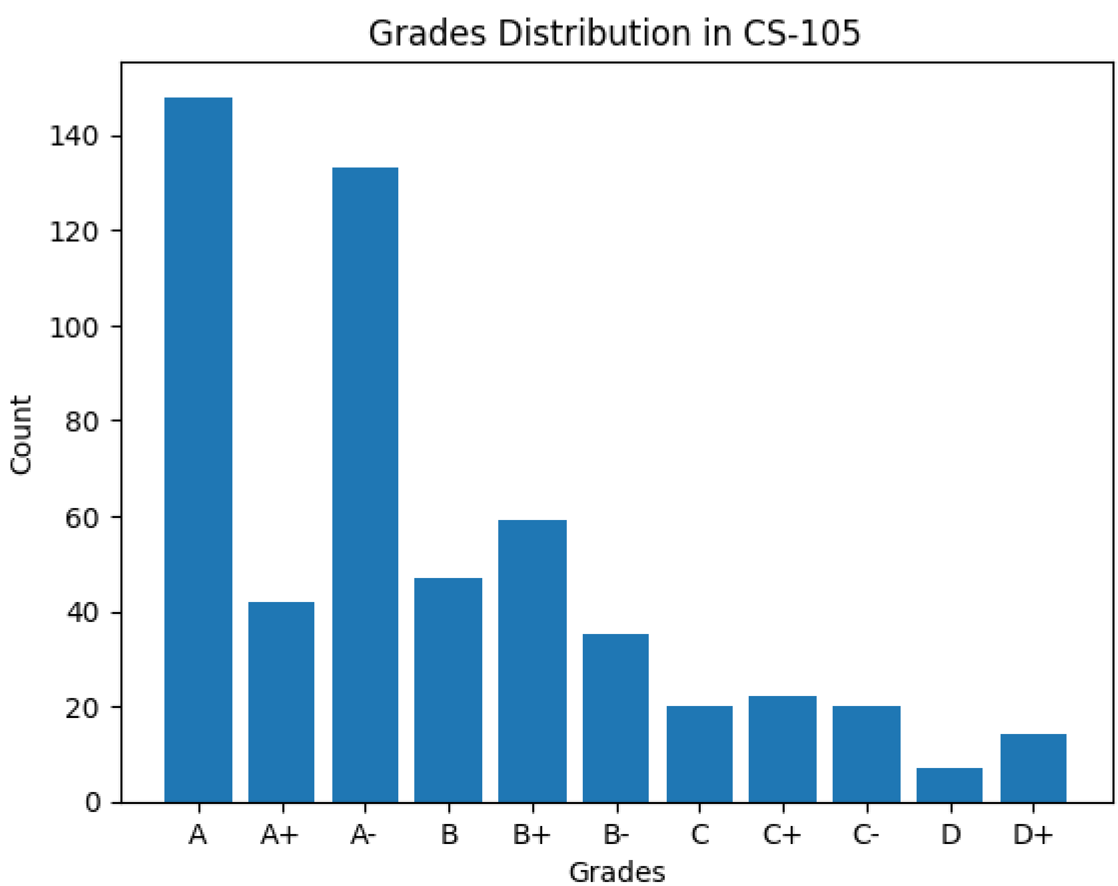



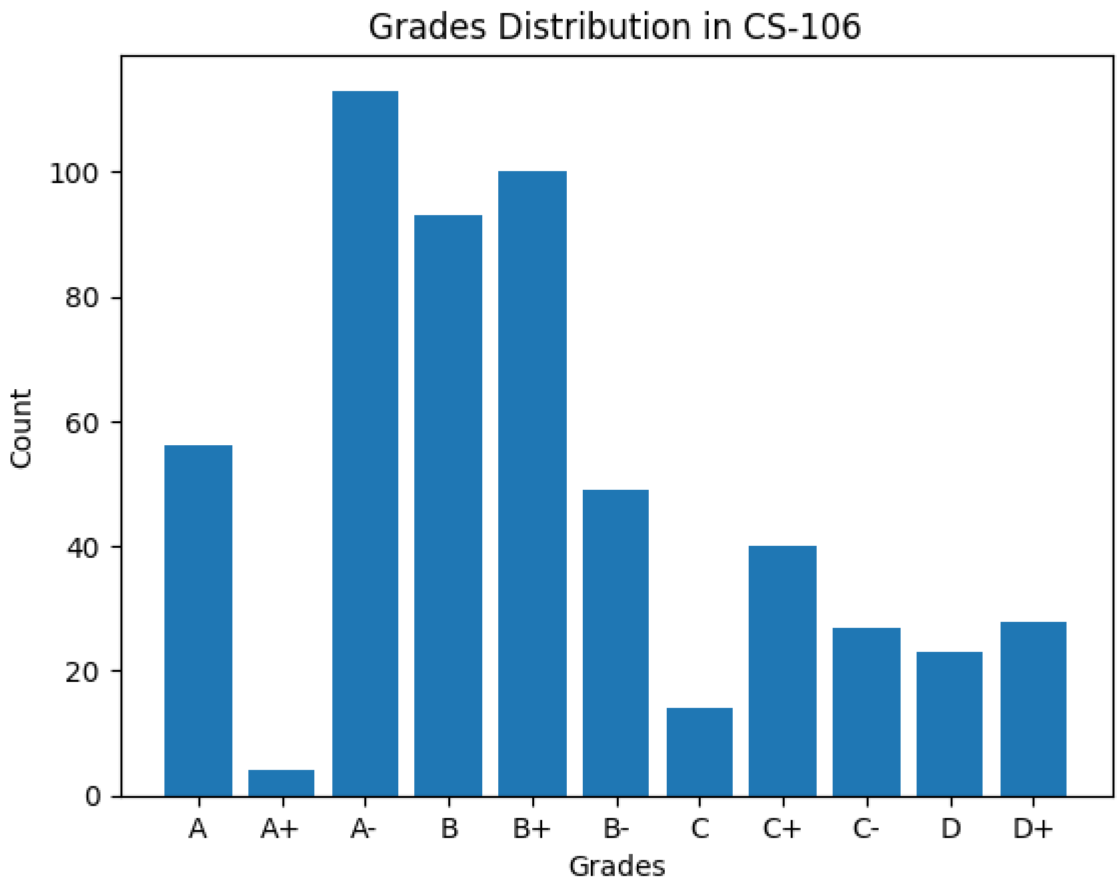
Grades Distribution in HS-101
Count
0
10
20
30
40
50
60
70
80
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F
Grades

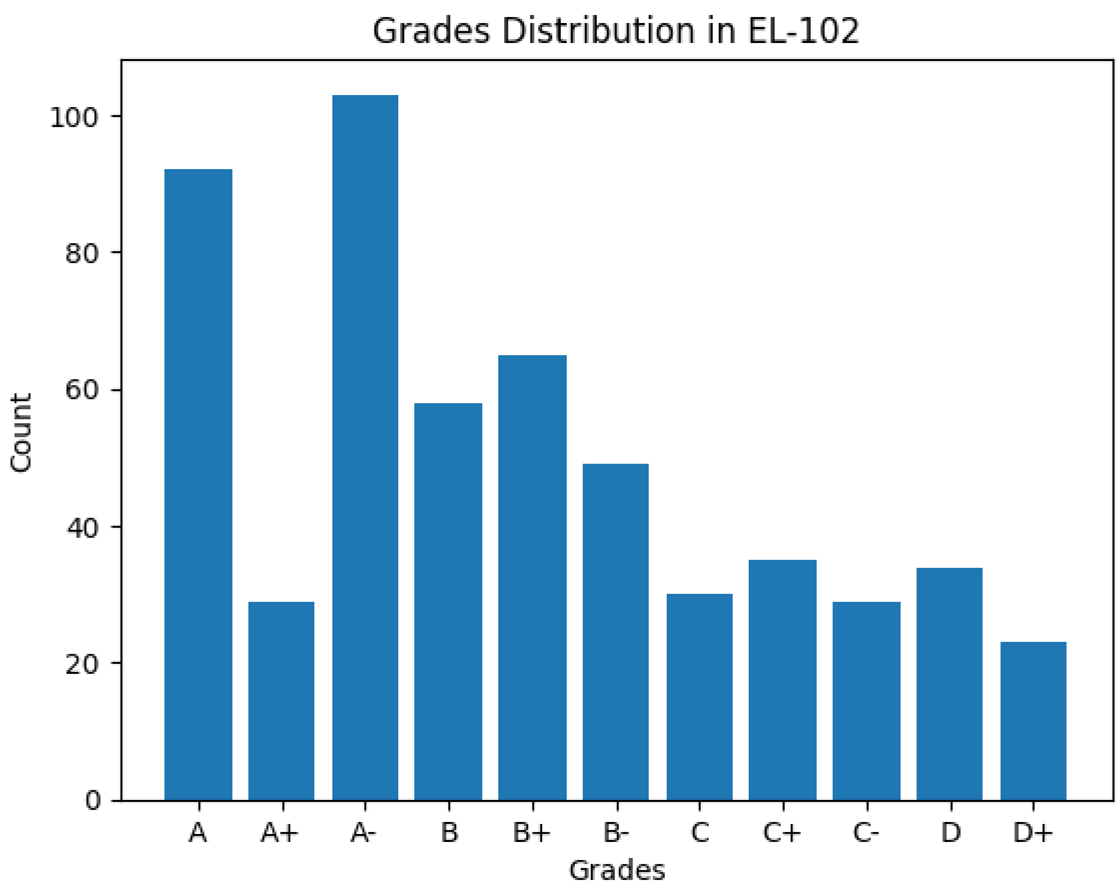

Grades Distribution in CY-105
Count
175
150
125
100
75
50
25
0
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
Grades

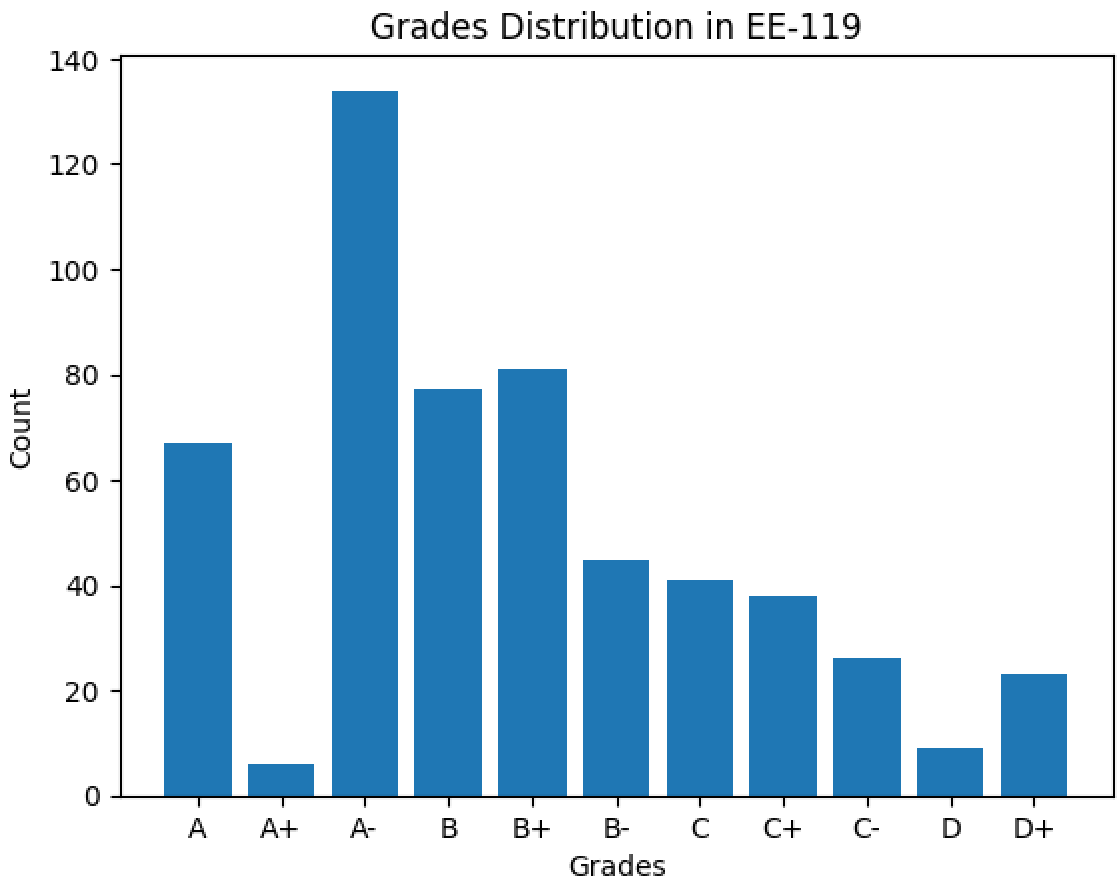

Grades Distribution in HS-105
Count
80
60
40
20
0
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
Grades

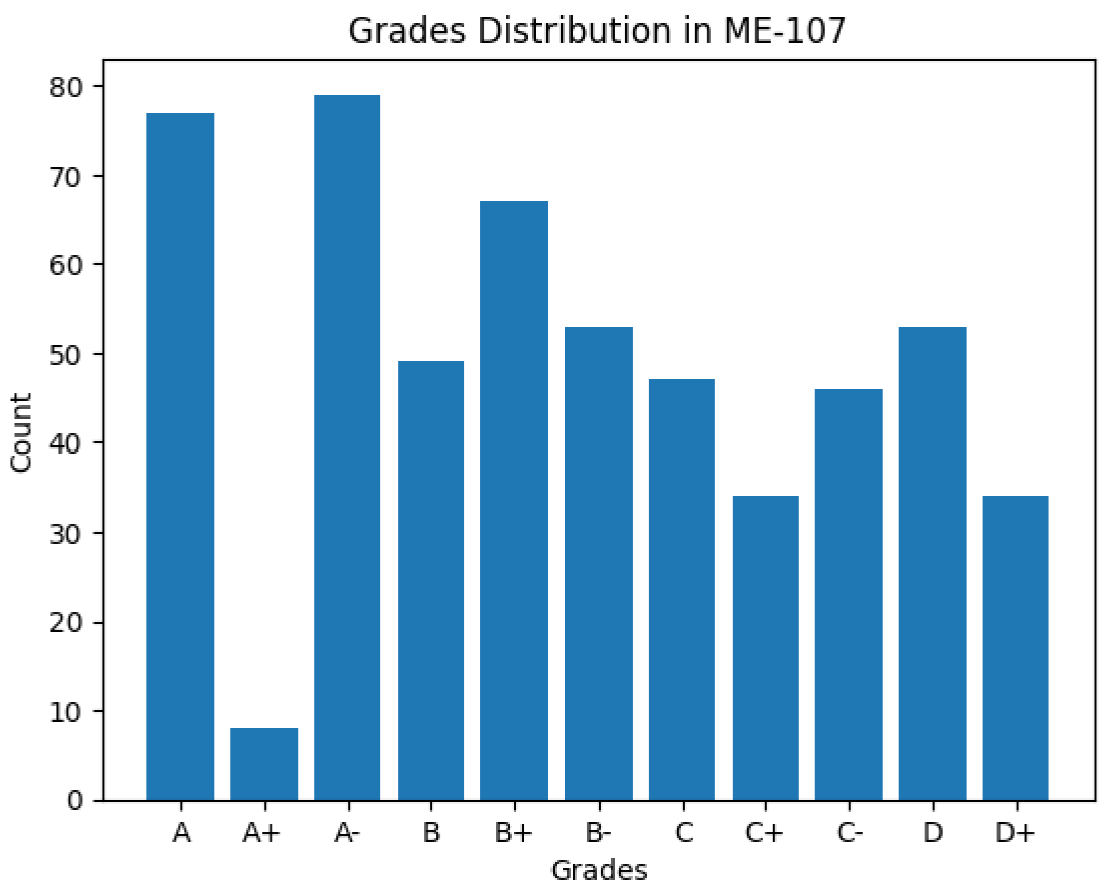

Grades Distribution in MT-111
Count
0
20
40
60
80
100
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F
Grades

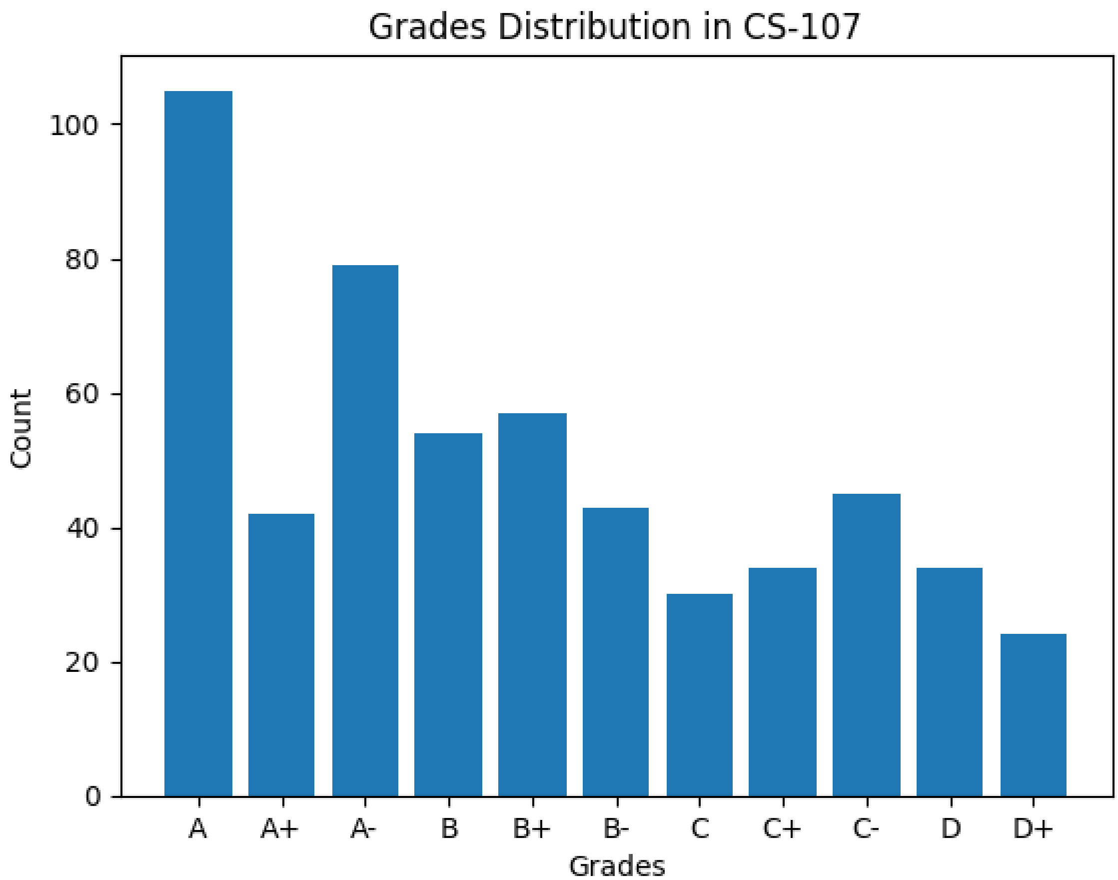

Grades Distribution in CS-105
Count
0
20
40
60
80
100
120
140
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
Grades

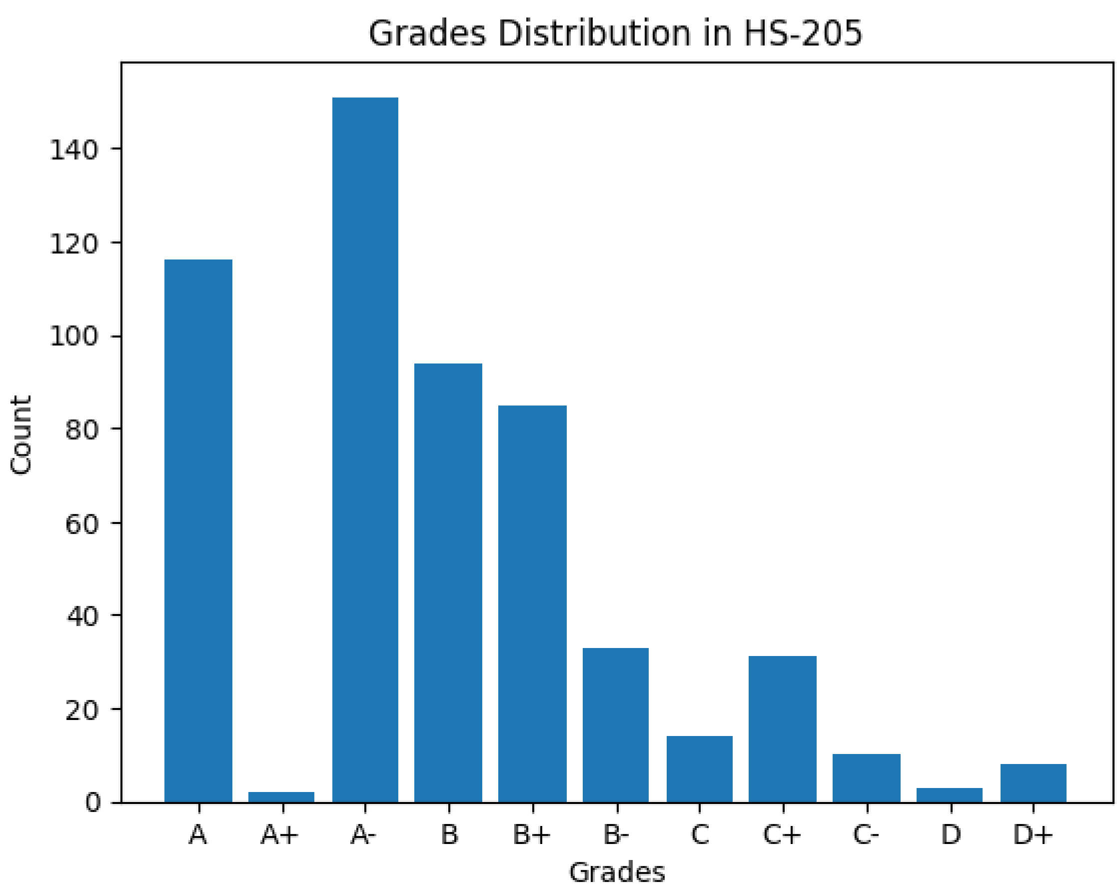

Grades Distribution in CS-106
Count
0
20
40
60
80
100
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
Grades

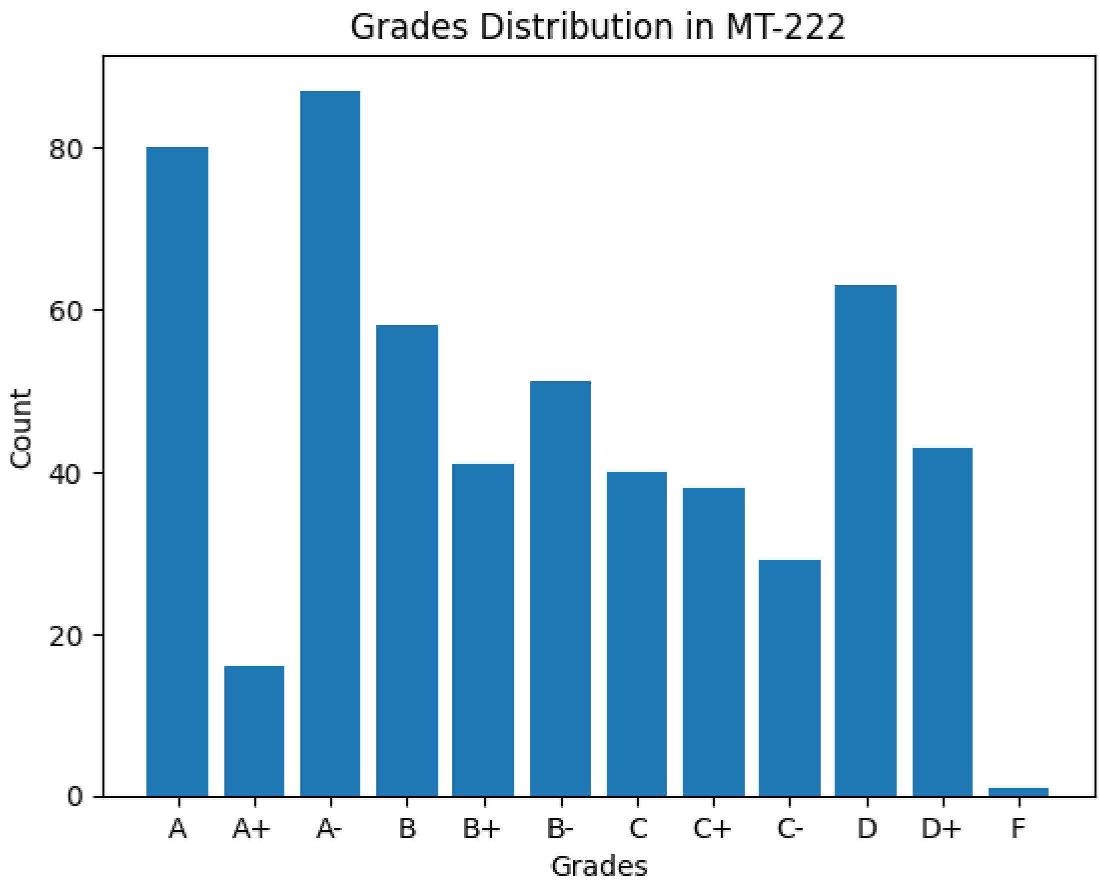

Grades Distribution in EL-102
Count
0
20
40
60
80
100
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
Grades

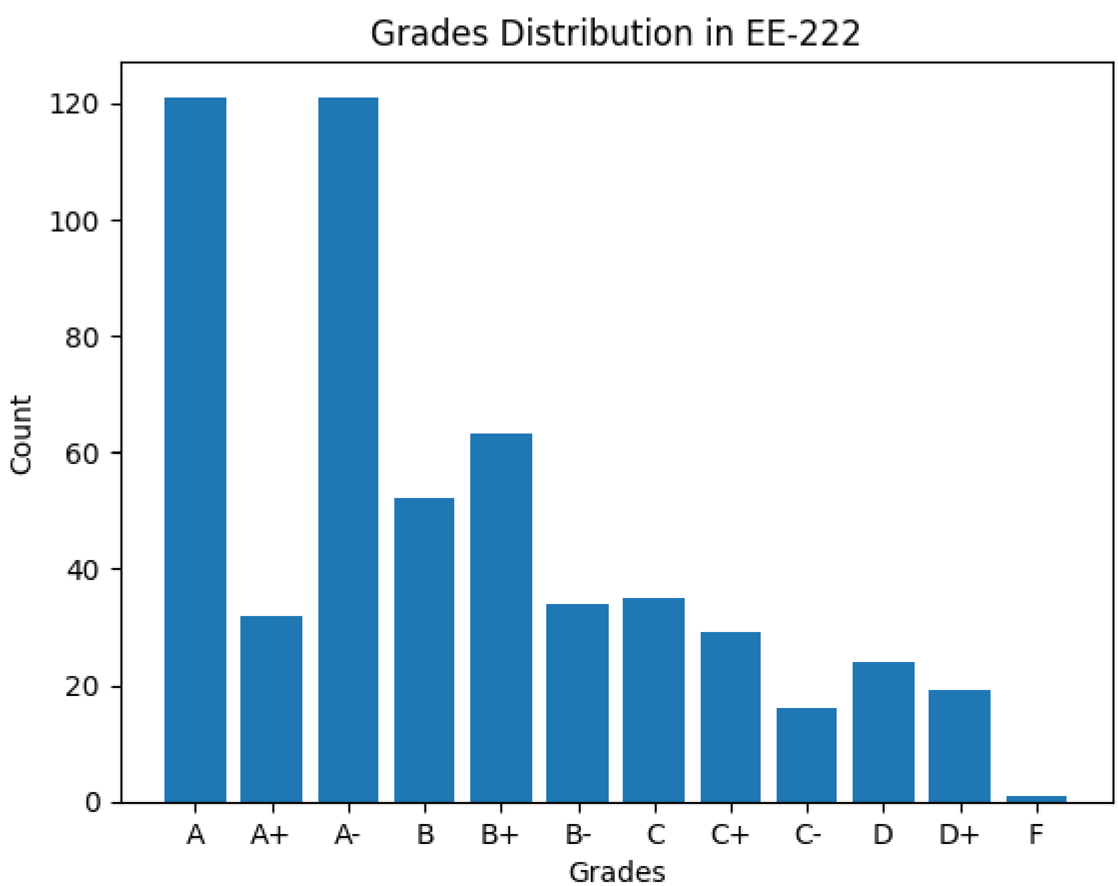

Grades Distribution in EE-119
Count
Grades
0
20
40
60
80
100
120
140
A
A+
A-
B
B+
B-
C
C+
C-
D
D+

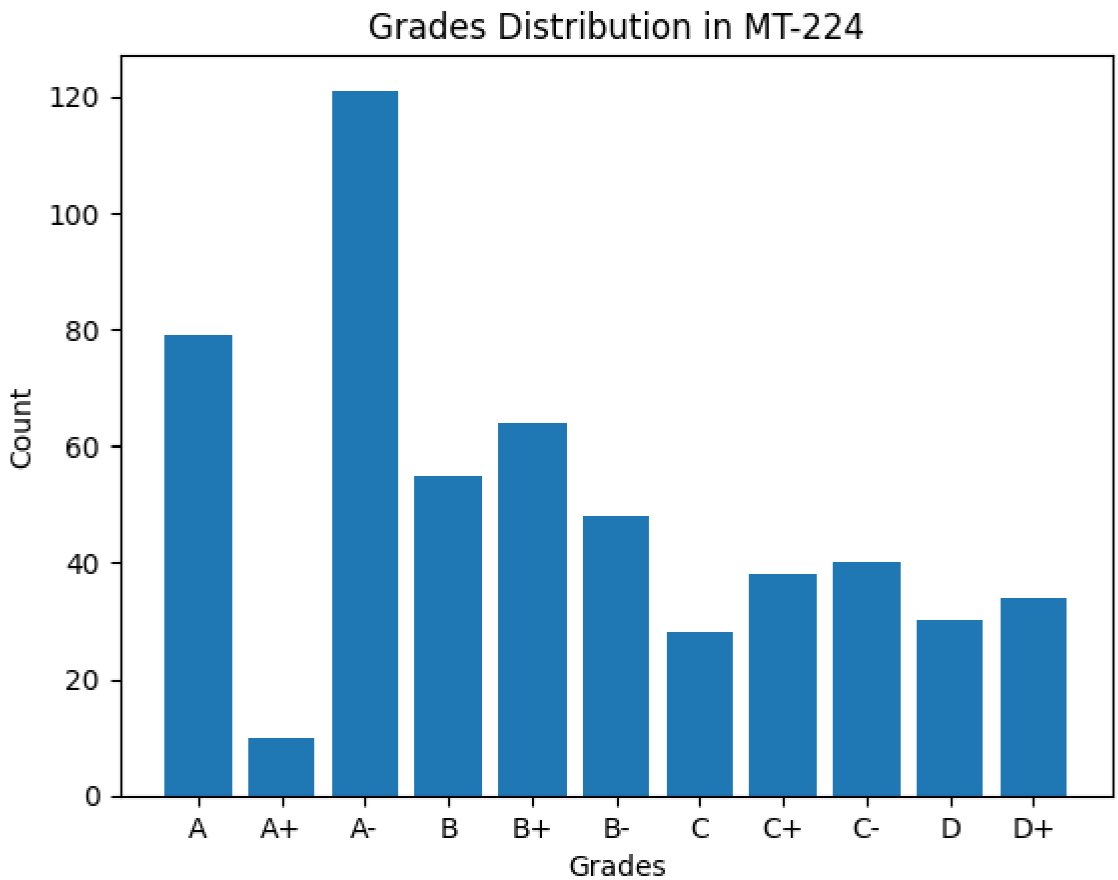

Grades Distribution in ME-107
Count
Grades
0
10
20
30
40
50
60
70
80
A
A+
A-
B
B+
B-
C
C+
C-
D
D+

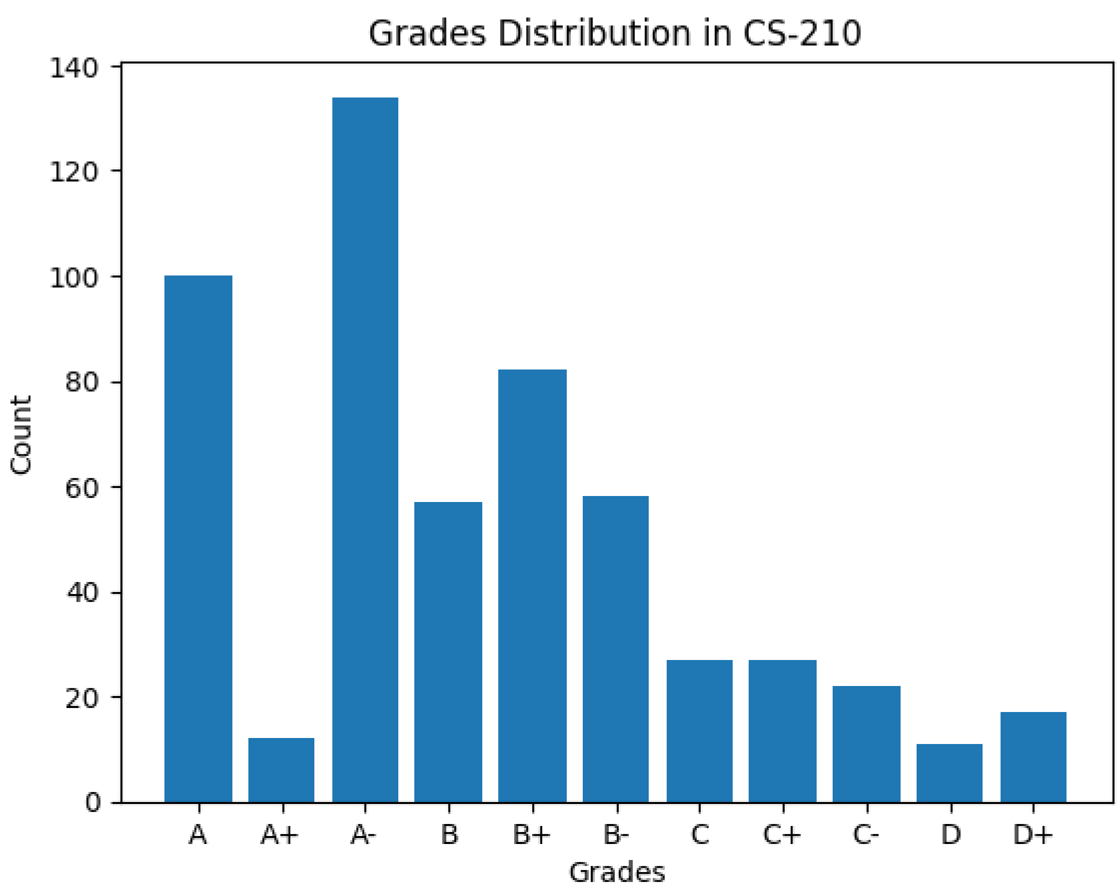

Grades Distribution in CS-107
Count
0
20
40
60
80
100
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
Grades

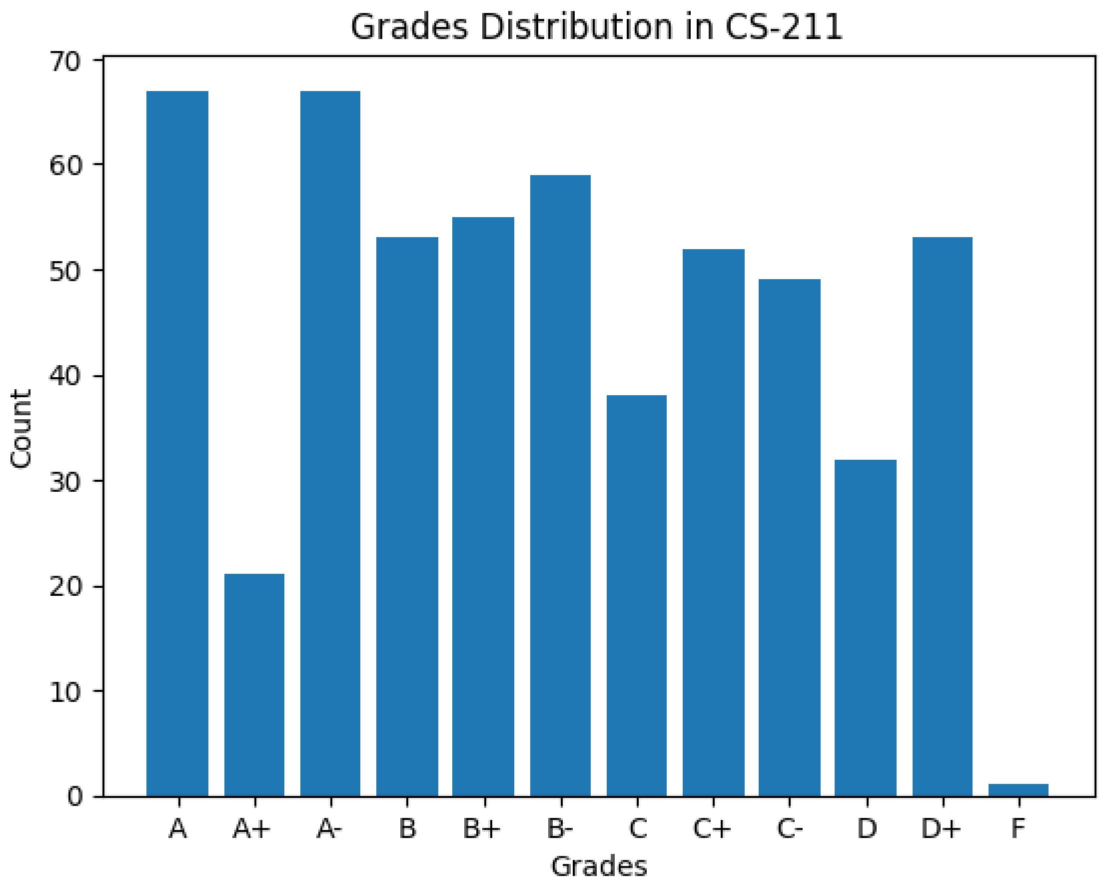

Grades Distribution in HS-205
Count
0
20
40
60
80
100
120
140
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
Grades

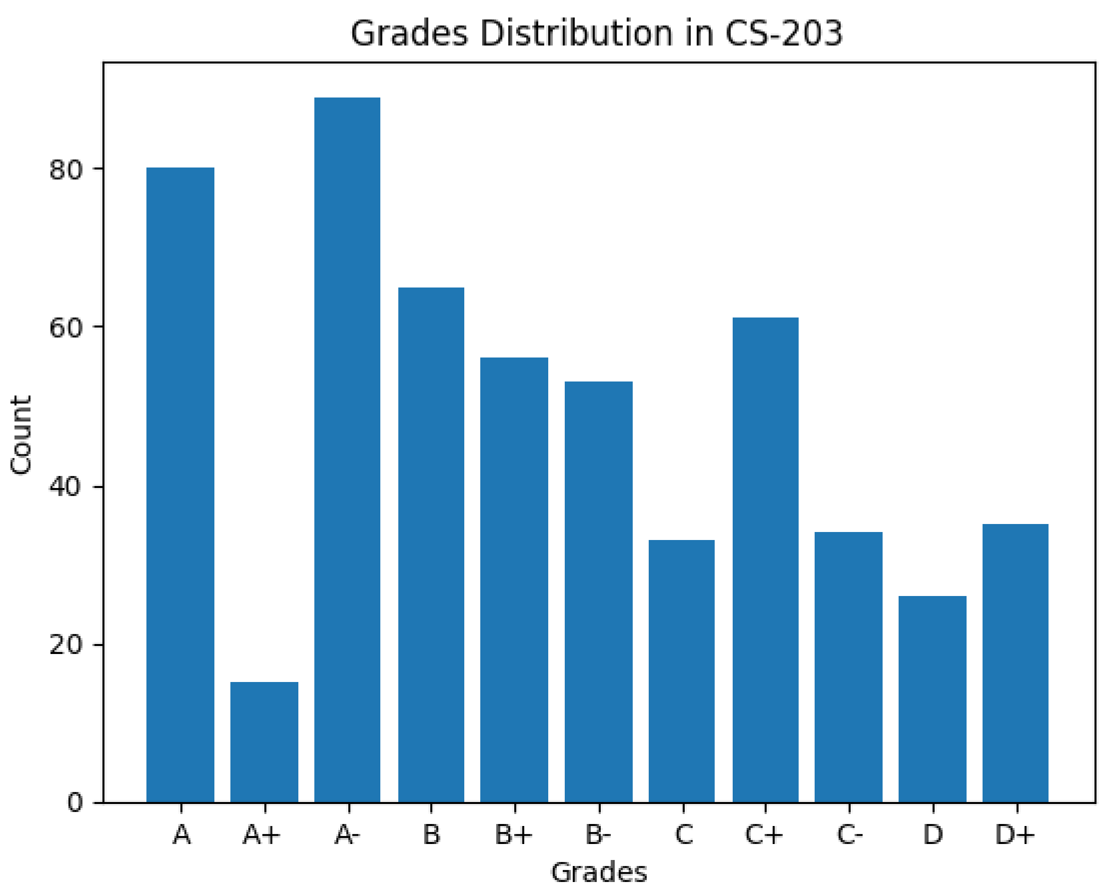

Grades Distribution in MT-222
Count
0
20
40
60
80
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F
Grades

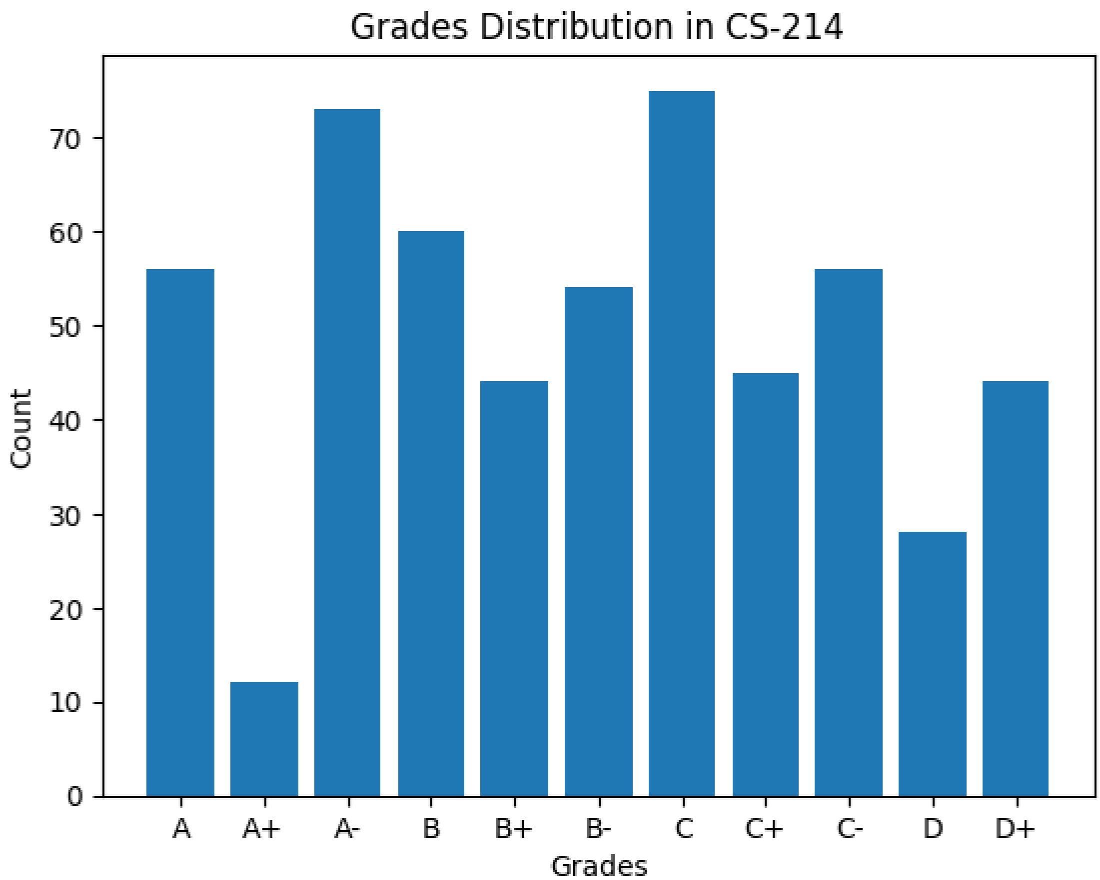

Grades Distribution in EE-222
Count
0
20
40
60
80
100
120
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F
Grades

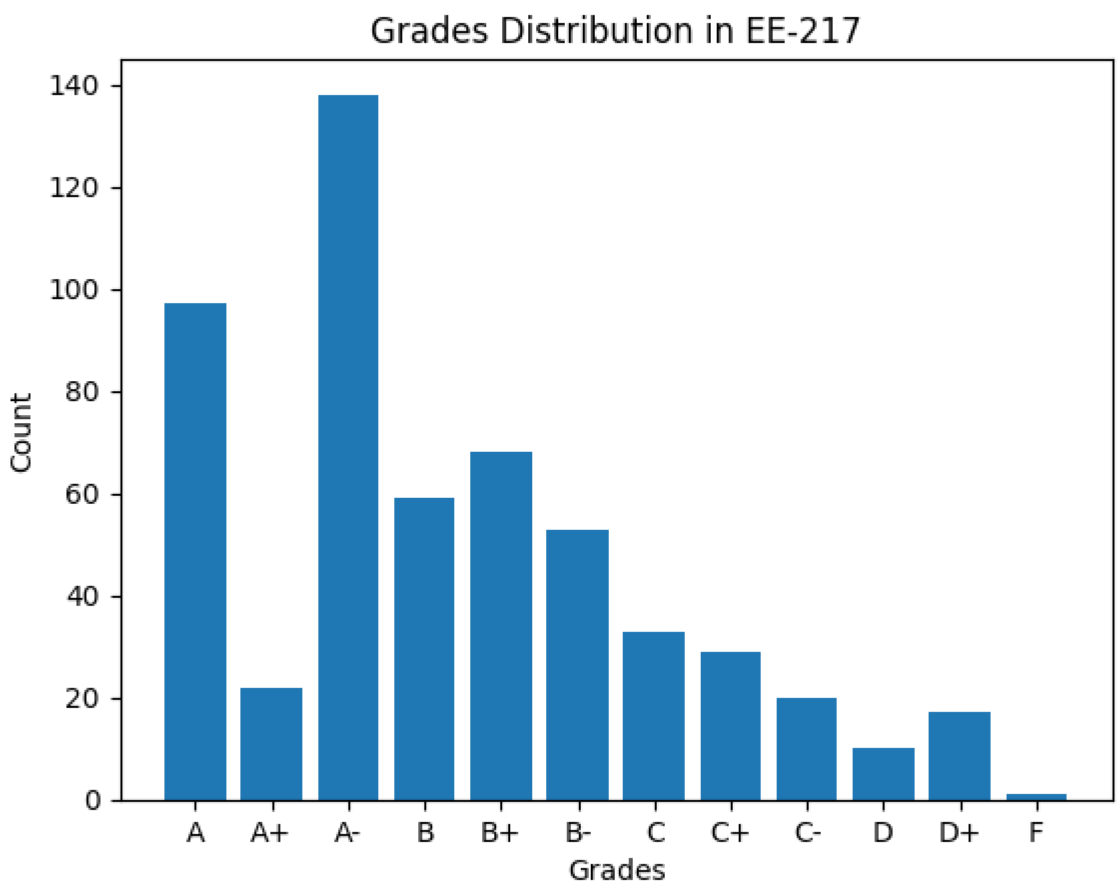

Grades Distribution in MT-224
Count
120
100
80
60
40
20
0
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
Grades

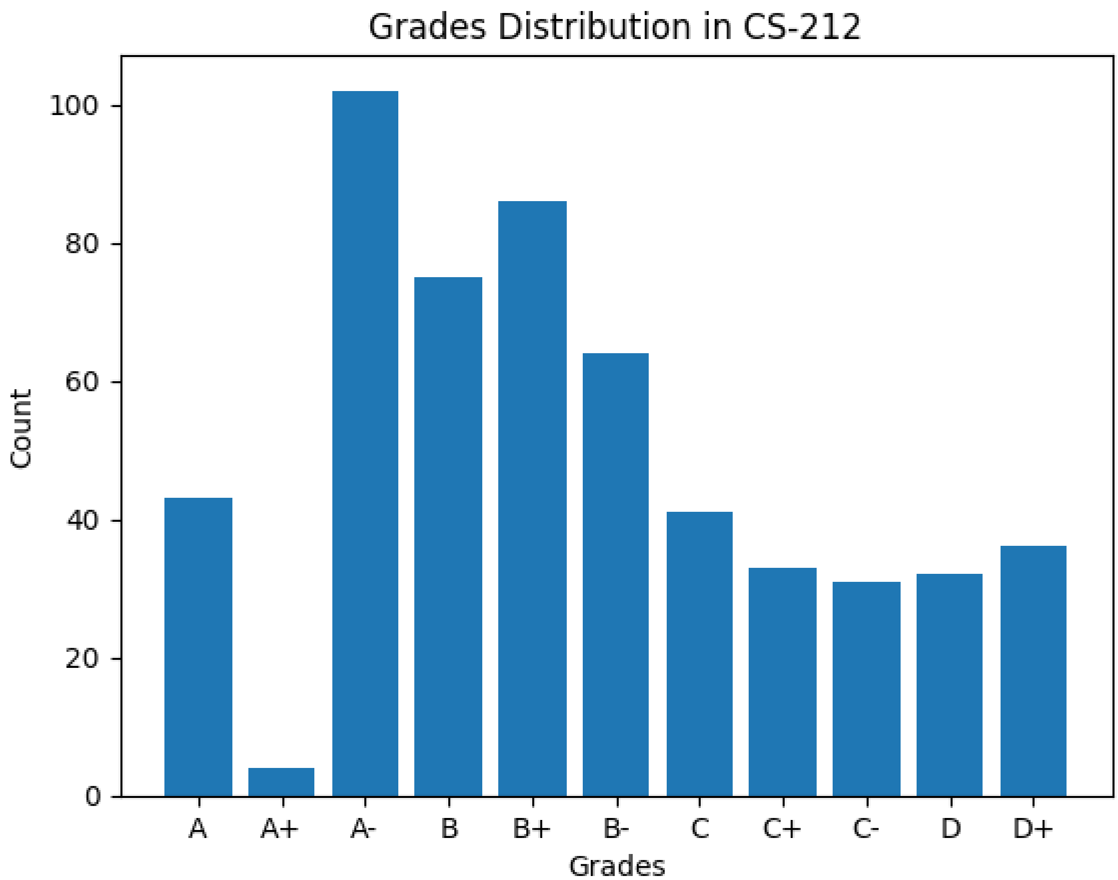

Grades Distribution in CS-210
Count
140
120
100
80
60
40
20
0
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
Grades

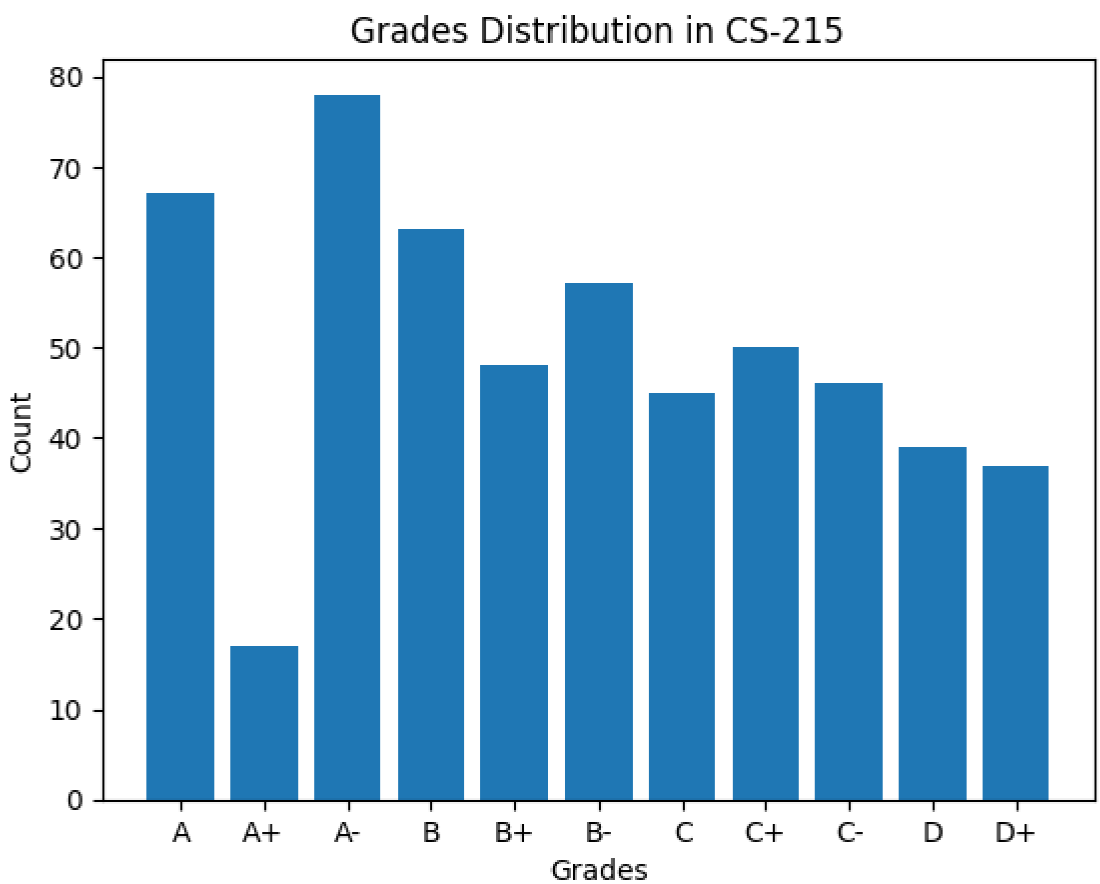

Grades Distribution in CS-211
Count
Grades
0
10
20
30
40
50
60
70
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F

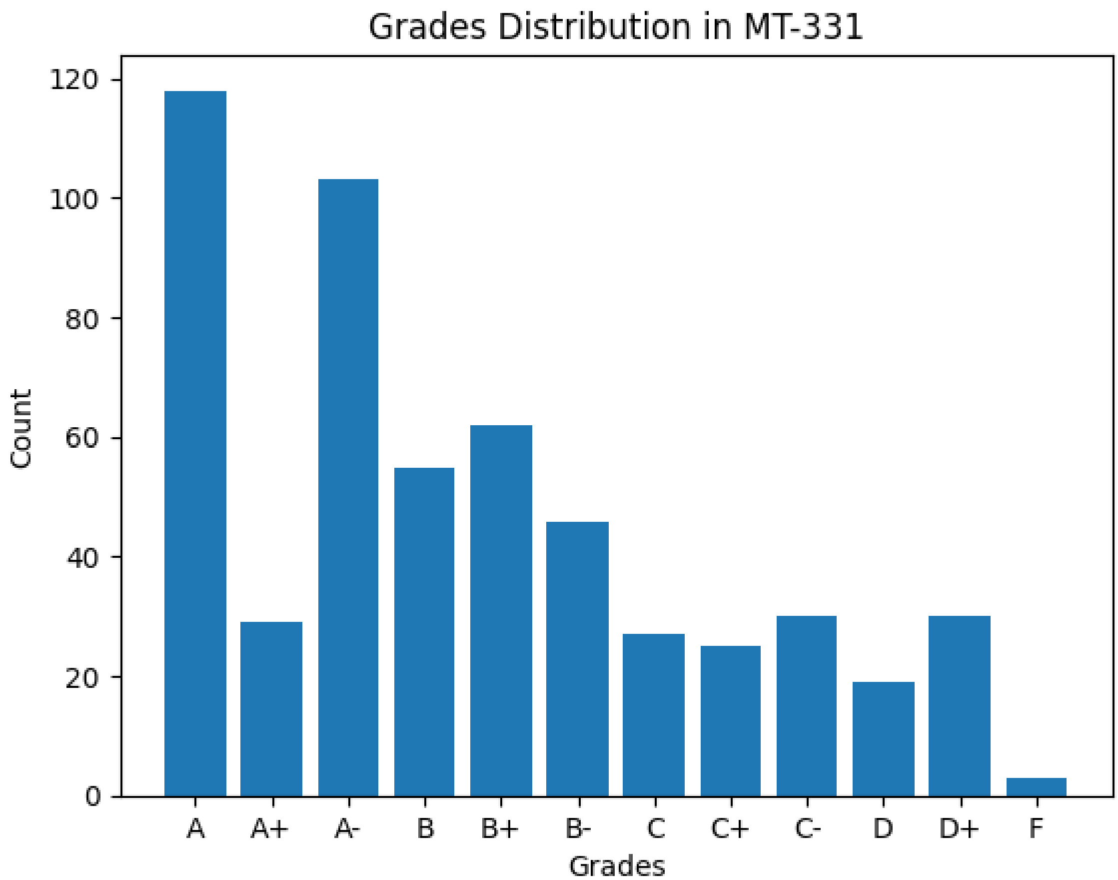

Grades Distribution in CS-203
Count
Grades
0
20
40
60
80
A
A+
A-
B
B+
B-
C
C+
C-
D
D+

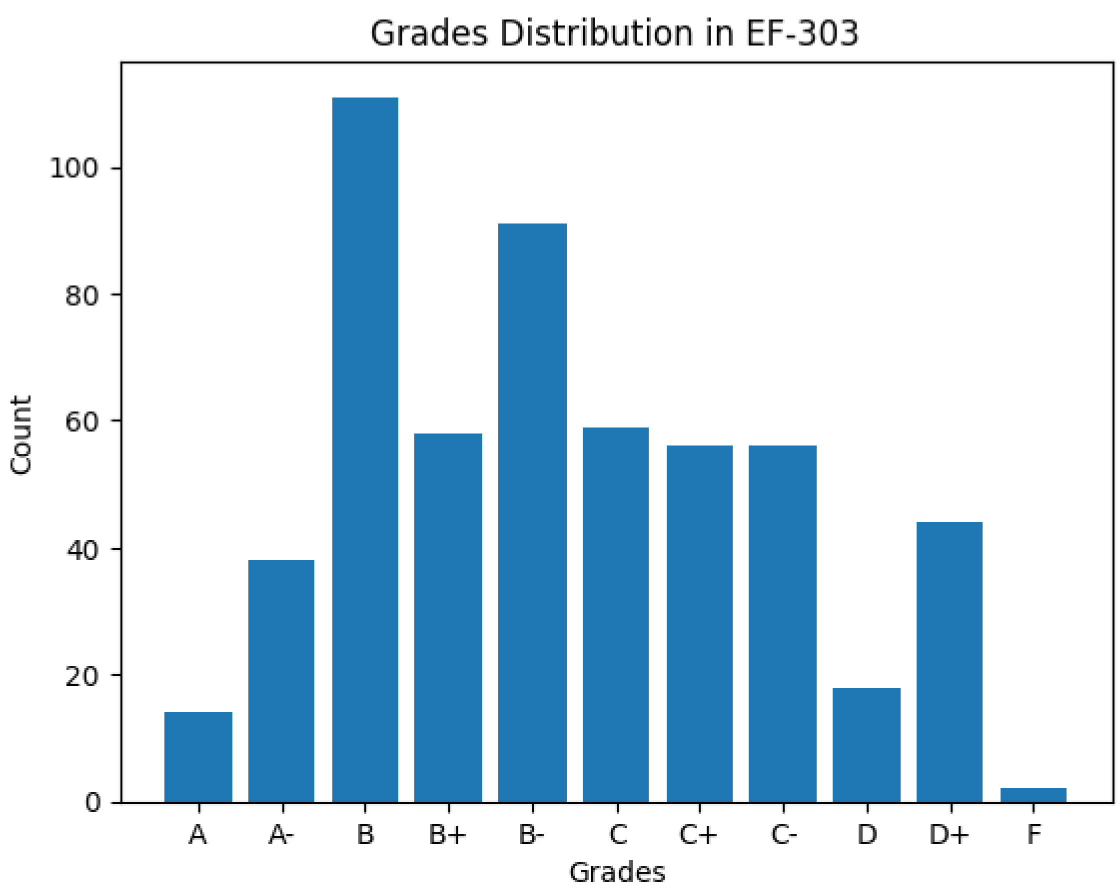

Grades Distribution in CS-214
Count
Grades
0
10
20
30
40
50
60
70
A
A+
A-
B
B+
B-
C
C+
C-
D
D+

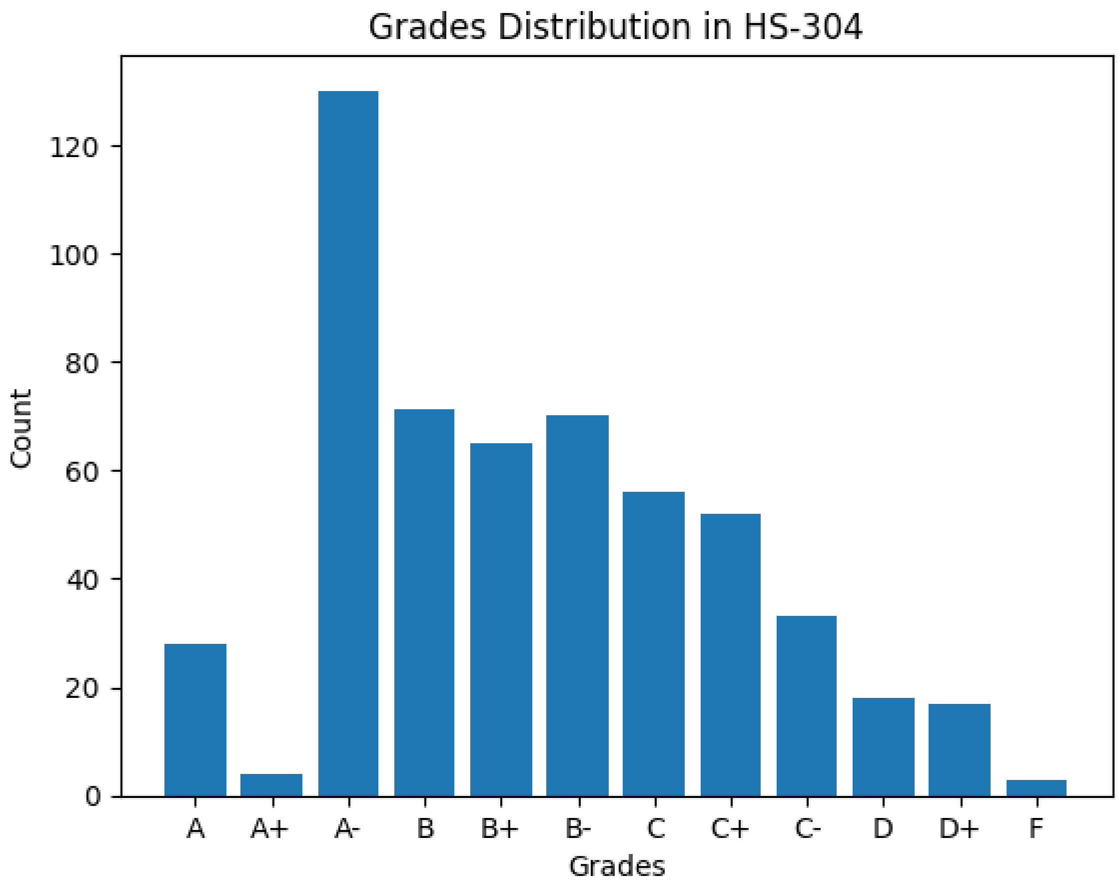

Grades Distribution in EE-217
Count
Grades
0
20
40
60
80
100
120
140
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F

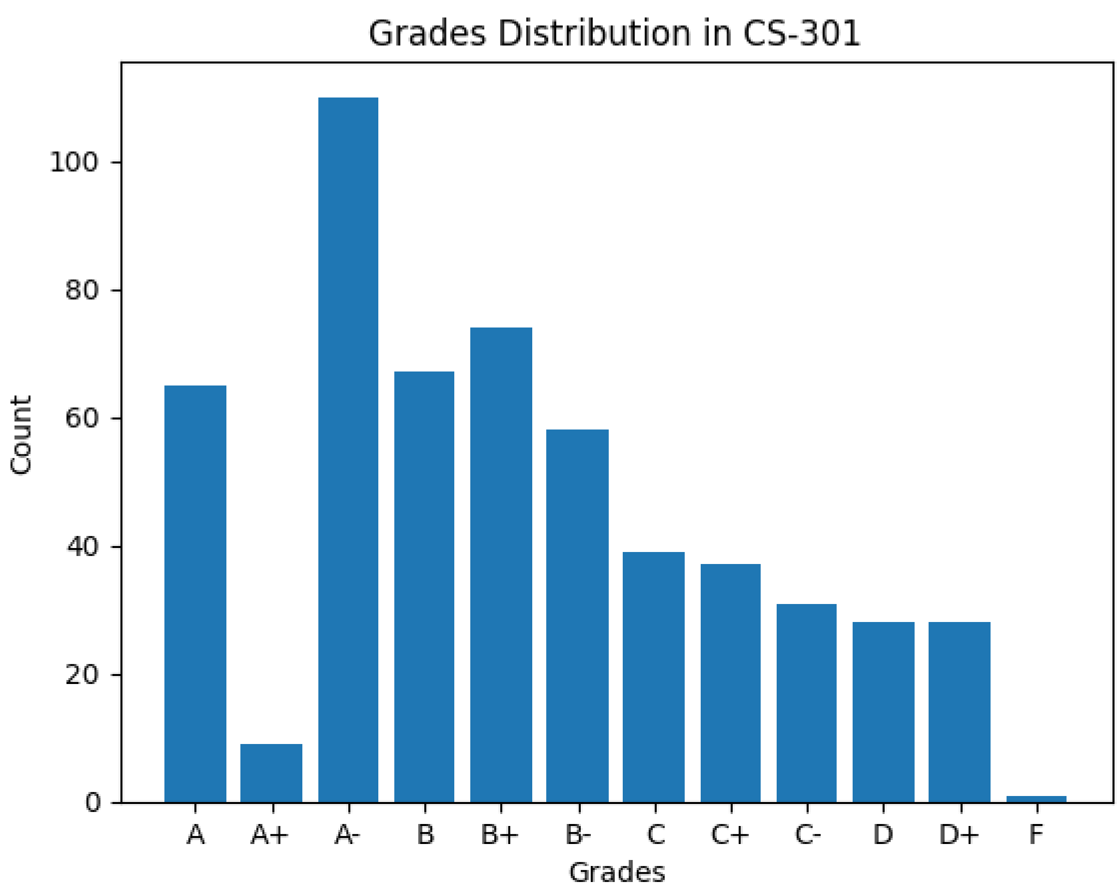

Grades Distribution in CS-212
Count
Grades
0
20
40
60
80
100
A
A+
A-
B
B+
B-
C
C+
C-
D
D+

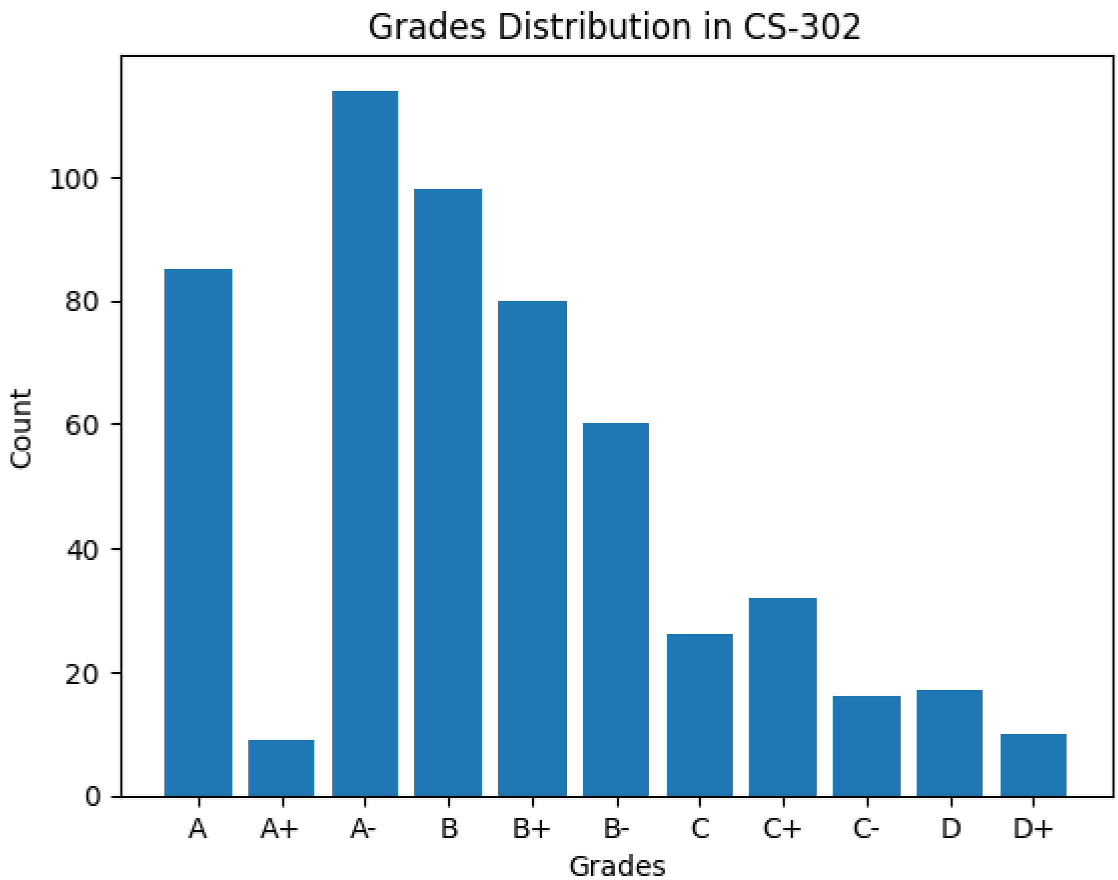

Grades Distribution in CS-215
Count
Grades
0
10
20
30
40
50
60
70
80
A
A+
A-
B
B+
B-
C
C+
C-
D
D+

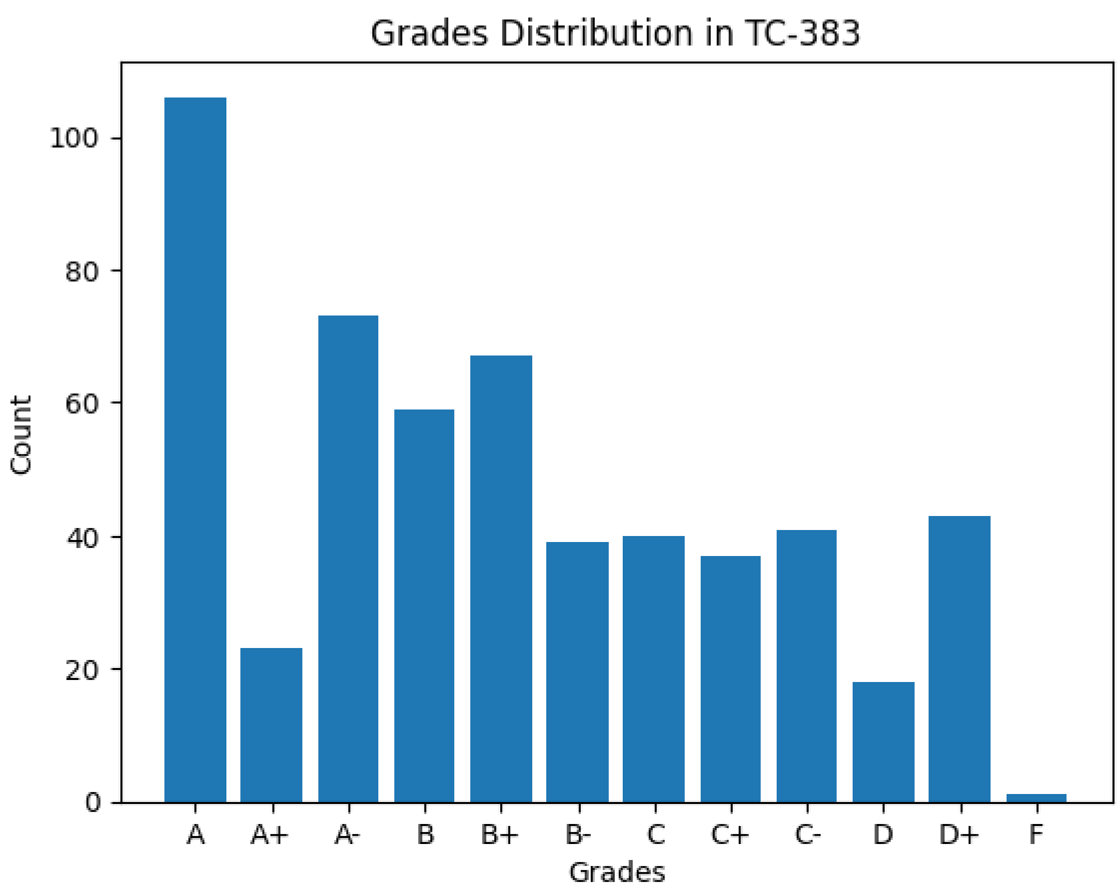

Grades Distribution in MT-331
Count
Grades
0
20
40
60
80
100
120
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F

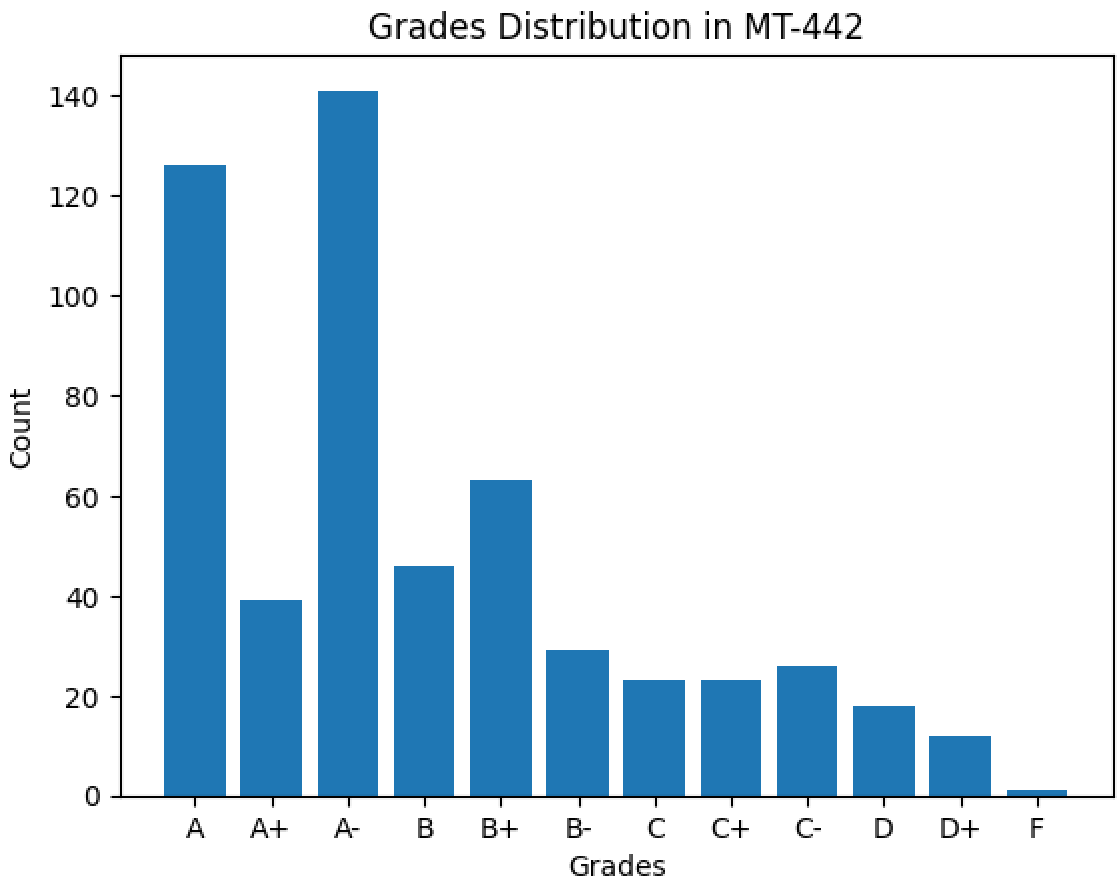

Grades Distribution in EF-303
Count
Grades
0
20
40
60
80
100
A
A-
B
B+
B-
C
C+
C-
D
D+
F

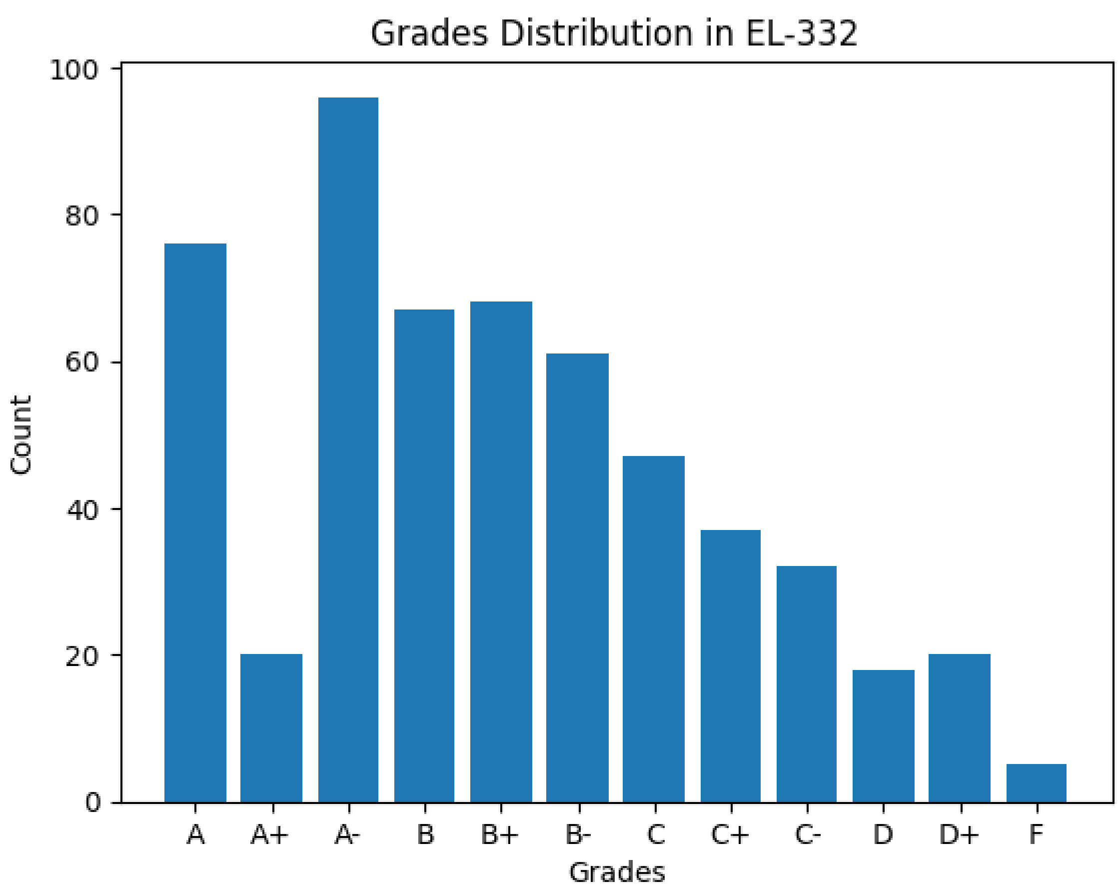

Grades Distribution in HS-304
Count
0
20
40
60
80
100
120
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F
Grades

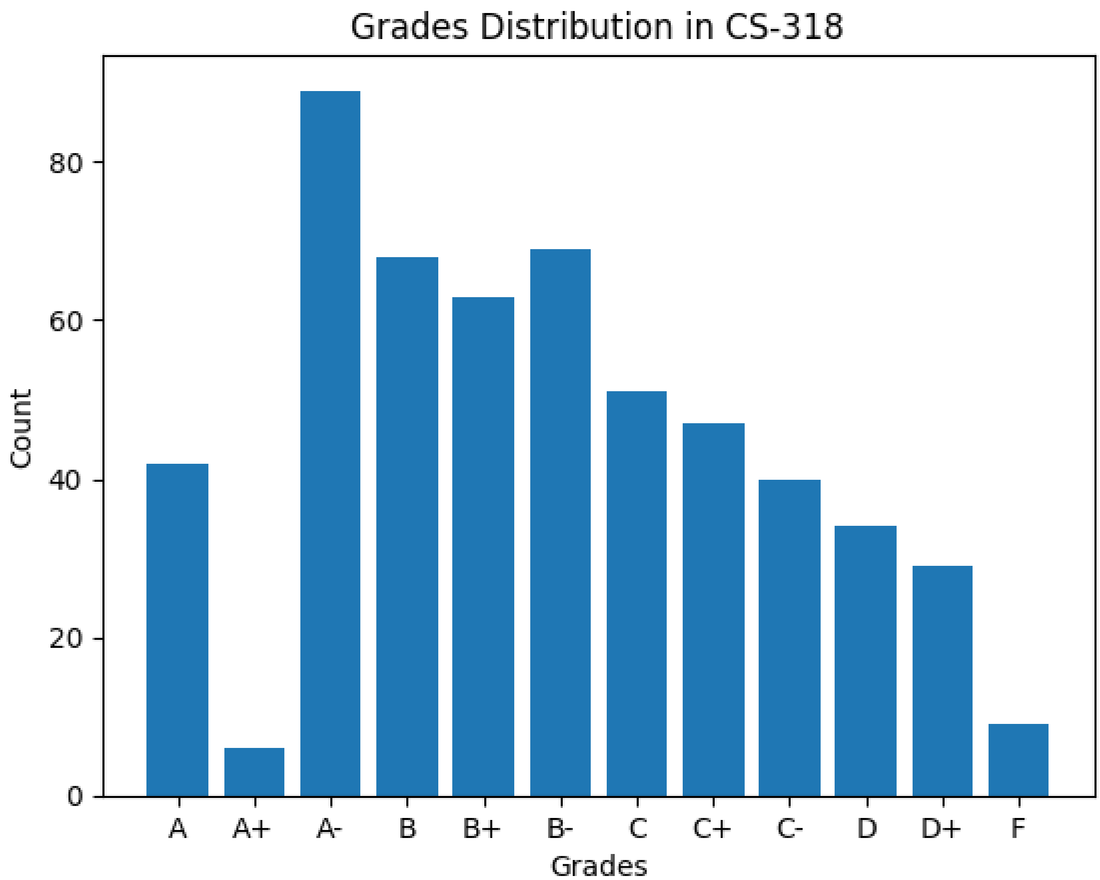

Grades Distribution in CS-301
Count
0
20
40
60
80
100
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F
Grades

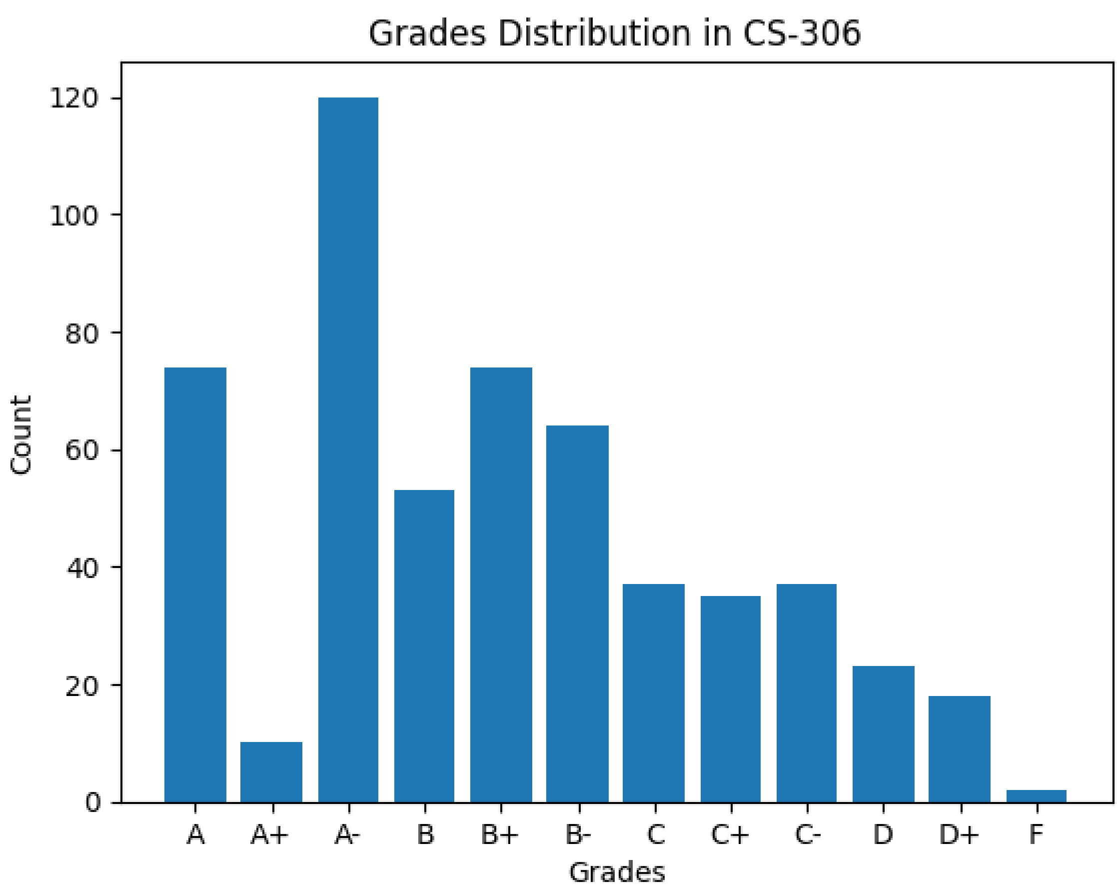

Grades Distribution in CS-302
Count
0
20
40
60
80
100
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
Grades

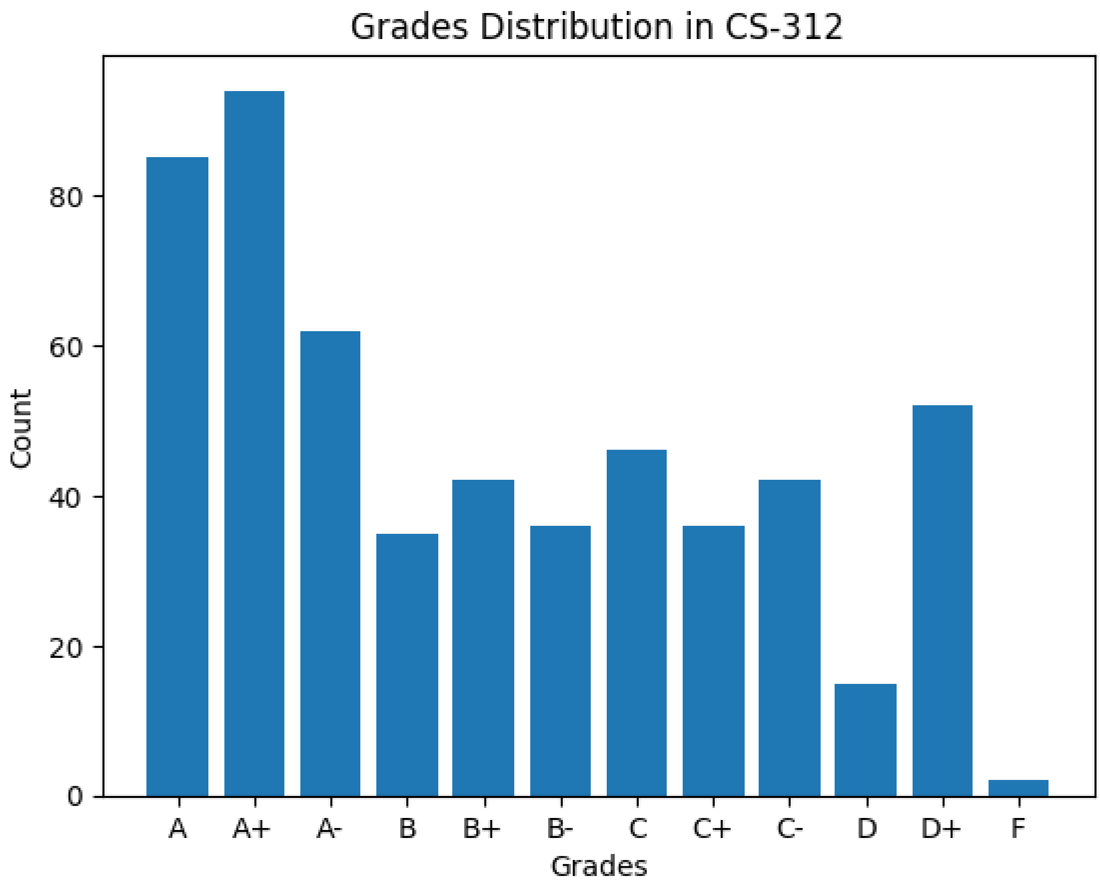

Grades Distribution in TC-383
Count
0
20
40
60
80
100
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F
Grades


Grades Distribution in MT-442
Count
0
20
40
60
80
100
120
140
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F
Grades

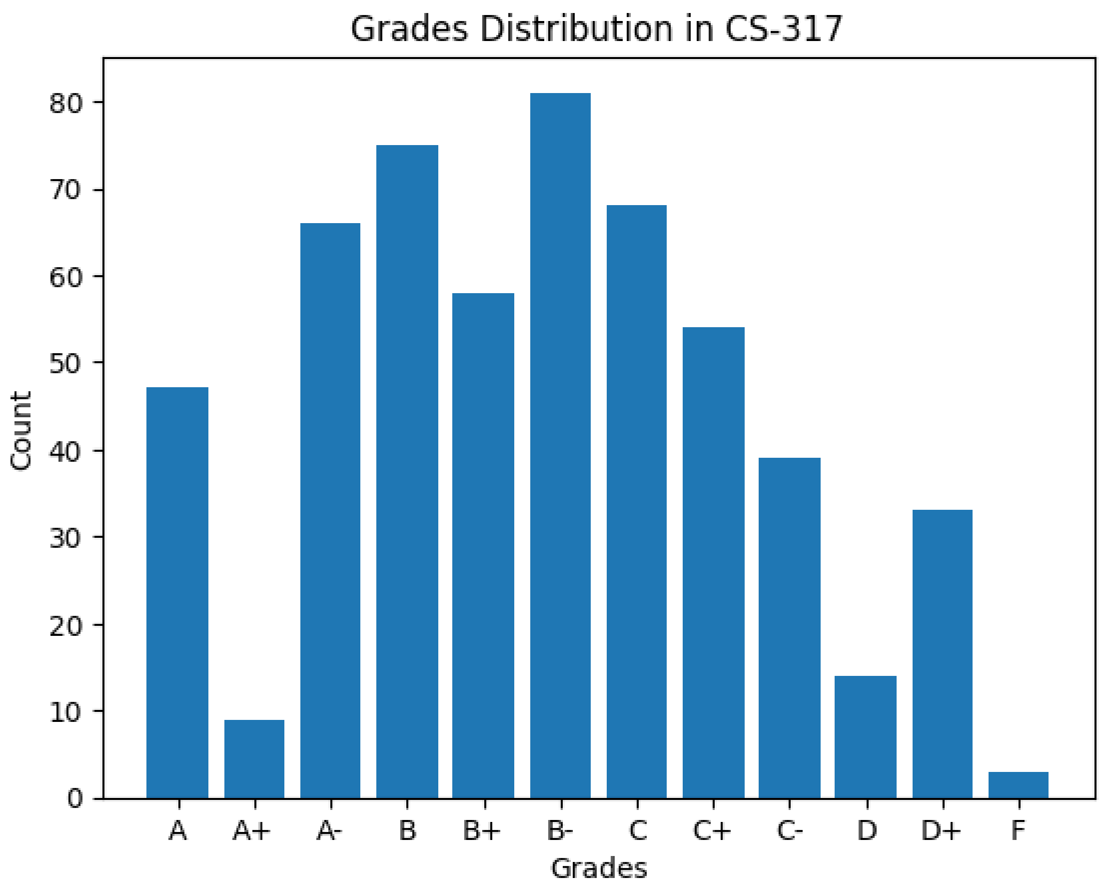

Grades Distribution in EL-332
Count
0
20
40
60
80
100
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F
Grades


Grades Distribution in CS-318
Count
0
20
40
60
80
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F
Grades

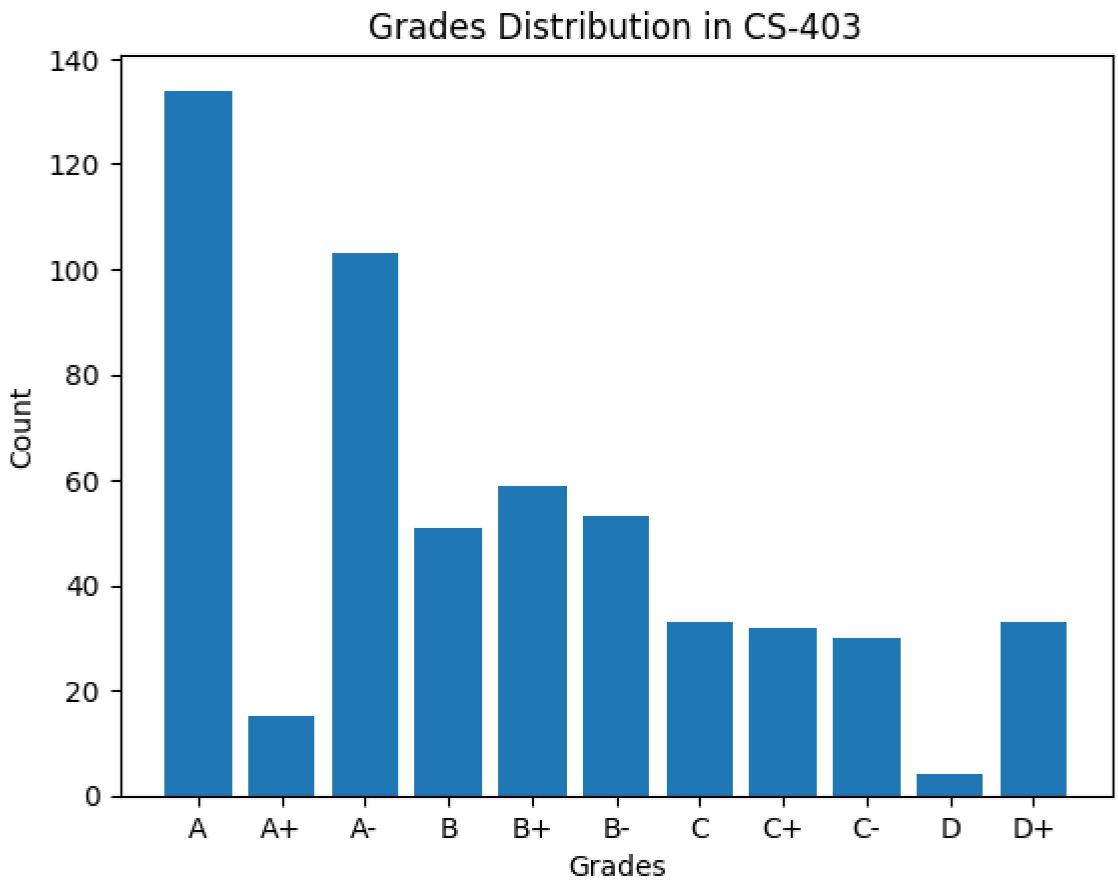

Grades Distribution in CS-306
Count
0
20
40
60
80
100
120
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F
Grades

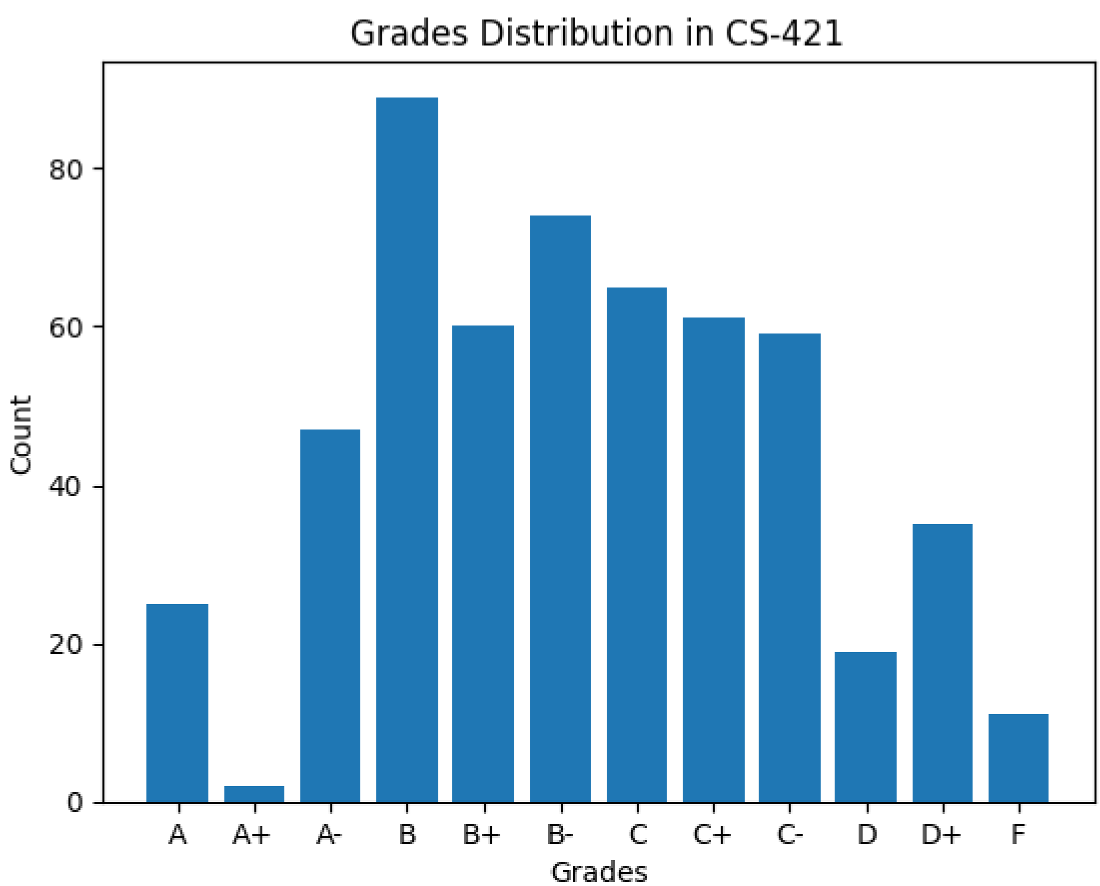


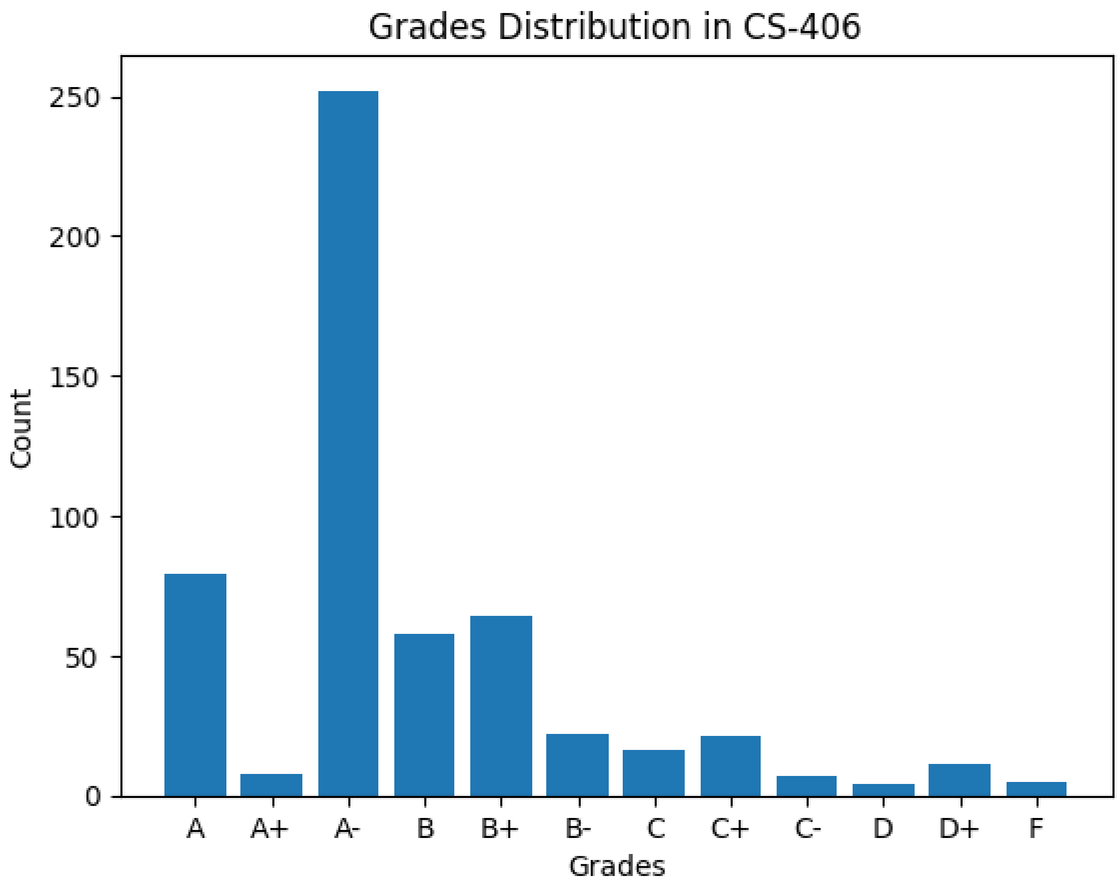
Grades Distribution in CS-312
Count
0
20
40
60
80
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F
Grades

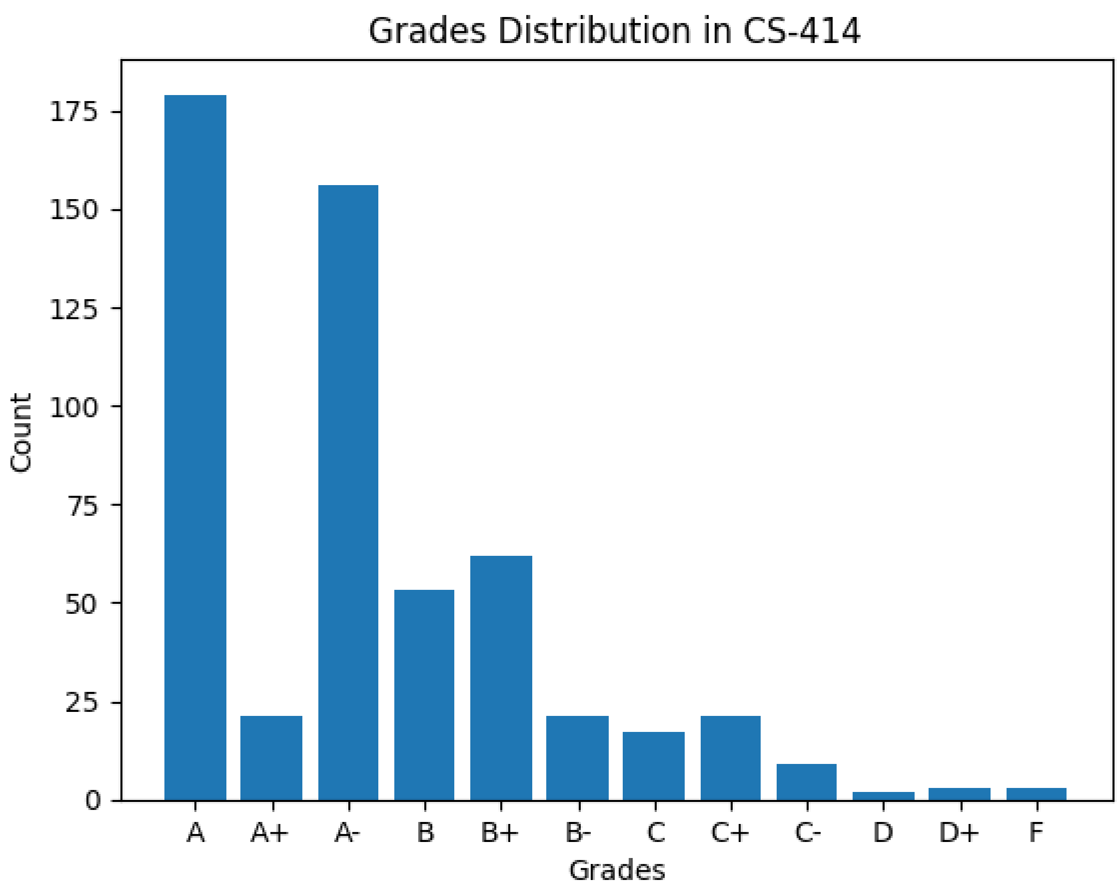
Grades Distribution in CS-317
Count
0
10
20
30
40
50
60
70
80
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F
Grades


Grades Distribution in CS-403
Count
Grades
0
20
40
60
80
100
120
140
A
A+
A-
B
B+
B-
C
C+
C-
D
D+

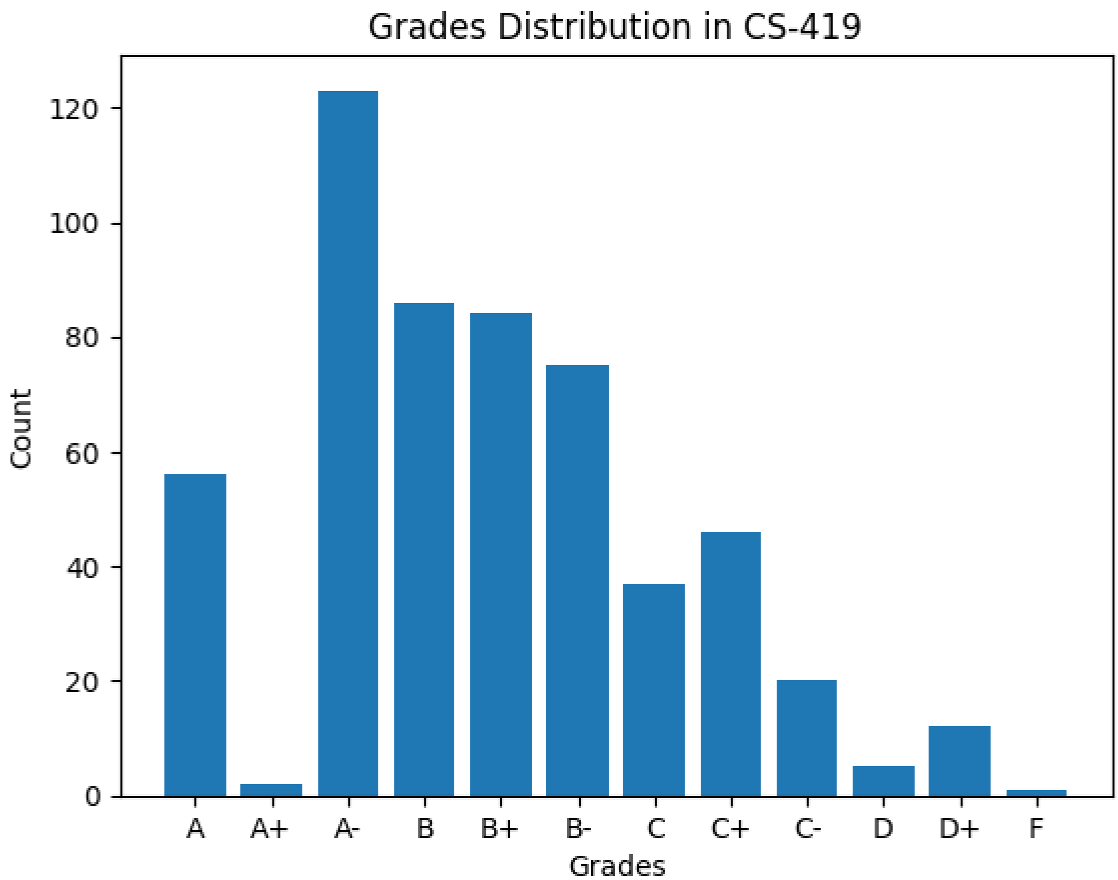

Grades Distribution in CS-421
Count
Grades
0
20
40
60
80
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F



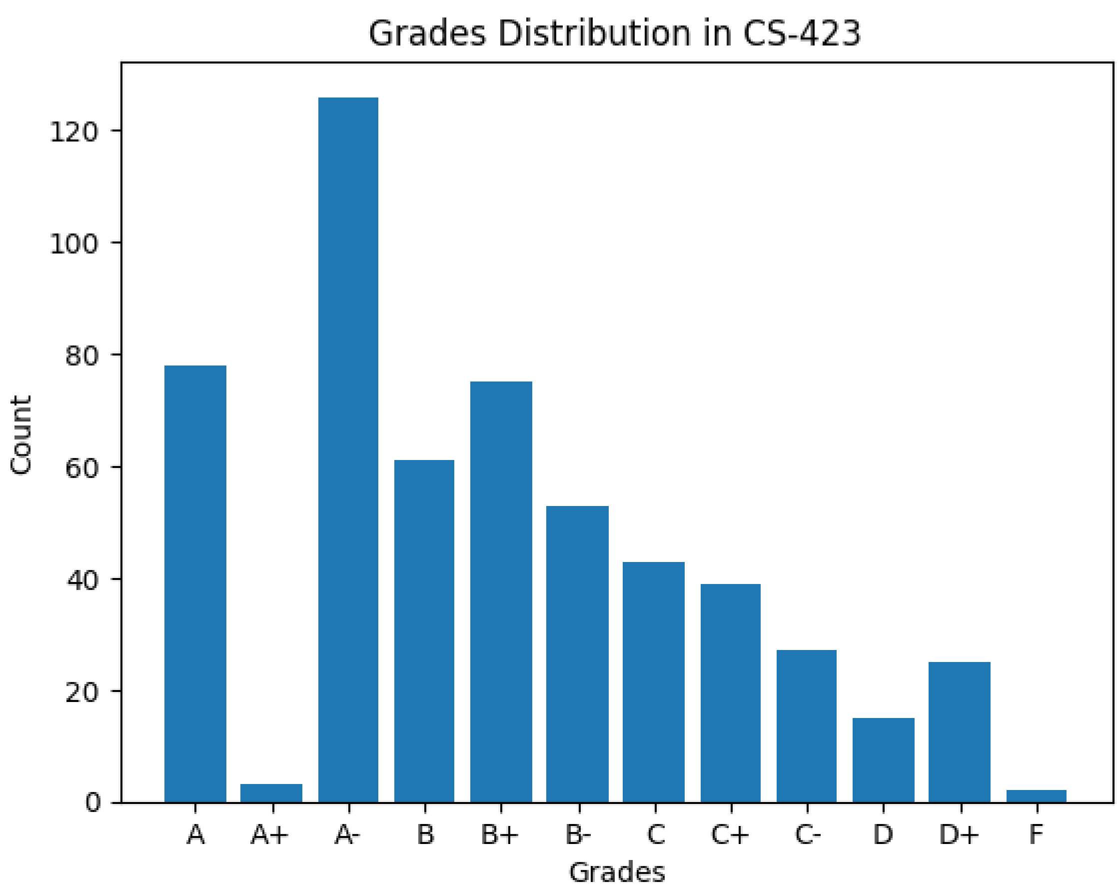
Grades Distribution in CS-406
Count
Grades
0
50
100
150
200
250
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F

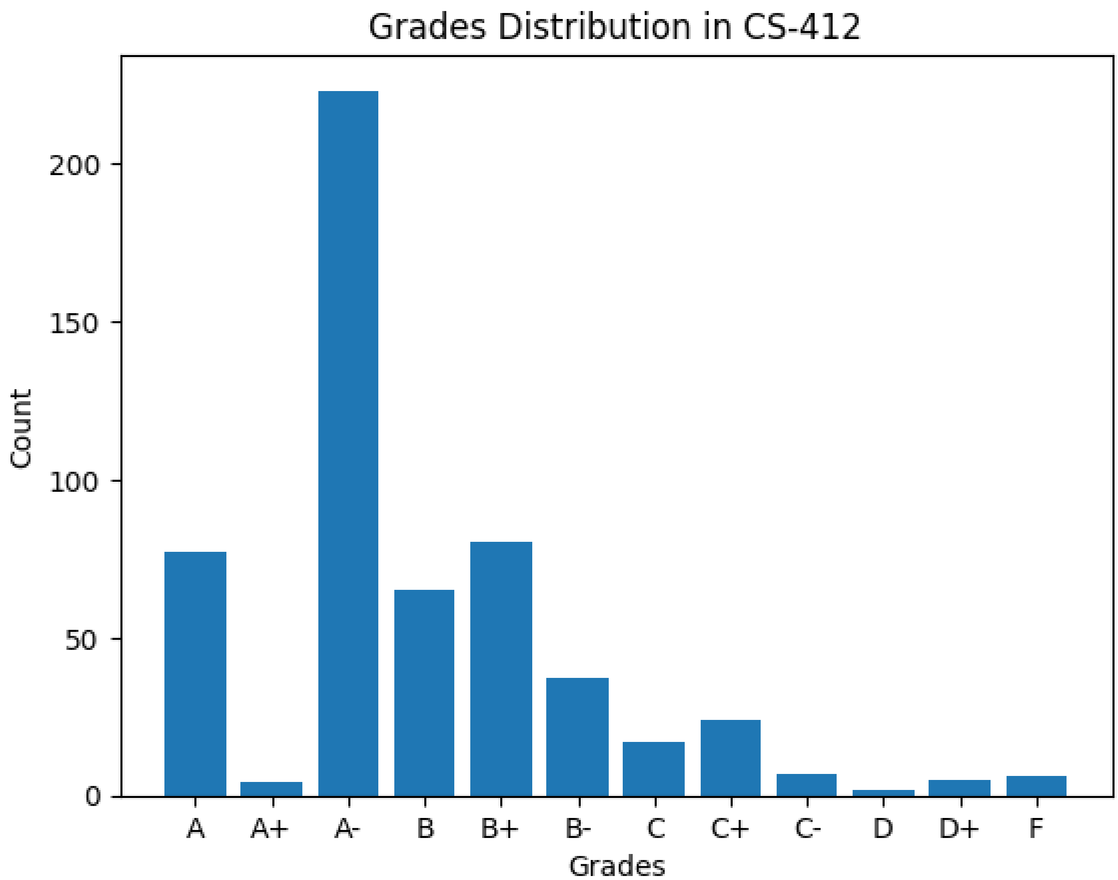
Grades Distribution in CS-414
Count
Grades
0
25
50
75
100
125
150
175
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F

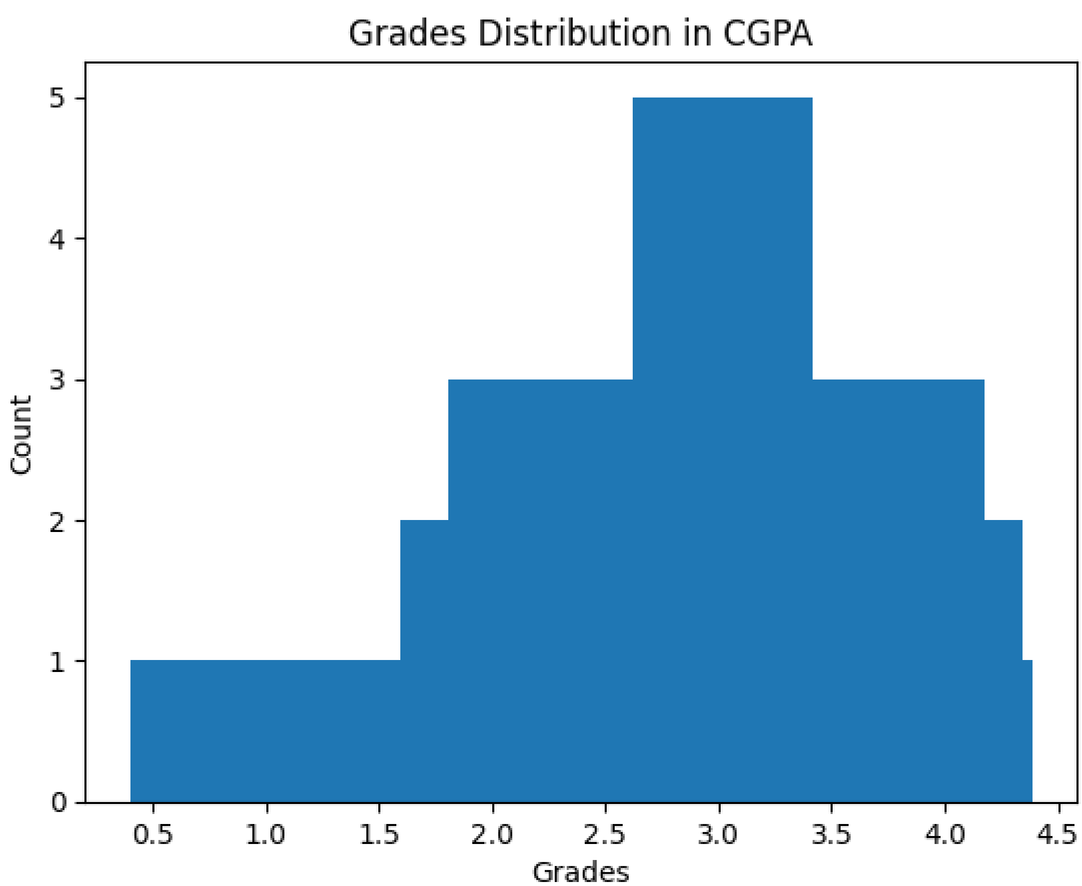


Grades Distribution in CS-419
Count
Grades
0
20
40
60
80
100
120
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F

Grades Distribution in CS-423
Count
Grades
0
20
40
60
80
100
120
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F


Grades Distribution in CS-412
Count
Grades
A
A+
A-
B
B+
B-
C
C+
C-
D
D+
F
0
50
100
150
200


Grades Distribution in CGPA
Count
Grades
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
0
1
2
3
4
5

## Dropping Unnecessary column

In [21]:
```
df = df.drop('Seat No.', axis=1)
```

## Encoding Categorical Values

In [22]:
```
# Define a dictionary to map letter grades to numerical values
grade_mapping = {
    'A+': 4.0,
    'A': 4.0,
    'A-': 3.7,
    'B+': 3.4,
    'B': 3.0,
    'B-': 2.7,
    'C+': 2.4,
    'C': 2.0,
    'C-': 1.7,
    'D+': 1.4,
    'D': 1.0,
    'F': 0.0
}

# Use the replace method to map grades to numerical values for all columns
df = df.replace(grade_mapping)
```

In [23]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 547 entries, 0 to 546
Data columns (total 42 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   PH-121  547 non-null    float64
 1   HS-101  547 non-null    float64
 2   CY-105  547 non-null    float64
 3   HS-105  547 non-null    float64
 4   MT-111  547 non-null    float64
 5   CS-105  547 non-null    float64
 6   CS-106  547 non-null    float64
 7   EL-102  547 non-null    float64
 8   EE-119  547 non-null    float64
 9   ME-107  547 non-null    float64
 10  CS-107  547 non-null    float64
 11  HS-205  547 non-null    float64
 12  MT-222  547 non-null    float64
 13  EE-222  547 non-null    float64
 14  MT-224  547 non-null    float64
 15  CS-210  547 non-null    float64
 16  CS-211  547 non-null    float64
 17  CS-203  547 non-null    float64
 18  CS-214  547 non-null    float64
 19  EE-217  547 non-null    float64
 20  CS-212  547 non-null    float64
 21  CS-215  547 non-null    float64
 22  MT-331  547 non-null    float64
 23  EF-303  547 non-null    float64
 24  HS-304  547 non-null    float64
 25  CS-301  547 non-null    float64
 26  CS-302  547 non-null    float64
 27  TC-383  547 non-null    float64
 28  MT-442  547 non-null    float64
 29  EL-332  547 non-null    float64
 30  CS-318  547 non-null    float64
 31  CS-306  547 non-null    float64
 32  CS-312  547 non-null    float64
 33  CS-317  547 non-null    float64
 34  CS-403  547 non-null    float64
 35  CS-421  547 non-null    float64
 36  CS-406  547 non-null    float64
 37  CS-414  547 non-null    float64
 38  CS-419  547 non-null    float64
 39  CS-423  547 non-null    float64
 40  CS-412  547 non-null    float64
 41  CGPA    547 non-null    float64
dtypes: float64(42)
memory usage: 179.6 KB
```

### Observations

- We have dropped "Seat no." from the dataframe as it wont help us in anyway to predict grades
- We have encoded categorical values to numerical values
- Now you can clear see that all columns has float values
- No missing values in our dataframe as count of all the columns are 547
- Memory usage is 179.6KB

In [24]:
```
#Statistical summary
df.describe()
```

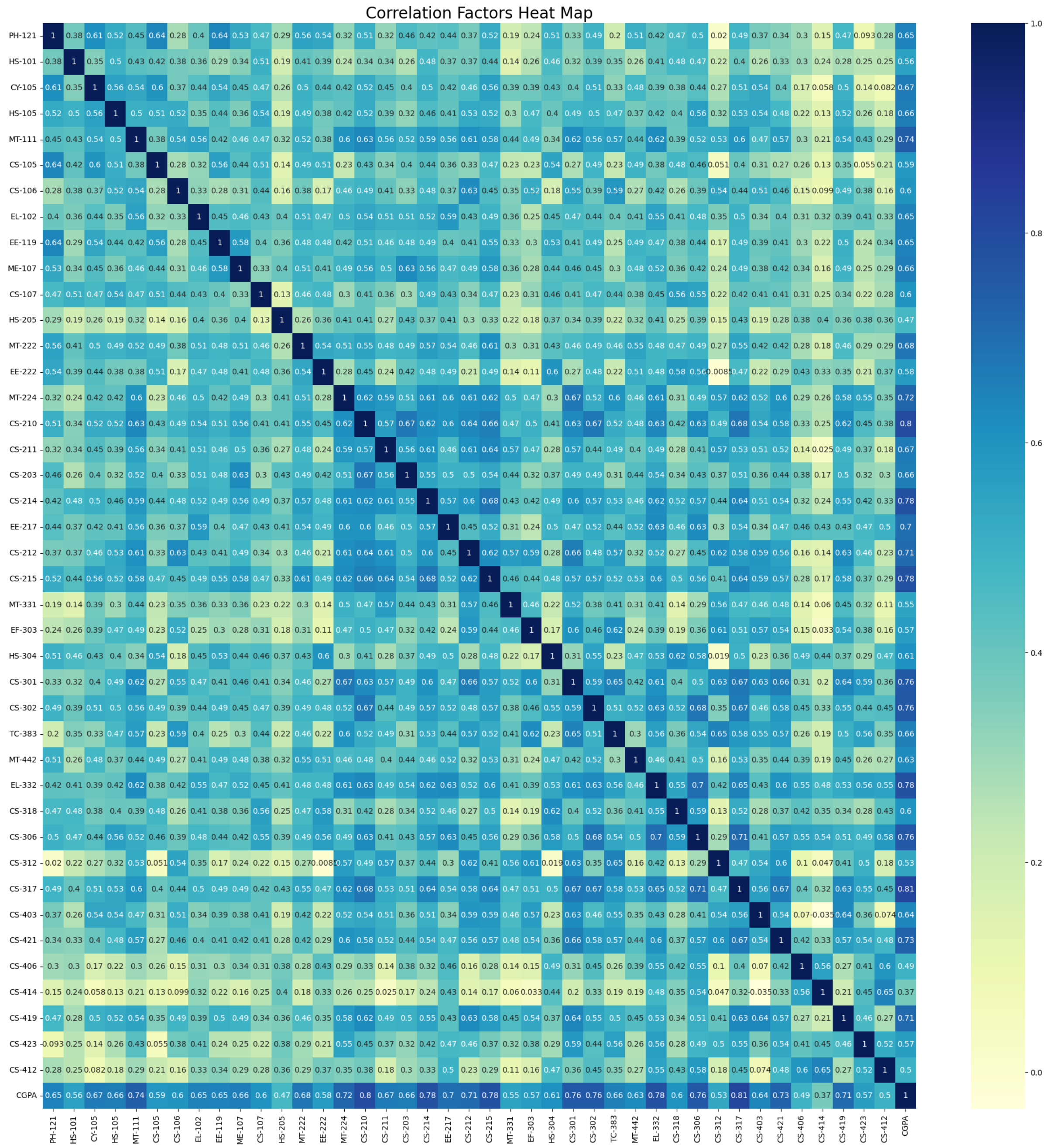
Out[24]:

| | PH-121 | HS-101 | CY-105 | HS-105 | MT-111 | CS-105 | CS-106 | E |
|---|---|---|---|---|---|---|---|---|
| **count** | 547.000000 | 547.000000 | 547.000000 | 547.000000 | 547.000000 | 547.000000 | 547.000000 | 547.0 |
| **mean** | 3.067642 | 2.657952 | 3.310055 | 2.916636 | 3.047166 | 3.366728 | 3.000914 | 3.0 |
| **std** | 0.936537 | 0.898506 | 0.873265 | 0.988782 | 0.896047 | 0.765265 | 0.827071 | 0.9 |
| **min** | 0.000000 | 0.000000 | 1.000000 | 1.000000 | 0.000000 | 1.000000 | 1.000000 | 1.0 |
| **25%** | 2.700000 | 2.000000 | 3.000000 | 2.000000 | 2.400000 | 3.000000 | 2.700000 | 2.4 |
| **50%** | 3.400000 | 2.700000 | 3.700000 | 3.000000 | 3.400000 | 3.700000 | 3.000000 | 3.4 |
| **75%** | 3.700000 | 3.400000 | 4.000000 | 3.700000 | 3.700000 | 4.000000 | 3.700000 | 3.7 |
| **max** | 4.000000 | 4.000000 | 4.000000 | 4.000000 | 4.000000 | 4.000000 | 4.000000 | 4.0 |

8 rows × 42 columns

In [25]:
```python
# visualization of correlation metrix
plt.figure(figsize=(25,25))
sns.heatmap(df.corr(), annot = True, cmap = 'YlGnBu').set_title('Correlation Fa
```

Out[25]: Text(0.5, 1.0, 'Correlation Factors Heat Map')



In [26]:

```
# correlation with label
correlation = df.corr()['CGPA'].drop('CGPA')

# Display the correlation values
print(correlation)
```

```
PH-121    0.645401
HS-101    0.556404
CY-105    0.669354
HS-105    0.660944
MT-111    0.738479
CS-105    0.588444
CS-106    0.597217
EL-102    0.651206
EE-119    0.647128
ME-107    0.659035
CS-107    0.601477
HS-205    0.470703
MT-222    0.680591
EE-222    0.577960
MT-224    0.722883
CS-210    0.799214
CS-211    0.672404
CS-203    0.662619
CS-214    0.782109
EE-217    0.699407
CS-212    0.706259
CS-215    0.780897
MT-331    0.550261
EF-303    0.568559
HS-304    0.608500
CS-301    0.755949
CS-302    0.756661
TC-383    0.655092
MT-442    0.626249
EL-332    0.784013
CS-318    0.603264
CS-306    0.758919
CS-312    0.534566
CS-317    0.814143
CS-403    0.639654
CS-421    0.728702
CS-406    0.494329
CS-414    0.370553
CS-419    0.711185
CS-423    0.565454
CS-412    0.495348
Name: CGPA, dtype: float64
```

### Observations:

- We can clearly visualize that all the columns are highly correlated to each other

## Data Spliting

In [27]:
```
X = df.drop('CGPA', axis=1) # List of all features
y = df['CGPA'] # Data of our label
```

In [28]:
```
X.shape
```

Out[28]:
```
(547, 41)
```

In [29]:
```
y.shape
```

Out[29]:
```
(547,)
```

### Observations:

- we have splitted the data into features and label.
- Now we have 41 features with 547 observations.
- As we y is our target it has only 574 observations

## Feature Scaling

In [30]:
```
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns = X.columns)
X
```

Out[30]:

| | PH-121 | HS-101 | CY-105 | HS-105 | MT-111 | CS-105 | CS-106 | EL-102 | EE-1 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | -0.392914 | -1.401331 | -1.845407 | -0.927884 | -1.504832 | -2.572347 | -2.421491 | -1.420098 | -0.4530 |
| **1** | 0.996450 | -1.846922 | -2.189260 | -1.940155 | -0.387797 | -1.787588 | -2.421491 | 1.054452 | -2.0848 |
| **2** | 0.996450 | 0.381033 | 0.790799 | -0.219295 | 0.394128 | 0.828278 | -0.364164 | 0.408917 | 0.8022 |
| **3** | -2.209774 | -0.287353 | -2.189260 | -1.940155 | -2.286756 | 0.435898 | -1.937414 | -1.420098 | -2.5869 |
| **4** | 0.675827 | 1.160817 | 0.446946 | 0.489295 | 1.064348 | 0.828278 | 0.846029 | 0.408917 | 1.1788 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **542** | -0.072291 | 1.495011 | 0.790799 | 0.792976 | 1.064348 | 0.828278 | 0.846029 | 0.731685 | 1.1788 |
| **543** | 0.996450 | 1.495011 | 0.790799 | 1.096658 | 1.064348 | 0.828278 | 1.209086 | 0.731685 | 1.1788 |
| **544** | -0.072291 | 1.495011 | 0.446946 | 0.489295 | 1.064348 | 0.828278 | 1.209086 | 1.054452 | 1.1788 |
| **545** | 0.996450 | 0.826624 | -2.647731 | 1.096658 | -2.286756 | -2.572347 | -0.364164 | -1.420098 | -0.4530 |
| **546** | -1.141032 | -1.846922 | -2.647731 | -0.927884 | -1.169721 | -2.572347 | -0.001106 | -0.666974 | -1.3316 |

547 rows × 41 columns

## Checking Best Random State

In [31]:
```
maxR2_score=0
maxRS=0
for i in range(1,200):
    X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.33, ranc
    lr=LinearRegression()
    lr.fit(X_train,y_train)
    y_pred=lr.predict(X_test)
    R2=r2_score(y_test,y_pred)
    if R2>maxR2_score:
        maxR2_score=R2
        maxRS=i
print('Best accuracy is', maxR2_score ,'on Random_state', maxRS)
```

```
Best accuracy is 0.9875788176922384 on Random_state 46
```

## Train_ Test_Split

In [32]:
```
x_train,x_test,y_train,y_test= train_test_split(X,y, test_size =0.2, random_sta
```

# Model Training & Testing

## LinearRegression

In [33]:
```
LR = LinearRegression()

# Perform cross-validation
cv_scores = cross_val_score(LR, x_train, y_train, cv=5, scoring='r2')

# Fit the model on the entire training set
LR.fit(x_train, y_train)

# Predict on the test set
y_pred = LR.predict(x_test)

# Print evaluation metrics
print('R2 Score:', r2_score(y_test, y_pred))
print('MAE:', mean_absolute_error(y_test, y_pred))

# Print cross-validation scores
print('Cross-Validation Scores:', cv_scores)
print('Mean Cross-Validation Score:', cv_scores.mean())
```

```
R2 Score: 0.9884894119471499
MAE: 0.05287669778724794
Cross-Validation Scores: [0.98972109 0.97168207 0.77195307 0.97381632 0.97827
864]
Mean Cross-Validation Score: 0.9370902366556093
```

### Ridge Regression

In [34]:
```
# Define the parameter grid
param_grid = {'alpha': [0.1, 1, 10, 100]}  # Example parameter values, you can

# Create the Ridge regression model
R = Ridge()

# Perform grid search with cross-validation
grid_search = GridSearchCV(R, param_grid, cv=5, scoring='r2')
grid_search.fit(x_train, y_train)

# Print the best parameters
print('Best Parameters:', grid_search.best_params_)

# Fit the model with the best parameters on the entire training set
best_R = grid_search.best_estimator_
best_R.fit(x_train, y_train)

# Predict on the test set
y_pred = best_R.predict(x_test)

# Print evaluation metrics
print('R2 Score:', r2_score(y_test, y_pred))
print('MAE:', mean_absolute_error(y_test, y_pred))

# Perform cross-validation with the best model
cv_scores = cross_val_score(best_R, x_train, y_train, cv=5, scoring='r2')

# Print cross-validation scores
print('Cross-Validation Scores:', cv_scores)
print('Mean Cross-Validation Score:', cv_scores.mean())
```

```
Best Parameters: {'alpha': 100}
R2 Score: 0.992129009223353
MAE: 0.040878438346774456
Cross-Validation Scores: [0.99406179 0.98239821 0.78052778 0.98514285 0.99237
896]
Mean Cross-Validation Score: 0.9469019170790579
```

### Lasso Regression

In [35]:
```
# Define the parameter grid
param_grid = {'alpha': [0.1, 1, 10, 100]}  # Example parameter values, you can

# Create the Lasso regression model
L = Lasso()

# Perform grid search with cross-validation
grid_search = GridSearchCV(L, param_grid, cv=5, scoring='r2')
grid_search.fit(x_train, y_train)

# Print the best parameters
print('Best Parameters:', grid_search.best_params_)

# Fit the model with the best parameters on the entire training set
best_L = grid_search.best_estimator_
best_L.fit(x_train, y_train)

# Predict on the test set
y_pred = best_L.predict(x_test)


# Print evaluation metrics
print('R2 Score:', r2_score(y_test, y_pred))
print('MAE:', mean_absolute_error(y_test, y_pred))

# Perform cross-validation with the best model
cv_scores = cross_val_score(best_L, x_train, y_train, cv=5, scoring='r2')

# Print cross-validation scores
print('Cross-Validation Scores:', cv_scores)
print('Mean Cross-Validation Score:', cv_scores.mean())
```

```
Best Parameters: {'alpha': 0.1}
R2 Score: 0.8979205182363923
MAE: 0.16203976784773086
Cross-Validation Scores: [0.90262411 0.88373868 0.72248202 0.87741962 0.90261
066]
Mean Cross-Validation Score: 0.8577750169235688
```

### Decision Tree Regressor

In [36]:
```
# Define the parameter grid
param_grid = {
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'sqrt', 'log2']
}

# Create the Decision Tree Regressor model
DTR = DecisionTreeRegressor()

# Perform grid search with cross-validation
grid_search = GridSearchCV(DTR, param_grid, cv=5, scoring='r2')
grid_search.fit(x_train, y_train)

# Print the best parameters
print('Best Parameters:', grid_search.best_params_)

# Fit the model with the best parameters on the entire training set
best_DTR = grid_search.best_estimator_
best_DTR.fit(x_train, y_train)

# Predict on the test set
y_pred = best_DTR.predict(x_test)

# Print evaluation metrics
print('R2 Score:', r2_score(y_test, y_pred))
print('MAE:', mean_absolute_error(y_test, y_pred))

# Perform cross-validation with the best model
cv_scores = cross_val_score(best_DTR, x_train, y_train, cv=5, scoring='r2')

# Print cross-validation scores
print('Cross-Validation Scores:', cv_scores)
print('Mean Cross-Validation Score:', cv_scores.mean())
```

```
Best Parameters: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_lea
f': 2, 'min_samples_split': 2}
R2 Score: 0.8041219851751172
MAE: 0.18967030893349074
Cross-Validation Scores: [0.84960142 0.82536029 0.62495131 0.8751598  0.80308
116]
Mean Cross-Validation Score: 0.7956307949264849
```

### Random Forest Regressor

In [37]:
```
# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt']
}

# Create the Random Forest Regressor model
RFR = RandomForestRegressor()

# Perform grid search with cross-validation
grid_search = GridSearchCV(RFR, param_grid, cv=5, scoring='r2')
grid_search.fit(x_train, y_train)

# Print the best parameters
print('Best Parameters:', grid_search.best_params_)

# Fit the model with the best parameters on the entire training set
best_RFR = grid_search.best_estimator_
best_RFR.fit(x_train, y_train)

# Predict on the test set
y_pred = best_RFR.predict(x_test)

# Print evaluation metrics
print('R2 Score:', r2_score(y_test, y_pred))
print('MAE:', mean_absolute_error(y_test, y_pred))

# Perform cross-validation with the best model
cv_scores = cross_val_score(best_RFR, x_train, y_train, cv=5, scoring='r2')

# Print cross-validation scores
print('Cross-Validation Scores:', cv_scores)
print('Mean Cross-Validation Score:', cv_scores.mean())
```

```
Best Parameters: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_lea
f': 1, 'min_samples_split': 2, 'n_estimators': 200}
R2 Score: 0.9430268670773639
MAE: 0.09561351183951926
Cross-Validation Scores: [0.96510473 0.95120716 0.73974802 0.96402785 0.97202
262]
Mean Cross-Validation Score: 0.9184220773387425
```

### Extra Tree Regressor

In [38]:
```
# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt']
}

# Create the Extra Trees Regressor model
ETR = ExtraTreesRegressor()

# Perform grid search with cross-validation
grid_search = GridSearchCV(ETR, param_grid, cv=5, scoring='r2')
grid_search.fit(x_train, y_train)

# Print the best parameters
print('Best Parameters:', grid_search.best_params_)

# Fit the model with the best parameters on the entire training set
best_ETR = grid_search.best_estimator_
best_ETR.fit(x_train, y_train)

# Predict on the test set
y_pred = best_ETR.predict(x_test)

# Print evaluation metrics
print('R2 Score:', r2_score(y_test, y_pred))
print('MAE:', mean_absolute_error(y_test, y_pred))

# Perform cross-validation with the best model
cv_scores = cross_val_score(best_ETR, x_train, y_train, cv=5, scoring='r2')

# Print cross-validation scores
print('Cross-Validation Scores:', cv_scores)
print('Mean Cross-Validation Score:', cv_scores.mean())
```

```
Best Parameters: {'max_depth': 20, 'max_features': 'sqrt', 'min_samples_lea
f': 1, 'min_samples_split': 2, 'n_estimators': 300}
R2 Score: 0.9545728090941457
MAE: 0.08417658879268887
Cross-Validation Scores: [0.97292878 0.95777007 0.74884279 0.9704137  0.97293
474]
Mean Cross-Validation Score: 0.9245780143030384
```

## Ada Boost Regressor

In [39]:
```
# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.1, 0.01, 0.001],
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt']
}

# Create the Gradient Boosting Regressor model
GBR = GradientBoostingRegressor()

# Perform grid search with cross-validation
grid_search = GridSearchCV(GBR, param_grid, cv=5, scoring='r2')
grid_search.fit(x_train, y_train)

# Print the best parameters
print('Best Parameters:', grid_search.best_params_)

# Fit the model with the best parameters on the entire training set
best_GBR = grid_search.best_estimator_
best_GBR.fit(x_train, y_train)

# Predict on the test set
y_pred = best_GBR.predict(x_test)

# Print evaluation metrics
print('R2 Score:', r2_score(y_test, y_pred))
print('MAE:', mean_absolute_error(y_test, y_pred))

# Perform cross-validation with the best model
cv_scores = cross_val_score(best_GBR, x_train, y_train, cv=5, scoring='r2')

# Print cross-validation scores
print('Cross-Validation Scores:', cv_scores)
print('Mean Cross-Validation Score:', cv_scores.mean())
```

```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'max_features': 'sqr
t', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}
R2 Score: 0.9624314018087885
MAE: 0.08048523532908322
Cross-Validation Scores: [0.97537463 0.95857745 0.77329232 0.97042696 0.96666
871]
Mean Cross-Validation Score: 0.9288680150630805
```

### XGB Regressor

In [41]:
```
# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.1, 0.01, 0.001],
    'max_depth': [3, 5, 10],
    'min_child_weight': [1, 3, 5],
    'gamma': [0, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

# Create the XGBoost Regressor model
XGBR = XGBRegressor()

# Perform grid search with cross-validation
grid_search = GridSearchCV(XGBR, param_grid, cv=5, scoring='r2')
grid_search.fit(x_train, y_train)

# Print the best parameters
print('Best Parameters:', grid_search.best_params_)

# Fit the model with the best parameters on the entire training set
best_XGBR = grid_search.best_estimator_
best_XGBR.fit(x_train, y_train)

# Predict on the test set
y_pred = best_XGBR.predict(x_test)

# Print evaluation metrics
print('R2 Score:', r2_score(y_test, y_pred))
print('MAE:', mean_absolute_error(y_test, y_pred))

# Perform cross-validation with the best model
cv_scores = cross_val_score(best_XGBR, x_train, y_train, cv=5, scoring='r2')

# Print cross-validation scores
print('Cross-Validation Scores:', cv_scores)
print('Mean Cross-Validation Score:', cv_scores.mean())
```

```
Best Parameters: {'colsample_bytree': 0.8, 'gamma': 0, 'learning_rate': 0.1,
'max_depth': 3, 'min_child_weight': 1, 'n_estimators': 300, 'subsample': 0.8}
R2 Score: 0.9680914803790569
MAE: 0.06646585891897029
Cross-Validation Scores: [0.97257635 0.947726   0.75845435 0.96550984 0.96851
745]
Mean Cross-Validation Score: 0.9225568005070667
```

## Saving The Model

In [42]:
```
filename = 'LinearRegression.pkl'
pickle.dump(LR, open(filename, 'wb'))
pickle.dump(scaler, open('scaler.pkl','wb'))
```

In [43]:
```
with open('LinearRegression.pkl', 'rb') as file:
    data = pickle.load(file)
```

In [44]:
```
data.predict(X_test)
```

Out[44]:
```
array([3.77068565, 3.75407909, 2.54638351, 2.09360857, 3.79929245,
       2.94961198, 2.09316688, 3.23065178, 2.9072488 , 2.03583939,
       3.62151759, 3.01708529, 3.54714463, 2.80159418, 3.42954448,
       3.7285125 , 3.14013385, 3.94642353, 2.82604138, 2.07526458,
       2.63434344, 3.08805893, 2.81268765, 3.55520011, 3.48156609,
       3.23621862, 3.20573641, 1.97585951, 3.1965465 , 3.19690244,
       3.32649891, 3.08564728, 2.35530664, 2.5222813 , 3.37919198,
       3.70367376, 3.51162938, 2.74767634, 1.89004226, 2.75939195,
       3.3905639 , 2.91442493, 2.2607173 , 2.79347019, 3.59859323,
       2.64477587, 2.11762028, 2.91221773, 3.36521106, 3.34981952,
       3.70027887, 3.21621348, 3.71687807, 3.43379881, 2.05197933,
       2.70001763, 2.80321349, 3.68061392, 2.75416651, 3.44985303,
       2.01609771, 2.49585468, 3.22180723, 2.66438405, 3.28641425,
       2.69450312, 3.18749046, 2.69617306, 3.7197727 , 2.79241047,
       2.68502294, 2.6333894 , 2.31021338, 3.85124306, 2.21982604,
       1.67976166, 3.22778617, 3.42832287, 2.60992268, 3.5804398 ,
       3.91321072, 2.36095309, 3.23243762, 3.64396685, 2.37045648,
       2.95291942, 3.01967734, 3.40840739, 3.73934669, 3.51014454,
       3.45246303, 3.33509822, 3.31931864, 2.88377436, 3.02355165,
       2.39303435, 2.34369662, 3.65112597, 2.07028116, 3.37433738,
       3.34307375, 2.97092875, 3.23823953, 2.95603269, 2.25407779,
       2.25122401, 2.71968032, 3.42939103, 2.81673461, 3.52555136,
       3.28047123, 2.4037378 , 3.33179508, 2.50166223, 2.44402452,
       3.4338544 , 2.97940816, 3.28026008, 3.72369396, 3.75522856,
       2.60246276, 2.79758156, 3.06762496, 3.07697157, 2.65655002,
       2.66119539, 2.510536  , 2.64403298, 3.3414724 , 3.31516961,
       2.26097537, 1.94256418, 2.11220247, 2.09339029, 2.50179947,
       2.78796   , 3.08631393, 1.88918392, 1.77854538, 3.52114735,
       2.92457503, 2.93211482, 3.15352026, 3.00933502, 2.49063925,
       3.3524095 , 2.37224261, 2.41506061, 3.04507289, 3.66893984,
       3.17324682, 3.21287961, 3.6440449 , 3.56021996, 2.89916625,
       2.75021942, 3.43468634, 2.58246441, 3.87106498, 3.576756  ,
       3.68477213, 2.98628938, 2.68814171, 3.46140601, 2.60150606,
       3.28057787, 2.72198083, 2.09389936, 3.79780698, 3.90488737,
       1.95304486, 3.10633644, 2.32663802, 3.01475513, 2.43784521,
       2.10324206, 2.37956128, 2.8256216 , 2.91374611, 3.63339214,
       3.26468148])
```

In [ ]:
```

```