In [277]:

```
import pandas as pd
df=pd.read_excel("MIDMARKS-MINOR1-EXAM.xlsx")
df
```

Out[277]:

```
     S.NO  SECTION  DV  M-II  PP  BEEE  FL  FIMS
  0     1    ALPHA  12     0  17     9  19    15
  1     2    ALPHA  19    12  16    16  18     3
  2     3    ALPHA  18    14  18    18  18    16
  3     4    ALPHA  15     9  19    17  19    15
  4     5    ALPHA  18    17  19    19  20    18
...   ...      ...  ...  ...  ...   ...  ...   ...
475   476      NaN  18     2  12     3  17    15
476   477      NaN  20     6  16    11  20    14
477   478      NaN  20   NaN  18    13  20    18
478   479      NaN  20    20   5    19  18    14
479   480      NaN  20    16  18    19  20    19

480 rows × 8 columns
```

In [278]:

```
df.head()
```

Out[278]:

```
   S.NO  SECTION  DV  M-II  PP  BEEE  FL  FIMS
0     1    ALPHA  12     0  17     9  19    15
1     2    ALPHA  19    12  16    16  18     3
2     3    ALPHA  18    14  18    18  18    16
3     4    ALPHA  15     9  19    17  19    15
4     5    ALPHA  18    17  19    19  20    18
```

```
In [279]: df.tail()
```

Out[279]:

	S.NO	SECTION	DV	M-II	PP	BEEE	FL	FIMS
475	476	NaN	18	2	12	3	17	15
476	477	NaN	20	6	16	11	20	14
477	478	NaN	20	NaN	18	13	20	18
478	479	NaN	20	20	5	19	18	14
479	480	NaN	20	16	18	19	20	19

```
In [280]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 480 entries, 0 to 479
Data columns (total 8 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   S.NO     480 non-null    int64 
 1   SECTION  439 non-null    object
 2   DV       479 non-null    object
 3   M-II     477 non-null    object
 4   PP       480 non-null    object
 5   BEEE     478 non-null    object
 6   FL       479 non-null    object
 7   FIMS     480 non-null    object
dtypes: int64(1), object(7)
memory usage: 30.1+ KB
```

In [281]:

```
df.describe()
```

Out[281]:

	S.NO
count	480.000000
mean	240.500000
std	138.708327
min	1.000000
25%	120.750000
50%	240.500000
75%	360.250000
max	480.000000

In [282]:

```
df.isnull().sum()
```

Out[282]:

```
S.NO        0
SECTION    41
DV          1
M-II        3
PP          0
BEEE        2
FL          1
FIMS        0
dtype: int64
```

```
In [283]: df.dtypes

Out[283]: S.NO        int64
          SECTION    object
          DV         object
          M-II       object
          PP         object
          BEEE       object
          FL         object
          FIMS       object
          dtype: object
```

```
In [284]: df.shape

Out[284]: (480, 8)
```

```
In [285]: len(df)

Out[285]: 480
```

```
In [286]: len(df.columns)

Out[286]: 8
```

```
In [287]: df.replace('A',0,inplace=True)
          df.replace('AB',0,inplace=True)
          df.replace('MP',0,inplace=True)
          df.replace('I',1,inplace=True)
          df.replace('II',11,inplace=True)
          df.replace('o',0,inplace=True)
```

```
In [288]: df.iloc[156]
```

```
Out[288]: S.NO           157
          SECTION      DELTA
          DV            15.0
          M-II           5.0
          PP               0
          BEEE           0.0
          FL            20.0
          FIMS            15
          Name: 156, dtype: object
```

```
In [289]: df.iloc[173]
```

```
Out[289]: S.NO           174
          SECTION      DELTA
          DV             0.0
          M-II           0.0
          PP              16
          BEEE          10.0
          FL            20.0
          FIMS            19
          Name: 173, dtype: object
```

```
In [290]: df.iloc[192]
```

```
Out[290]: S.NO             193
          SECTION      EPSILON
          DV              16.0
          M-II            18.0
          PP                15
          BEEE             NaN
          FL              18.0
          FIMS              18
          Name: 192, dtype: object
```

In [291]:
```
df.iloc[378]
```

Out[291]:
```
S.NO        379
SECTION    ZETA
DV          8.0
M-II        0.0
PP            2
BEEE        6.0
FL         15.0
FIMS          8
Name: 378, dtype: object
```

In [292]:
```
df.iloc[293]
```

Out[292]:
```
S.NO         294
SECTION    GAMMA
DV          16.0
M-II        11.0
PP            11
BEEE        19.0
FL          15.0
FIMS          13
Name: 293, dtype: object
```

In [293]:
```
df.iloc[366]
```

Out[293]:
```
S.NO        367
SECTION    ZETA
DV         11.0
M-II        1.0
PP            7
BEEE       19.0
FL         15.0
FIMS         11
Name: 366, dtype: object
```

```
In [294]: df.columns
```

```
Out[294]: Index(['S.NO', 'SECTION', 'DV', 'M-II', 'PP', 'BEEE', 'FL', 'FIMS'], dtype='object')
```

```
In [295]: df.value_counts
```

```
Out[295]: <bound method DataFrame.value_counts of      S.NO SECTION    DV  M-II  PP  BEEE    FL  FIMS
0       1   ALPHA  12.0   0.0  17   9.0  19.0    15
1       2   ALPHA  19.0  12.0  16  16.0  18.0     3
2       3   ALPHA  18.0  14.0  18  18.0  18.0    16
3       4   ALPHA  15.0   9.0  19  17.0  19.0    15
4       5   ALPHA  18.0  17.0  19  19.0  20.0    18
..    ...     ...   ...   ...  ..   ...   ...   ...
475   476     NaN  18.0   2.0  12   3.0  17.0    15
476   477     NaN  20.0   6.0  16  11.0  20.0    14
477   478     NaN  20.0   NaN  18  13.0  20.0    18
478   479     NaN  20.0  20.0   5  19.0  18.0    14
479   480     NaN  20.0  16.0  18  19.0  20.0    19

[480 rows x 8 columns]>
```

In [321]:

```
# List of subject columns
subjects = ["DV", "M-II", "PP", "BEEE", "FL", "FIMS"]

# Loop through each row
for index, row in df.iterrows():
    # Compute mean of available (non-null) marks
    valid_marks = row[subjects].dropna()

    if not valid_marks.empty:  # Ensure there are valid marks to compute the mean
        row_mean = valid_marks.mean()

        # Fill missing values with the row-wise mean
        for subject in subjects:
            if pd.isna(row[subject]):  # Check if the value is missing
                df.at[index, subject] = row_mean  # Assign row-wise mean

# Convert marks back to integers
df[subjects] = df[subjects].astype(int)

print("Missing marks filled correctly with row-wise mean.")
```

```
Missing marks filled correctly with row-wise mean.
```

In [323]:

```
df.iloc[477]
```

Out[323]:

```
S.NO       478
SECTION    NaN
DV          20
M-II        17
PP          18
BEEE        13
FL          20
FIMS        18
Name: 477, dtype: object
```

In [298]:

```
subjects = ["DV", "M-II", "PP", "BEEE", "FL", "FIMS"]
df[subjects] = df[subjects].apply(pd.to_numeric, errors='coerce').fillna(0).astype(int)
df
```

Out[298]:

	S.NO	SECTION	DV	M-II	PP	BEEE	FL	FIMS
0	1	ALPHA	12	0	17	9	19	15
1	2	ALPHA	19	12	16	16	18	3
2	3	ALPHA	18	14	18	18	18	16
3	4	ALPHA	15	9	19	17	19	15
4	5	ALPHA	18	17	19	19	20	18
...	...	...	...	...	...	...	...	...
475	476	NaN	18	2	12	3	17	15
476	477	NaN	20	6	16	11	20	14
477	478	NaN	20	17	18	13	20	18
478	479	NaN	20	20	5	19	18	14
479	480	NaN	20	16	18	19	20	19

480 rows × 8 columns

In [325]:

```
df.dtypes
```

Out[325]:

```
S.NO         int64
SECTION     object
DV           int32
M-II         int32
PP           int32
BEEE         int32
FL           int32
FIMS         int32
dtype: object
```

In [329]:

```
# List of sections in order
sections = ["ALPHA", "BETA", "DELTA", "EPSILON", "GAMMA", "OMEGA", "SIGMA", "ZETA"]

# Assign section names in blocks of 60 records
df["SECTION"] = [sections[i // 60] for i in range(len(df))]

print("Missing SECTION values filled correctly.")
df
```

```
Missing SECTION values filled correctly.
```

Out[329]:

	S.NO	SECTION	DV	M-II	PP	BEEE	FL	FIMS
0	1	ALPHA	12	0	17	9	19	15
1	2	ALPHA	19	12	16	16	18	3
2	3	ALPHA	18	14	18	18	18	16
3	4	ALPHA	15	9	19	17	19	15
4	5	ALPHA	18	17	19	19	20	18
...	...	...	...	...	...	...	...	...
475	476	ZETA	18	2	12	3	17	15
476	477	ZETA	20	6	16	11	20	14
477	478	ZETA	20	17	18	13	20	18
478	479	ZETA	20	20	5	19	18	14
479	480	ZETA	20	16	18	19	20	19

480 rows × 8 columns

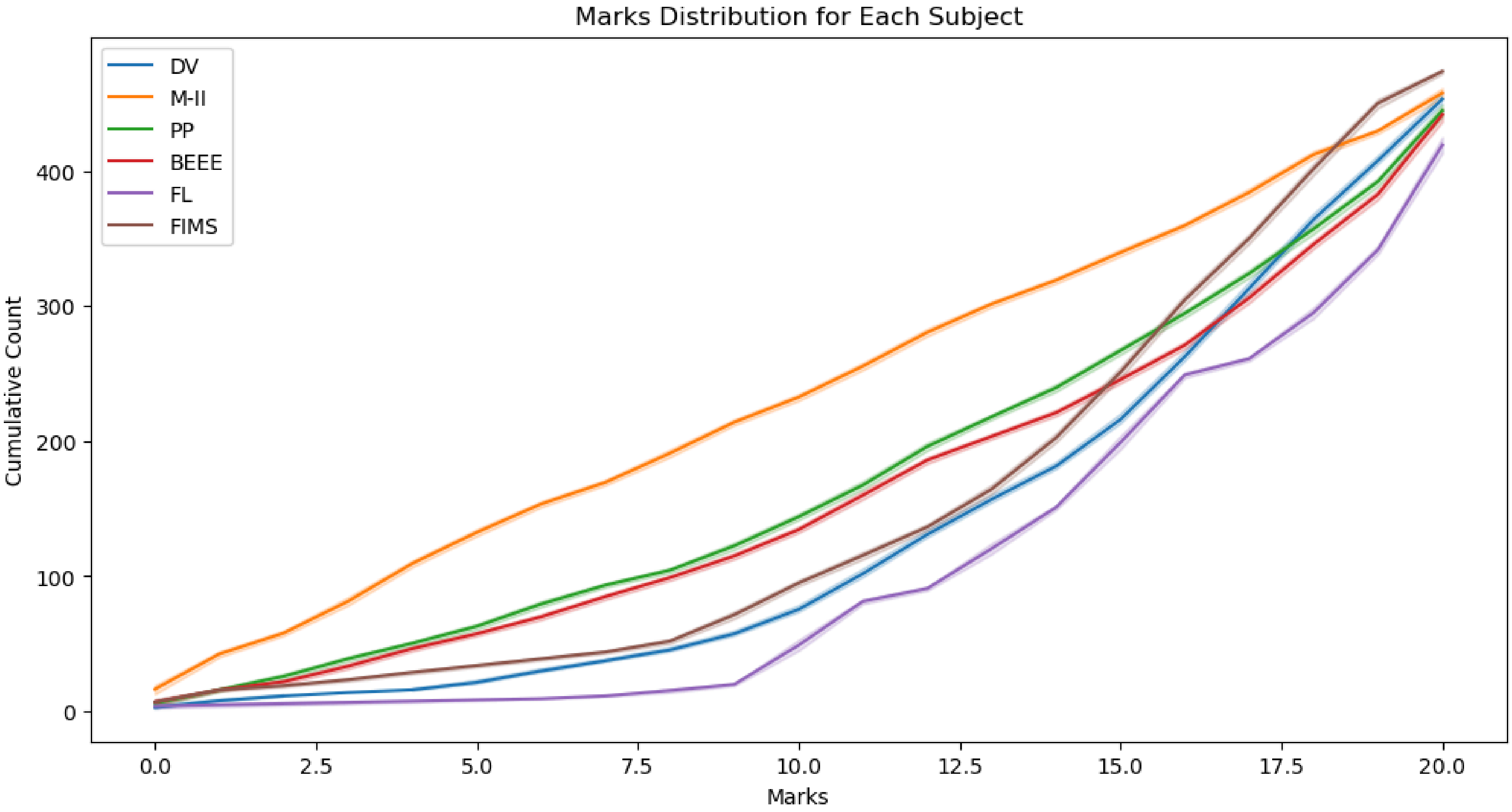
```
In [331]: df.isnull().sum()
```

```
Out[331]: S.NO       0
          SECTION    0
          DV         0
          M-II       0
          PP         0
          BEEE       0
          FL         0
          FIMS       0
          dtype: int64
```

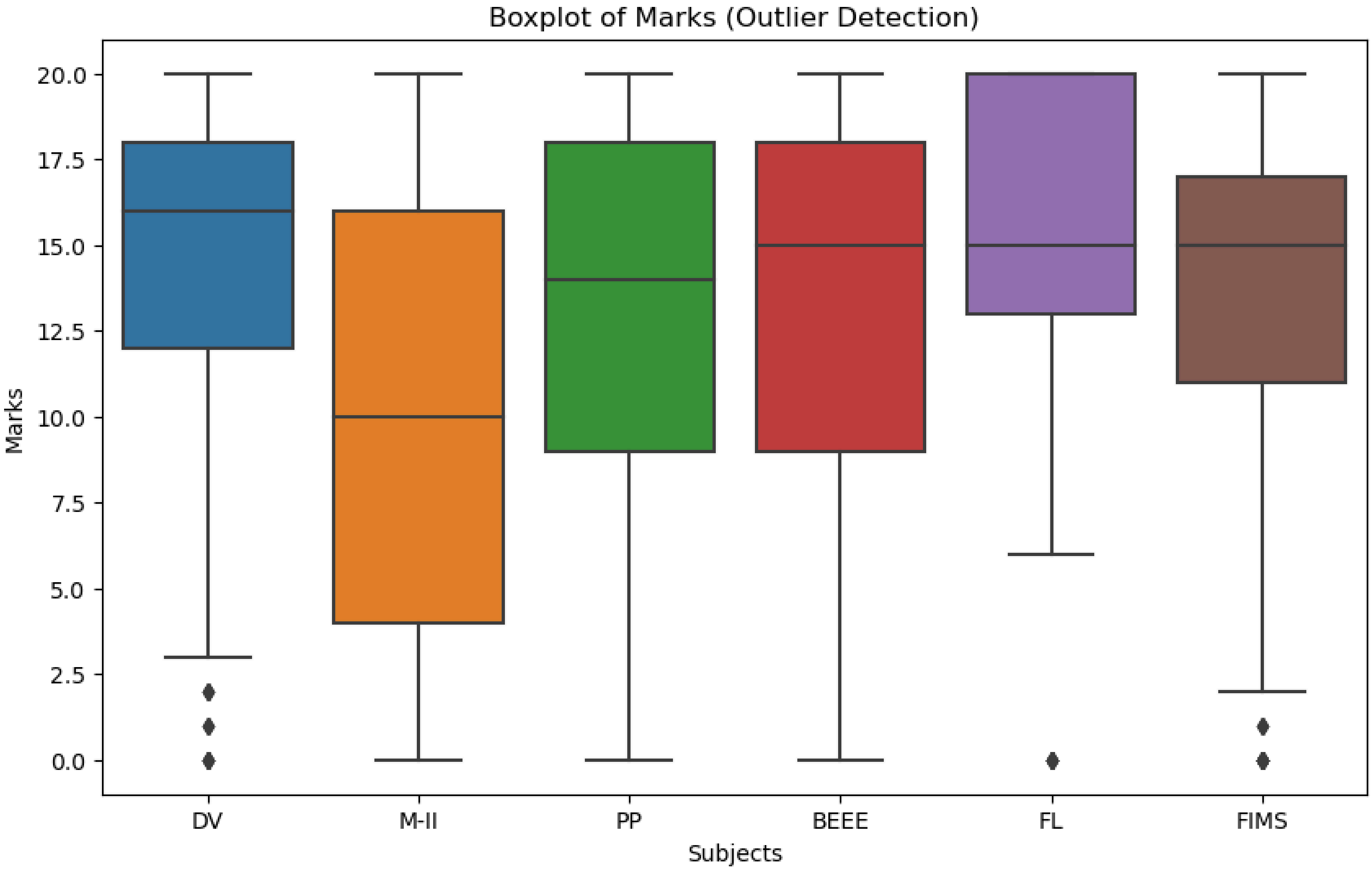
```
In [335]: import seaborn as sns
          import matplotlib.pyplot as plt
```

In [347]:

```
plt.figure(figsize=(12, 6))

# Plot subject-wise line chart for marks distribution
for subject in ["DV", "M-II", "PP", "BEEE", "FL", "FIMS"]:
    sns.lineplot(x=sorted(df[subject]), y=range(len(df)), label=subject)

plt.title("Marks Distribution for Each Subject")
plt.xlabel("Marks")
plt.ylabel("Cumulative Count")
plt.legend()
plt.show()
```


Marks Distribution for Each Subject
DV
M-II
PP
BEEE
FL
FIMS
Cumulative Count
0
100
200
300
400
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Marks

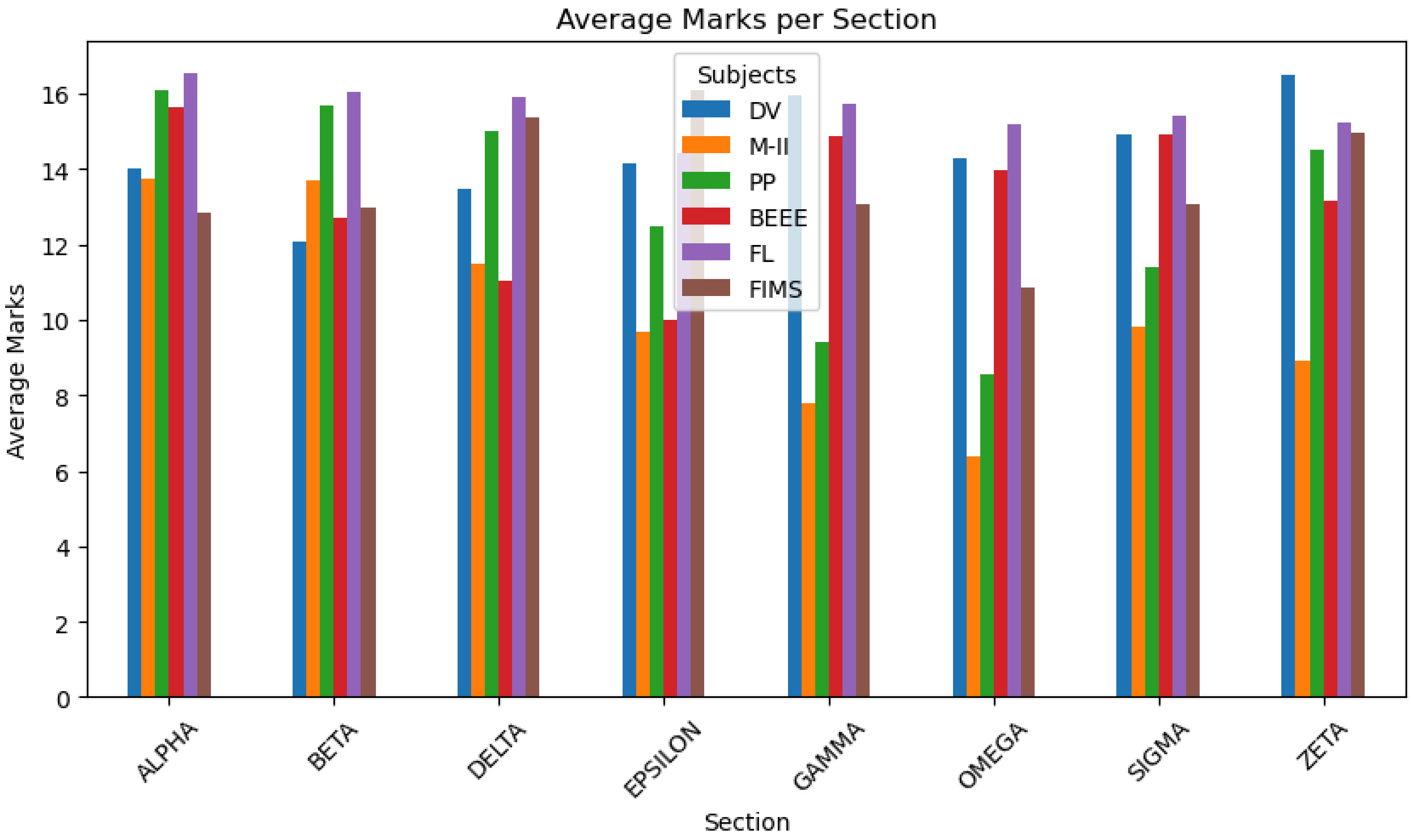
```
In [339]: plt.figure(figsize=(10, 6))
          sns.boxplot(data=df[subjects])
          plt.title("Boxplot of Marks (Outlier Detection)")
          plt.xlabel("Subjects")
          plt.ylabel("Marks")
          plt.show()
```



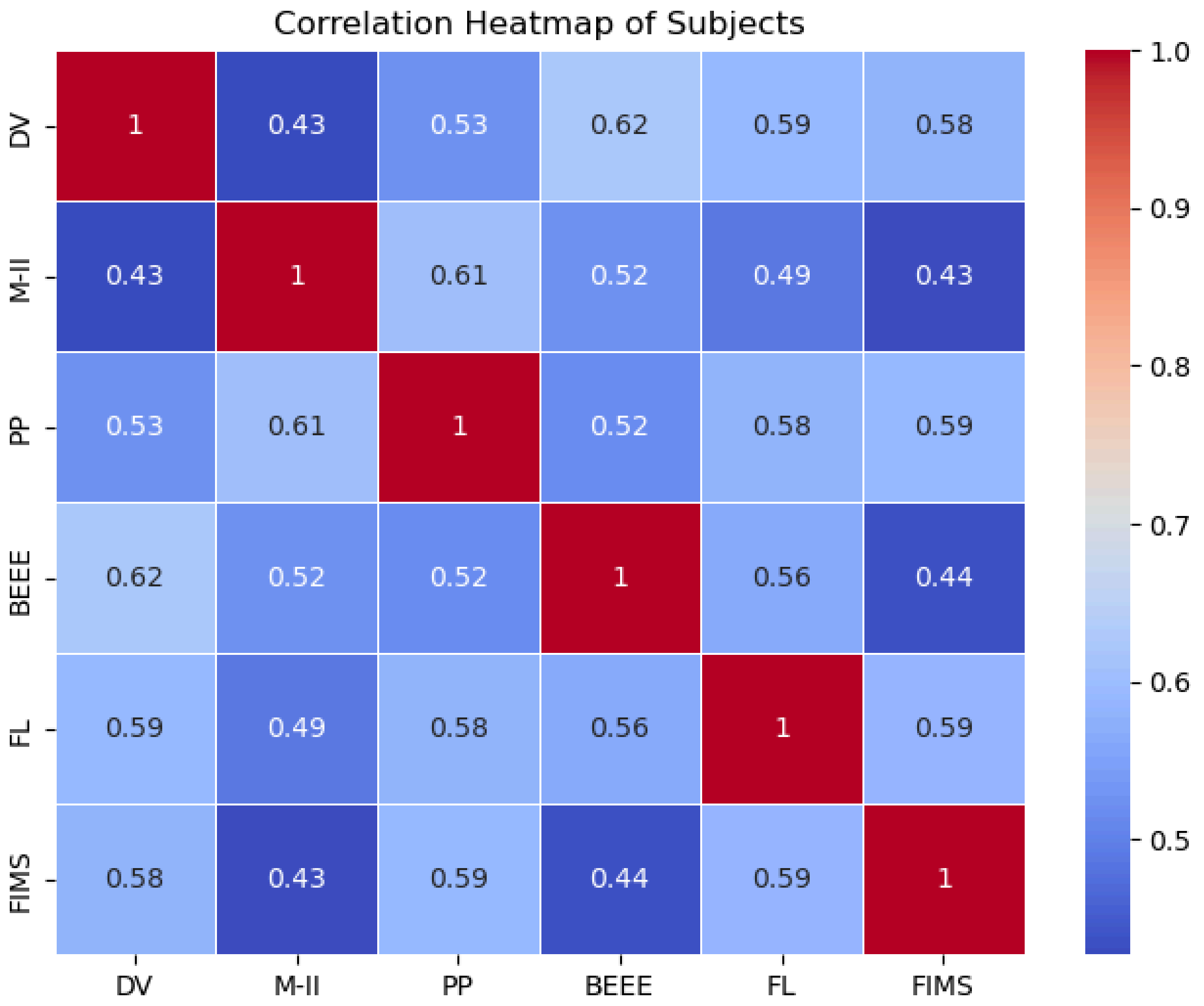
In [349]:

```
plt.figure(figsize=(10, 5))

# Calculate average marks per section and plot
df.groupby("SECTION")[subjects].mean().plot(kind="bar", figsize=(10, 5))

plt.title("Average Marks per Section")
plt.xlabel("Section")
plt.ylabel("Average Marks")
plt.xticks(rotation=45)
plt.legend(title="Subjects")
plt.show()
```

```
<Figure size 1000x500 with 0 Axes>
```

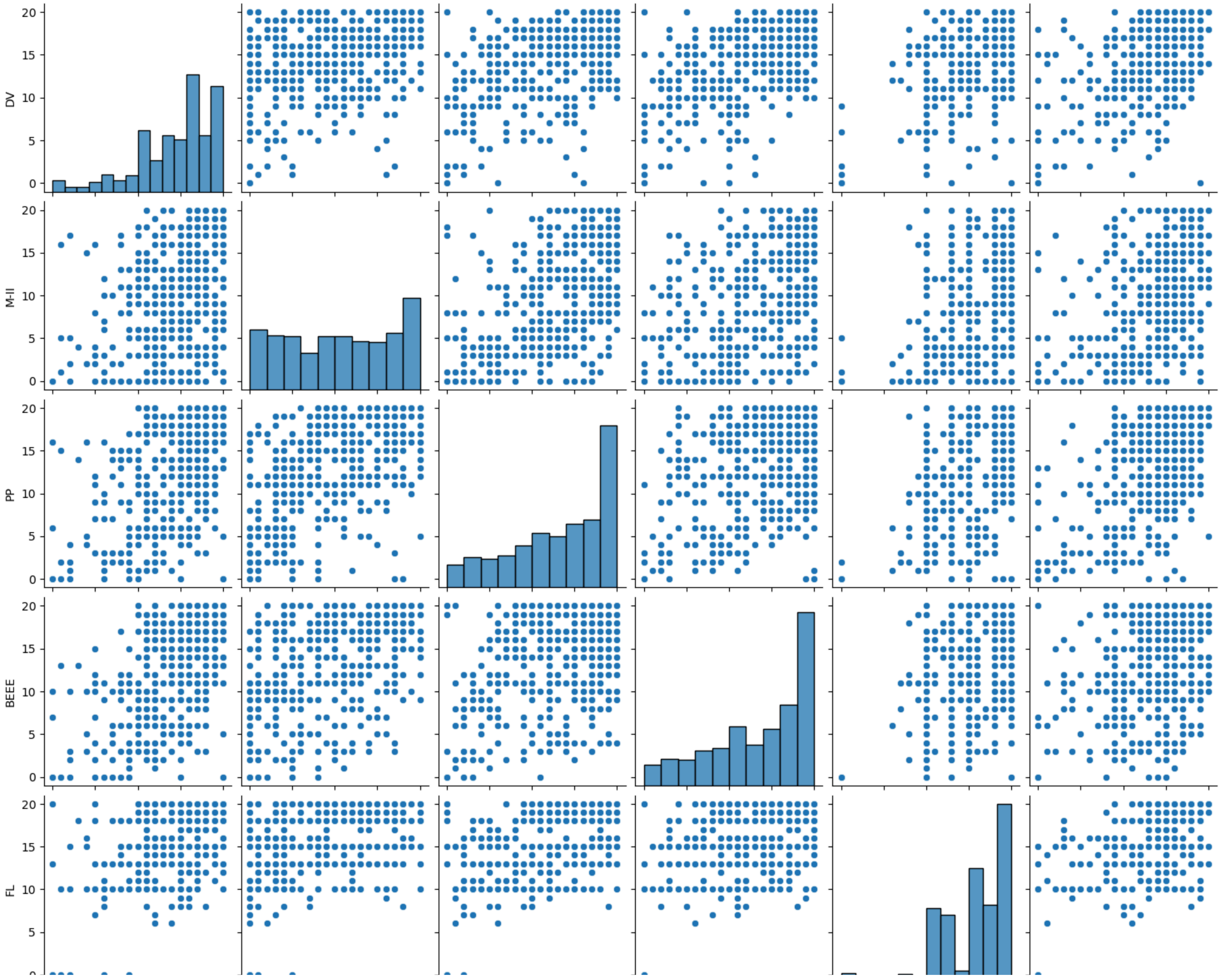

Average Marks per Section
Subjects
DV
M-II
PP
BEEE
FL
FIMS
Average Marks
0
2
4
6
8
10
12
14
16
ALPHA
BETA
DELTA
EPSILON
GAMMA
OMEGA
SIGMA
ZETA
Section

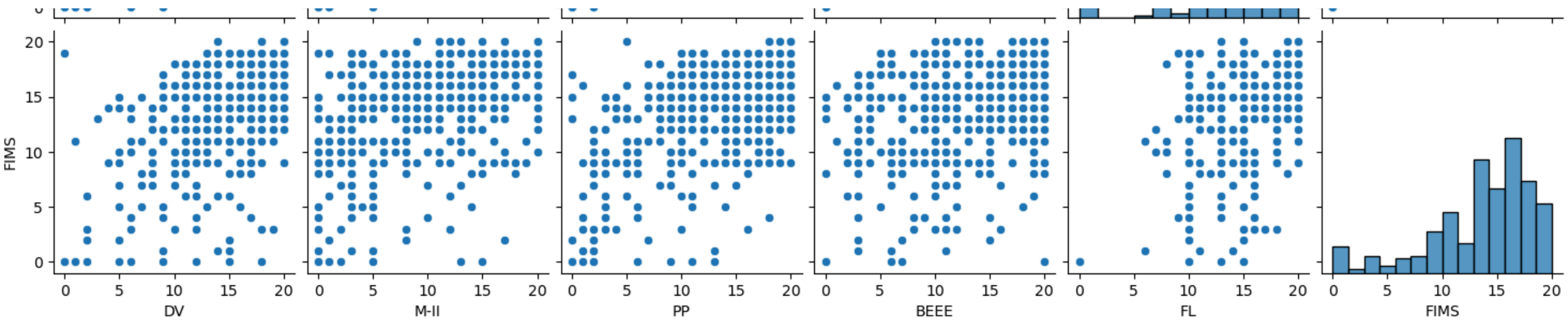
```
In [343]: plt.figure(figsize=(8, 6))
          sns.heatmap(df[subjects].corr(), annot=True, cmap="coolwarm", linewidths=0.5)
          plt.title("Correlation Heatmap of Subjects")
          plt.show()
```



In [345]:

```
sns.pairplot(df[subjects])
plt.show()
```

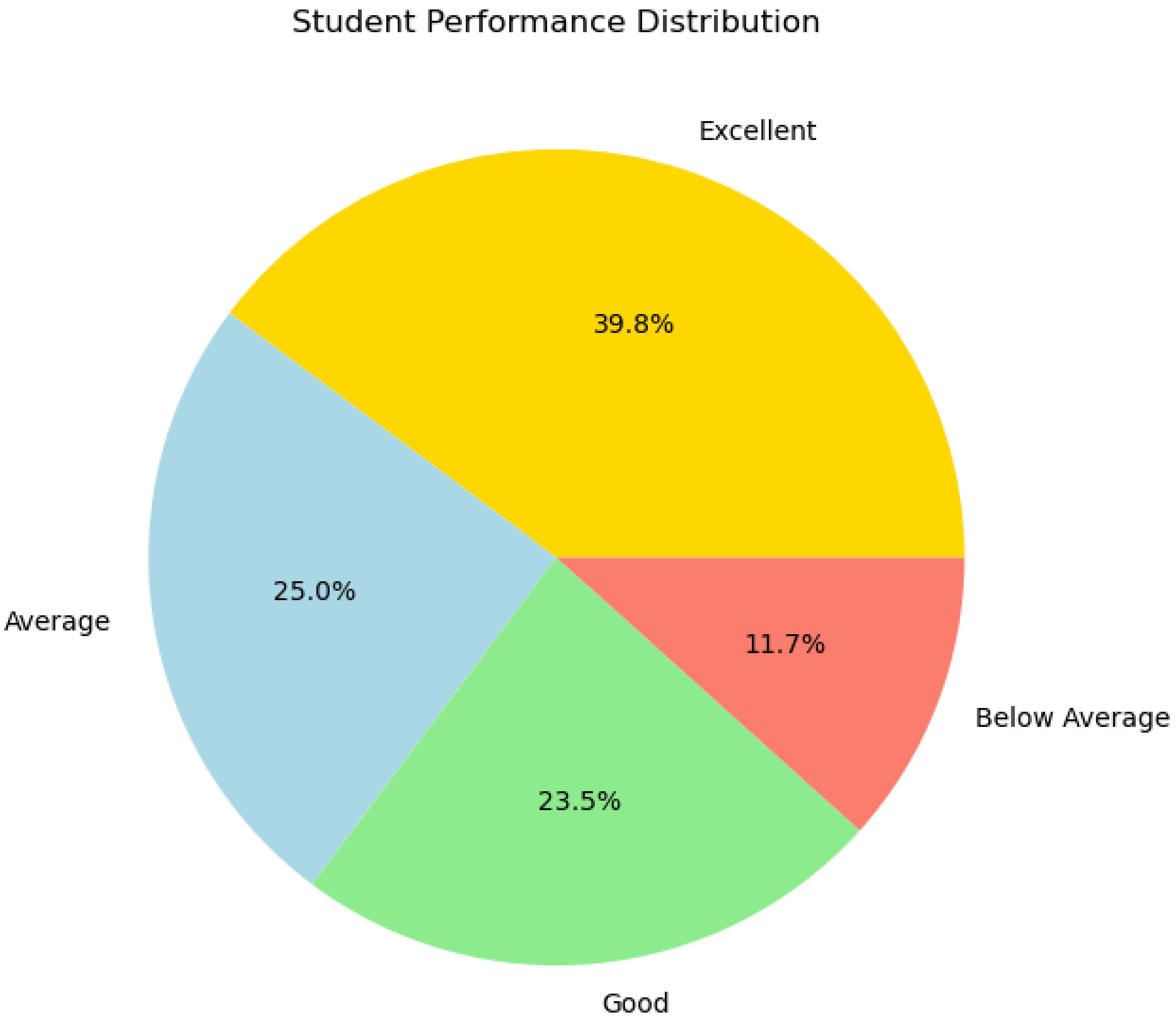

DV
20
15
10
5
0
M-II
20
15
10
5
0
PP
20
15
10
5
0
BEEE
20
15
10
5
0
FL
20
15
10
5
0



0
20
15
10
5
0
FIMS
0
5
10
15
20
DV
M-II
PP
BEEE
FL
FIMS

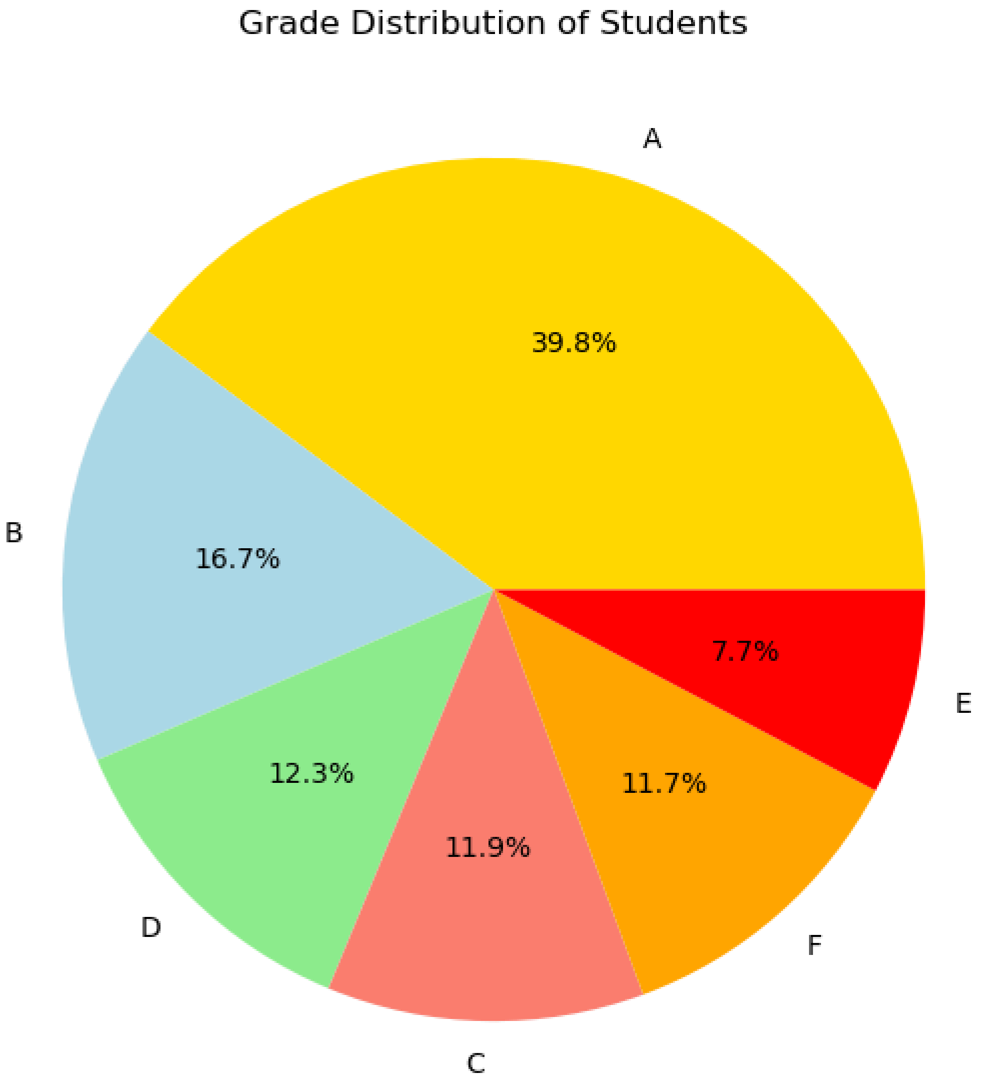
In [355]:

```
import matplotlib.pyplot as plt

# Define categories based on total marks (out of 20 per subject, 120 total)
def categorize_marks(total):
    if total >= 90:
        return "Excellent"
    elif total >= 75:
        return "Good"
    elif total >= 50:
        return "Average"
    else:
        return "Below Average"

# Calculate total marks for each student
df["Total Marks"] = df[["DV", "M-II", "PP", "BEEE", "FL", "FIMS"]].sum(axis=1)

# Categorize students
df["Category"] = df["Total Marks"].apply(categorize_marks)

# Count students in each category
category_counts = df["Category"].value_counts()

# Plot pie chart
plt.figure(figsize=(7, 7))
plt.pie(category_counts, labels=category_counts.index, autopct="%1.1f%%", colors=["gold", "lightblue", "lightgreen", "
plt.title("Student Performance Distribution")
plt.show()
```


Student Performance Distribution
Excellent
39.8%
Average
25.0%
Good
23.5%
11.7%
Below Average

In [357]:

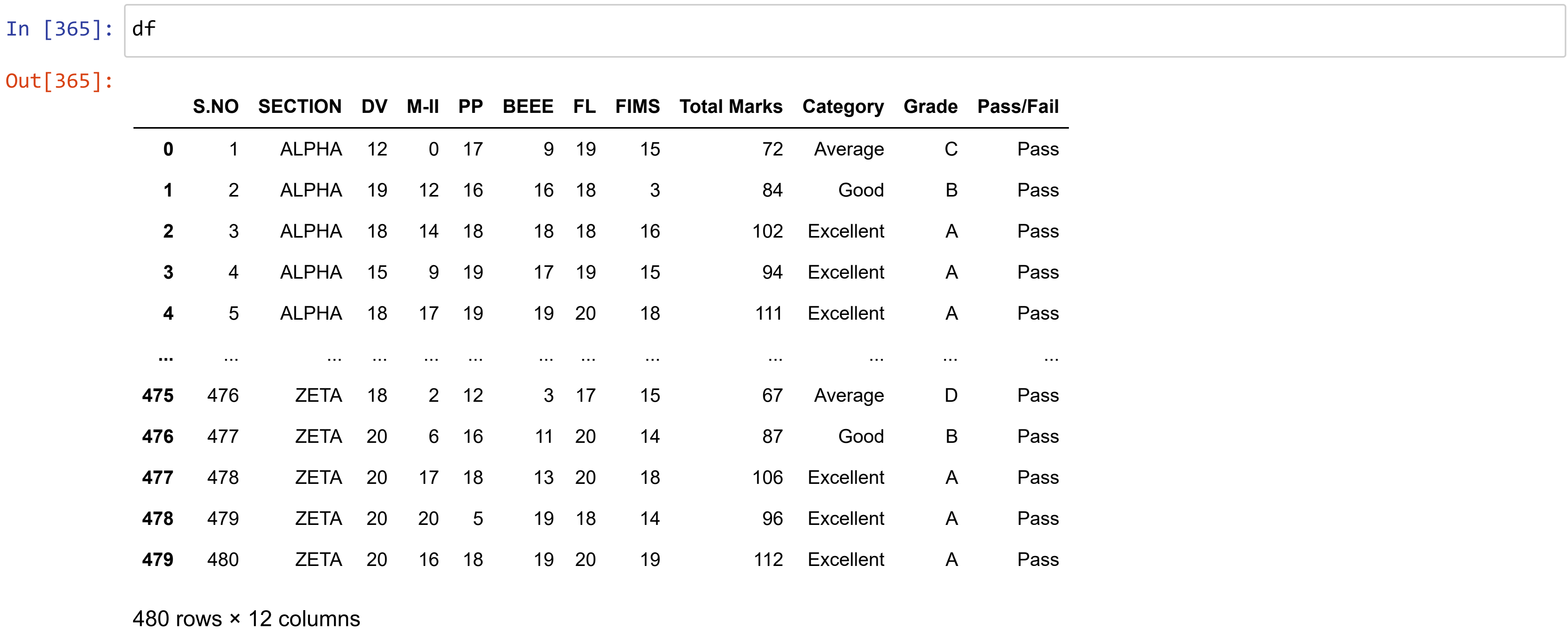
```
import matplotlib.pyplot as plt

# Define grade categories
def assign_grade(total):
    if total >= 90:
        return "A"
    elif total >= 80:
        return "B"
    elif total >= 70:
        return "C"
    elif total >= 60:
        return "D"
    elif total >= 50:
        return "E"
    else:
        return "F"

# Calculate total marks for each student
df["Total Marks"] = df[["DV", "M-II", "PP", "BEEE", "FL", "FIMS"]].sum(axis=1)

# Assign grades
df["Grade"] = df["Total Marks"].apply(assign_grade)

# Count students in each grade
grade_counts = df["Grade"].value_counts()

# Plot pie chart
plt.figure(figsize=(7, 7))
plt.pie(grade_counts, labels=grade_counts.index, autopct="%1.1f%%", colors=["gold", "lightblue", "lightgreen", "salmon
plt.title("Grade Distribution of Students")
plt.show()
```


Grade Distribution of Students
A
39.8%
B
16.7%
D
12.3%
C
11.9%
F
11.7%
E
7.7%

In [361]:

```python
# Define pass/fail criteria (assuming passing marks are 50/120)
def pass_fail(total):
    return "Pass" if total >= 50 else "Fail"

# Calculate total marks for each student
df["Total Marks"] = df[["DV", "M-II", "PP", "BEEE", "FL", "FIMS"]].sum(axis=1)

# Assign Pass/Fail status and create a new column
df["Pass/Fail"] = df["Total Marks"].apply(pass_fail)

# Count Pass and Fail students
result_counts = df["Pass/Fail"].value_counts()
print(result_counts)
```

```
Pass    424
Fail     56
Name: Pass/Fail, dtype: int64
```



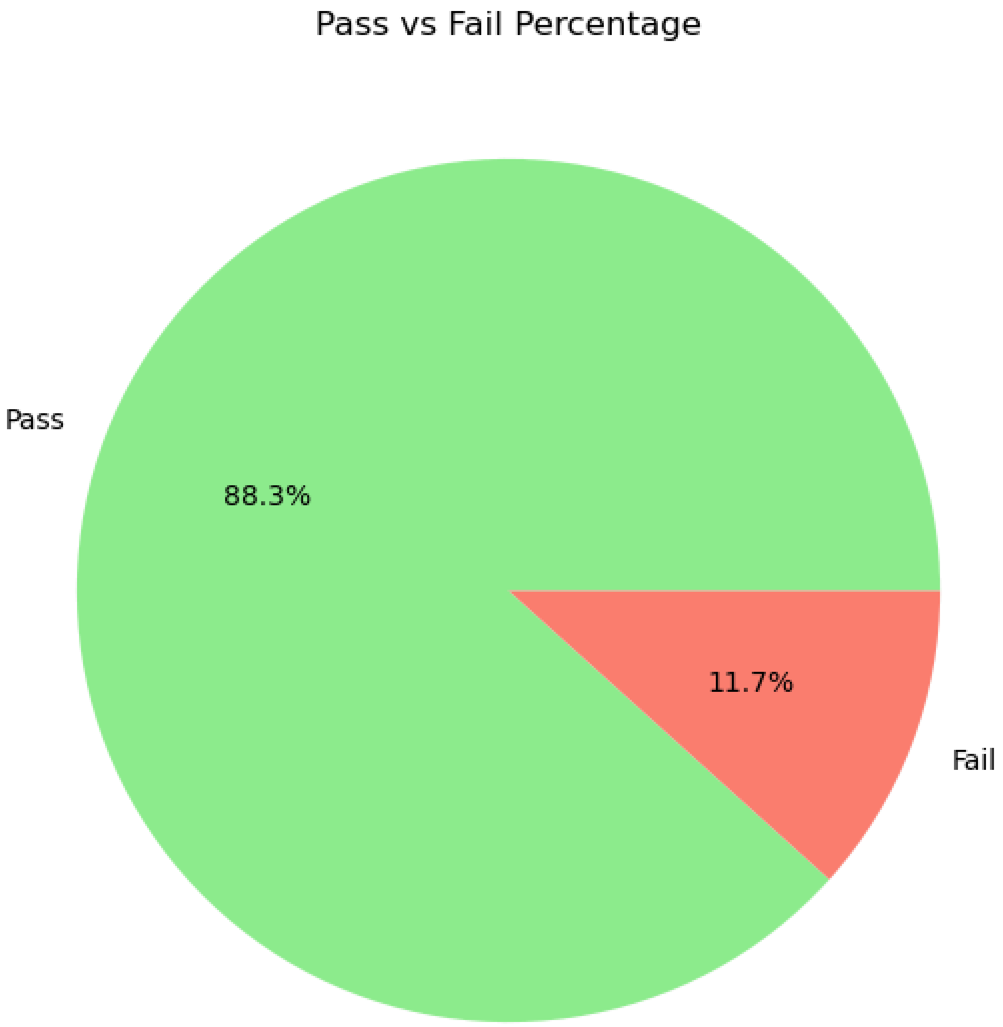
In [365]:
```
df
```

Out[365]:

	S.NO	SECTION	DV	M-II	PP	BEEE	FL	FIMS	Total Marks	Category	Grade	Pass/Fail
0	1	ALPHA	12	0	17	9	19	15	72	Average	C	Pass
1	2	ALPHA	19	12	16	16	18	3	84	Good	B	Pass
2	3	ALPHA	18	14	18	18	18	16	102	Excellent	A	Pass
3	4	ALPHA	15	9	19	17	19	15	94	Excellent	A	Pass
4	5	ALPHA	18	17	19	19	20	18	111	Excellent	A	Pass
...	...	...	...	...	...	...	...	...	...	...	...	...
475	476	ZETA	18	2	12	3	17	15	67	Average	D	Pass
476	477	ZETA	20	6	16	11	20	14	87	Good	B	Pass
477	478	ZETA	20	17	18	13	20	18	106	Excellent	A	Pass
478	479	ZETA	20	20	5	19	18	14	96	Excellent	A	Pass
479	480	ZETA	20	16	18	19	20	19	112	Excellent	A	Pass

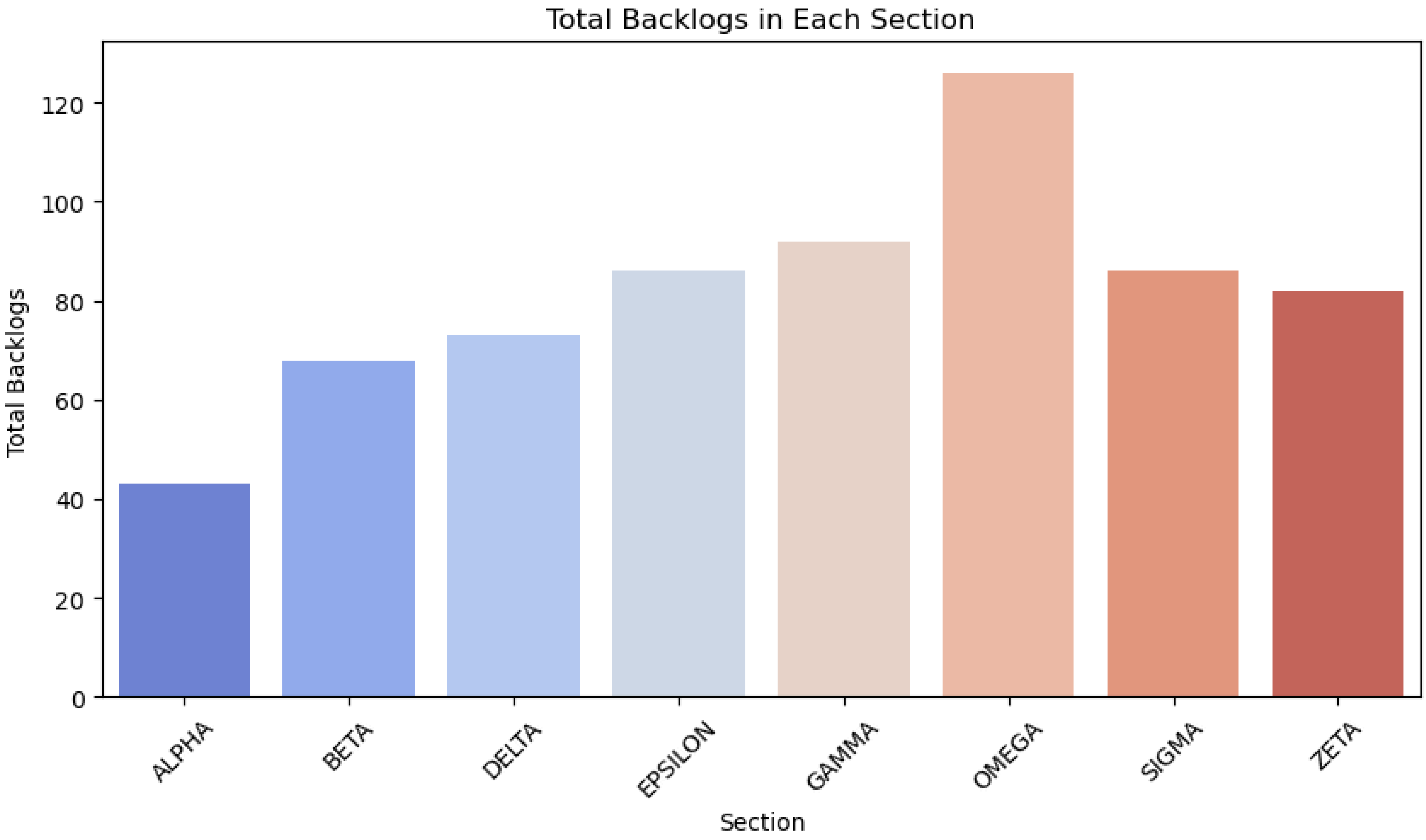
480 rows × 12 columns

In [363]:
```
# Plot pie chart
plt.figure(figsize=(7, 7))
plt.pie(result_counts, labels=result_counts.index, autopct="%1.1f%%", colors=["lightgreen", "salmon"])
plt.title("Pass vs Fail Percentage")
plt.show()
```



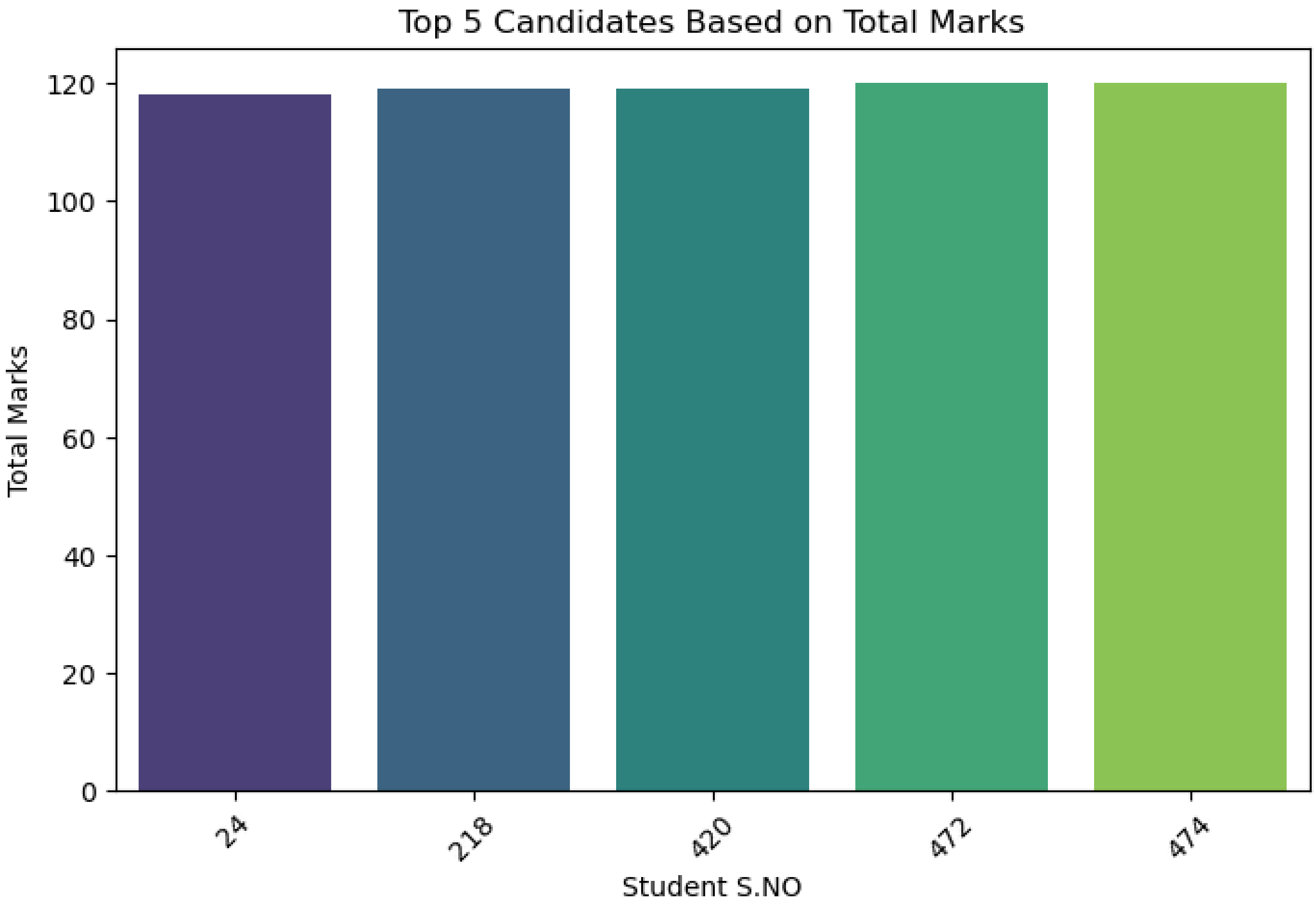
In [371]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Define backlog count (subjects with marks < 10 are considered backlogs)
df["Backlogs"] = (df[["DV", "M-II", "PP", "BEEE", "FL", "FIMS"]] < 10).sum(axis=1)

# Group by section to count total backlogs
backlog_counts = df.groupby("SECTION")["Backlogs"].sum().reset_index()

# Plot bar chart
plt.figure(figsize=(10, 5))
sns.barplot(x="SECTION", y="Backlogs", data=backlog_counts, palette="coolwarm")

plt.title("Total Backlogs in Each Section")
plt.xlabel("Section")
plt.ylabel("Total Backlogs")
plt.xticks(rotation=45)
plt.show()
```


Total Backlogs in Each Section
Total Backlogs
0
20
40
60
80
100
120
ALPHA
BETA
DELTA
EPSILON
GAMMA
OMEGA
SIGMA
ZETA
Section

In [373]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate total marks for each student
df["Total Marks"] = df[["DV", "M-II", "PP", "BEEE", "FL", "FIMS"]].sum(axis=1)

# Select the top 5 candidates
top_5 = df.nlargest(5, "Total Marks")

# Plot bar chart
plt.figure(figsize=(8, 5))
sns.barplot(x=top_5["S.NO"], y=top_5["Total Marks"], palette="viridis")

plt.title("Top 5 Candidates Based on Total Marks")
plt.xlabel("Student S.NO")
plt.ylabel("Total Marks")
plt.xticks(rotation=45)
plt.show()
```


Top 5 Candidates Based on Total Marks
Total Marks
0
20
40
60
80
100
120
24
218
420
472
474
Student S.NO

In [375]:

```
aggregated_data=df.groupby(['SECTION']).mean()
aggregated_data
```

```
C:\Users\subha\AppData\Local\Temp\ipykernel_22864\113419422.py:1: FutureWarning: The default value of numeric_only in
DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_
only or select only columns which should be valid for the function.
  aggregated_data=df.groupby(['SECTION']).mean()
```

Out[375]:

	S.NO	DV	M-II	PP	BEEE	FL	FIMS	Total Marks	Backlogs
SECTION									
ALPHA	30.5	14.033333	13.733333	16.066667	15.616667	16.550000	12.850000	88.850000	0.716667
BETA	90.5	12.083333	13.683333	15.666667	12.716667	16.033333	12.983333	83.166667	1.133333
DELTA	150.5	13.483333	11.466667	15.016667	11.050000	15.916667	15.350000	82.283333	1.216667
EPSILON	210.5	14.150000	9.683333	12.483333	10.016667	14.400000	16.066667	76.800000	1.433333
GAMMA	270.5	15.933333	7.800000	9.400000	14.866667	15.716667	13.050000	76.766667	1.533333
OMEGA	330.5	14.300000	6.400000	8.533333	13.983333	15.183333	10.866667	69.266667	2.100000
SIGMA	390.5	14.900000	9.833333	11.400000	14.933333	15.433333	13.050000	79.550000	1.433333
ZETA	450.5	16.483333	8.900000	14.516667	13.166667	15.250000	14.950000	83.266667	1.366667

In [377]:

```
std_data=df.groupby(['SECTION']).std()
std_data
```

```
C:\Users\subha\AppData\Local\Temp\ipykernel_22864\475094170.py:1: FutureWarning: The default value of numeric_only in
DataFrameGroupBy.std is deprecated. In a future version, numeric_only will default to False. Either specify numeric_o
nly or select only columns which should be valid for the function.
  std_data=df.groupby(['SECTION']).std()
```

Out[377]:

	S.NO	DV	M-II	PP	BEEE	FL	FIMS	Total Marks	Backlogs
SECTION									
ALPHA	17.464249	4.654018	5.161351	5.085262	4.476271	3.402018	4.037221	20.844725	1.090664
BETA	17.464249	4.465657	5.484931	5.183634	6.031251	3.817740	4.343285	22.804376	1.395716
DELTA	17.464249	4.268496	6.091023	4.942008	5.664115	3.585714	3.545467	21.055288	1.316025
EPSILON	17.464249	4.070668	5.887337	5.163403	5.697135	4.373262	4.095747	23.883758	1.418601
GAMMA	17.464249	2.208356	5.885345	3.945390	4.537851	3.884309	4.350823	19.912833	1.346265
OMEGA	17.464249	4.412233	5.793626	4.928185	5.447723	4.118959	4.563921	24.623860	1.633339
SIGMA	17.464249	5.417173	6.636230	6.436654	6.273070	4.618924	5.359689	29.925005	1.835310
ZETA	17.464249	5.309064	6.321070	5.961719	6.284678	5.503851	5.221939	29.601229	1.625711

In [379]:

```
group1=df[df['SECTION']=='ALPHA']['DV']
print(group1)
```

```
0     12
1     19
2     18
3     15
4     18
5     17
6     15
7     17
8     10
9     18
10    17
11    20
12    16
13    17
14    19
15    13
16    15
17    11
18    14
19    19
20     4
21    14
22    17
23    20
24    15
25     6
26    17
27     5
28    19
29     8
30    11
31    12
32    17
33    14
34    17
35     8
36    11
37    15
38    19
39    20
40    18
```

```
41    16
42    16
43    11
44    18
45    11
46    14
47    16
48    16
49    15
50     1
51     6
52    17
53     8
54    14
55    15
56    10
57     2
58    10
59    19
Name: DV, dtype: int32
```

In [381]:

```
group2=df[df['SECTION']=='BETA']['DV']
print(group2)
```

```
60      19
61       8
62      12
63      11
64      12
65       9
66      12
67      12
68      16
69      20
70       4
71      17
72       7
73      10
74      17
75       5
76      17
77      13
78      19
79      19
80      19
81      18
82       2
83      10
84      12
85       3
86      17
87      13
88       2
89      10
90      17
91      14
92      11
93      14
94      12
95      16
96       8
97       8
98       6
99       9
100     10
```

```
101    13
102    10
103    11
104    17
105    12
106     9
107    11
108    10
109    13
110     8
111    10
112    16
113    15
114    11
115    20
116    13
117    12
118     9
119    15
Name: DV, dtype: int32
```

In [383]:

```
from scipy.stats import ttest_ind
ttest_ind(group1, group2, equal_var=False)
```

Out[383]:

```
Ttest_indResult(statistic=2.34181859243181, pvalue=0.020869348905772172)
```

In [385]:

```
from scipy.stats import ttest_rel
ttest_rel(group1, group2)
```

Out[385]:

```
TtestResult(statistic=2.3172456109384103, pvalue=0.023979527821469917, df=59)
```

In [387]:

```
df[df['SECTION']=='ALPHA']['DV'].mean()
```

Out[387]:

```
14.033333333333333
```

In [389]:

```
from scipy.stats import ttest_1samp
t_statistic,p_value=ttest_1samp(df[df['SECTION']=='ALPHA']['DV'],popmean=14.41)
print(t_statistic,p_value)
```

```
-0.6269093116996493 0.5331371479713868
```

In [391]:

```
import pandas as pd
from scipy.stats import chi2_contingency

# Create a contingency table (Cross-tab of Pass/Fail vs Section)
contingency_table = pd.crosstab(df["Pass/Fail"], df["SECTION"])

# Perform Chi-Square test
chi2_stat, p_value, dof, expected = chi2_contingency(contingency_table)

print(f"Chi-Square Test: chi2 = {chi2_stat:.3f}, p-value = {p_value:.3f}")
```

```
Chi-Square Test: chi2 = 18.437, p-value = 0.010
```

In [393]:

```
import pandas as pd
from scipy.stats import chi2_contingency

# Create a contingency table for Pass/Fail vs Section
contingency_table = pd.crosstab(df["Pass/Fail"], df["SECTION"])

# Perform Chi-Square test
stat, p, dof, expected = chi2_contingency(contingency_table)

# Set significance level
alpha = 0.05

# Print p-value
print(f"p-value is {p:.5f}")

# Decision based on p-value
if p <= alpha:
    print("Pass/Fail is dependent on Section (Reject H0)")
else:
    print("Pass/Fail is independent of Section (H0 holds true)")
```

```
p-value is 0.01015
Pass/Fail is dependent on Section (Reject H0)
```