DA Case Study-2

Name: Subhapreet Patro

Roll No.: 2211CS10547

Group: 3

Dataset: MIDMARKS.xlsx

In [3]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

This kernel imports the necessary libraries: `pandas` for data manipulation, `matplotlib.pyplot` for plotting, and `seaborn` for statistical data visualization.

In [5]:
```python
file_path = 'MIDMARKS(1).xlsx'
try:
    df = pd.read_excel(file_path, sheet_name='SEM2 MID 1 - ALPHA')
except FileNotFoundError:
    print(f"Error: File '{file_path}' not found.")
    raise
except ValueError:
    print(f"Error: Sheet name 'SEM2 MID 1 - ALPHA' not found in the file.")
    raise

marks_columns = ['DV', 'M-II', 'PP', 'BEEE', 'FL', 'FIMS']

def clean_marks(value):
    if isinstance(value, str):
        if value in ['A', 'AB']:
            return 0
        elif value == 'MP':
            return None
    return pd.to_numeric(value, errors='coerce')

df[marks_columns] = df[marks_columns].applymap(clean_marks)
df=df.drop(index=[717])
df.dropna(subset=marks_columns, inplace=True)
df
```

Out[5]:

	S.NO	SECTION	DV	M-II	PP	BEEE	FL	FIMS
0	1.0	ALPHA	12.0	0.0	17.0	9.0	19.0	15.0
1	2.0	ALPHA	19.0	12.0	16.0	16.0	18.0	3.0
2	3.0	ALPHA	18.0	14.0	18.0	18.0	18.0	16.0
3	4.0	ALPHA	15.0	9.0	19.0	17.0	19.0	15.0
4	5.0	ALPHA	18.0	17.0	19.0	19.0	20.0	18.0
...	...	...	...	...	...	...	...	...
712	NaN	ZETA	15.0	10.0	7.0	18.0	18.0	16.0
713	NaN	ZETA	19.0	8.0	8.0	19.0	17.0	18.0
714	NaN	ZETA	12.0	1.0	7.0	10.0	20.0	8.0
715	NaN	ZETA	17.0	6.0	14.0	14.0	17.0	18.0
716	NaN	ZETA	12.0	1.0	6.0	7.0	15.0	12.0

712 rows × 8 columns

In [6]:

```
df.info()
```

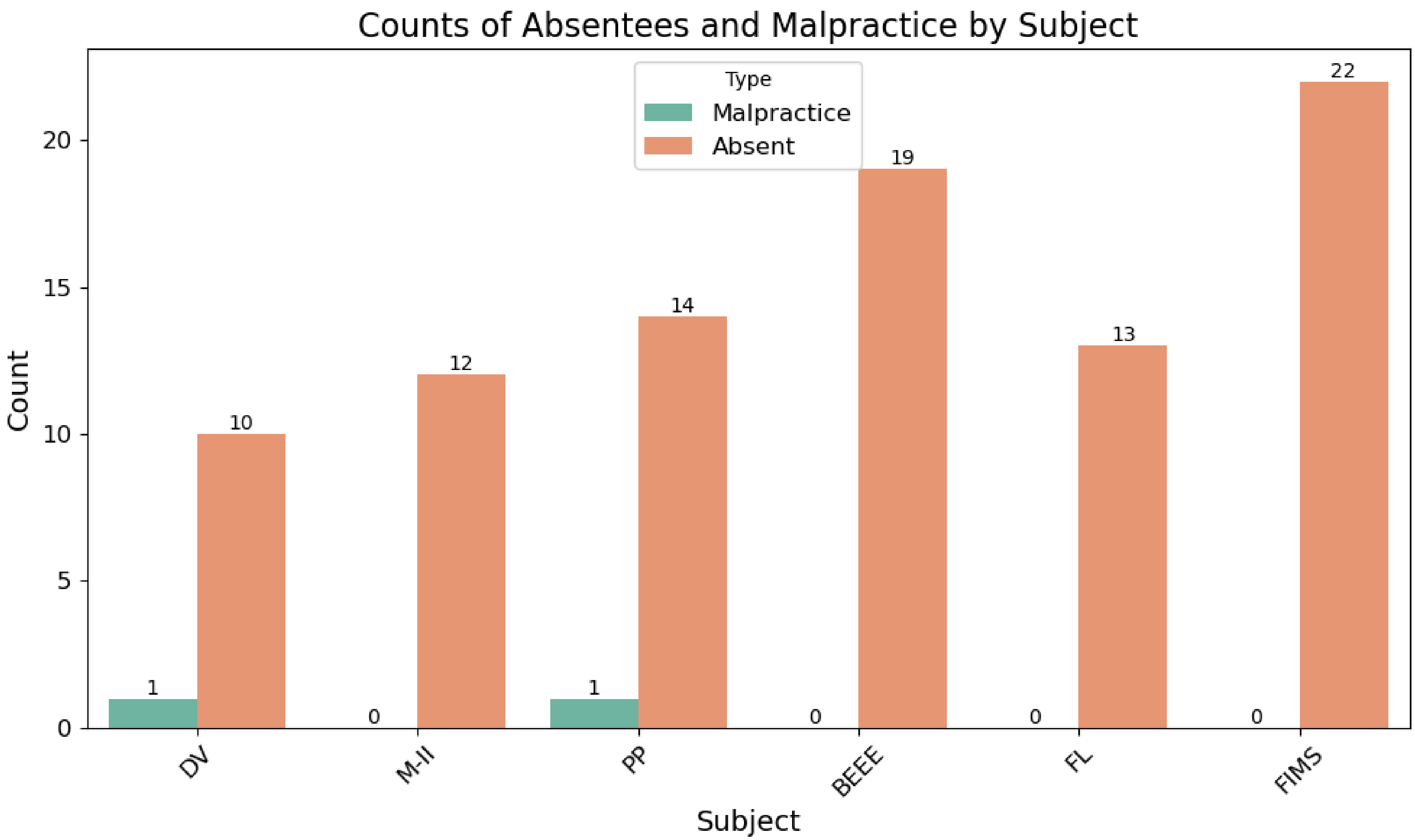
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 712 entries, 0 to 716
Data columns (total 8 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   S.NO     597 non-null    float64
 1   SECTION  686 non-null    object 
 2   DV       712 non-null    float64
 3   M-II     712 non-null    float64
 4   PP       712 non-null    float64
 5   BEEE     712 non-null    float64
 6   FL       712 non-null    float64
 7   FIMS     712 non-null    float64
dtypes: float64(7), object(1)
memory usage: 50.1+ KB
```

This kernel reads the Excel file `MIDMARKS.xlsx`, extracts the sheet 'SEM2 MID 1 - ALPHA', and cleans the marks data by converting absent ('A', 'AB') to 0, malpractice ('MP') to NaN, and then drops rows with missing marks.

In [8]:
```
file_path = 'MIDMARKS(1).xlsx'
sheet_name = 'SEM2 MID 1 - ALPHA'

try:
    df = pd.read_excel(file_path, sheet_name=sheet_name)
except Exception as e:
    print(f"Error loading file: {e}")

marks_columns = ['DV', 'M-II', 'PP', 'BEEE', 'FL', 'FIMS']

missing_columns = [col for col in marks_columns if col not in df.columns]
if missing_columns:
    raise ValueError(f"The following columns are missing from the dataset: {missing_columns}")

def clean_marks(value):
    if isinstance(value, str):
        if value in ['A', 'AB']:
            return 'Absent'
        elif value == 'MP':
            return 'Malpractice'
    return pd.to_numeric(value, errors='coerce')

df_cleaned = df.copy()
df_cleaned[marks_columns] = df_cleaned[marks_columns].applymap(clean_marks)

absentees_count = (df_cleaned[marks_columns] == 'Absent').sum().sum()
malpractice_count = (df_cleaned[marks_columns] == 'Malpractice').sum().sum()

print(f"Number of absentees: {absentees_count}")
print(f"Number of malpractice cases: {malpractice_count}")

df_cleaned_numeric = df_cleaned.copy()
df_cleaned_numeric[marks_columns] = df_cleaned_numeric[marks_columns].replace(
    {'Absent': 0, 'Malpractice': 0}
)

df_melted = df_cleaned.melt(
    id_vars=['S.NO', 'SECTION'],
    value_vars=marks_columns,
    var_name='Subject',
    value_name='Marks'
```

```python
)

absent_malpractice_counts = df_melted[df_melted['Marks'].isin(['Absent', 'Malpractice'])]

plt.figure(figsize=(10, 6))
sns.countplot(data=absent_malpractice_counts, x='Subject', hue='Marks', palette='Set2')
plt.title('Counts of Absentees and Malpractice by Subject', fontsize=16)
plt.xlabel('Subject', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.legend(title='Type', loc='upper center', fontsize=12)
plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)

for container in plt.gca().containers:
    plt.gca().bar_label(container, label_type='edge', fontsize=10)

plt.tight_layout()
plt.show()
```

```
Number of absentees: 90
Number of malpractice cases: 2
```

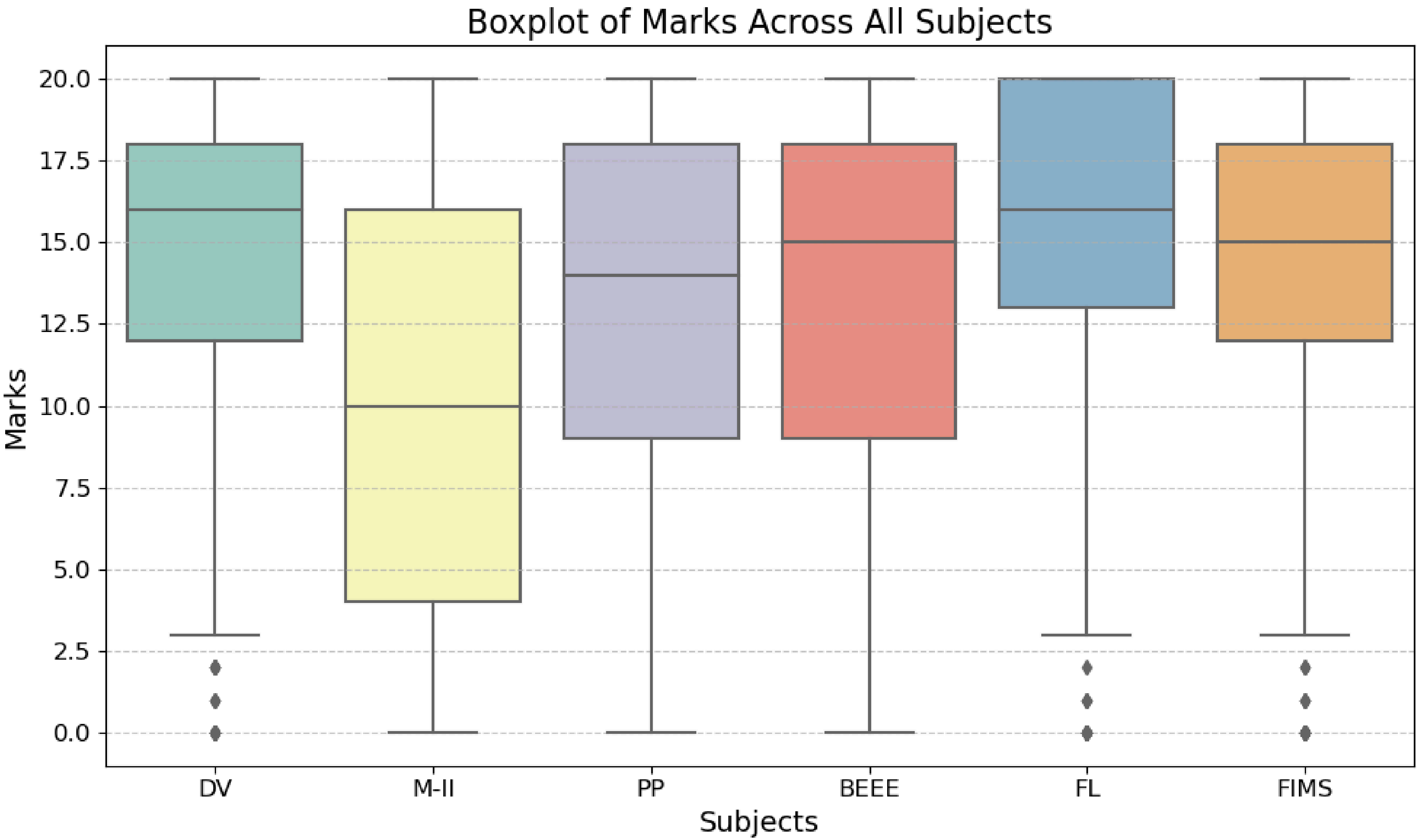


This kernel loads the `MIDMARKS.xlsx` file and sheet `SEM2 MID 1 - ALPHA`, cleans the marks data by categorizing absentees and malpractice cases, counts the absentees and malpractice instances, and visualizes the counts of these cases for each subject in a bar chart.

In [10]:

```
print("Summary Statistics for Marks:")
print(df_cleaned_numeric[marks_columns].describe())

plt.figure(figsize=(10, 6))
sns.boxplot(data=df_cleaned_numeric[marks_columns], palette='Set3')

plt.title("Boxplot of Marks Across All Subjects", fontsize=16)
plt.xlabel("Subjects", fontsize=14)
plt.ylabel("Marks", fontsize=14)
plt.xticks(ticks=range(len(marks_columns)), labels=marks_columns, fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```

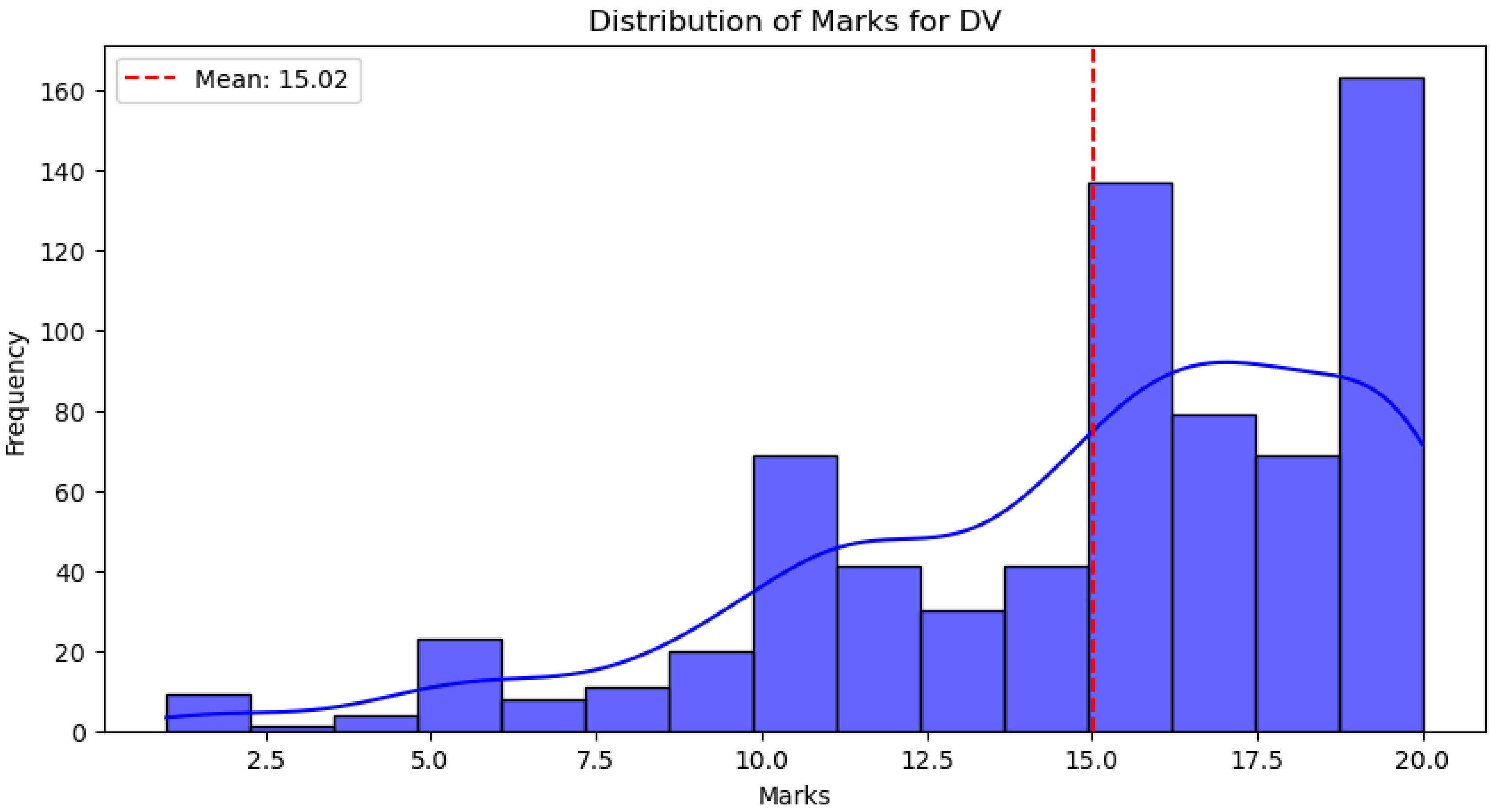
```
Summary Statistics for Marks:
               DV        M-II          PP        BEEE          FL        FIMS
count  716.000000  716.000000  716.000000  716.000000  715.000000  716.000000
mean    14.787709    9.949721   12.769553   13.287709   15.566434   14.047486
std      4.569545    6.599236    5.817381    5.783112    4.441275    4.709815
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%     12.000000    4.000000    9.000000    9.000000   13.000000   12.000000
50%     16.000000   10.000000   14.000000   15.000000   16.000000   15.000000
75%     18.000000   16.000000   18.000000   18.000000   20.000000   18.000000
max     20.000000   20.000000   20.000000   20.000000   20.000000   20.000000
```

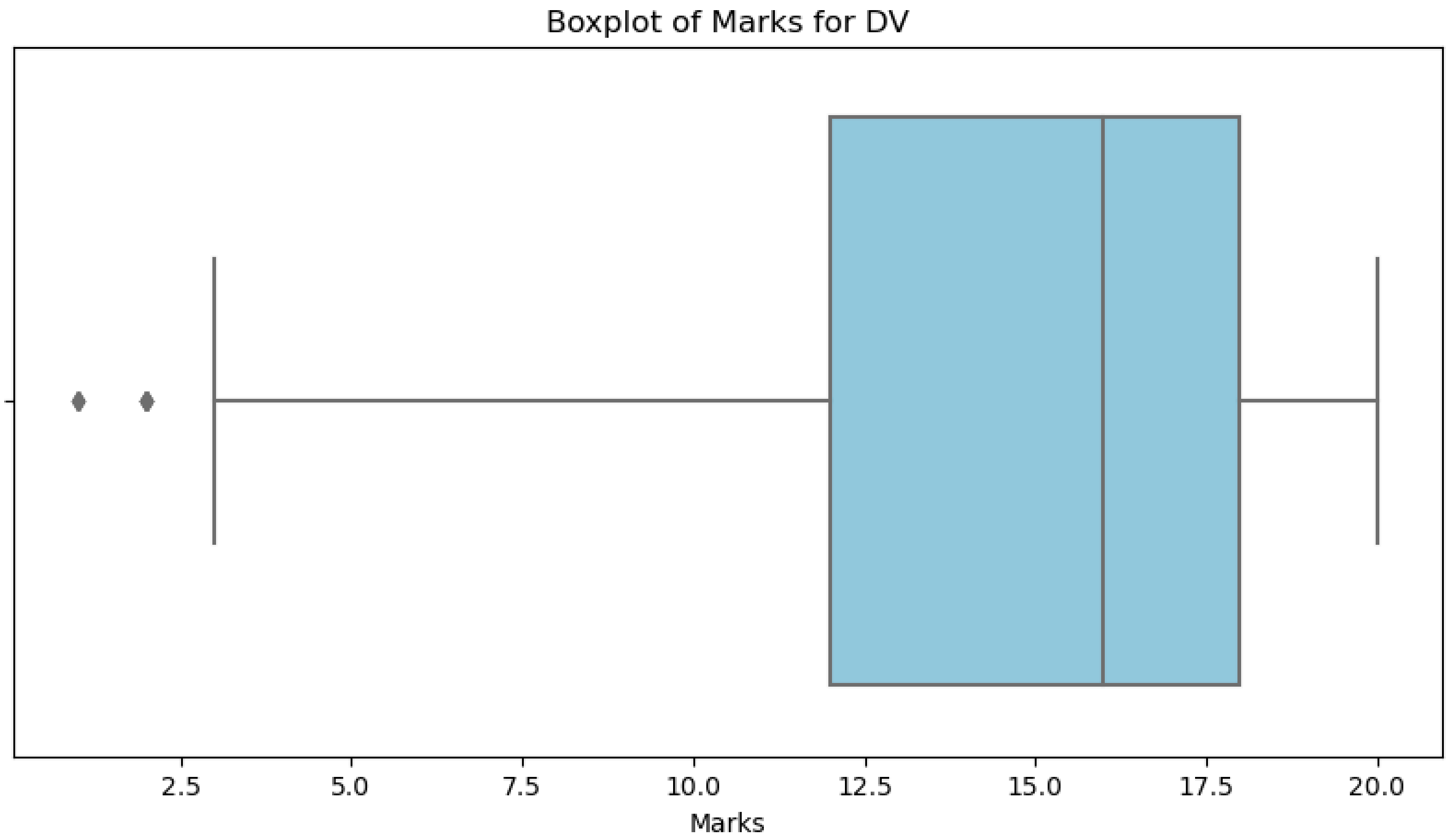


This kernel displays summary statistics (such as mean, min, max, and standard deviation) for the marks across all subjects, and visualizes the distribution of marks through a boxplot to show the spread and outliers for each subject.

In [12]:

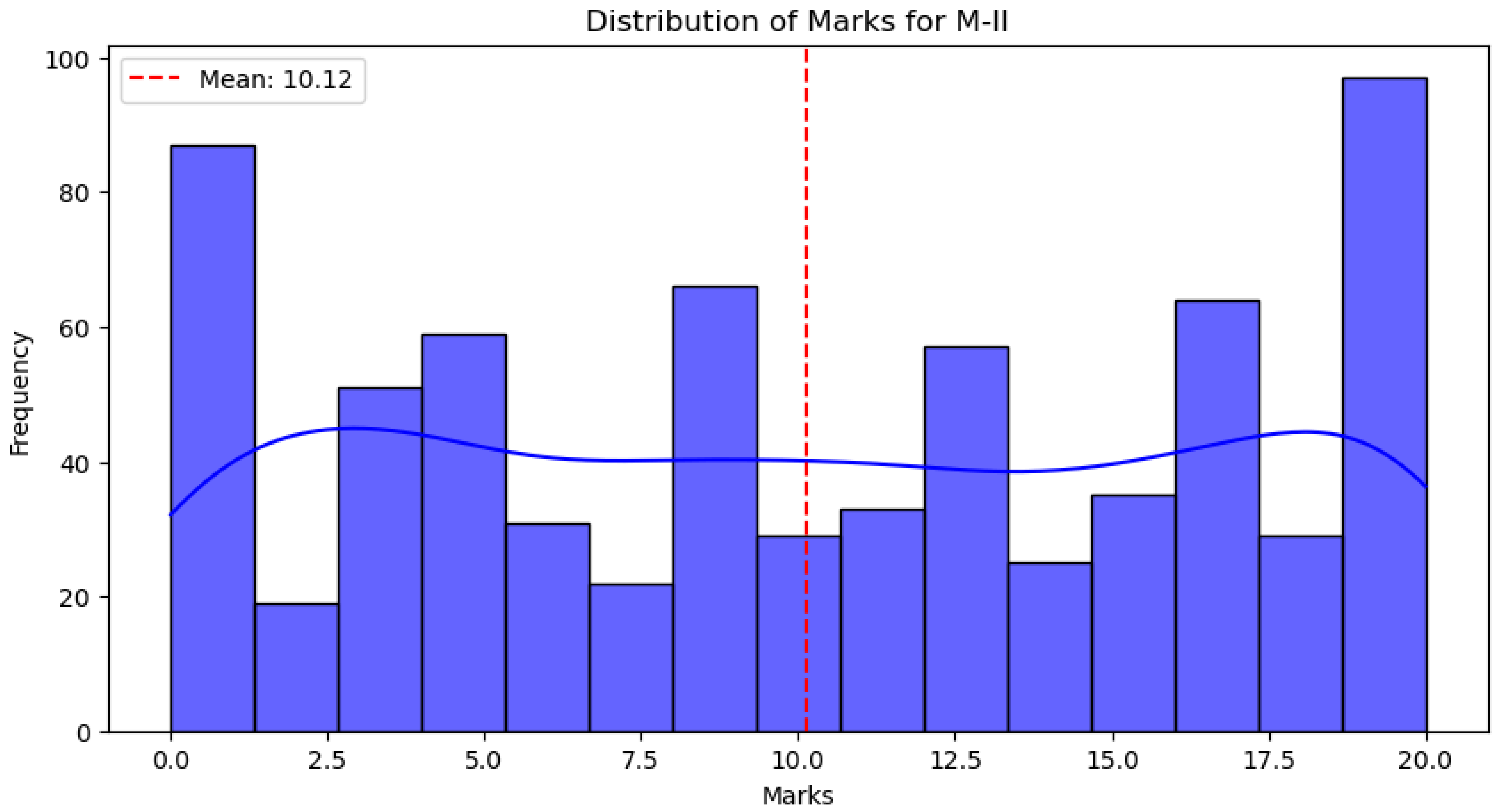
```
df['DV'] = pd.to_numeric(df['DV'], errors='coerce')

df_filtered = df.dropna(subset=['DV'])

plt.figure(figsize=(10, 5))
sns.histplot(df_filtered['DV'], kde=True, bins=15, color='blue', alpha=0.6, edgecolor='black')
plt.title('Distribution of Marks for DV')
plt.xlabel('Marks')
plt.ylabel('Frequency')
plt.axvline(df_filtered['DV'].mean(), color='red', linestyle='--', linewidth=1.5, label=f'Mean: {df_filtered["DV"].mea
plt.legend()
plt.show()

plt.figure(figsize=(10, 5))
sns.boxplot(x=df_filtered['DV'], color='skyblue')
plt.title('Boxplot of Marks for DV')
plt.xlabel('Marks')
plt.show()
```

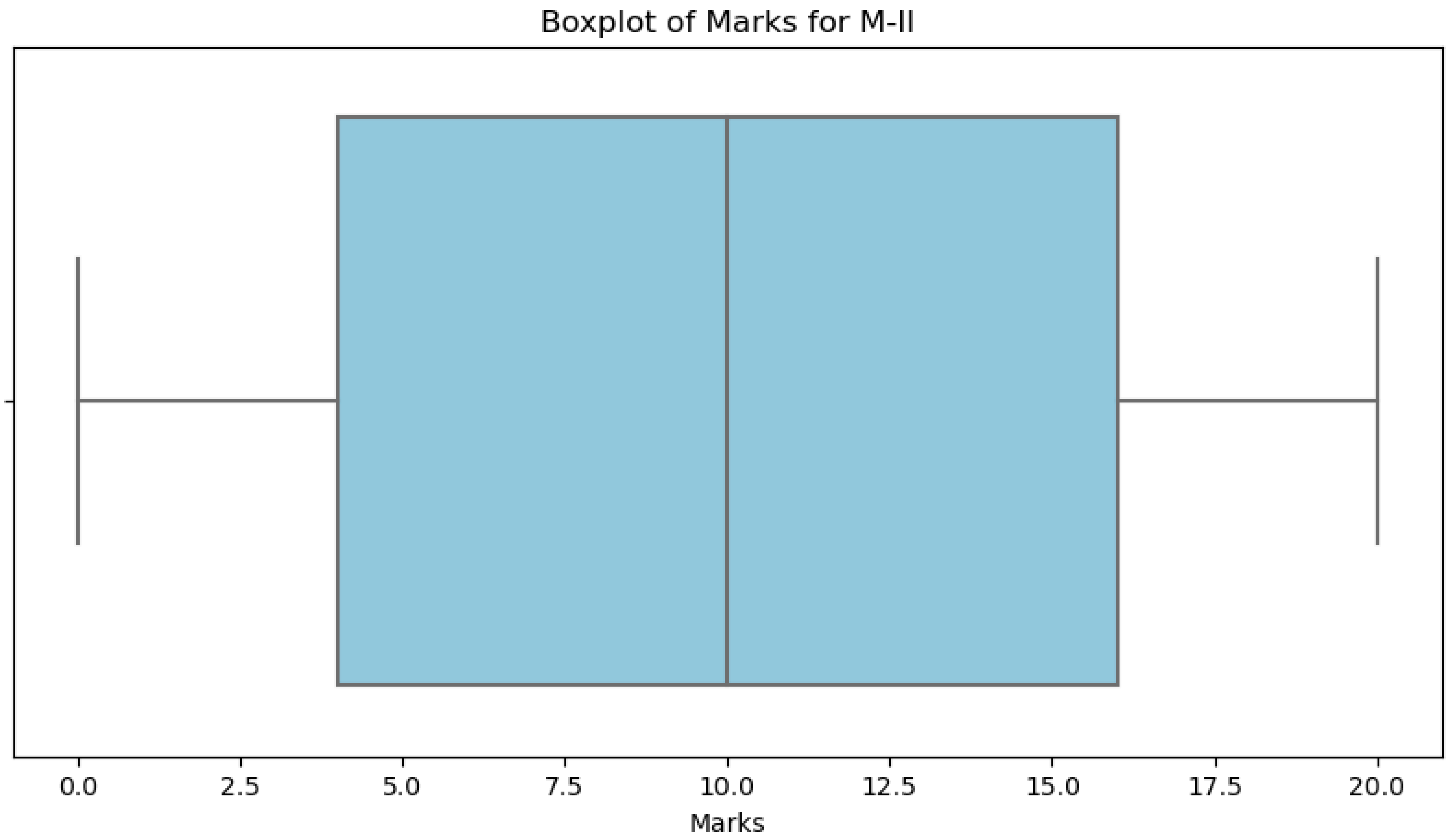

Distribution of Marks for DV
Mean: 15.02
Frequency
0
20
40
60
80
100
120
140
160
Marks
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



This kernel converts the 'DV' column to numeric, filters out rows with missing 'DV' marks, and then visualizes the distribution of the marks through a histogram (with KDE) and a boxplot, highlighting the mean with a vertical line on the histogram.

In [14]:
```
df['M-II'] = pd.to_numeric(df['M-II'], errors='coerce')

df_filtered = df.dropna(subset=['M-II'])

plt.figure(figsize=(10, 5))
sns.histplot(df_filtered['M-II'], kde=True, bins=15, color='blue', alpha=0.6, edgecolor='black')
plt.title('Distribution of Marks for M-II')
plt.xlabel('Marks')
plt.ylabel('Frequency')
plt.axvline(df_filtered['M-II'].mean(), color='red', linestyle='--', linewidth=1.5, label=f'Mean: {df_filtered["M-II"]
plt.legend()
plt.show()

plt.figure(figsize=(10, 5))
sns.boxplot(x=df_filtered['M-II'], color='skyblue')
plt.title('Boxplot of Marks for M-II')
plt.xlabel('Marks')
plt.show()
```

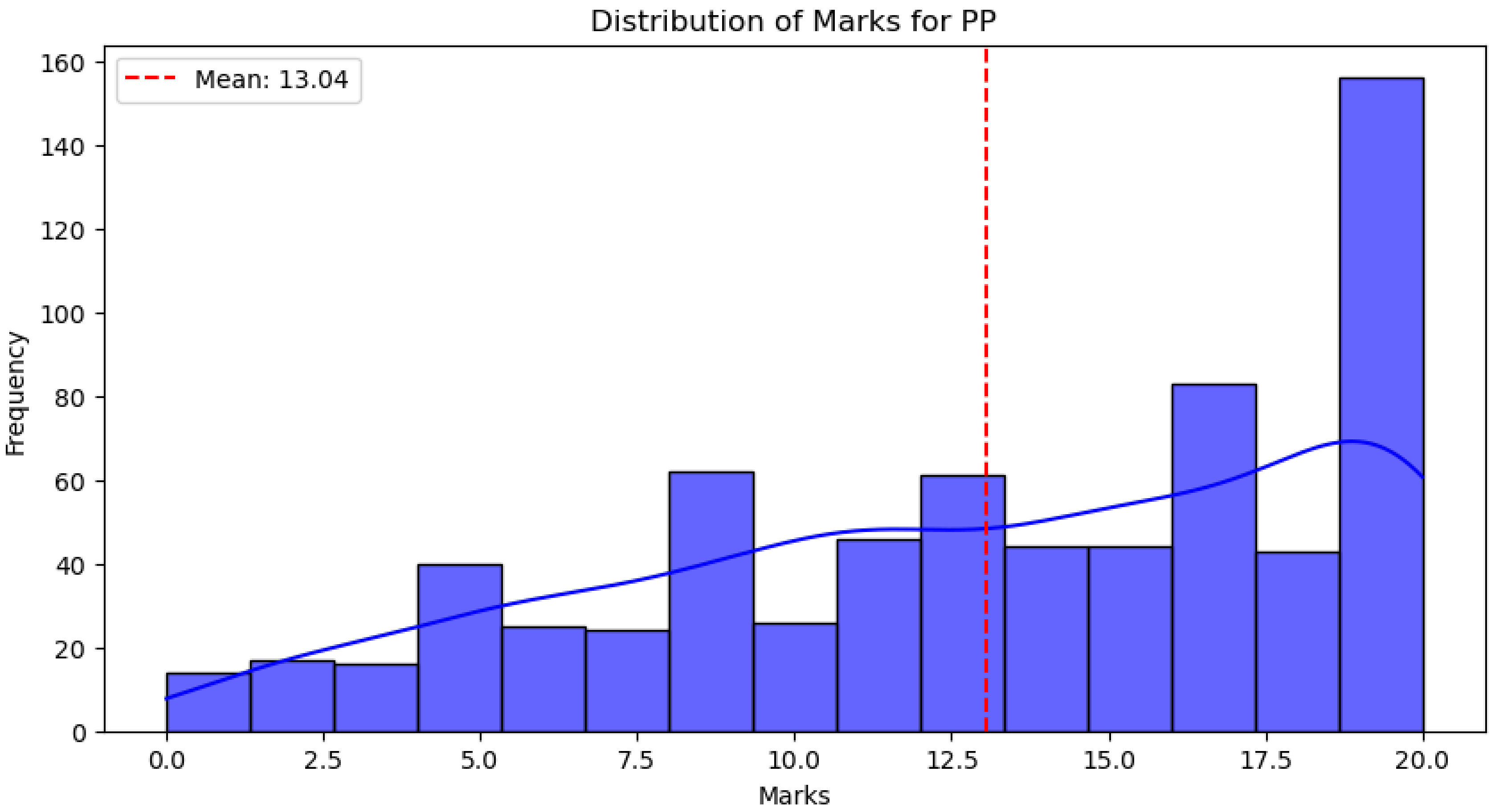

Distribution of Marks for M-II
Mean: 10.12
Frequency
Marks
0
20
40
60
80
100
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



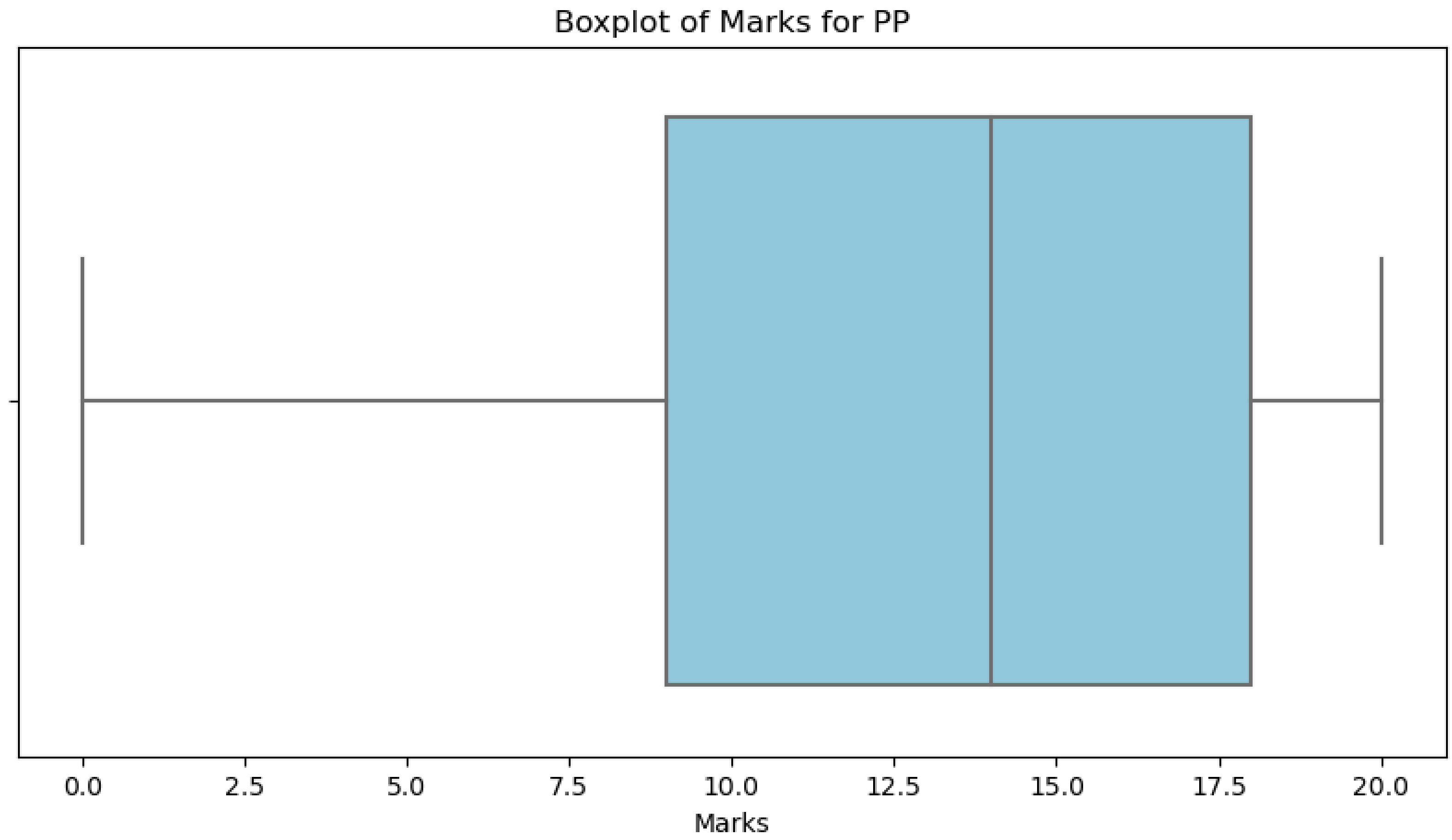
This kernel converts the 'M-II' column to numeric, filters out rows with missing 'M-II' marks, and then visualizes the distribution of the marks through a histogram (with KDE) and a boxplot, highlighting the mean with a vertical line on the histogram.

In [16]:

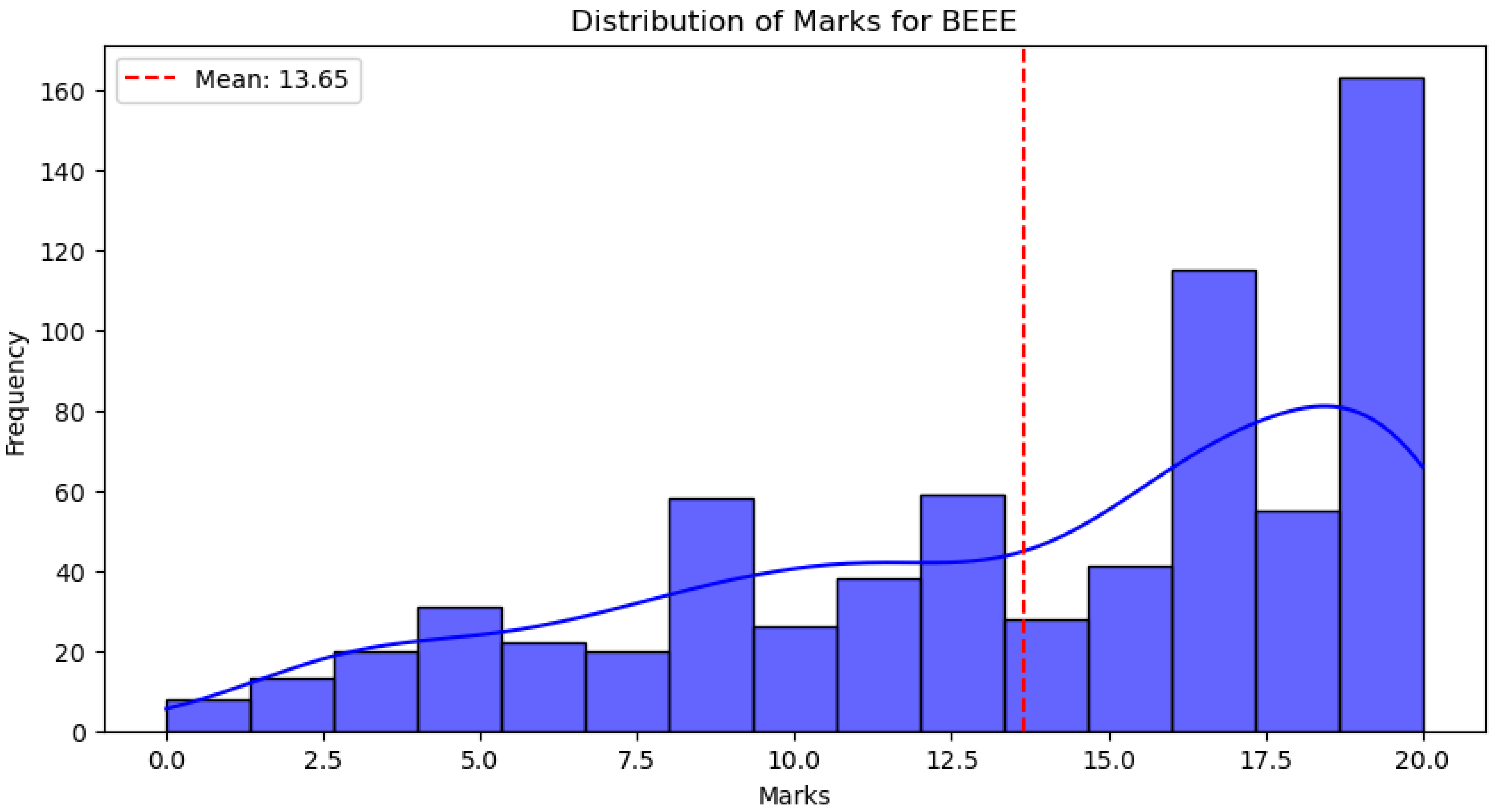
```
df['PP'] = pd.to_numeric(df['PP'], errors='coerce')

df_filtered_pp = df.dropna(subset=['PP'])

plt.figure(figsize=(10, 5))
sns.histplot(df_filtered_pp['PP'], kde=True, bins=15, color='blue', alpha=0.6, edgecolor='black')
plt.title('Distribution of Marks for PP')
plt.xlabel('Marks')
plt.ylabel('Frequency')
plt.axvline(df_filtered_pp['PP'].mean(), color='red', linestyle='--', linewidth=1.5, label=f'Mean: {df_filtered_pp["PP
plt.legend()
plt.show()

plt.figure(figsize=(10, 5))
sns.boxplot(x=df_filtered_pp['PP'], color='skyblue')
plt.title('Boxplot of Marks for PP')
plt.xlabel('Marks')
plt.show()
```



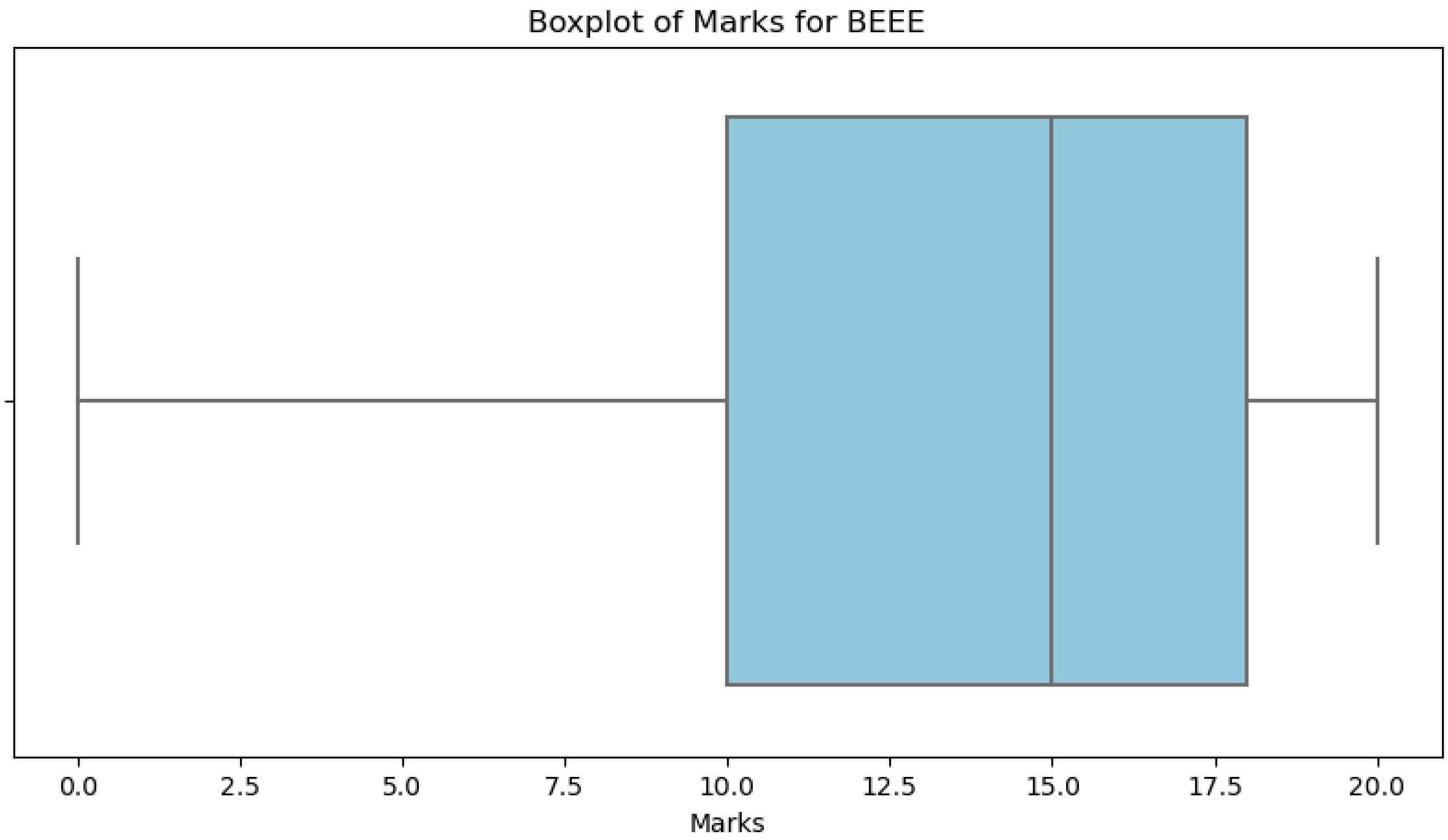
Distribution of Marks for PP
Mean: 13.04
Frequency
0
20
40
60
80
100
120
140
160
Marks
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Boxplot of Marks for PP
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Marks

In [17]:
```
df['BEEE'] = pd.to_numeric(df['BEEE'], errors='coerce')

df_filtered_beee = df.dropna(subset=['BEEE'])

plt.figure(figsize=(10, 5))
sns.histplot(df_filtered_beee['BEEE'], kde=True, bins=15, color='blue', alpha=0.6, edgecolor='black')
plt.title('Distribution of Marks for BEEE')
plt.xlabel('Marks')
plt.ylabel('Frequency')
plt.axvline(df_filtered_beee['BEEE'].mean(), color='red', linestyle='--', linewidth=1.5, label=f'Mean: {df_filtered_be
plt.legend()
plt.show()

plt.figure(figsize=(10, 5))
sns.boxplot(x=df_filtered_beee['BEEE'], color='skyblue')
plt.title('Boxplot of Marks for BEEE')
plt.xlabel('Marks')
plt.show()
```

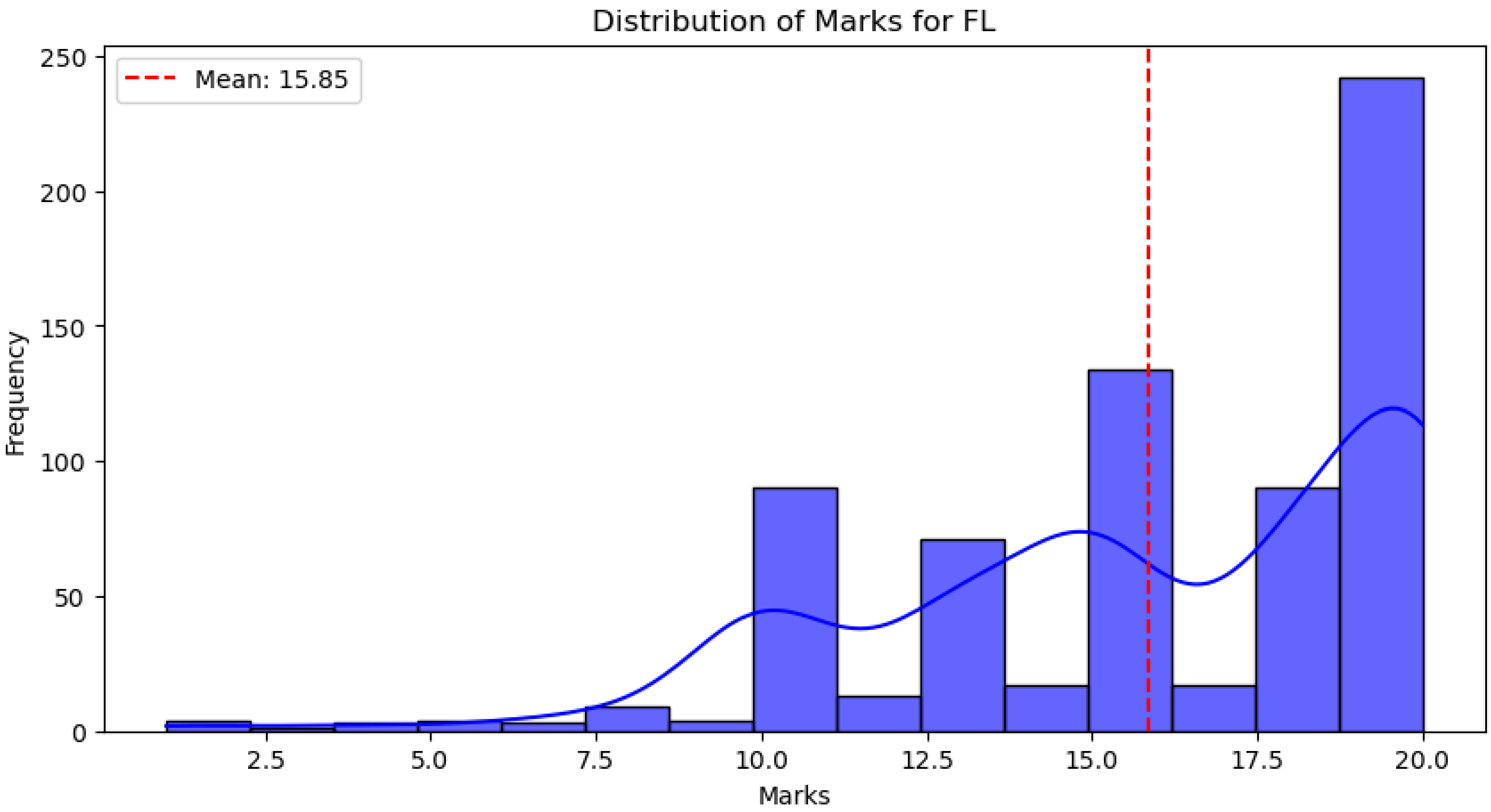


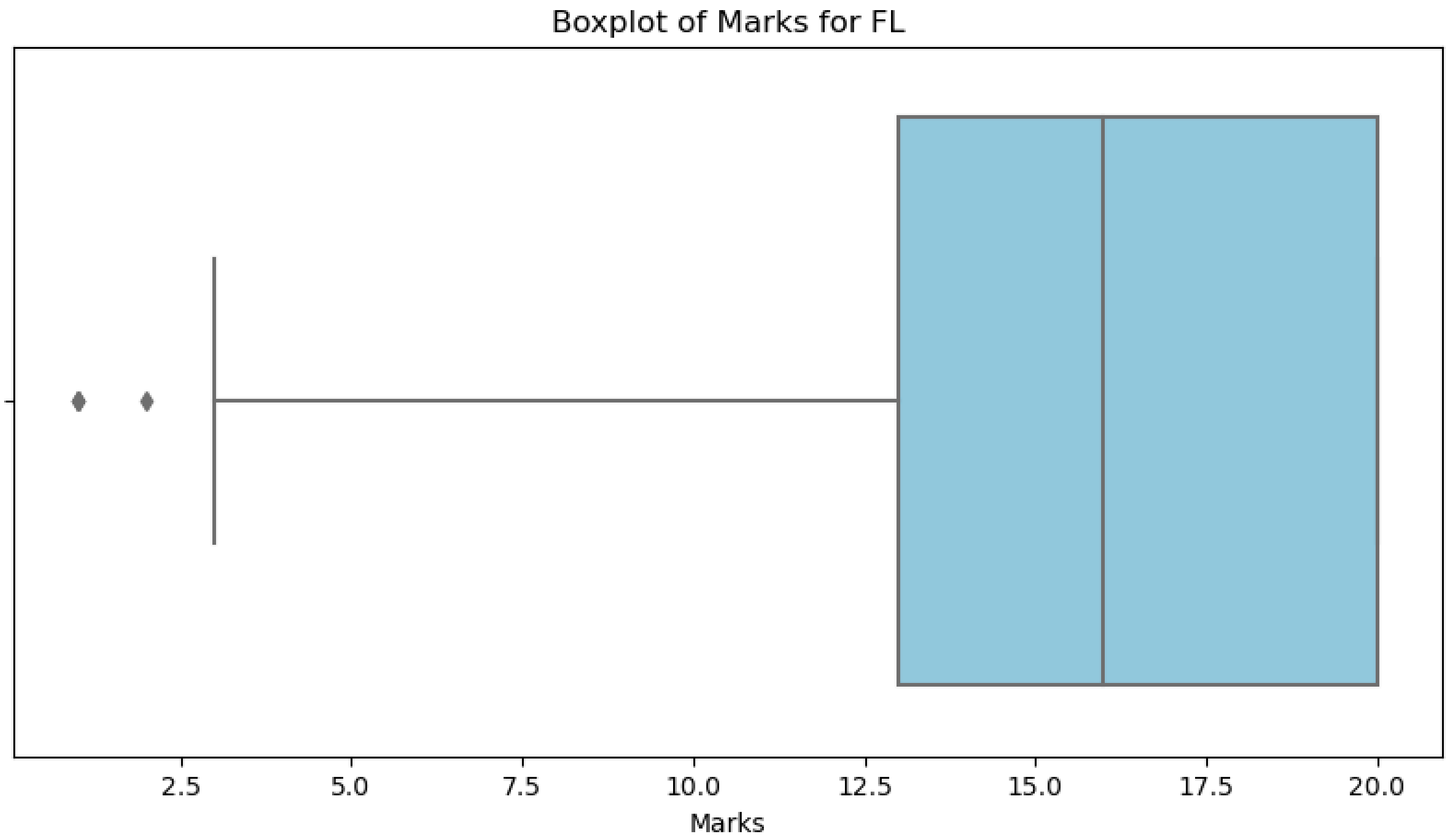
Distribution of Marks for BEEE
Mean: 13.65
Frequency
Marks
0
20
40
60
80
100
120
140
160
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



This kernel converts the 'BEEE' column to numeric, filters out rows with missing 'BEEE' marks, and then visualizes the distribution of the marks through a histogram (with KDE) and a boxplot, highlighting the mean with a vertical line on the histogram.

```
In [19]: df['FL'] = pd.to_numeric(df['FL'], errors='coerce')

df_filtered_fl = df.dropna(subset=['FL'])

plt.figure(figsize=(10, 5))
sns.histplot(df_filtered_fl['FL'], kde=True, bins=15, color='blue', alpha=0.6, edgecolor='black')
plt.title('Distribution of Marks for FL')
plt.xlabel('Marks')
plt.ylabel('Frequency')
plt.axvline(df_filtered_fl['FL'].mean(), color='red', linestyle='--', linewidth=1.5, label=f'Mean: {df_filtered_fl["FL
plt.legend()
plt.show()

plt.figure(figsize=(10, 5))
sns.boxplot(x=df_filtered_fl['FL'], color='skyblue')
plt.title('Boxplot of Marks for FL')
plt.xlabel('Marks')
plt.show()
```

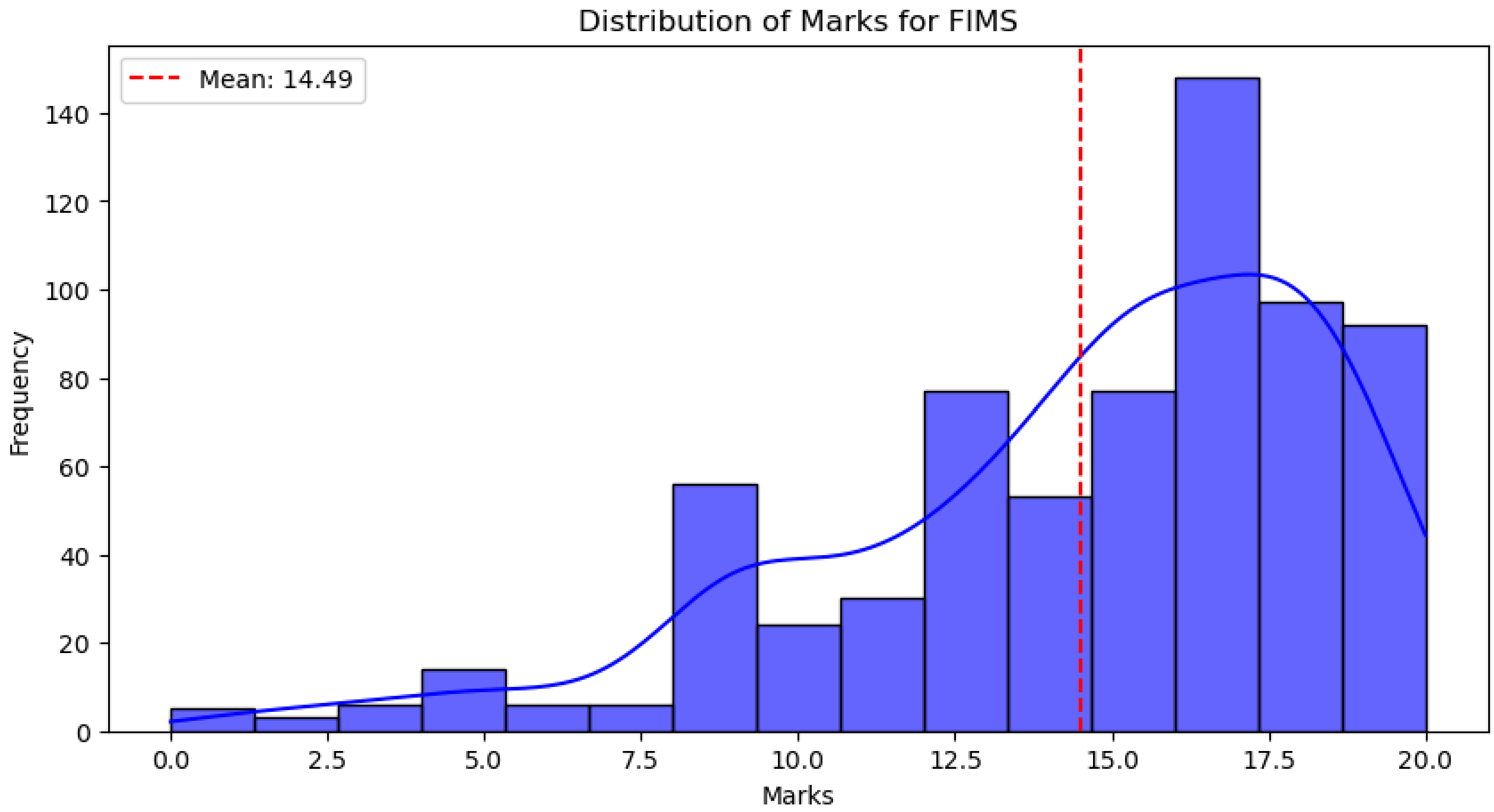

Distribution of Marks for FL
Mean: 15.85
Frequency
Marks
0
50
100
150
200
250
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

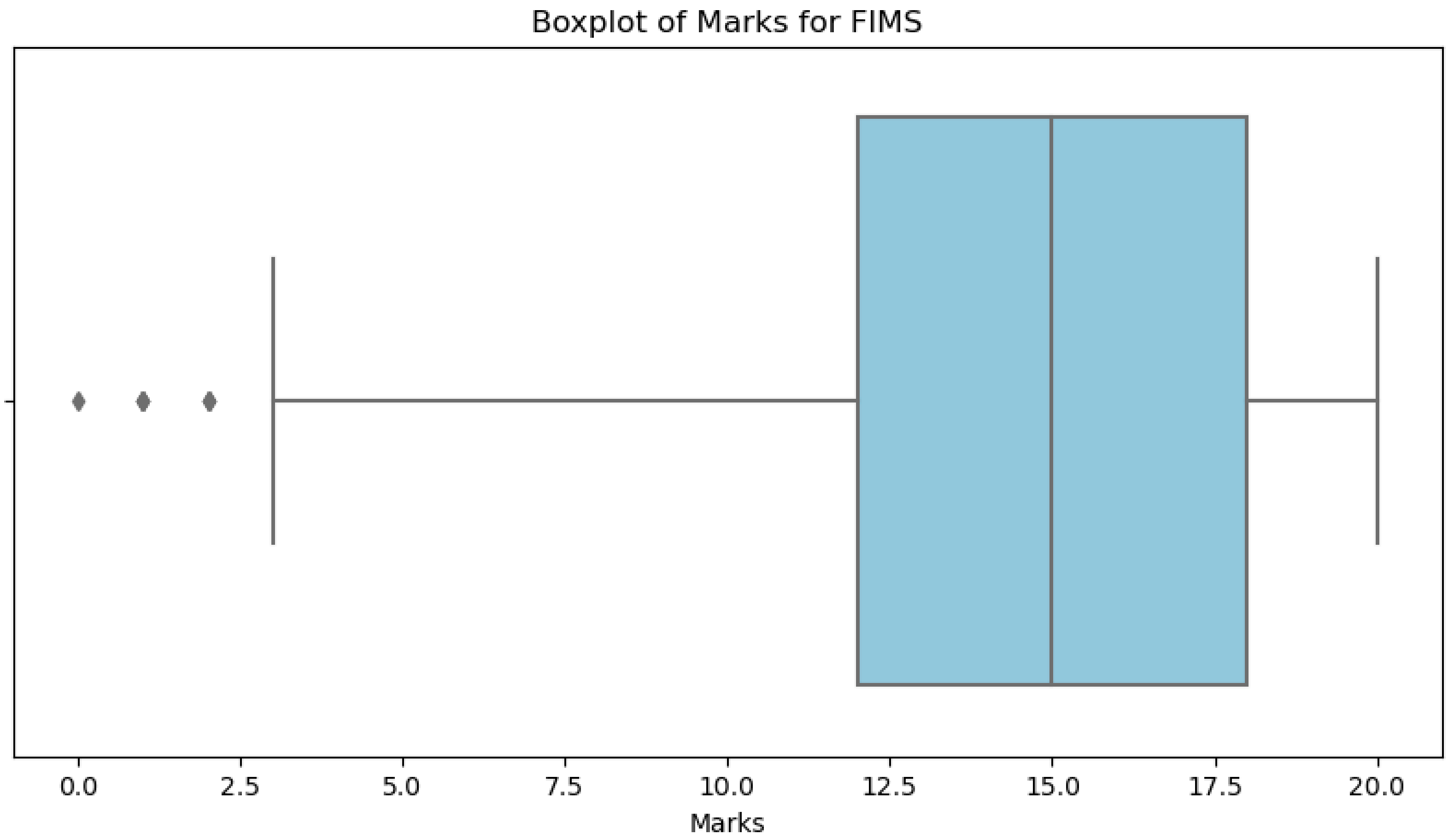


This kernel converts the 'FL' column to numeric, filters out rows with missing 'FL' marks, and then visualizes the distribution of the marks through a histogram (with KDE) and a boxplot, highlighting the mean with a vertical line on the histogram.

In [21]:

```
df['FIMS'] = pd.to_numeric(df['FIMS'], errors='coerce')

df_filtered_fims = df.dropna(subset=['FIMS'])

plt.figure(figsize=(10, 5))
sns.histplot(df_filtered_fims['FIMS'], kde=True, bins=15, color='blue', alpha=0.6, edgecolor='black')
plt.title('Distribution of Marks for FIMS')
plt.xlabel('Marks')
plt.ylabel('Frequency')
plt.axvline(df_filtered_fims['FIMS'].mean(), color='red', linestyle='--', linewidth=1.5, label=f'Mean: {df_filtered_fi
plt.legend()
plt.show()

plt.figure(figsize=(10, 5))
sns.boxplot(x=df_filtered_fims['FIMS'], color='skyblue')
plt.title('Boxplot of Marks for FIMS')
plt.xlabel('Marks')
plt.show()
```


Distribution of Marks for FIMS
Mean: 14.49
Frequency
Marks
0
20
40
60
80
100
120
140
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



This kernel converts the 'FIMS' column to numeric, filters out rows with missing 'FIMS' marks, and then visualizes the distribution of the marks through a histogram (with KDE) and a boxplot, highlighting the mean with a vertical line on the histogram.

In [23]:
```
df['Total Marks'] = df[marks_columns].sum(axis=1)
df
```

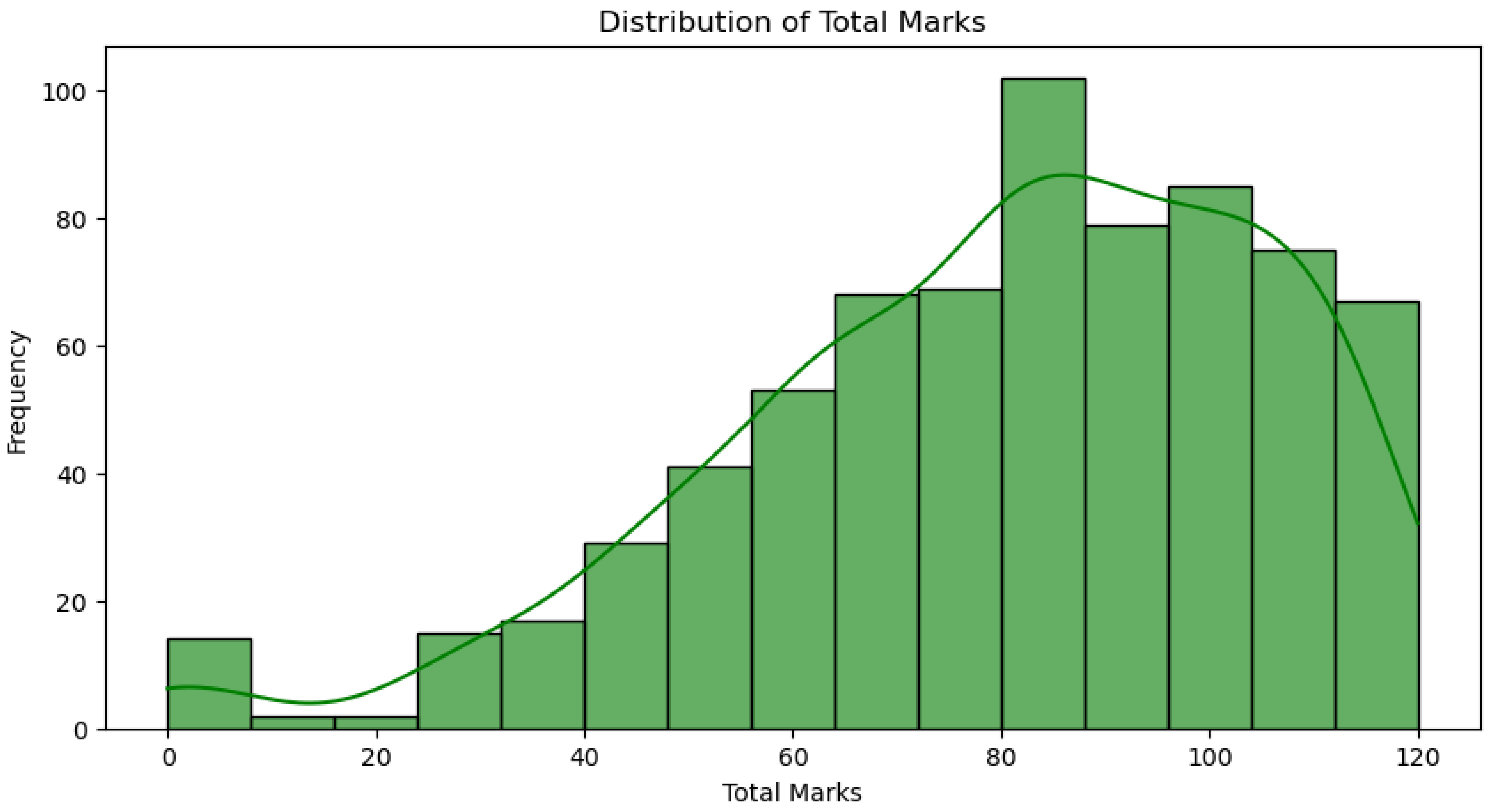
Out[23]:

	S.NO	SECTION	DV	M-II	PP	BEEE	FL	FIMS	Total Marks
0	1.0	ALPHA	12.0	0.0	17.0	9.0	19.0	15.0	72.0
1	2.0	ALPHA	19.0	12.0	16.0	16.0	18.0	3.0	84.0
2	3.0	ALPHA	18.0	14.0	18.0	18.0	18.0	16.0	102.0
3	4.0	ALPHA	15.0	9.0	19.0	17.0	19.0	15.0	94.0
4	5.0	ALPHA	18.0	17.0	19.0	19.0	20.0	18.0	111.0
...	...	...	...	...	...	...	...	...	...
713	NaN	ZETA	19.0	8.0	8.0	19.0	17.0	18.0	89.0
714	NaN	ZETA	12.0	1.0	7.0	10.0	20.0	8.0	58.0
715	NaN	ZETA	17.0	6.0	14.0	14.0	17.0	18.0	86.0
716	NaN	ZETA	12.0	1.0	6.0	7.0	15.0	12.0	53.0
717	NaN	ZETA	19.0	14.0	17.0	16.0	20.0	19.0	105.0

718 rows × 9 columns

This kernel calculates the total marks for each student by summing the marks across all specified subjects (`marks_columns`) and adds a new column, `Total Marks`, to the DataFrame.

In [25]:
```
plt.figure(figsize=(10, 5))
sns.histplot(df['Total Marks'], kde=True, bins=15, color='green', alpha=0.6)
plt.title('Distribution of Total Marks')
plt.xlabel('Total Marks')
plt.ylabel('Frequency')
plt.show()
```



This kernel visualizes the distribution of total marks for all students using a histogram with a kernel density estimate (KDE), which helps in

understanding the overall spread and central tendency of the total

In [27]:
```
df['Average Marks'] = df[marks_columns].mean(axis=1)
df
```

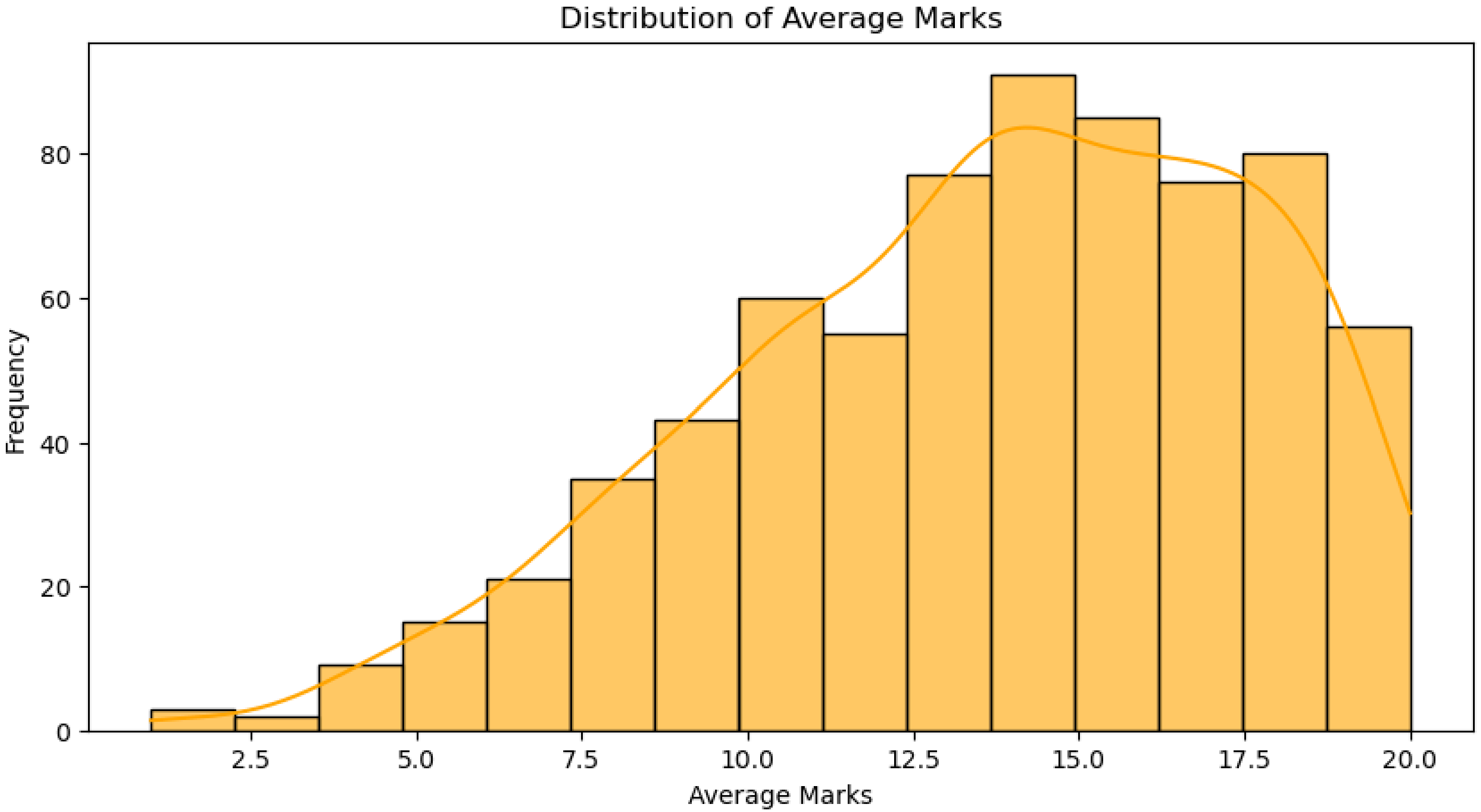
Out[27]:

	S.NO	SECTION	DV	M-II	PP	BEEE	FL	FIMS	Total Marks	Average Marks
0	1.0	ALPHA	12.0	0.0	17.0	9.0	19.0	15.0	72.0	12.000000
1	2.0	ALPHA	19.0	12.0	16.0	16.0	18.0	3.0	84.0	14.000000
2	3.0	ALPHA	18.0	14.0	18.0	18.0	18.0	16.0	102.0	17.000000
3	4.0	ALPHA	15.0	9.0	19.0	17.0	19.0	15.0	94.0	15.666667
4	5.0	ALPHA	18.0	17.0	19.0	19.0	20.0	18.0	111.0	18.500000
...	...	...	...	...	...	...	...	...	...	...
713	NaN	ZETA	19.0	8.0	8.0	19.0	17.0	18.0	89.0	14.833333
714	NaN	ZETA	12.0	1.0	7.0	10.0	20.0	8.0	58.0	9.666667
715	NaN	ZETA	17.0	6.0	14.0	14.0	17.0	18.0	86.0	14.333333
716	NaN	ZETA	12.0	1.0	6.0	7.0	15.0	12.0	53.0	8.833333
717	NaN	ZETA	19.0	14.0	17.0	16.0	20.0	19.0	105.0	17.500000

718 rows × 10 columns

This kernel calculates the average marks for each student by taking the mean of their marks across the specified subjects and adds it as a new column `Average Marks` in the DataFrame.

In [29]:
```
plt.figure(figsize=(10, 5))
sns.histplot(df['Average Marks'], kde=True, bins=15, color='orange', alpha=0.6)
plt.title('Distribution of Average Marks')
plt.xlabel('Average Marks')
plt.ylabel('Frequency')
plt.show()
```



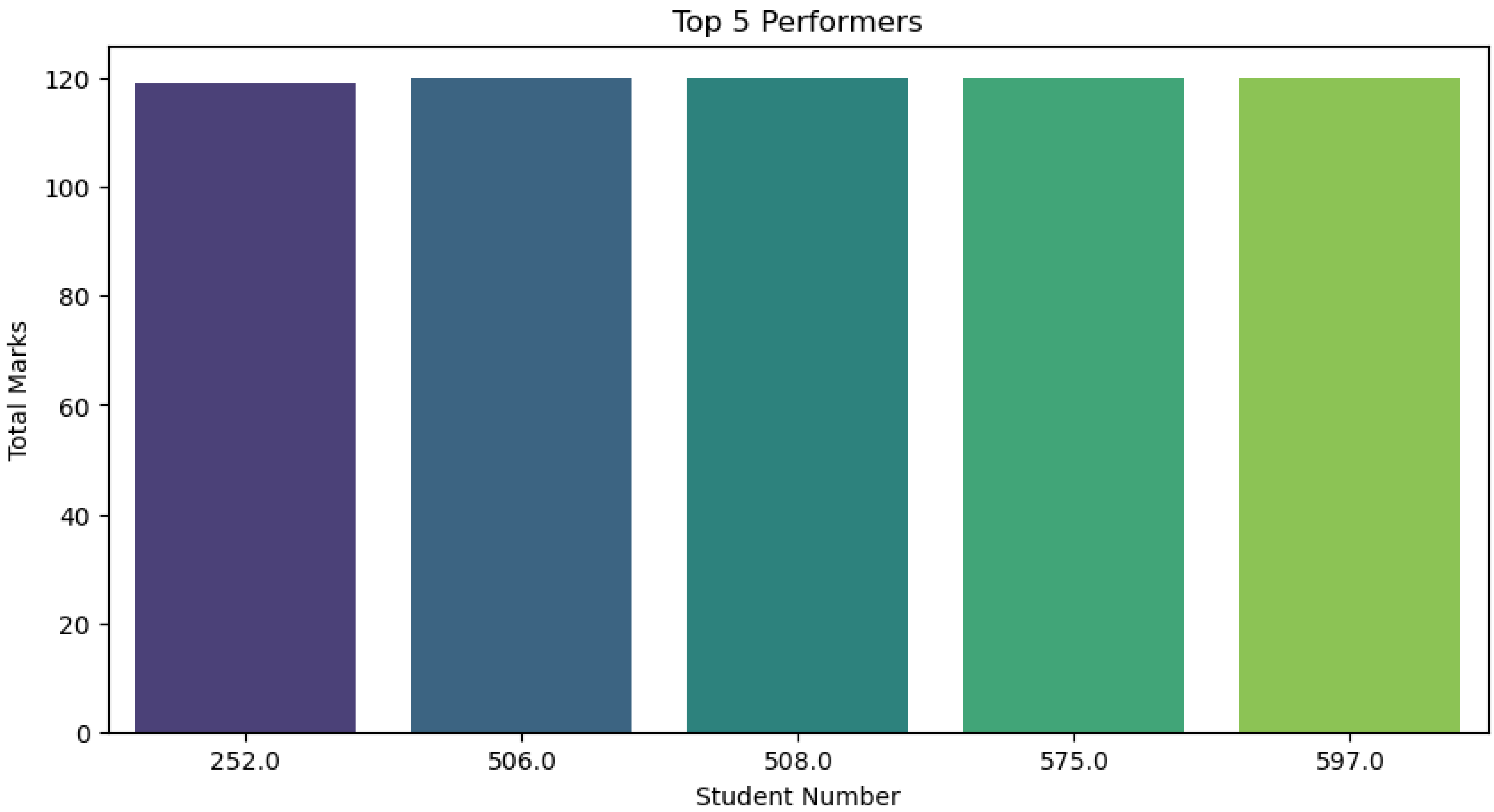
This kernel generates a histogram with a kernel density estimate (KDE) for the distribution of the average marks across all students, using the

`Average Marks` column. The plot helps visualize the spread and central

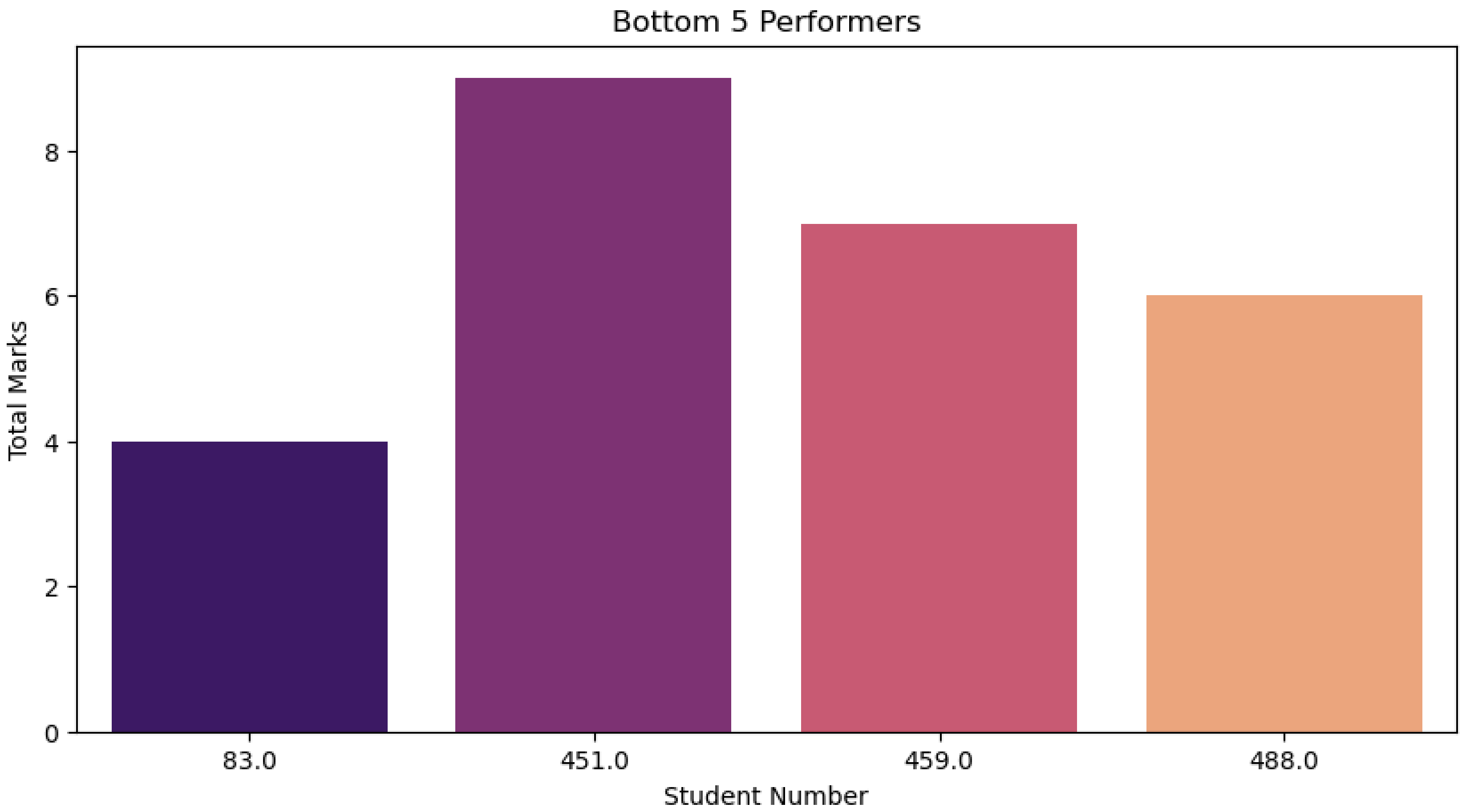
In [31]:

```
print("Top 5 Performers:")
top_5 = df.nlargest(5, 'Total Marks')[['S.NO', 'Total Marks', 'Average Marks']]
print(top_5)

plt.figure(figsize=(10, 5))
sns.barplot(
    x='S.NO',
    y='Total Marks',
    data=top_5,
    palette='viridis'
)
plt.title("Top 5 Performers")
plt.xlabel("Student Number")
plt.ylabel("Total Marks")
plt.show()

print("\nBottom 5 Performers:")

bottom_5 = df[df['Total Marks'].notna() & (df['Total Marks'] > 0)].nsmallest(5, 'Total Marks')[['S.NO', 'Total Marks',

if not bottom_5.empty:
    print(bottom_5)

    plt.figure(figsize=(10, 5))
    sns.barplot(
        x='S.NO',
        y='Total Marks',
        data=bottom_5,
        palette='magma'
    )
    plt.title("Bottom 5 Performers")
    plt.xlabel("Student Number")
    plt.ylabel("Total Marks")
    plt.show()
else:
    print("Not enough valid data for bottom 5 performers.")
```

```
Top 5 Performers:
      S.NO  Total Marks  Average Marks
505  506.0        120.0      20.000000
507  508.0        120.0      20.000000
574  575.0        120.0      20.000000
596  597.0        120.0      20.000000
251  252.0        119.0      19.833333
```



```
Bottom 5 Performers:
      S.NO  Total Marks  Average Marks
82    83.0          4.0       1.333333
673    NaN          5.0       1.000000
487  488.0          6.0       2.000000
458  459.0          7.0       3.500000
450  451.0          9.0       4.500000
```



This kernel identifies and displays the top 5 performers (students with the highest total marks) and the bottom 5 performers (students with the

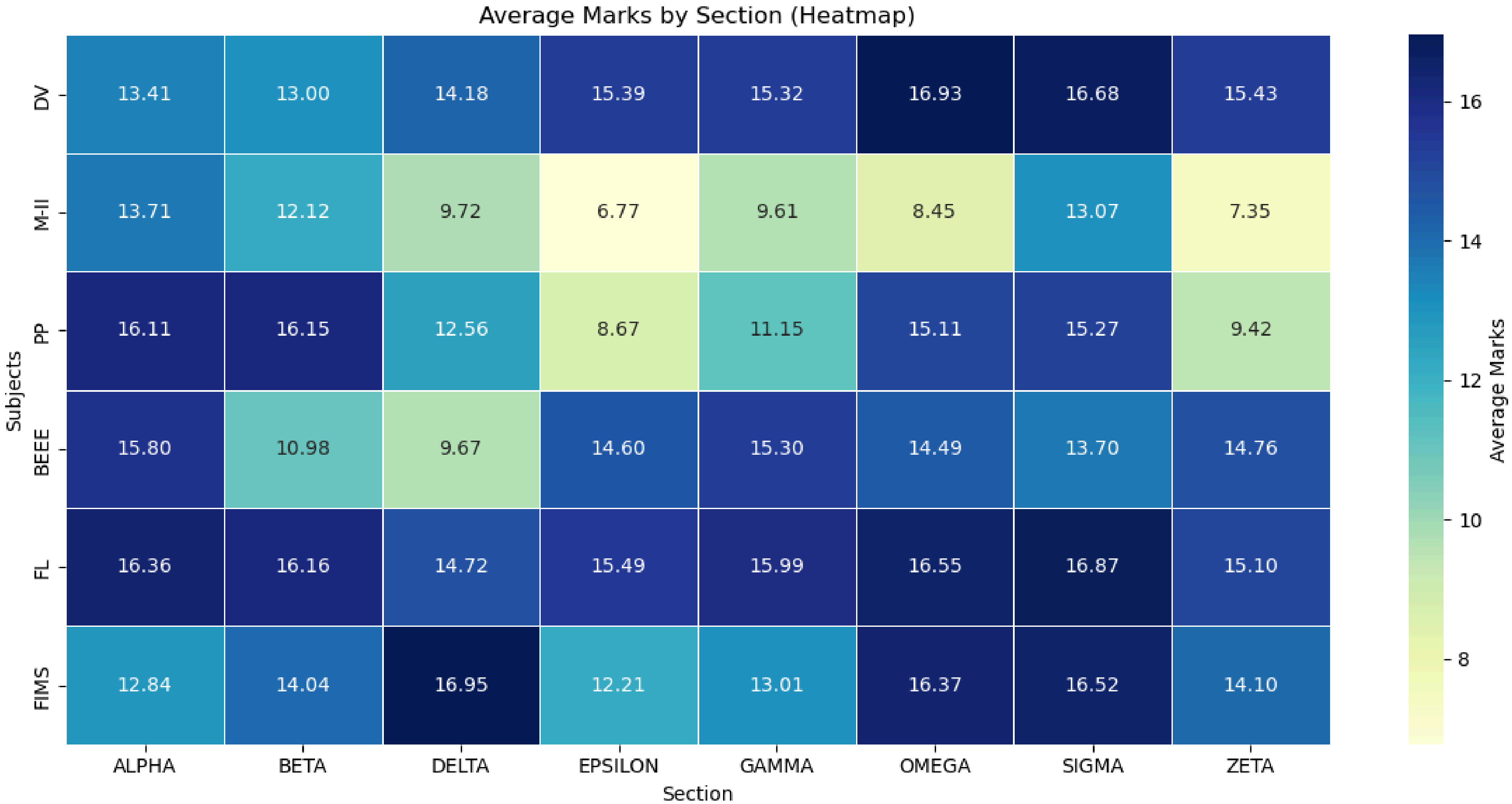
lowest total marks, excluding invalid or missing data), and visualizes their total marks using bar plots.

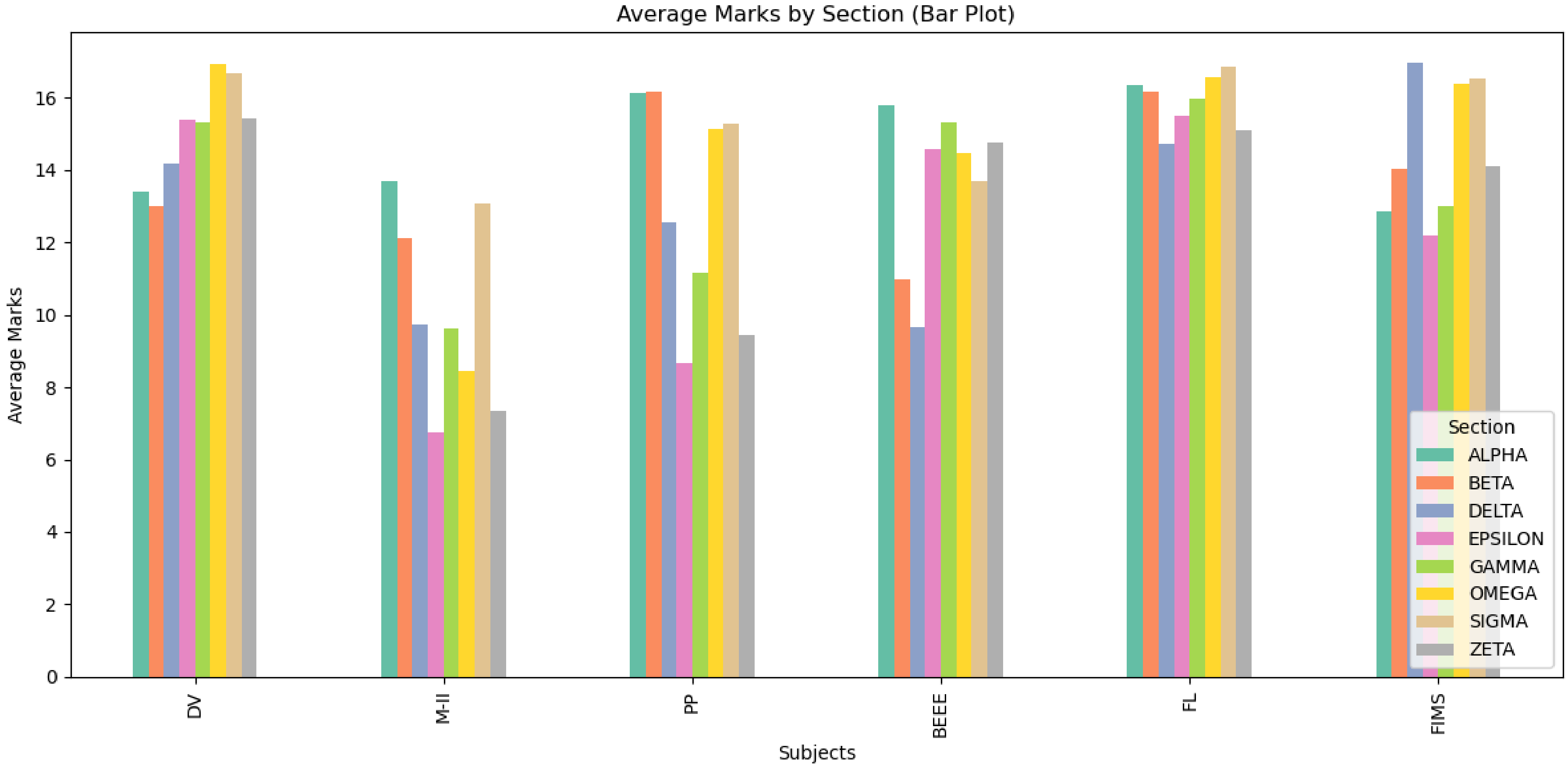
In [33]:
```python
if 'SECTION' in df.columns:
    missing_marks_columns = [col for col in marks_columns if col not in df.columns]
    if missing_marks_columns:
        print(f"Missing marks columns: {missing_marks_columns}")
    else:
        section_summary = df.groupby('SECTION')[marks_columns + ['Total Marks', 'Average Marks']].mean()
        print("Section-Wise Summary:")
        print(section_summary)

        section_means = section_summary[marks_columns].T
        plt.figure(figsize=(12, 6))
        sns.heatmap(section_means, annot=True, fmt=".2f", cmap="YlGnBu", linewidths=0.5, cbar_kws={'label': 'Average M
        plt.title('Average Marks by Section (Heatmap)')
        plt.xlabel('Section')
        plt.ylabel('Subjects')
        plt.tight_layout()
        plt.show()

        section_means.plot(kind='bar', figsize=(12, 6), colormap='Set2')
        plt.title('Average Marks by Section (Bar Plot)')
        plt.ylabel('Average Marks')
        plt.xlabel('Subjects')
        plt.legend(title='Section', loc='lower right')
        plt.tight_layout()
        plt.show()

else:
    print("No 'SECTION' column found in the dataset.")
```

Section-Wise Summary:

SECTION	DV	M-II	PP	BEEE	FL	FIMS \
ALPHA	13.411111	13.711111	16.112360	15.797753	16.359551	12.842697
BETA	13.000000	12.122222	16.146067	10.977528	16.157303	14.044444
DELTA	14.181818	9.715909	12.561798	9.670455	14.719101	16.954545
EPSILON	15.390805	6.770115	8.666667	14.597701	15.494253	12.206897
GAMMA	15.321839	9.609195	11.149425	15.298851	15.988636	13.011628
OMEGA	16.931818	8.454545	15.114943	14.488095	16.552941	16.373494
SIGMA	16.683333	13.066667	15.271186	13.700000	16.866667	16.517241
ZETA	15.426966	7.352273	9.420455	14.761364	15.101124	14.103448

SECTION	Total Marks	Average Marks
ALPHA	87.555556	14.630370
BETA	81.966667	13.735741
DELTA	76.377778	12.968352
EPSILON	72.295455	12.187739
GAMMA	77.733333	13.376515
OMEGA	83.688889	14.300749
SIGMA	86.952381	15.294444
ZETA	74.655556	12.666479

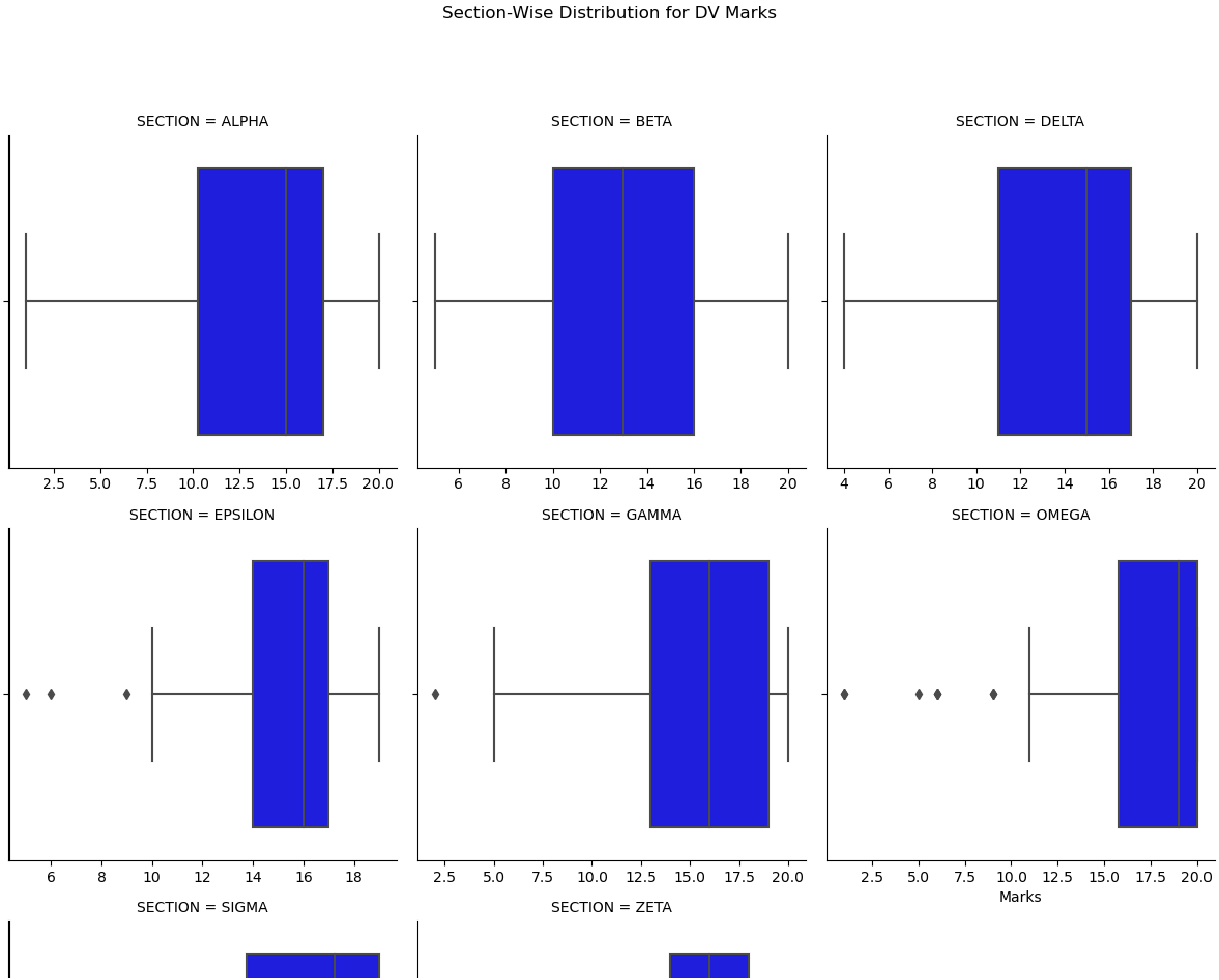

Average Marks by Section (Heatmap)
Subjects
DV
M-II
PP
BEEE
FL
FIMS
ALPHA
BETA
DELTA
EPSILON
GAMMA
OMEGA
SIGMA
ZETA
Section
Average Marks
16
14
12
10
8
13.41 13.00 14.18 15.39 15.32 16.93 16.68 15.43
13.71 12.12 9.72 6.77 9.61 8.45 13.07 7.35
16.11 16.15 12.56 8.67 11.15 15.11 15.27 9.42
15.80 10.98 9.67 14.60 15.30 14.49 13.70 14.76
16.36 16.16 14.72 15.49 15.99 16.55 16.87 15.10
12.84 14.04 16.95 12.21 13.01 16.37 16.52 14.10



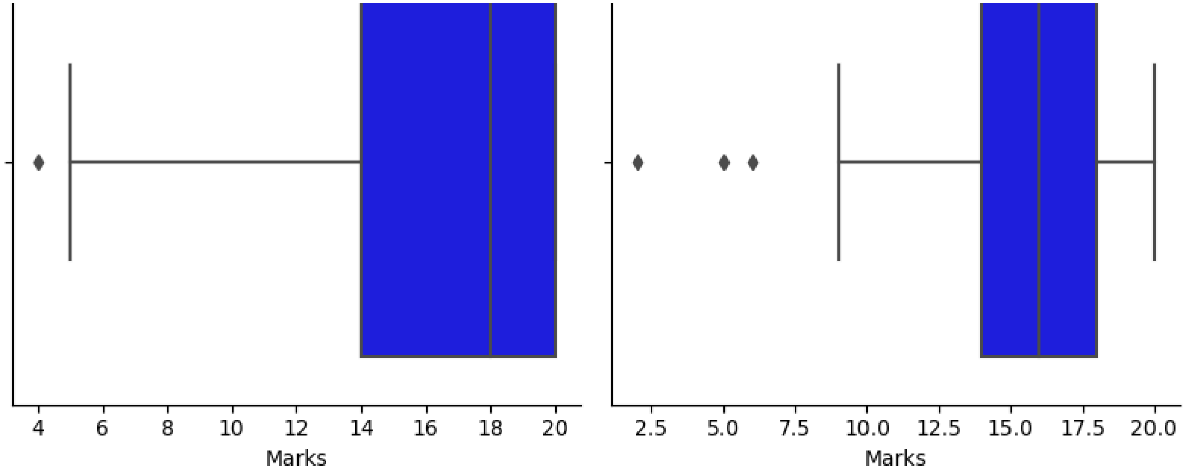
This kernel computes and visualizes the section-wise average marks for each subject through both a heatmap and bar plot. It also ensures that the required marks columns are present.

In [35]:
```python
if 'SECTION' in df.columns:
    df_cleaned = df.dropna(subset=['DV'])

    g = sns.FacetGrid(df_cleaned, col='SECTION', col_wrap=3, height=4, sharex=False, sharey=True)
    g.map(sns.boxplot, 'DV', color='blue')

    g.fig.suptitle('Section-Wise Distribution for DV Marks', y=1.05)
    g.set_axis_labels('Marks', '')
    plt.tight_layout()
    plt.show()
else:
    print("No 'SECTION' column found in the dataset.")
```

```
C:\Users\subha\anaconda3\Lib\site-packages\seaborn\axisgrid.py:712: UserWarning: Using the boxplot function without s
pecifying `order` is likely to produce an incorrect plot.
  warnings.warn(warning)
```



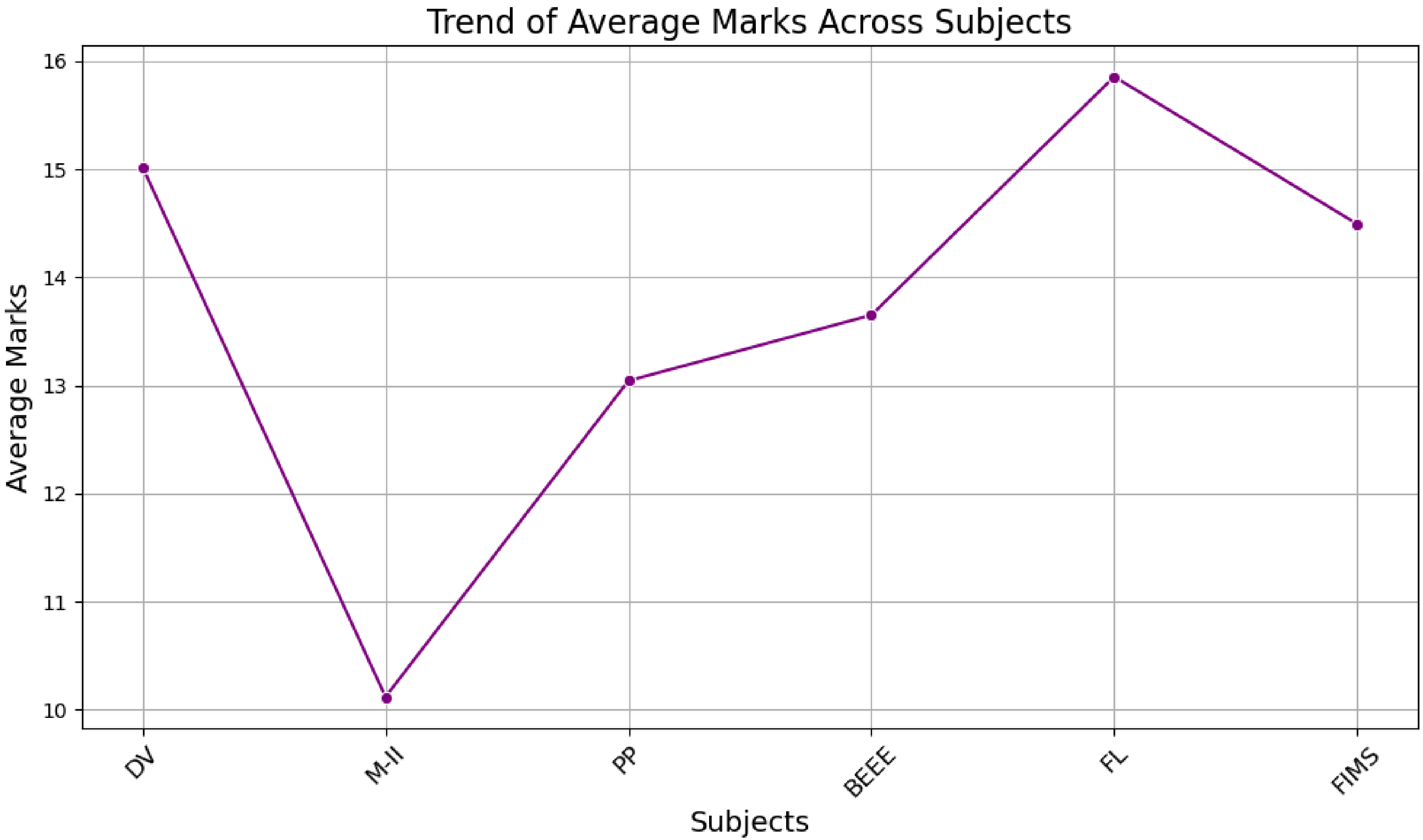
Section-Wise Distribution for DV Marks
SECTION = ALPHA
SECTION = BETA
SECTION = DELTA
SECTION = EPSILON
SECTION = GAMMA
SECTION = OMEGA
SECTION = SIGMA
SECTION = ZETA
Marks



This kernel filters the dataset to exclude rows with missing 'DV' marks and creates a FacetGrid to display section-wise boxplots of the 'DV' marks for each section, if the 'SECTION' column exists in the dataset.

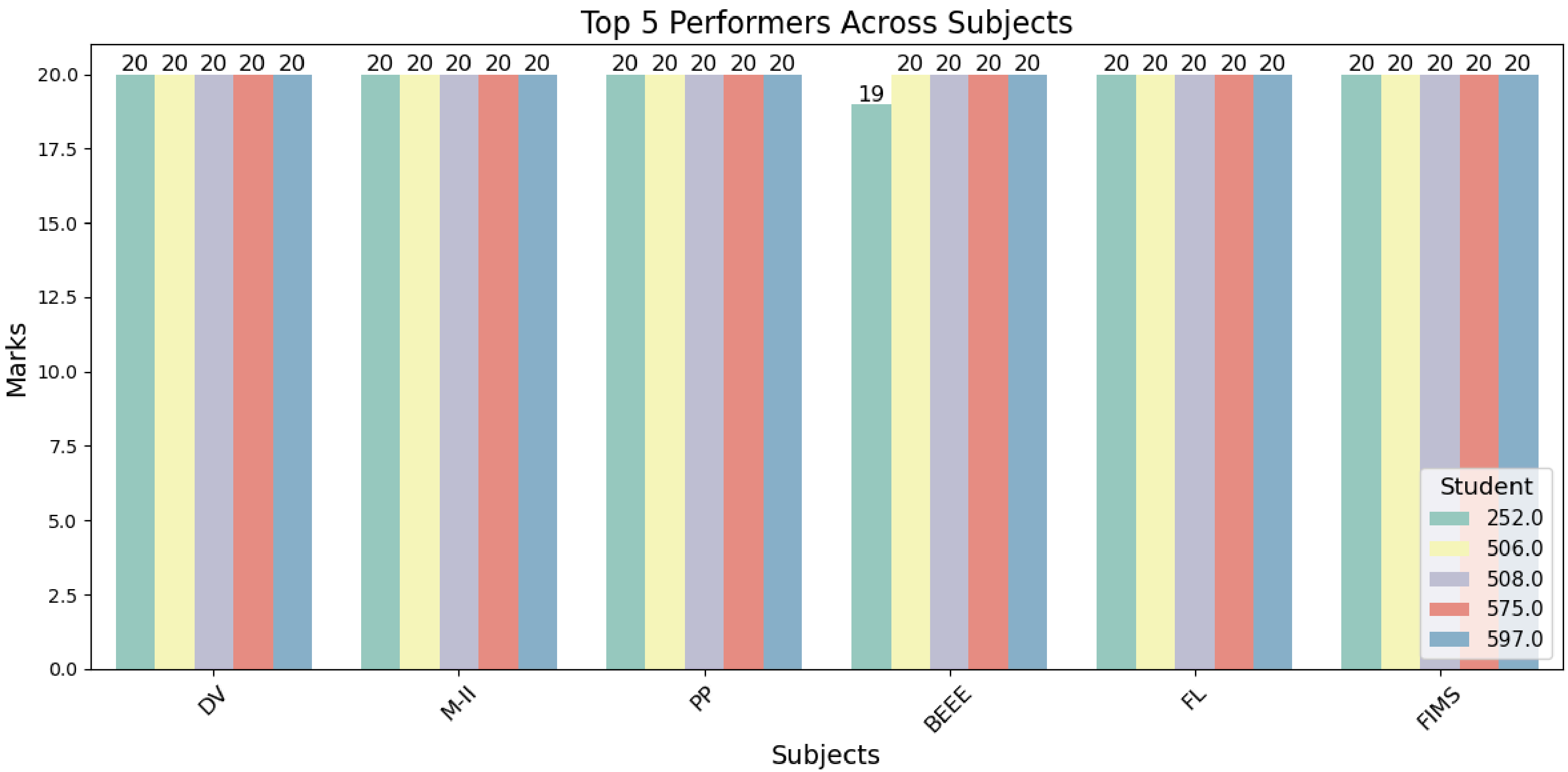
In [37]:

```
subject_averages = df[marks_columns].mean()

plt.figure(figsize=(10, 6))
sns.lineplot(x=subject_averages.index, y=subject_averages.values, marker='o', linestyle='-', color='purple')

plt.title('Trend of Average Marks Across Subjects', fontsize=16)
plt.ylabel('Average Marks', fontsize=14)
plt.xlabel('Subjects', fontsize=14)

plt.xticks(ticks=range(len(marks_columns)), labels=marks_columns, rotation=45, fontsize=12)

plt.grid(True)

plt.tight_layout()
plt.show()
```



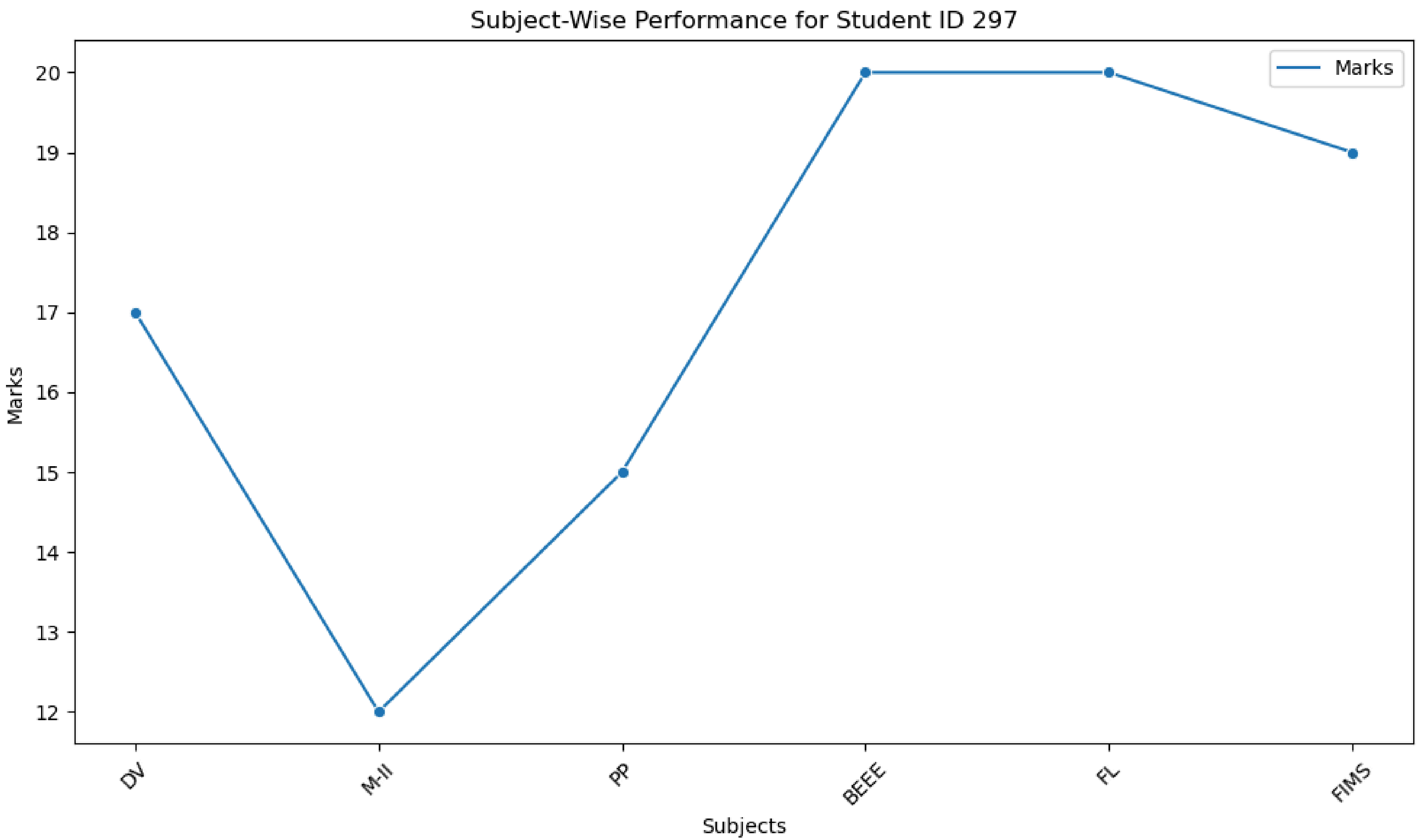
This kernel calculates and visualizes the trend of average marks across different subjects by plotting a line graph with markers, showcasing the subject-wise performance in the dataset.

In [39]:
```python
top_students = df.nlargest(5, 'Total Marks')
plt.figure(figsize=(12, 6))
ax = sns.barplot(data=top_students.melt(id_vars=['S.NO'], value_vars=marks_columns),
                 x='variable', y='value', hue='S.NO', palette='Set3', dodge=True)
plt.title('Top 5 Performers Across Subjects', fontsize=16)
plt.ylabel('Marks', fontsize=14)
plt.xlabel('Subjects', fontsize=14)
plt.legend(title='Student', title_fontsize='13', fontsize='11', loc='lower right')
plt.xticks(rotation=45, fontsize=12)

for p in ax.patches:
    height = p.get_height()
    if not pd.isna(height):
        ax.annotate(f'{int(height)}',
                    (p.get_x() + p.get_width() / 2., height),
                    ha='center', va='center', fontsize=12, color='black',
                    xytext=(0, 5), textcoords='offset points')

plt.tight_layout()
plt.show()
```



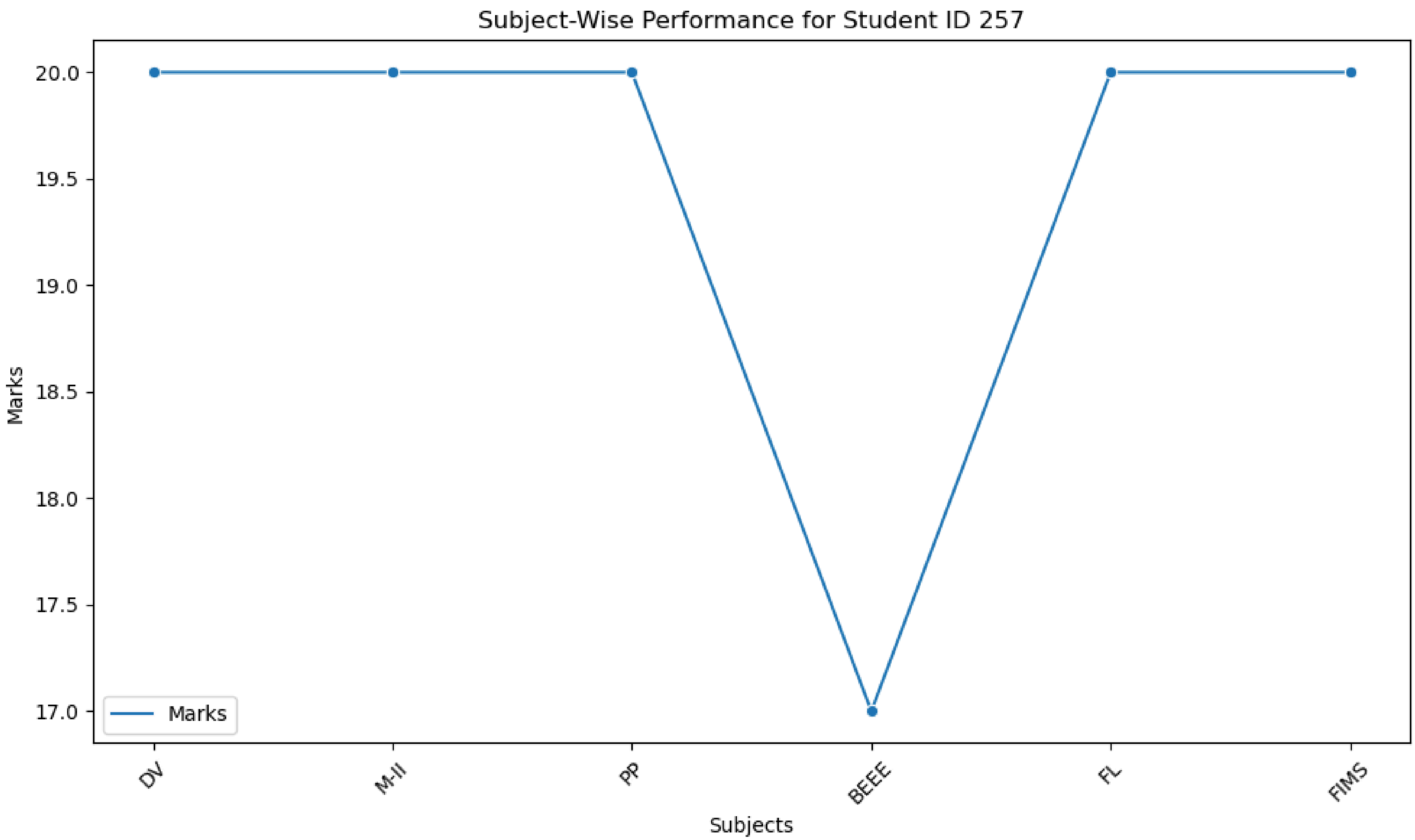
This kernel identifies the top 5 students based on total marks and visualizes their performance across various subjects using a bar plot with annotations showing the exact marks for each student in each subject.

In [41]:
```
student_id = 297
student_data = df.loc[df['S.NO'] == student_id, marks_columns].T
student_data.columns = ['Marks']

plt.figure(figsize=(10, 6))
sns.lineplot(data=student_data, marker='o', color='teal')
plt.title(f'Subject-Wise Performance for Student ID {student_id}')
plt.ylabel('Marks')
plt.xlabel('Subjects')
plt.xticks(ticks=range(len(marks_columns)), labels=marks_columns, rotation=45)
plt.tight_layout()
plt.show()
```



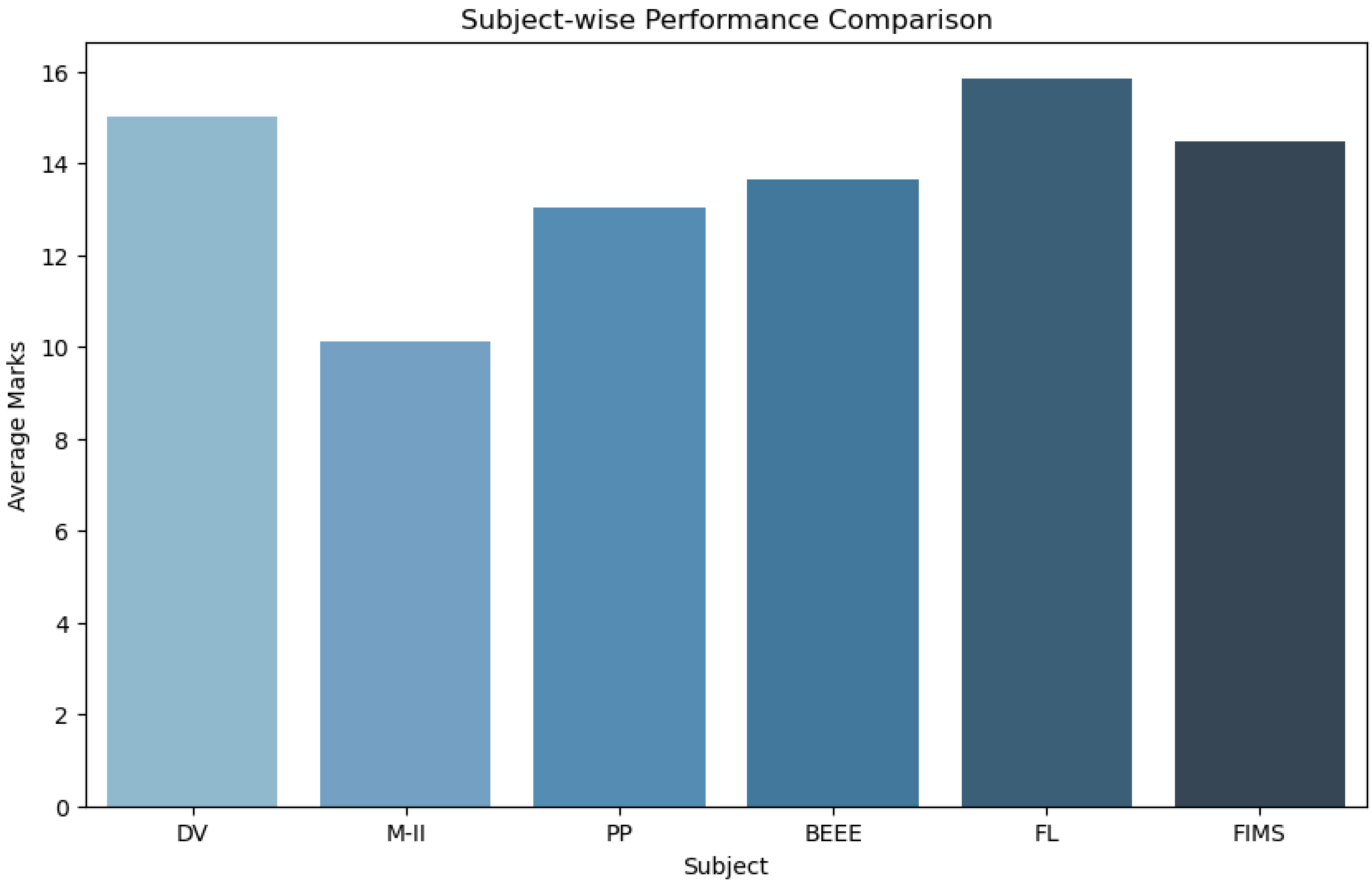
This kernel extracts and visualizes the subject-wise performance of a specific student (Roll no.: 297) through a line plot, displaying their marks across different subjects.

In [43]:
```
student_id = 257
student_data = df.loc[df['S.NO'] == student_id, marks_columns].T
student_data.columns = ['Marks']

plt.figure(figsize=(10, 6))
sns.lineplot(data=student_data, marker='o', color='orange')
plt.title(f'Subject-Wise Performance for Student ID {student_id}')
plt.ylabel('Marks')
plt.xlabel('Subjects')
plt.xticks(ticks=range(len(marks_columns)), labels=marks_columns, rotation=45)
plt.tight_layout()
plt.show()
```



This kernel extracts and visualizes the subject-wise performance of a specific student (Roll no.: 257) through a line plot, displaying their marks across different subjects.

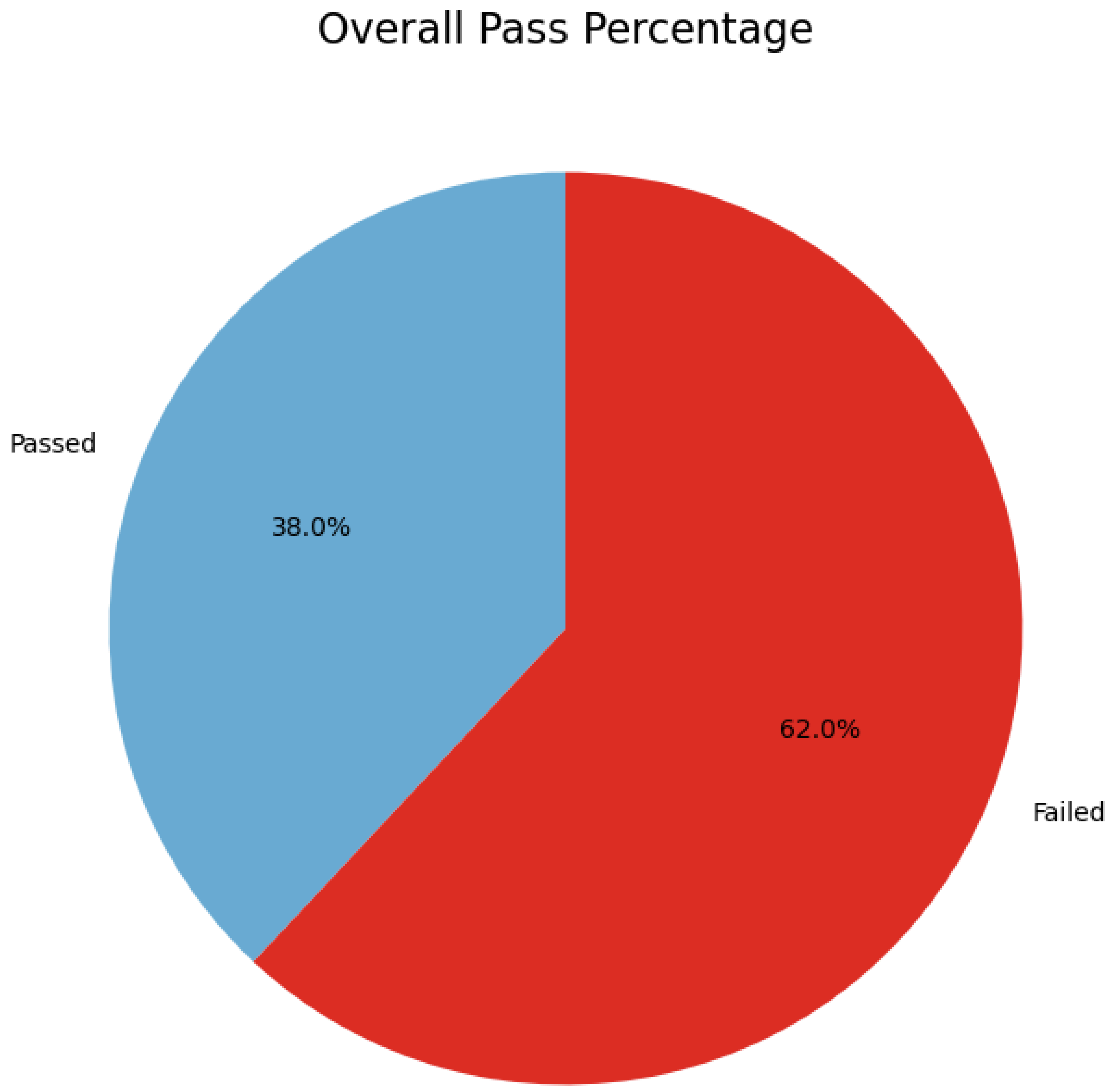
In [45]:
```
subject_means = df[marks_columns].mean()

plt.figure(figsize=(10, 6))
sns.barplot(x=subject_means.index, y=subject_means.values, palette='Blues_d')
plt.title('Subject-wise Performance Comparison')
plt.xlabel('Subject')
plt.ylabel('Average Marks')
plt.show()
```



This kernel calculates and visualizes the average marks for each subject using a bar plot to compare the subject-wise performance.

In [47]:
```
pass_mark = 10
df['Pass Count'] = (df[marks_columns] >= pass_mark).sum(axis=1)
students_passed = (df['Pass Count'] == len(marks_columns)).sum()
total_students = len(df)
pass_percentage = (students_passed / total_students) * 100

print(f"Total Students: {total_students}")
print(f"Students Passed: {students_passed}")
print(f"Pass Percentage: {pass_percentage:.2f}%")

plt.figure(figsize=(8, 8))
plt.pie(
    [students_passed, total_students - students_passed],
    labels=['Passed', 'Failed'],
    autopct='%1.1f%%',
    startangle=90,
    colors=['#6baed6', '#de2d26']
)
plt.title('Overall Pass Percentage', fontsize=16)
plt.show()
```

```
Total Students: 718
Students Passed: 273
Pass Percentage: 38.02%
```


Overall Pass Percentage
Passed
38.0%
62.0%
Failed

In [72]:
```
df.replace('A',0)
df.replace('AB',0)
df.replace('MP',0)
```

Out[72]:

	S.NO	SECTION	DV	M-II	PP	BEEE	FL	FIMS	Total Marks	Average Marks	Pass Count
0	1.0	ALPHA	12.0	0.0	17.0	9.0	19.0	15.0	72.0	12.000000	4
1	2.0	ALPHA	19.0	12.0	16.0	16.0	18.0	3.0	84.0	14.000000	5
2	3.0	ALPHA	18.0	14.0	18.0	18.0	18.0	16.0	102.0	17.000000	6
3	4.0	ALPHA	15.0	9.0	19.0	17.0	19.0	15.0	94.0	15.666667	5
4	5.0	ALPHA	18.0	17.0	19.0	19.0	20.0	18.0	111.0	18.500000	6
...	...	...	...	...	...	...	...	...	...	...	...
713	NaN	ZETA	19.0	8.0	8.0	19.0	17.0	18.0	89.0	14.833333	4
714	NaN	ZETA	12.0	1.0	7.0	10.0	20.0	8.0	58.0	9.666667	3
715	NaN	ZETA	17.0	6.0	14.0	14.0	17.0	18.0	86.0	14.333333	5
716	NaN	ZETA	12.0	1.0	6.0	7.0	15.0	12.0	53.0	8.833333	3
717	NaN	ZETA	19.0	14.0	17.0	16.0	20.0	19.0	105.0	17.500000	6

718 rows × 11 columns

In [74]:
```
data=pd.read_excel("MIDMARKS(1).xlsx")
data['PP']=pd.to_numeric(data['PP'],errors='coerce')
count=data[data['PP']<=10]['PP'].count()
print(count)
```

```
224
```

In [76]:
```
data=pd.read_excel("MIDMARKS(1).xlsx")
data['DV']=pd.to_numeric(data['DV'],errors='coerce')
count=data[data['DV']<=10]['DV'].count()
print(count)
```

```
102
```

In [78]:
```
data=pd.read_excel("MIDMARKS(1).xlsx")
data['M-II']=pd.to_numeric(data['M-II'],errors='coerce')
count=data[data['M-II']<=10]['M-II'].count()
print(count)
```

```
364
```

In [80]:
```
data=pd.read_excel("MIDMARKS(1).xlsx")
data['BEEE']=pd.to_numeric(data['BEEE'],errors='coerce')
count=data[data['BEEE']<=10]['BEEE'].count()
print(count)
```

```
198
```

In [82]:
```
data=pd.read_excel("MIDMARKS(1).xlsx")
data['FL']=pd.to_numeric(data['FL'],errors='coerce')
count=data[data['FL']<=10]['FL'].count()
print(count)
```

```
105
```

In [84]:
```
data=pd.read_excel("MIDMARKS(1).xlsx")
data['FIMS']=pd.to_numeric(data['FIMS'],errors='coerce')
count=data[data['FIMS']<=10]['FIMS'].count()
print(count)
```

```
120
```

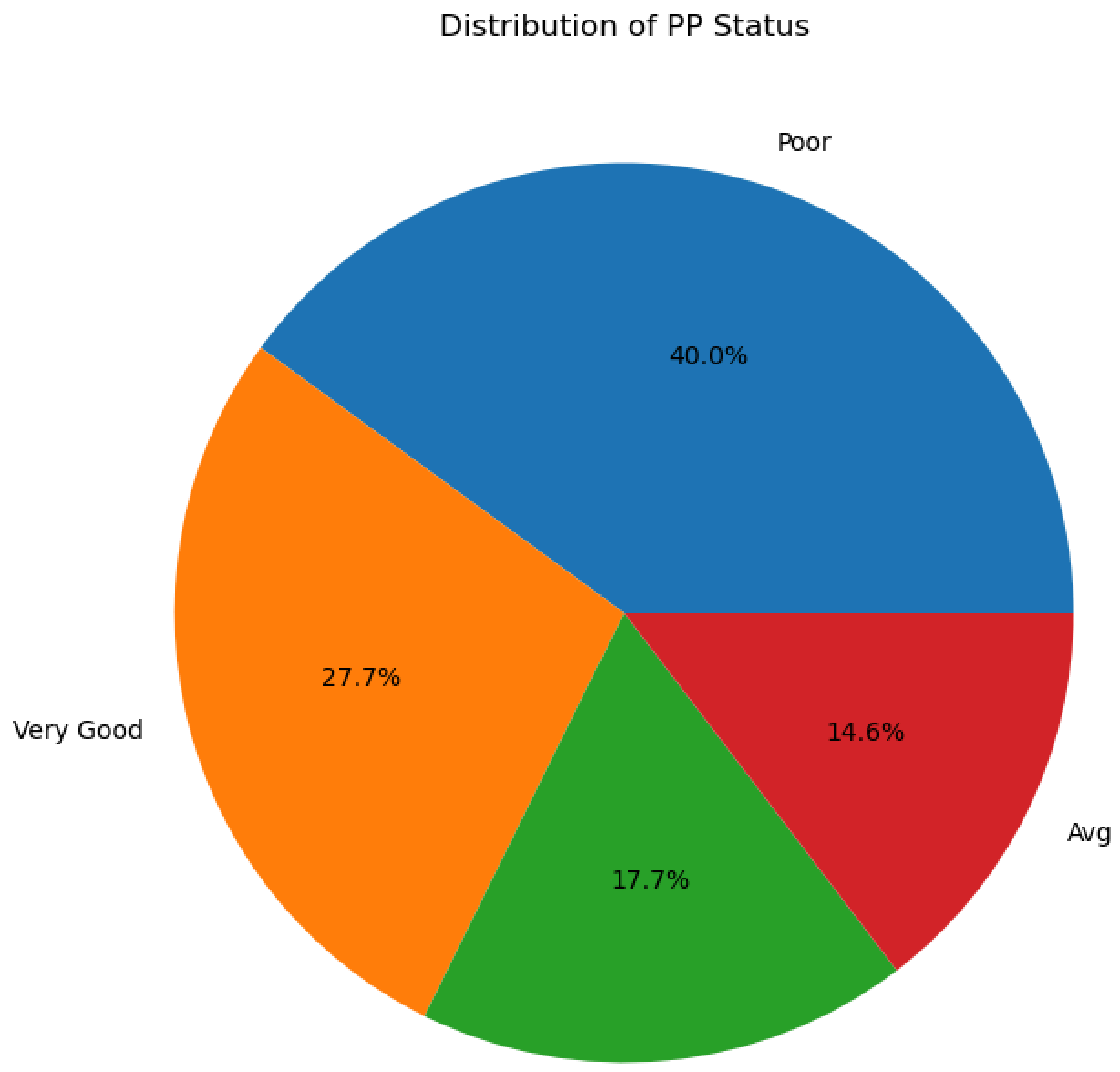
In [86]:
```
df['PP']=pd.to_numeric(df['PP'],errors='coerce')
```

In [88]:
```python
def programming_skills(PP):
    if PP >= 18:
        return 'Very Good'
    elif PP >= 15:
        return 'Good'
    elif PP >= 12:
        return 'Avg'
    else:
        return 'Poor'

df.fillna(0)
df['PP_Status'] = df['PP'].apply(programming_skills)
df.PP_Status.value_counts()
```

Out[88]:
```
Poor         287
Very Good    199
Good         127
Avg          105
Name: PP_Status, dtype: int64
```

In [92]:

```
# Create pie chart of PP_Status distribution
plt.figure(figsize=(8, 8))
plt.pie(df['PP_Status'].value_counts(),
        labels=df['PP_Status'].value_counts().index,
        autopct='%1.1f%%')
plt.title('Distribution of PP Status')
plt.show()
```


Distribution of PP Status
Poor
40.0%
Very Good
27.7%
Good
17.7%
Avg
14.6%

In [94]:
```
df['DV']=pd.to_numeric(df['DV'],errors='coerce')
```

In [96]:
```
def analytical_skills(PP):
    if PP >= 18:
        return 'Very Good'
    elif PP >= 15:
        return 'Good'
    elif PP >= 12:
        return 'Avg'
    else:
        return 'Poor'

df['Analytical_Skills'] = df['DV'].apply(programming_skills)
df.Analytical_Skills.value_counts()
```

Out[96]:
```
Very Good    232
Good         216
Poor         158
Avg          112
Name: Analytical_Skills, dtype: int64
```

In [98]:

```python
import pandas as pd

df = pd.read_excel("MIDMARKS.xlsx")

df['DV'] = pd.to_numeric(df['DV'], errors='coerce')
df['M-II'] = pd.to_numeric(df['M-II'], errors='coerce')
df['PP'] = pd.to_numeric(df['PP'], errors='coerce')
df['BEEE'] = pd.to_numeric(df['BEEE'], errors='coerce')
df['FL'] = pd.to_numeric(df['FL'], errors='coerce')
df['FIMS'] = pd.to_numeric(df['FIMS'], errors='coerce')

df_20 = df[(df['DV'] == 20) | (df['M-II'] == 20) | (df['PP'] == 20) |
           (df['BEEE'] == 20) | (df['FL'] == 20) | (df['FIMS'] == 20)]

count_20 = df_20.shape[0]

subject_counts = {
    'DV': (df['DV'] == 20).sum(),
    'M-II': (df['M-II'] == 20).sum(),
    'PP': (df['PP'] == 20).sum(),
    'BEEE': (df['BEEE'] == 20).sum(),
    'FL': (df['FL'] == 20).sum(),
    'FIMS': (df['FIMS'] == 20).sum()
}

print("Number of students who scored 20 in at least one subject:", count_20)
print(df_20)
most_common_subject = max(subject_counts, key=subject_counts.get)

print("Most students scored 20 in:", most_common_subject)
print("Number of students who scored 20 in each subject:", subject_counts)
```

```
Number of students who scored 20 in at least one subject: 253
     S.NO SECTION    DV  M-II    PP  BEEE    FL  FIMS
4       5   ALPHA  18.0  17.0  19.0  19.0  20.0  18.0
6       7   ALPHA  15.0  10.0  20.0  20.0  15.0  14.0
7       8   ALPHA  17.0  17.0  19.0  20.0  19.0  13.0
8       9   ALPHA  10.0  18.0   NaN  20.0  19.0  15.0
9      10   ALPHA  18.0  19.0  20.0  20.0  20.0  15.0
..    ...     ...   ...   ...   ...   ...   ...   ...
595   596   SIGMA  17.0  14.0  16.0  18.0  20.0  18.0
596   597   SIGMA  20.0  20.0  20.0  20.0  20.0  20.0
597   598   SIGMA  20.0  20.0  20.0  19.0  19.0  18.0
598   599   SIGMA  20.0  20.0  17.0  17.0  19.0  18.0
600   601   SIGMA  20.0  19.0  20.0  18.0  18.0  19.0

[253 rows x 8 columns]
Most students scored 20 in: FL
Number of students who scored 20 in each subject: {'DV': 88, 'M-II': 56, 'PP': 104, 'BEEE': 89, 'FL': 159, 'FIMS': 2
7}
```

In [100]:

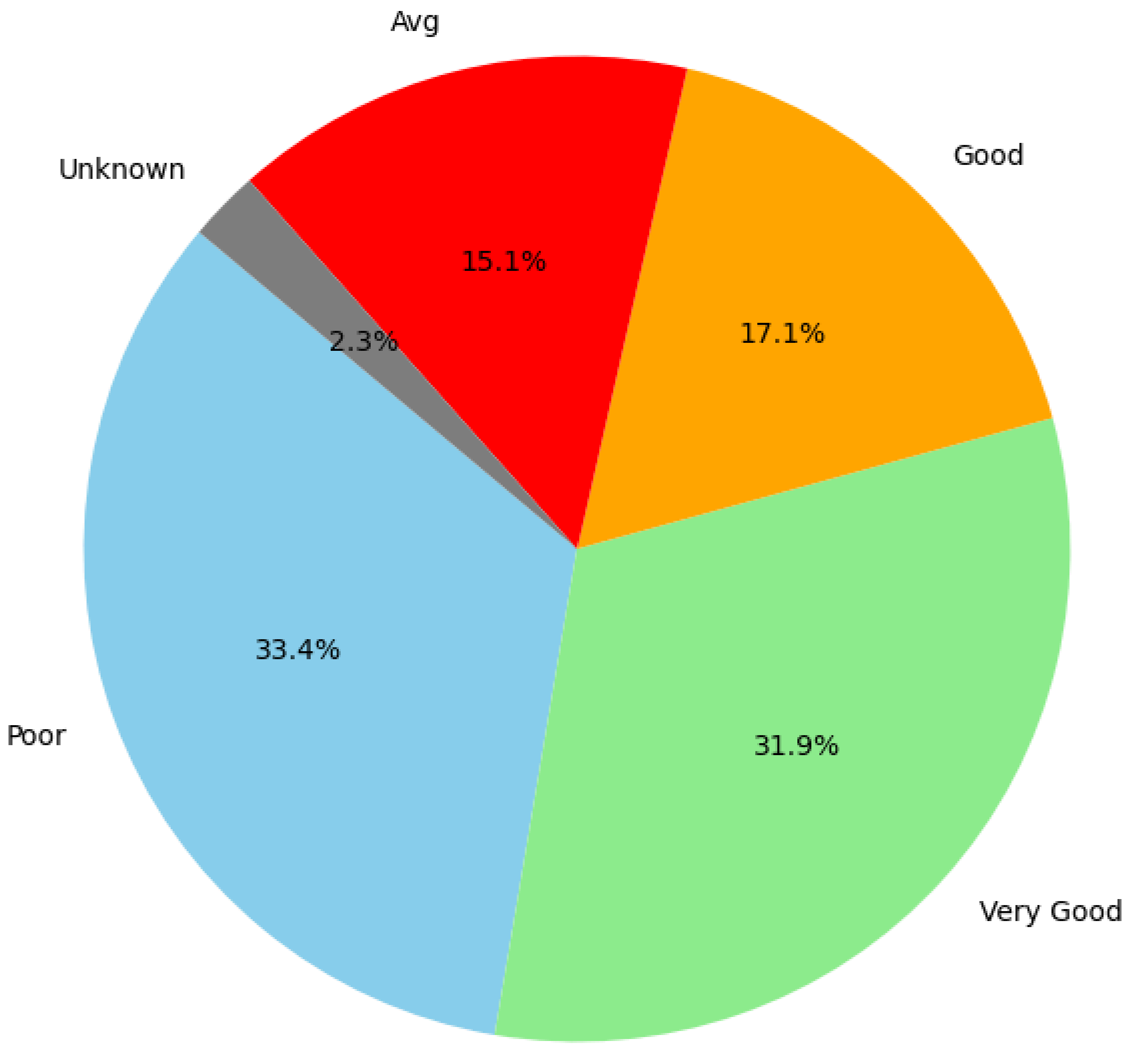
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_excel("MIDMARKS.xlsx")
df['PP'] = pd.to_numeric(df['PP'], errors='coerce')

def ppstatus(PP):
    if pd.isna(PP):
        return 'Unknown'
    elif PP >= 18:
        return 'Very Good'
    elif PP >= 15:
        return 'Good'
    elif PP >= 12:
        return 'Avg'
    else:
        return 'Poor'

df['programing_skills'] = df['PP'].apply(ppstatus)
df.rename(columns=lambda x: x.strip(), inplace=True)
```

In [104]:

```
skill_counts = df['programing_skills'].value_counts()
```

In [106]:

```
plt.figure(figsize=(8, 8))
plt.pie(skill_counts, labels=skill_counts.index, autopct='%1.1f%%', startangle=140, colors=['skyblue', 'lightgreen', '
plt.show()
```


Avg
15.1%
Good
17.1%
Unknown
2.3%
Poor
33.4%
31.9%
Very Good

In [108]:
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_excel("MIDMARKS.xlsx")
df['DV'] = pd.to_numeric(df['DV'], errors='coerce')

def dvstatus(PP):
    if pd.isna(PP):
        return 'Unknown'
    elif PP >= 18:
        return 'Very Good'
    elif PP >= 15:
        return 'Good'
    elif PP >= 12:
        return 'Avg'
    else:
        return 'Poor'

df['analytical_skills'] = df['DV'].apply(ppstatus)
df.rename(columns=lambda x: x.strip(), inplace=True)

if 'anaytical_skills' in df.columns:
    skill_counts = df['analytical_skills'].value_counts()
    plt.figure(figsize=(8, 8))
    plt.pie(skill_counts, labels=skill_counts.index, autopct='%1.1f%%', startangle=140, colors=['skyblue', 'lightgreen
    plt.show()
```

In [110]:
```
df = pd.read_excel("MIDMARKS(1).xlsx")
```

In [112]:
```
df.describe()
```

Out[112]:

	S.NO
count	601.000000
mean	301.000000
std	173.638033
min	1.000000
25%	151.000000
50%	301.000000
75%	451.000000
max	601.000000

In [114]:
```
df.drop(df.tail(2).index,inplace=True)
```

In [116]:
```
df.tail()
```

Out[116]:

	S.NO	SECTION	DV	M-II	PP	BEEE	FL	FIMS
711	NaN	ZETA	18	9	12	20	16	16
712	NaN	ZETA	15	10	7	18	18	16
713	NaN	ZETA	19	8	8	19	17	18
714	NaN	ZETA	12	1	7	10	20	8
715	NaN	ZETA	17	6	14	14	17	18

In [118]:
```
df['DV'] = pd.to_numeric(df['DV'], errors='coerce')
df['DV'].mean()
```

Out[118]: 15.017069701280228

In [120]:
```
df['PP'] = pd.to_numeric(df['PP'], errors='coerce')
df['PP'].mean()
```

Out[120]: 13.047210300429185

In [122]:
```
df['FIMS'] = pd.to_numeric(df['FIMS'], errors='coerce')
df['FIMS'].mean()
```

Out[122]: 14.489884393063583

In [124]:
```
df['FL'] = pd.to_numeric(df['FL'], errors='coerce')
df['FL'].mean()
```

Out[124]: 15.85

In [126]:
```
df['M-II'] = pd.to_numeric(df['M-II'], errors='coerce')
df['M-II'].mean()
```

Out[126]: 10.126780626780628

In [128]:
```
df['BEEE'] = pd.to_numeric(df['BEEE'], errors='coerce')
df['BEEE'].mean()
```

Out[128]: 13.656115107913669

In [130]:
```
df.head()
```

Out[130]:

	S.NO	SECTION	DV	M-II	PP	BEEE	FL	FIMS
0	1.0	ALPHA	12.0	0.0	17.0	9.0	19.0	15.0
1	2.0	ALPHA	19.0	12.0	16.0	16.0	18.0	3.0
2	3.0	ALPHA	18.0	14.0	18.0	18.0	18.0	16.0
3	4.0	ALPHA	15.0	9.0	19.0	17.0	19.0	15.0
4	5.0	ALPHA	18.0	17.0	19.0	19.0	20.0	18.0

In [132]:
```
df['SECTION']
```

Out[132]:
```
0      ALPHA
1      ALPHA
2      ALPHA
3      ALPHA
4      ALPHA
       ...
711     ZETA
712     ZETA
713     ZETA
714     ZETA
715     ZETA
Name: SECTION, Length: 716, dtype: object
```

In [134]:
```
alpha_mean=df[df.SECTION=='ALPHA'].mean()
print(alpha_mean)
```

```
S.NO    45.500000
DV      13.411111
M-II    13.711111
PP      16.112360
BEEE    15.797753
FL      16.359551
FIMS    12.842697
dtype: float64

C:\Users\subha\AppData\Local\Temp\ipykernel_18440\119993082.py:1: FutureWarning: The default value of numeric_only in
DataFrame.mean is deprecated. In a future version, it will default to False. In addition, specifying 'numeric_only=No
ne' is deprecated. Select only valid columns or specify the value of numeric_only to silence this warning.
  alpha_mean=df[df.SECTION=='ALPHA'].mean()
```

In [136]:
```
df[df.SECTION=='BETA'].mean()
```

```
C:\Users\subha\AppData\Local\Temp\ipykernel_18440\1771889873.py:1: FutureWarning: The default value of numeric_only i
n DataFrame.mean is deprecated. In a future version, it will default to False. In addition, specifying 'numeric_only=
None' is deprecated. Select only valid columns or specify the value of numeric_only to silence this warning.
  df[df.SECTION=='BETA'].mean()
```

Out[136]:
```
S.NO    135.500000
DV       13.000000
M-II     12.122222
PP       16.146067
BEEE     10.977528
FL       16.157303
FIMS     14.044444
dtype: float64
```

In [138]:
```
df[df.SECTION=='GAMMA'].mean()
```

```
C:\Users\subha\AppData\Local\Temp\ipykernel_18440\1603944844.py:1: FutureWarning: The default value of numeric_only i
n DataFrame.mean is deprecated. In a future version, it will default to False. In addition, specifying 'numeric_only=
None' is deprecated. Select only valid columns or specify the value of numeric_only to silence this warning.
  df[df.SECTION=='GAMMA'].mean()
```

Out[138]:
```
S.NO    403.500000
DV       15.321839
M-II      9.609195
PP       11.149425
BEEE     15.298851
FL       15.988636
FIMS     13.011628
dtype: float64
```

In [140]:
```
df[df.SECTION=='DELTA'].mean()
```

```
C:\Users\subha\AppData\Local\Temp\ipykernel_18440\734469687.py:1: FutureWarning: The default value of numeric_only in
DataFrame.mean is deprecated. In a future version, it will default to False. In addition, specifying 'numeric_only=No
ne' is deprecated. Select only valid columns or specify the value of numeric_only to silence this warning.
  df[df.SECTION=='DELTA'].mean()
```

Out[140]:
```
S.NO    225.500000
DV       14.181818
M-II      9.715909
PP       12.561798
BEEE      9.670455
FL       14.719101
FIMS     16.954545
dtype: float64
```

In [142]:
```
df[df.SECTION=='SIGMA'].mean()
```

```
C:\Users\subha\AppData\Local\Temp\ipykernel_18440\2631436304.py:1: FutureWarning: The default value of numeric_only i
n DataFrame.mean is deprecated. In a future version, it will default to False. In addition, specifying 'numeric_only=
None' is deprecated. Select only valid columns or specify the value of numeric_only to silence this warning.
  df[df.SECTION=='SIGMA'].mean()
```

Out[142]:
```
S.NO    570.000000
DV       16.683333
M-II     13.066667
PP       15.271186
BEEE     13.700000
FL       16.866667
FIMS     16.517241
dtype: float64
```

In [144]:
```
df[df.SECTION=='ZETA'].mean()
```

```
C:\Users\subha\AppData\Local\Temp\ipykernel_18440\1235761348.py:1: FutureWarning: The default value of numeric_only i
n DataFrame.mean is deprecated. In a future version, it will default to False. In addition, specifying 'numeric_only=
None' is deprecated. Select only valid columns or specify the value of numeric_only to silence this warning.
  df[df.SECTION=='ZETA'].mean()
```

Out[144]:
```
S.NO          NaN
DV      15.425287
M-II     7.348837
PP       9.372093
BEEE    14.837209
FL      15.045977
FIMS    14.070588
dtype: float64
```

In [146]:
```
df[df.SECTION=='EPSILON'].mean()
```

```
C:\Users\subha\AppData\Local\Temp\ipykernel_18440\3856739591.py:1: FutureWarning: The default value of numeric_only i
n DataFrame.mean is deprecated. In a future version, it will default to False. In addition, specifying 'numeric_only=
None' is deprecated. Select only valid columns or specify the value of numeric_only to silence this warning.
  df[df.SECTION=='EPSILON'].mean()
```

Out[146]:
```
S.NO    314.500000
DV       15.390805
M-II      6.770115
PP        8.666667
BEEE     14.597701
FL       15.494253
FIMS     12.206897
dtype: float64
```

In [148]:
```
dv_mean=df['DV'].mean()
print(dv_mean)
```

```
15.017069701280228
```

In [150]:
```
std=alpha_mean.std()
print(std)
```

```
11.725031271935967
```

In [152]:
```
alpha_sample_size=len(df[df['SECTION']=='ALPHA'])
print(alpha_sample_size)
```

```
90
```

In [154]:
```
# T-test for alpha section for DV subject
t=(alpha_mean-dv_mean)/(alpha_sample_size/std**0.5)
print(t)
```

```
S.NO    1.159768
DV     -0.061101
M-II   -0.049687
PP      0.041672
BEEE    0.029702
FL      0.051077
FIMS   -0.082727
dtype: float64
```

In [156]:
```
pm=df[["DV","M-II","PP","BEEE","FL","FIMS"]].mean()
pm
```

Out[156]:
```
DV      15.017070
M-II    10.126781
PP      13.047210
BEEE    13.656115
FL      15.850000
FIMS    14.489884
dtype: float64
```

In [158]:
```
c=df[df["SECTION"]=="ALPHA"]
sm=c[["DV","M-II","PP","BEEE","FL","FIMS"]].mean()
print(sm)
sd=c[["DV","M-II","PP","BEEE","FL","FIMS"]].std()

print(sd)
```

```
DV      13.411111
M-II    13.711111
PP      16.112360
BEEE    15.797753
FL      16.359551
FIMS    12.842697
dtype: float64
DV      4.991891
M-II    5.595432
PP      5.095538
BEEE    4.530653
FL      3.415364
FIMS    4.314086
dtype: float64
```

In [160]:
```
# One sample t-test
import math
(sm-pm)/(sd/math.sqrt(90))
```

Out[160]:
```
DV     -3.052042
M-II    6.077090
PP      5.706671
BEEE    4.484422
FL      1.415375
FIMS   -3.622226
dtype: float64
```