A1

1. Code:
   Student_List.StudentID.count() # Count number of occurrences of a StudentID in Student_List

   Output: 1500

2. Code:
   Min = Student_List['Age'].min() # Returns item with lowest value
   Max = Student_List['Age'].max() # Returns item with highest value
   Age_range = Max - Min
   print(Min, "-", Max)
   print("Age range:", Age_range)

   Output:
   15 - 18
   Age range: 3

3. Code:
   Student_List.dtypes # Returns the data type of each column.

   Output:

StudentID	int64
Age	int64
ParentalEducation	object
StudyTimeWeekly	float64
Absences	int64
Tutoring	object
ParentalSupport	int64
Extracurricular	object
Sports	object
Music	object
Volunteering	object
GPA	float64
GradeClass	object

   dtype: object

   Explanation
   Data type: int64
   Int represents numeric characters.
   64 represents the 64 bits memory allocated to store data.

Data type: object
General dtype. This dtype is assigned to the column if the column has mixed types like numbers and strings.

Data type: float64
Numeric characters with decimals.
64 represents the 64 bits memory allocated to store data.

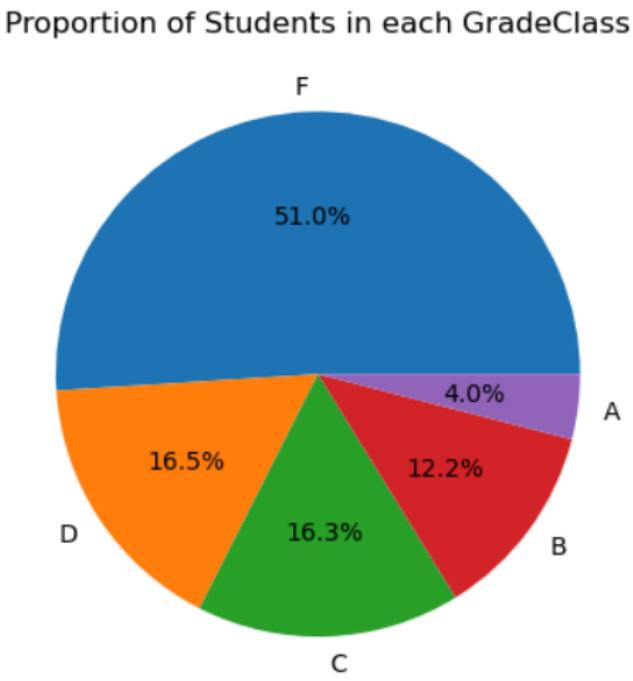
4. Code:
```
Grade_Class = Student_List.groupby('GradeClass')['StudentID'].size()
Student_Total = Student_List.StudentID.count()
Percentage = (Grade_Class/Student_Total) * 100
Percentage.apply(lambda x: round(x, 2))
```

Output:
```
GradeClass
A       4.00
B       12.20
C       16.27
D       16.53
F       51.00
Name: StudentID, dtype: float64
```

Explanation
```
Grade_Class = Student_List.groupby('GradeClass')['StudentID'].size() # split a Student_List into groups
Student_Total = Student_List.StudentID.count() # Count number of occurrences of a StudentID in Student_List
Percentage = (Grade_Class/Student_Total) * 100 # Calculates percentage of students in each grade category
Percentage.apply(lambda x: round(x, 2)) # Round values to two decimals
```

5. Code:
```
Grade_Count = Student_List['GradeClass'].value_counts()
plt.pie(Grade_Count, labels = Grade_Count.index, autopct='%1.1f%%')
plt.title('Proportion of Students in each GradeClass')
plt.show()
```

Output:



Explanation

Grade_Count = Student_List['GradeClass'].value_counts() # Count the number of occurrences of each Gradeclass in the Student_List

plt.pie(Grade_Count, labels = Grade_Count.index, autopct='%1.1f%%') # Create pie chart of the Grade_Count with labels and percentage display

Observation

- More than half of the students scored F in this dataset, 765 out of 1500 students (Shown by 51% in the pie chart).
- Only 4% of students scored A, with a number of 60 students in this dataset.
- Around the same number of students scored C or D in this dataset of 1500 students. Where 244 students scored C and 248 students scored D.

The overall trend of the pie chart shows a large number of students failed whereas only a small percentage of students scored the highest grade.

A2

1. a)

   Code:

   Student_List['ParentalEducation'].value_counts().get('Higher') # Get the count of students by 'Higher' in ParentalEducation

   Output:

   77

1. b)
   Code:
   Student_List['ParentalEducation'].value_counts().get('No Education') # Get the count of students by ‘No Education’ in ParentalEducation

   Output:
   154

1. c)
   Code:
   Student_List['ParentalEducation'].mode() # Find the most common parental education level

   Output:
   0 Some College
   Name: ParentalEducation, dtype: object

2. Code:
   ```
   Student_List['ParentalEducation']
   map = {
           'No Education': 0,
           'High School': 1,
           'Some College': 2,
           "Bachelor's": 3,
           'Higher': 4
   }
   Student_List['ParentalEducation'] = Student_List['ParentalEducation'].map(map)
   Student_List
   ```

   Output:

	ParentalEducation
0	2
1	3
2	2
3	1
4	2
...	...
1495	0
1496	2
1497	2
1498	2

1499 4

Explanation

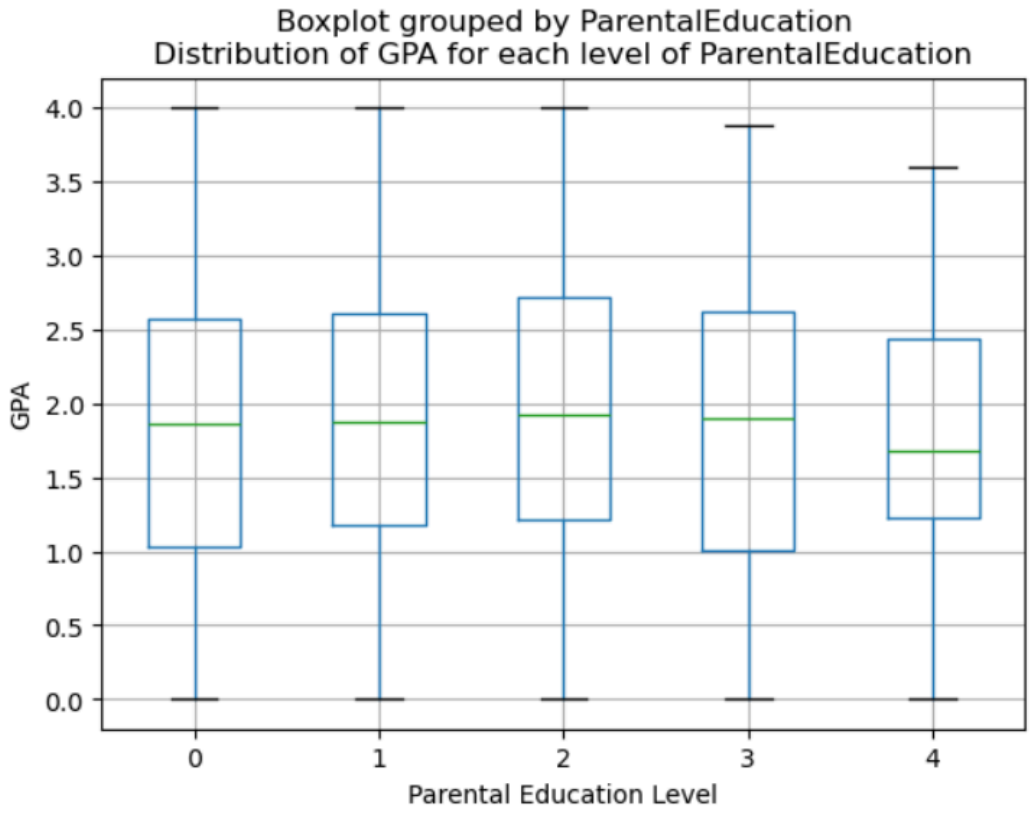
```
Student_List['ParentalEducation']
map = {
   'No Education': 0,
   'High School': 1,
   'Some College': 2,
   "Bachelor's": 3,
   'Higher': 4
}
Student_List['ParentalEducation'] = Student_List['ParentalEducation'].map(map)
# Replace the values according to map in the Student_List

Student_List[['ParentalEducation']] # Display Student_List with only ParentalEducation column
```

3. Code:

```
Student_List.boxplot(column = 'GPA', by = 'ParentalEducation')
plt.xlabel('Parental Education Level')
plt.ylabel('GPA')
plt.title('Distribution of GPA for each level of ParentalEducation')
```

Output:

```
Text(0.5, 1.0, 'Distribution of GPA for each level of ParentalEducation')
```



Observation

- The median GPA is similar across every Parental Education Level, around 1.5 to 2.0 GPA. This shows that the median of GPA is consistent regardless of the Parental Education Level.
- The lower quartile and higher quartile range shows that the GPA is fairly evenly spread across every Parental Education Level. This suggests that the GPA does not change exponentially with a higher Parental Education Level.
- The whiskers show that the range of GPA is consistent across every Parental Education Level, with the minimum at 0.0 and maximum around 4.0.

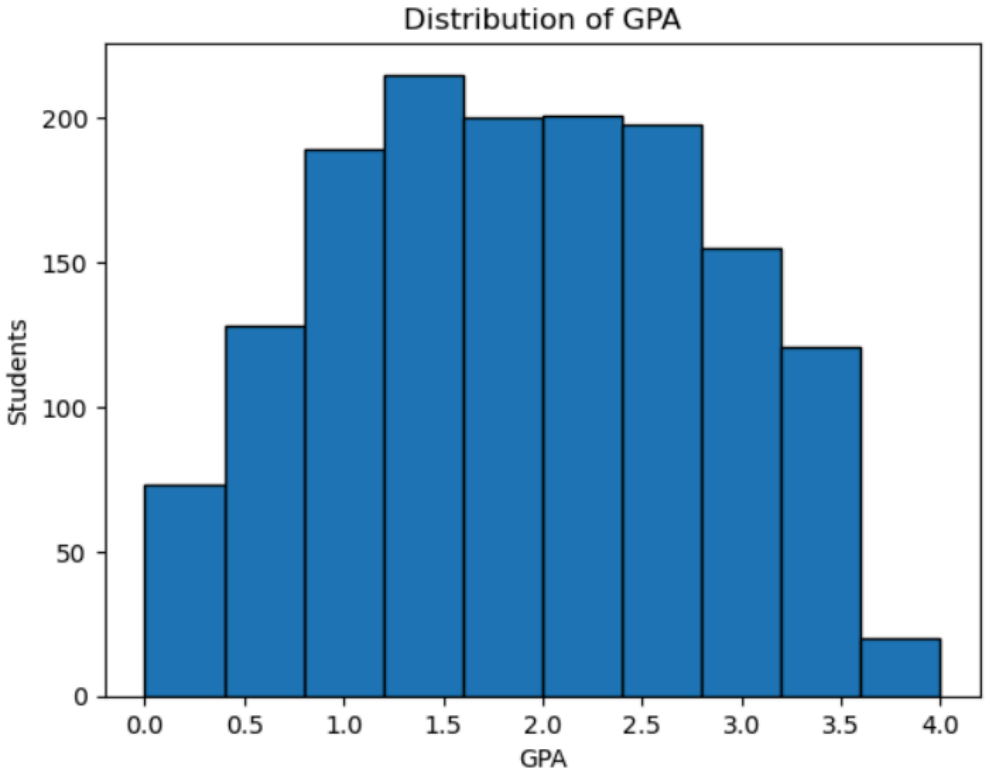
The relationship between Parental Education Level and GPA shown in the box plot indicates that there is no strong correlation between the two factors as the distribution of GPA is similar amongst every Parental Education Level.

A3

1. Code:

```
plt.hist(Student_List['GPA'], edgecolor = 'black')
plt.xlabel('GPA')
plt.ylabel('Students')
plt.title('Distribution of GPA')
plt.show()
```

Output:



Observation

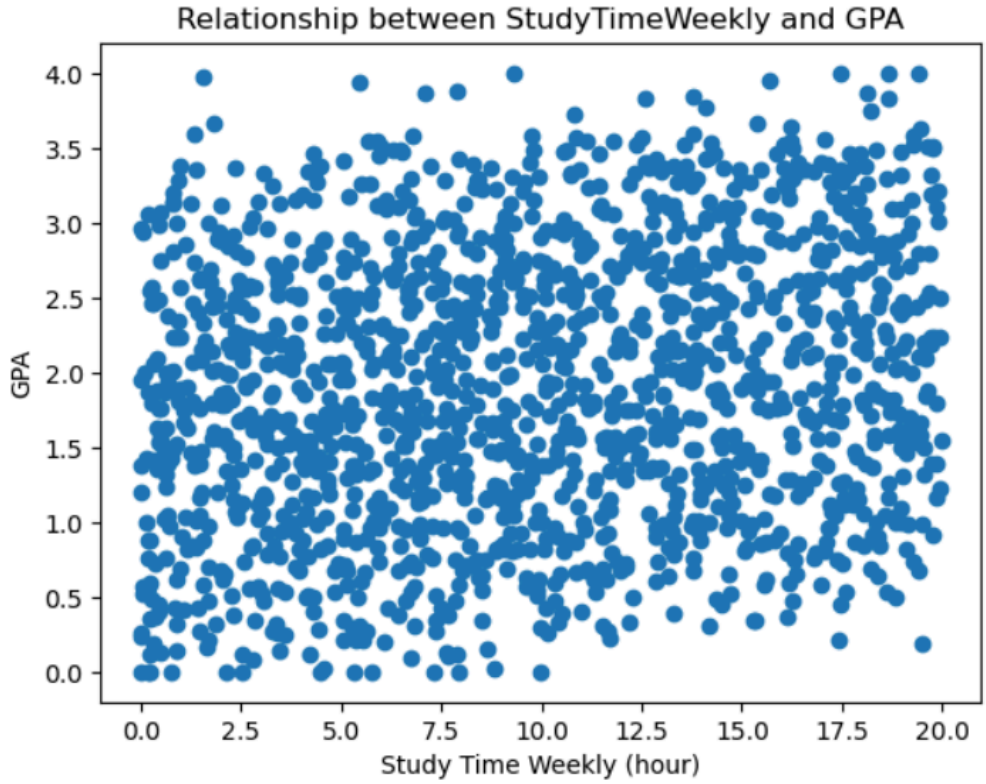
- The shape of the distribution is bell-shaped and slightly positively skewed.
- The most common GPA range is around 1.2 and 1.6.
- The spread of GPA shows that there are fewer students with low GPAs or high GPAs.
- The distribution tail suggests that there are fewer students that have a higher GPA compared to the lower GPA.

The distribution shows that most students have an average GPA between 1.5 and 2.5 with a few instances of students with lower or higher GPA.

2. Code:

```
plt.scatter(Student_List['StudyTimeWeekly'], Student_List['GPA'])
plt.title('Relationship between StudyTimeWeekly and GPA')
plt.xlabel('Study Time Weekly (hour)')
plt.ylabel('GPA')
plt.show()
```

Output:



Code:

```
correlation = Student_List['StudyTimeWeekly'].corr(Student_List['GPA'])
print(f"Correlation coefficient: {correlation}")
```

Output:
Correlation coefficient: 0.1904931303711251

Explanation

correlation = Student_List['StudyTimeWeekly'].corr(Student_List['GPA']) # .corr() is used to calculate the Pearson correlation coefficient
# Measures how StudyTimeWeekly is related to GPA

Observation

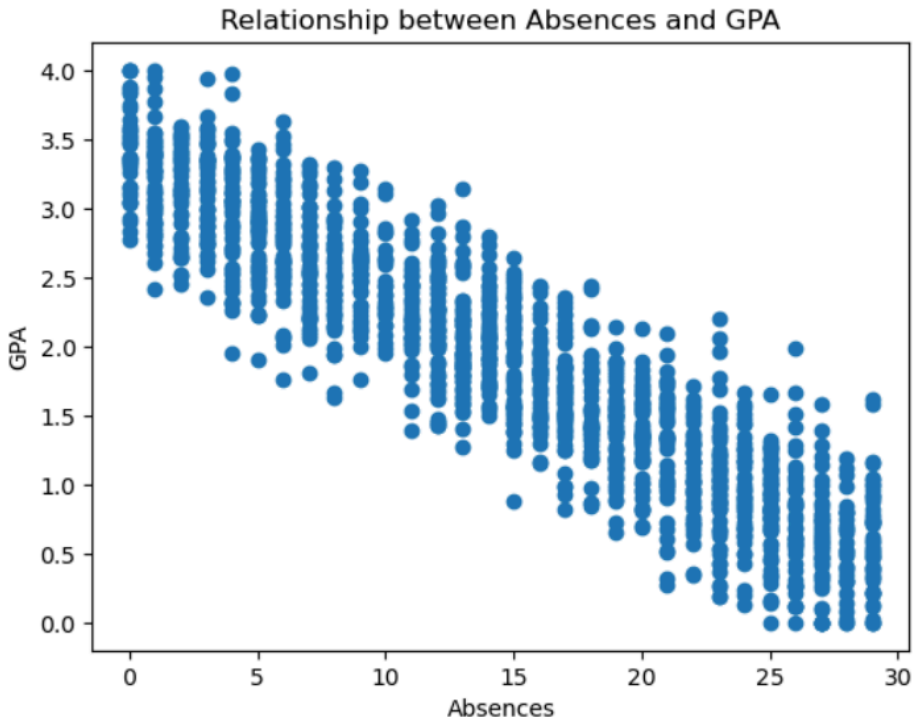
- The scatter plot suggests that the relationship between StudyTimeWeekly and GPA is very dispersed as data is scattered with no clear linear relationship.
- Correlation coefficient of 0.1904931303711251 suggests a very weak positive correlation.

Both the scatter plot and correlation coefficient shows that there is a very weak relationship between StudyTimeWeekly and GPA. This indicates that students that study for more hours weekly do not strongly correlate to their GPA.

3. Code:

```
plt.scatter(Student_List['Absences'], Student_List['GPA'])
plt.title('Relationship between Absences and GPA')
plt.xlabel('Absences')
plt.ylabel('GPA')
plt.show()
```

Output:



Code:
```
correlation = Student_List['Absences'].corr(Student_List['GPA'])
print(f"Correlation coefficient: {correlation}")
```

Output:
Correlation coefficient: -0.9194876943290947

Observation

- The scatter plot shows a clear negative linear relationship between Absences and GPA
- The correlation coefficient of -0.9194876943290947 indicates a strong negative correlation.

Both the scatter plot and correlation coefficient suggests a strong negative relationship between absences and GPA. This shows that students with a higher number of absences have a strong correlation to a lower GPA.

A4

1. Code:
```
Group_A = Student_List[
   (Student_List['Sports'] == 'Yes') &
   (Student_List['Music'] == 'Yes') &
   (Student_List['Volunteering'] == 'Yes') &
   (Student_List['Extracurricular'] == 'Yes')
]
Group_A_Num = Group_A.shape[0]
print(f"Group A has {Group_A_Num} of students")
```

Output:
Group A has 5 of students

Explanation
```
Group_A = Student_List[
   (Student_List['Sports'] == 'Yes') &
   (Student_List['Music'] == 'Yes') &
   (Student_List['Volunteering'] == 'Yes') &
   (Student_List['Extracurricular'] == 'Yes')
] # Filter students who participates in all activities
Group_A_Num = Group_A.shape[0] # Get number of rows for students in group A
```

2. Code:

```
Group_B = Student_List[
    (Student_List['Sports'] == 'No') &
    (Student_List['Music'] == 'No') &
    (Student_List['Volunteering'] == 'No') &
    (Student_List['Extracurricular'] == 'No')
]
Group_B_Num = Group_B.shape[0]
print(f"Group B has {Group_B_Num} of students")
```

Output:
Group B has 432 of students

3. Code:
```
mean_gpa_A = Group_A['GPA'].mean()
print(f"Mean GPA of Group A: {mean_gpa_A}")
mean_gpa_B = Group_B['GPA'].mean()
print(f"Mean GPA of Group B: {mean_gpa_B}")
```

Output:
Mean GPA of Group A: 2.4475262217999996
Mean GPA of Group B: 1.7312226005532407

Observation
Comparing the mean GPA of group A and group B, it shows that Students that participate in Sports, Music, Volunteering and Extracurricular Activities tend to have a higher GPA than students who don't participate in any activities.

A5

1. Code:
```
Aggregate_Data = Student_List.groupby('ParentalSupport').agg(
    Mean_GPA = ('GPA', 'mean'),
    Median_GPA = ('GPA', 'median'),
    Students_Age18 = ('Age', lambda x: (x == 18).sum())
)
Aggregate_Data
```

Output:

ParentalSupport	Mean_GPA	Median_GPA	Students_Age18

0	1.521602	1.471672	33
1	1.735855	1.740455	80
2	1.845914	1.817007	98
3	2.068174	2.070669	116
4	2.227639	2.215516	35

Explanation

```
Aggregate_Data = Student_List.groupby('ParentalSupport').agg(
    Mean_GPA = ('GPA', 'mean'), # Calculate the mean GPA for each level of ParentalSupport
    Median_GPA = ('GPA', 'median'), # Calculate the median GPA for each level of ParentalSupport
    Students_Age18 = ('Age', lambda x: (x == 18).sum()) # Count the number of students who are aged 18 in each level of ParentalSupport
)
```

('Age', lambda x: (x == 18).sum()):

1. 'Age' is the column to aggregate
2. x == 18: Creates a boolean the indicates True if 'Age' is 18, False otherwise
3. .sum(): Adds all the True value to calculate how many students are 18 years old

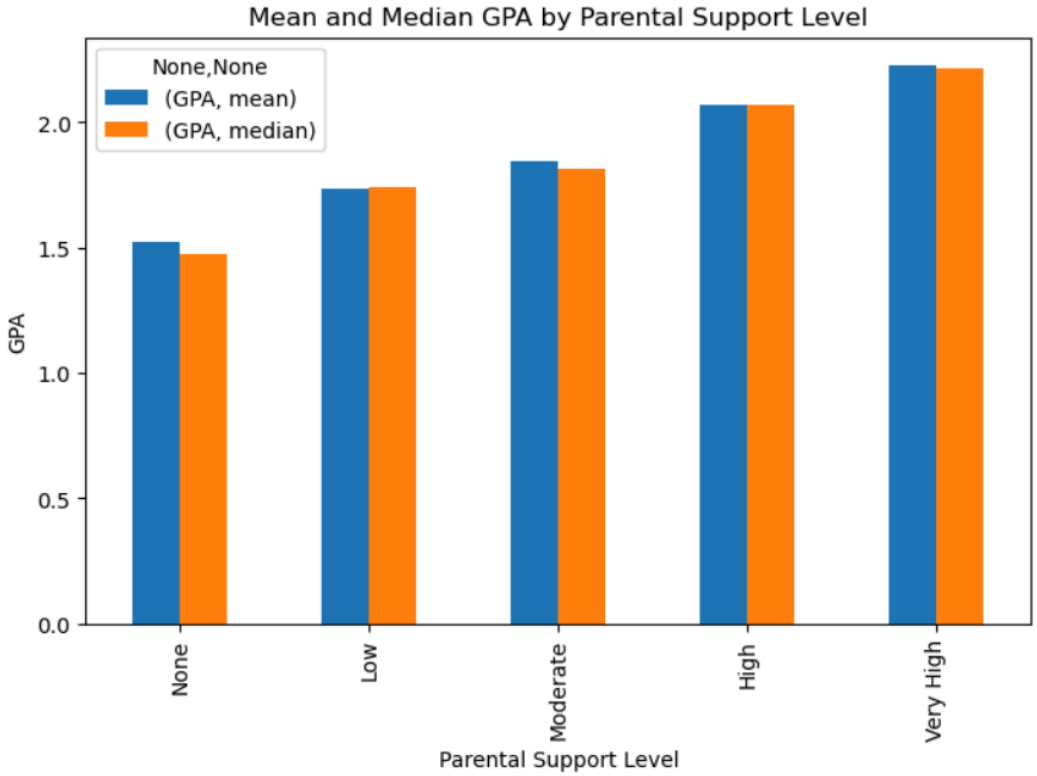
2. Code:

```
Data = {
    'GPA': ['mean', 'median']
}
Aggregate_Data2 = Student_List.groupby('ParentalSupport').agg(Data)
Aggregate_Data2 = Aggregate_Data2.reset_index()

new_label = {
    0: 'None',
    1: 'Low',
    2: 'Moderate',
    3: 'High',
    4: 'Very High'
}
```

```
Aggregate_Data2['ParentalSupport'] =
Aggregate_Data2['ParentalSupport'].map(new_label)
Aggregate_Data2.set_index('ParentalSupport', inplace=True)

Aggregate_Data2.plot.bar(figsize=(8,5))
plt.xlabel('Parental Support Level')
plt.ylabel('GPA')
plt.title('Mean and Median GPA by Parental Support Level')
```

Output:

```
Text(0.5, 1.0, 'Mean and Median GPA by Parental Support Level')
```



Explanation

1. Data = {
   'GPA': ['mean', 'median']
   } # Aggregation dictionary to calculate mean and median GPA

2. Aggregate_Data2 = Student_List.groupby('ParentalSupport').agg(Data) # Group Student_List by 'ParentalSupport'
   # Aggregate the data

3. Aggregate_Data2 = Aggregate_Data2.reset_index() # Reset index for 'ParentalSupport to turn into a column

4. new_label = {
   0: 'None',
   1: 'Low',
   2: 'Moderate',
   3: 'High',
   4: 'Very High'
   } # Dictionary to map labels

5. Aggregate_Data2['ParentalSupport'] = Aggregate_Data2['ParentalSupport'].map(new_label) # Map the labels to 'ParentalSupport'

6. Aggregate_Data2.set_index('ParentalSupport', inplace=True)
- set_index(): # Set 'ParentalSupport' as index of Aggregate_Data2
- Inplace = True: # Modify Aggregate_Data2 without creating new DataFrame

7. Aggregate_Data2.plot.bar(figsize=(8,5)) # Plot bar chart 8 inches width, 5 inches height

Observation

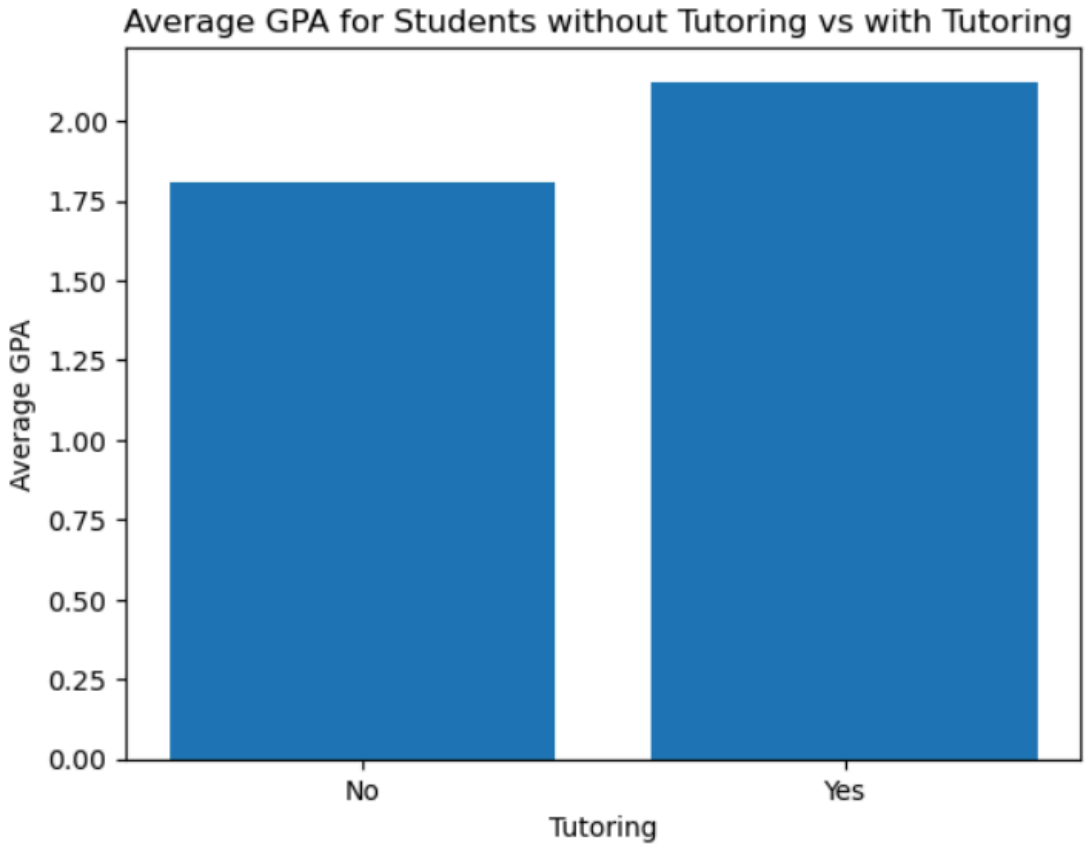
- The trend of the bar graph shows that the mean and median GPA increases as parental support increases. This suggests a positive correlation between parental support and GPA.
- The mean and median GPAs are almost evenly distributed across all parental support levels.
- Students with 'None' parental support has a mean and median around 1.5 GPA.
- Students with 'Very High' parental support have a mean and median around 2.2 GPA.

The overall bar graph shows a positive relationship between parent support and GPA. This suggests that parental support impacts a student's performance academically.

3. Code:

```
Average_GPA = Student_List.groupby('Tutoring')['GPA'].mean().reset_index()
plt.bar(Average_GPA['Tutoring'], Average_GPA['GPA'])
plt.xlabel('Tutoring')
plt.ylabel('Average GPA')
plt.title('Average GPA for Students without Tutoring vs with Tutoring ')
plt.show()
```

Output:



Explanation

Average_GPA = Student_List.groupby('Tutoring')['GPA'].mean().reset_index() # Calculate average GPA of students with and without 'Tutoring'

plt.bar(Average_GPA['Tutoring'], Average_GPA['GPA']) # Plot a bar graph using columns 'Tutoring' of the x-axis and Average GPA for the y-axis.

Observation

- Students that did not receive tutoring have an average GPA of 1.8
- Students that received tutoring have an average GPA of 2.1
- The bar graph shows that students who receive tutoring have a higher average GPA than students who did not receive tutoring.

The bar graph suggests that tutoring has an impact on students as students who received tutoring have a higher GPA compared to those without tutoring.