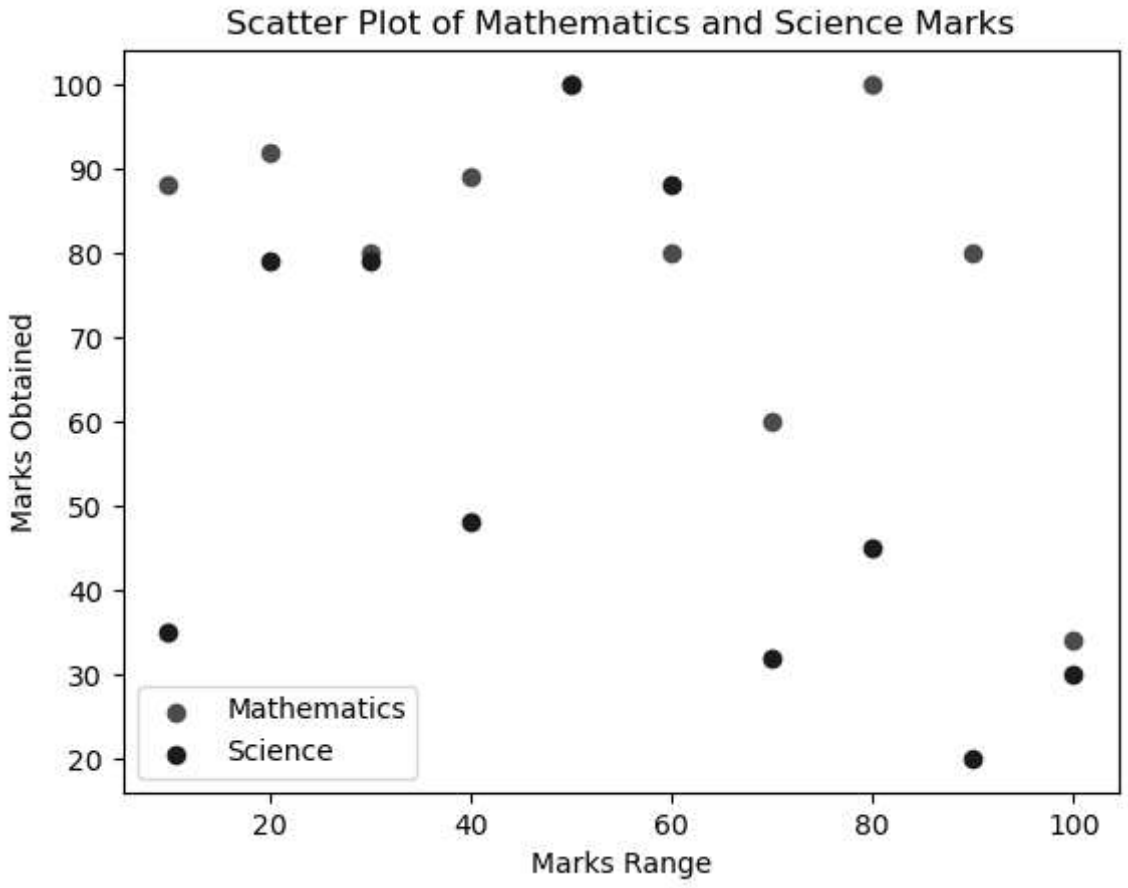
Q. 743 Write a Python program to draw a scatter plot comparing two subject marks of Mathematics and Science. Use marks of 10

students. Test Data: math_marks = [88, 92, 80, 89, 100, 80, 60, 100, 80, 34] science_marks = [35, 79, 79, 48, 100, 88, 32, 45, 20, 30] marks_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100] Add appropriate labels, title and legend.

In [1]:
```
import matplotlib.pyplot as plt

math_marks = [88, 92, 80, 89, 100, 80, 60, 100, 80, 34]
science_marks = [35, 79, 79, 48, 100, 88, 32, 45, 20, 30]
marks_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

plt.scatter(marks_range, math_marks, color='red', label='Mathematics')
plt.scatter(marks_range, science_marks, color='blue', label='Science')

plt.title('Scatter Plot of Mathematics and Science Marks')
plt.xlabel('Marks Range')
plt.ylabel('Marks Obtained')
plt.legend()

plt.show()
```

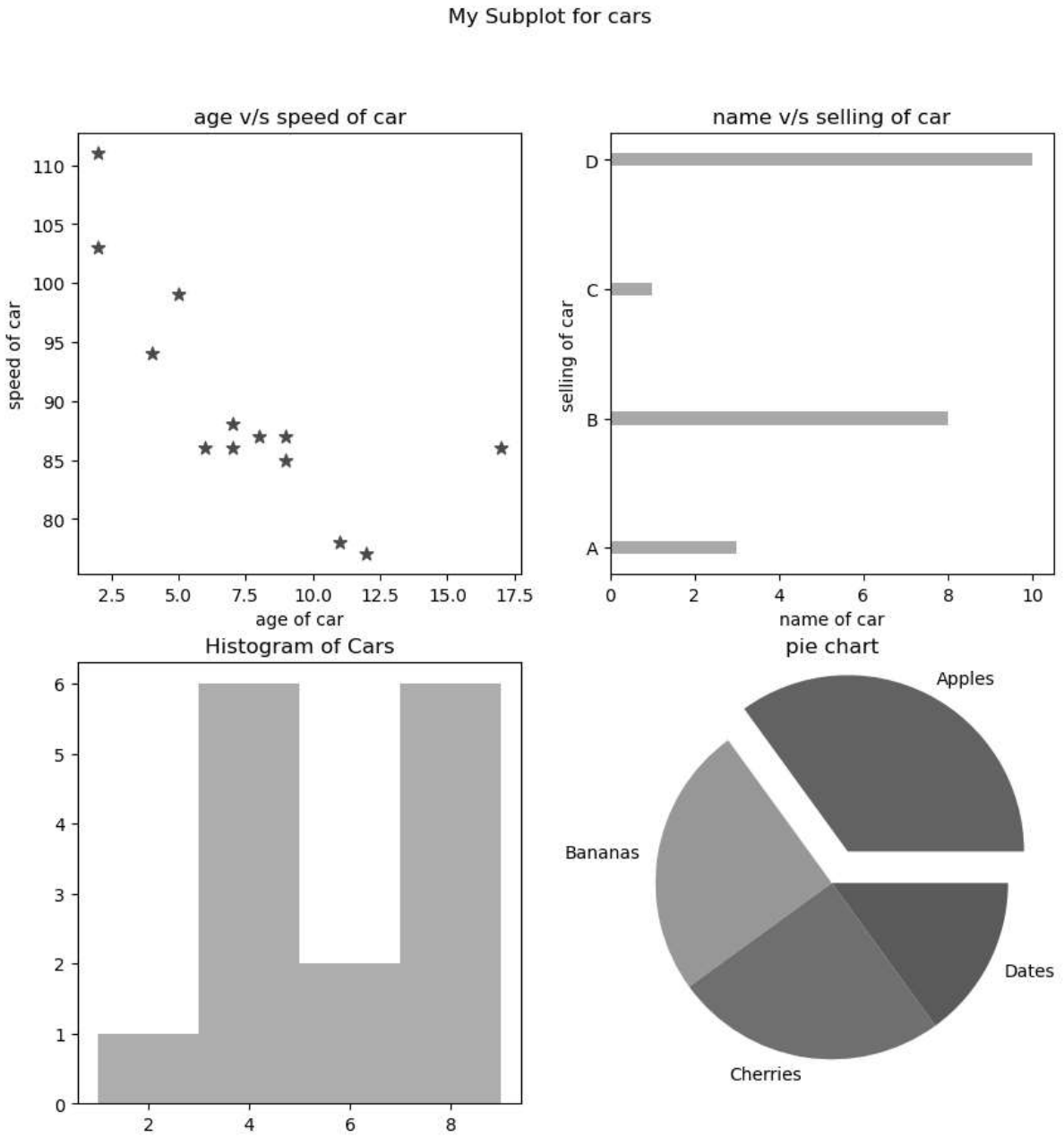


Q. 744 Draw multiple plots in one figure using subplot function. The multiple plots include below according to order:

1. Plot a scatter plot with following data: x = [5,7,8,7,2,17,2,9,4,11,12,9,6] y = [99,86,87,88,111,86,103,87,94,78,77,85,86] The x axis represents the age of car while y axis represents the speed of car. The title of the graph should be age v/s speed of car. Also in graph there should be x and y labels. The marker used should be star. The marker color should be green. The marker size should be 60. (Entire Scatter plot 2 marks)
2. Plot a horizontal bar with following data: x=["A", "B", "C", "D"] y=[3, 8, 1, 10] The x axis represents the name of car while y axis represents the selling of car. The title of the graph should be name v/s selling of car. Also in graph there should be x and y label. The horizontal bar chart's height should be 0.1. The color of bar should be yellow. (Entire bar plot 2 marks)
3. Plot a histogram with following data: data=[1,3,3,3,3,9,9,5,4,4,8,8,8,6,7] bins=4, the title of the graph should be histogram of cars. The orientation should be vertical. The color of plot should be violet (Entire histogram plot 2 marks)
4. Plot a pie with following data: y=[35,25,25,15] mylabels= ['Apple','Bananas','Cherries','Dates'] The title of the graph should be pie chart. The exploded view should be shown with 0.2 value for 'Apple' (Entire pie chart of 2 marks) Also need to provide a superior title to the subplot prepared i.e 'My Subplot for cars'(0.5 marks) and subplot preparation (0.5 mark) For clear visualization can use the following syntax after importing matplotlib plt.figure(figsize=(10,10))

In [2]:
```
import matplotlib.pyplot as plt

# plotting element that you can use anywhere
plt.figure(figsize=(10,10))
plt.suptitle('My Subplot for cars')

# for first plot
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.subplot(2,2,1)
plt.scatter(x,y,marker='*',c='g',s=60)
plt.title('age v/s speed of car')
plt.xlabel('age of car')
plt.ylabel('speed of car')

# for second plot
x = ["A","B","C","D"]
y = [3,8,1,10]

plt.subplot(2,2,2)
plt.barh(x,y,height=0.1,color='y')
plt.title('name v/s selling of car')
plt.xlabel('name of car')
plt.ylabel('selling of car')

# for third plot
```

```
data = [1,3,3,3,3,9,9,5,4,4,8,8,8,6,7]

plt.subplot(2,2,3)
plt.hist(data,bins=4,orientation='vertical',color='violet')
plt.title('Histogram of Cars')

# for fourth plot
y = [35,25,25,15]

plt.subplot(2,2,4)
mylabels = ['Apples','Bananas','Cherries','Dates']
myexplode = [0.2,0,0,0]
plt.pie(y,labels=mylabels,explode=myexplode)
plt.title('pie chart')



plt.show()
```

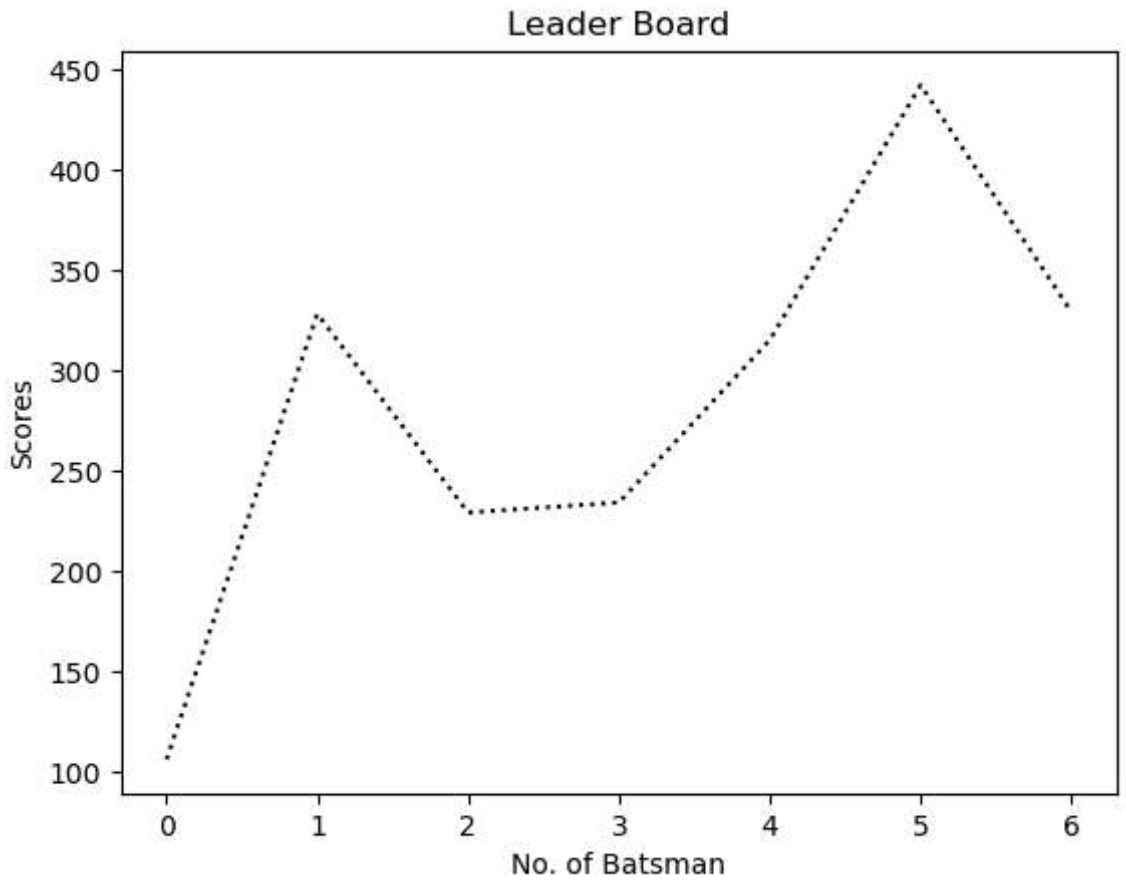


In [8]:
```
import numpy as np
import matplotlib.pyplot as plt
# Original array
Scores = np.array([[13, 10, 9, 33],
```

```
                    [63, 46, 90, 42],
                    [39, 76, 13, 29],
                    [82, 9, 29, 78],
                    [67, 61, 59, 36]])

# Scores of the 5th Match
Match_6 = np.array([41, 87, 72, 36, 92])

# Adding scores of the 5th Match to the original array without column_stack
Scores_with_5th_match = np.concatenate((Scores, Match_6.reshape(-1, 1)), axis=1)

# Print the result
print("Scores with 5th Match:")
print(Scores_with_5th_match)

# Print the result
print("Scores with 5th Match:")
print(Scores_with_5th_match)
Batsman_6= np.array([77, 83, 98, 95, 89])
Batsman_7= np.array([92, 71, 52, 61, 53])

Scores_with_6thBats=np.concatenate((Scores_with_5th_match,Batsman_6.reshape(1,-1)))
Scores_with_7thBats=np.concatenate((Scores_with_6thBats,Batsman_7.reshape(1,-1)))
print(Scores_with_7thBats)

import numpy as np

import numpy as np

# Given array


# Calculate the sum of the first five columns for each row
row_sums = np.sum(Scores_with_7thBats[: :], axis=1)

# Append the sum as a new column to the existing array without column_stack
Scores_with_sum = np.concatenate((Scores_with_7thBats, row_sums.reshape(-1, 1)), ax

# Print the result
print("Array with Sum Column:")
print(Scores_with_sum)
plt.plot(Scores_with_sum[0:7,5],color='black',linestyle=':')
plt.xlabel('No. of Batsman')
plt.ylabel('Scores')
plt.title('Leader Board')
```

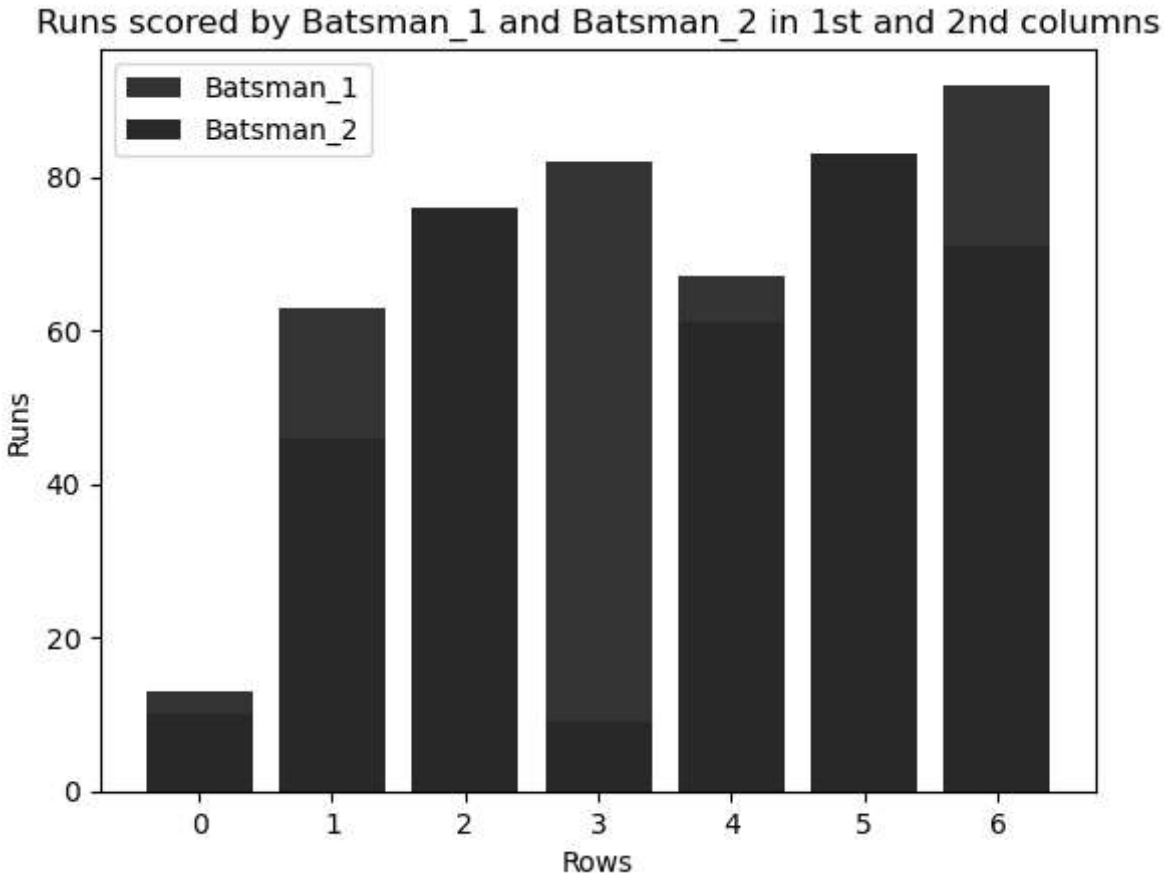
```
Scores with 5th Match:
[[13 10  9 33 41]
 [63 46 90 42 87]
 [39 76 13 29 72]
 [82  9 29 78 36]
 [67 61 59 36 92]]
Scores with 5th Match:
[[13 10  9 33 41]
 [63 46 90 42 87]
 [39 76 13 29 72]
 [82  9 29 78 36]
 [67 61 59 36 92]]
[[13 10  9 33 41]
 [63 46 90 42 87]
 [39 76 13 29 72]
 [82  9 29 78 36]
 [67 61 59 36 92]
 [77 83 98 95 89]
 [92 71 52 61 53]]
Array with Sum Column:
[[ 13  10   9  33  41 106]
 [ 63  46  90  42  87 328]
 [ 39  76  13  29  72 229]
 [ 82   9  29  78  36 234]
 [ 67  61  59  36  92 315]
 [ 77  83  98  95  89 442]
 [ 92  71  52  61  53 329]]
```

Out[8]: Text(0.5, 1.0, 'Leader Board')



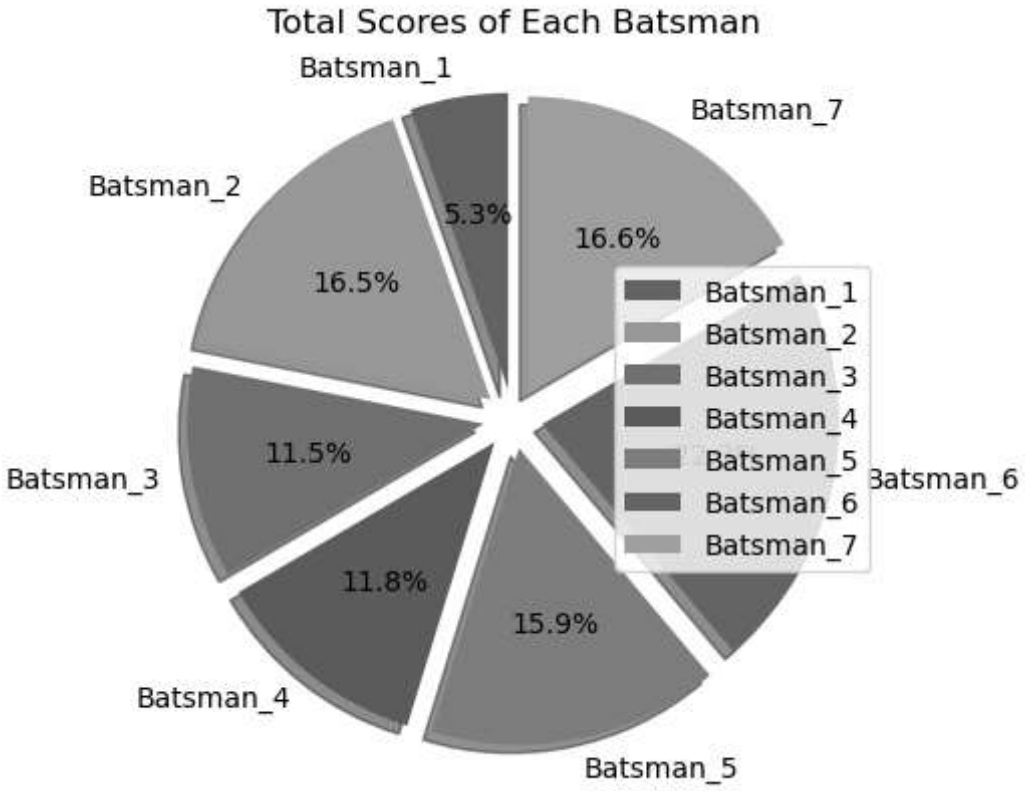
In [17]:
```
import matplotlib.pyplot as plt
import numpy as np

# Given data
```

```
# Extracting the 1st and 2nd columns
batsman_1 = Scores_with_sum[:, 0]
batsman_2 = Scores_with_sum[:, 1]

# Creating the bar chart
plt.bar(range(len(batsman_1)), batsman_1, color='purple', label='Batsman_1')
plt.bar(range(len(batsman_2)), batsman_2, color='darkred', label='Batsman_2')

# Adding labels and title
plt.xlabel('Rows')
plt.ylabel('Runs')
plt.title('Runs scored by Batsman_1 and Batsman_2 in 1st and 2nd columns')

# Adding legend
plt.legend()

# Display the plot
plt.show()
```



In [18]:
```
import matplotlib.pyplot as plt
import numpy as np



# Extracting the total scores (last column)
total_scores = Scores_with_sum[:, -1]

# Labels for legend
batsman_names = ['Batsman_1', 'Batsman_2', 'Batsman_3', 'Batsman_4', 'Batsman_5', '

# Exploded view
explode = [0.1] * len(batsman_names)

# Creating the pie chart
plt.pie(total_scores, labels=batsman_names, autopct='%1.1f%%', explode=explode, sta
```

```
# Adding title
plt.title('Total Scores of Each Batsman')

# Adding legend
plt.legend()

# Display the pie chart
plt.show()
```



Q. 746 import matplotlib.pyplot as plt

import numpy as np

Subplot 1

x_values = [5, 10, 25, 60, 80] y_values = [5, 17, 25, 40, 30]

plt.subplot(3, 2, 1) plt.plot(x_values, y_values, label='Line Plot', color='green', linestyle='dotted', marker='D') plt.xlabel('X Axis') plt.ylabel('Y Axis') plt.title('Line Plot') plt.legend()

Subplot 2

Scores_men = (22, 30, 35, 35, 26) Scores_women = (25, 32, 30, 35, 29) groups = np.arange(len(Scores_men))

plt.subplot(3, 2, 2) plt.bar(groups - 0.2, Scores_men, width=0.4, label='Men', color='green') plt.bar(groups + 0.2, Scores_women, width=0.4, label='Women', color='red') plt.xlabel('Groups') plt.ylabel('Scores') plt.title('Bar Plot') plt.legend()

Subplot 3

car_labels = ['Maruti Suzuki', 'Hyundai', 'Kia', 'Toyota', 'Honda'] popularity = [25, 50, 30, 20, 35]

plt.subplot(3, 2, 3) plt.pie(popularity, labels=car_labels, autopct='%1.1f%%') plt.title('Popularity of Car Company')

Subplot 4

math_marks = [88, 92, 80, 89, 100, 80, 60, 100, 80, 34] science_marks = [35, 79, 79, 48, 100, 88, 32, 45, 20, 30]

plt.subplot(3, 2, 4) plt.scatter(range(1, 11), math_marks, marker='o', color='yellow', label='Mathematics') plt.scatter(range(1, 11), science_marks, marker='*', color='blue', label='Science') plt.xlabel('Students') plt.ylabel('Marks') plt.title('Scatter Plot') plt.legend()

Subplot 5

languages = ['Java', 'Python', 'PHP', 'JavaScript', 'C', 'C++'] popularity_languages = [20, 100, 25, 30, 45, 50]

plt.subplot(3, 2, 5) plt.barh(languages, popularity_languages, color=['red', 'blue', 'green', 'orange', 'purple', 'brown']) plt.xlabel('Popularity') plt.ylabel('Programming Languages') plt.title('Horizontal Bar Chart')

Subplot 6

data = [10, 20, 20, 30, 30, 30, 40, 40, 40, 40, 50, 50, 50, 60, 60, 70]

plt.subplot(3, 2, 6) plt.hist(data, color='red', bins=10, edgecolor='black') plt.xlabel('Data') plt.ylabel('Frequency') plt.title('Histogram Chart')

Adjust layout to prevent clipping

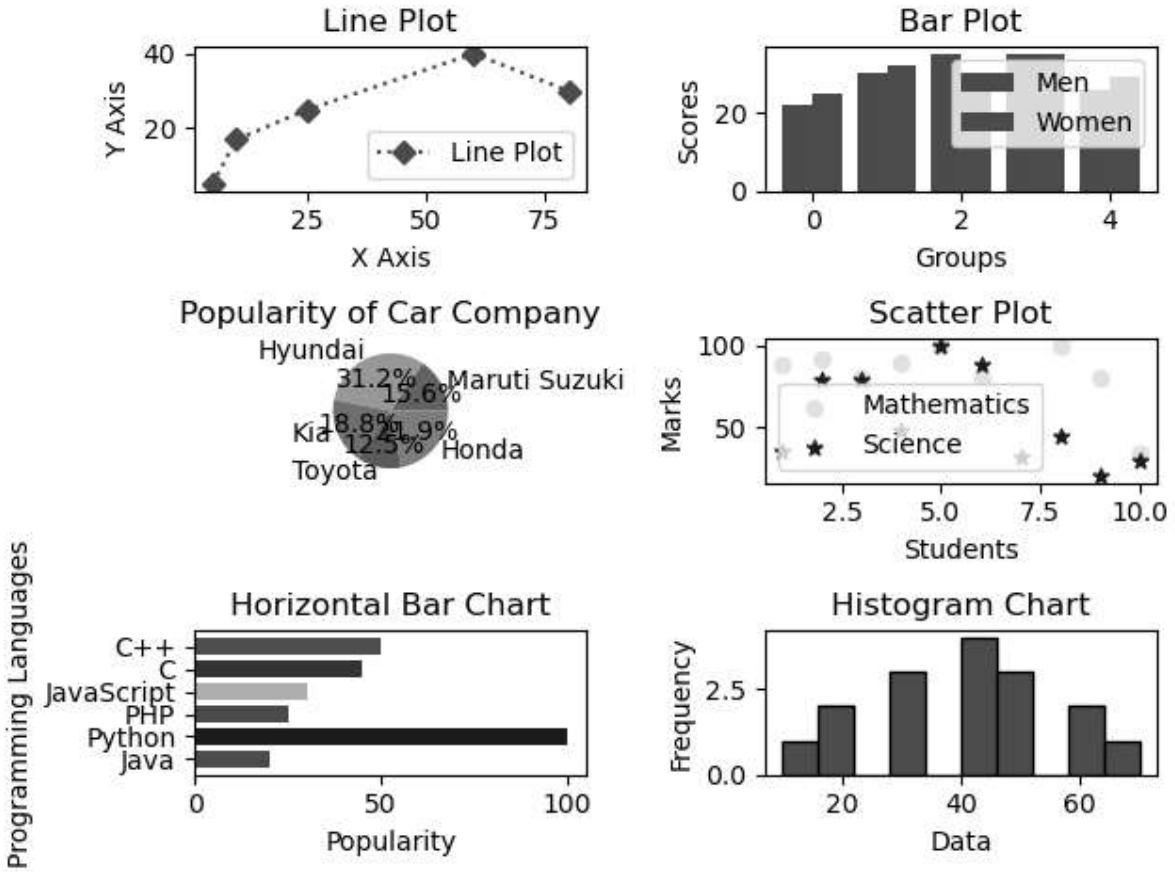
plt.tight_layout()

Display the plots

plt.show()

In [11]:
```
import matplotlib.pyplot as plt
import numpy as np

# Subplot 1
x_values = [5, 10, 25, 60, 80]
y_values = [5, 17, 25, 40, 30]

plt.subplot(3, 2, 1)
plt.plot(x_values, y_values, label='Line Plot', color='green', linestyle='dotted',
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Line Plot')
plt.legend()

# Subplot 2
Scores_men = (22, 30, 35, 35, 26)
Scores_women = (25, 32, 30, 35, 29)
groups = np.arange(len(Scores_men))

plt.subplot(3, 2, 2)
plt.bar(groups - 0.2, Scores_men, width=0.4, label='Men', color='green')
plt.bar(groups + 0.2, Scores_women, width=0.4, label='Women', color='red')
plt.xlabel('Groups')
plt.ylabel('Scores')
plt.title('Bar Plot')
plt.legend()

# Subplot 3
car_labels = ['Maruti Suzuki', 'Hyundai', 'Kia', 'Toyota', 'Honda']
popularity = [25, 50, 30, 20, 35]

plt.subplot(3, 2, 3)
plt.pie(popularity, labels=car_labels, autopct='%1.1f%%')
plt.title('Popularity of Car Company')

# Subplot 4
math_marks = [88, 92, 80, 89, 100, 80, 60, 100, 80, 34]
science_marks = [35, 79, 79, 48, 100, 88, 32, 45, 20, 30]

plt.subplot(3, 2, 4)
plt.scatter(range(1, 11), math_marks, marker='o', color='yellow', label='Mathematic
plt.scatter(range(1, 11), science_marks, marker='*', color='blue', label='Science')
plt.xlabel('Students')
plt.ylabel('Marks')
plt.title('Scatter Plot')
plt.legend()

# Subplot 5
languages = ['Java', 'Python', 'PHP', 'JavaScript', 'C', 'C++']
popularity_languages = [20, 100, 25, 30, 45, 50]

plt.subplot(3, 2, 5)
plt.barh(languages, popularity_languages, color=['red', 'blue', 'green', 'orange',
plt.xlabel('Popularity')
plt.ylabel('Programming Languages')
plt.title('Horizontal Bar Chart')

# Subplot 6
data = [10, 20, 20, 30, 30, 30, 40, 40, 40, 40, 50, 50, 50, 60, 60, 70]

plt.subplot(3, 2, 6)
```

```
plt.hist(data, color='red', bins=10, edgecolor='black')
plt.xlabel('Data')
plt.ylabel('Frequency')
plt.title('Histogram Chart')

# Adjust layout to prevent clipping
plt.tight_layout()

# Display the plots
plt.show()
```

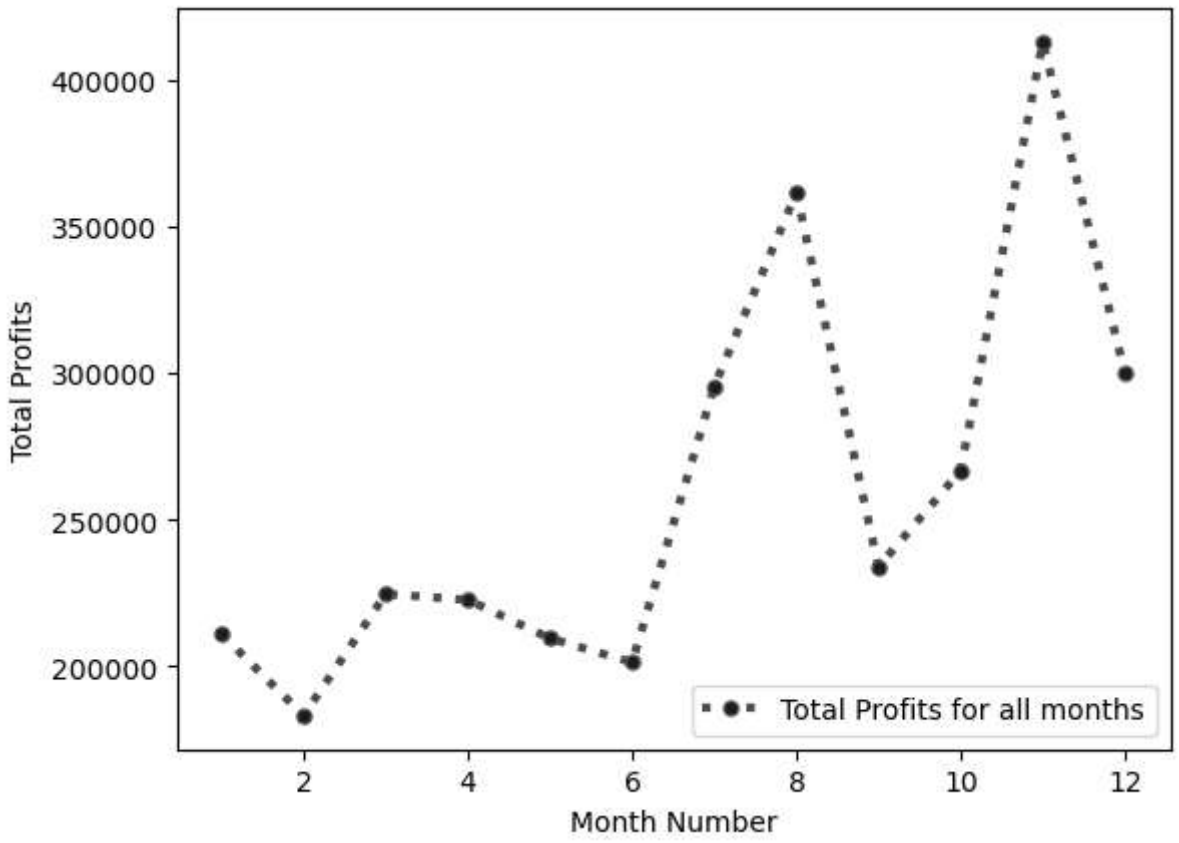


Q. 747 Consider the following datasheet to visualize Company Sales Data

Get total profit of all months and show line plot with the following Style properties: • Line Style dotted and Line-color should be red • Show legend at the lower right location. • X label name = Month Number • Y label name = Total Profits • Add a circle marker • Line marker color as blue • Line marker size as 5 • Line width should be 3

In [14]:
```
import matplotlib.pyplot as plt
import numpy as np
month_number=[1,2,3,4,5,6,7,8,9,10,11,12]
total_units=np.array([21100,18330,22470,22270,20960,20140,29550,36140,23400,26670,4
total_profit=total_units*10
print(total_profit)
plt.plot(month_number,total_profit,'o:r',ms=5,linewidth=3,markerfacecolor='b')
plt.xlabel('Month Number')
plt.ylabel('Total Profits')
plt.legend(['Total Profits for all months'],loc='lower right')
plt.show()
```

```
[211000 183300 224700 222700 209600 201400 295500 361400 234000 266700
 412800 300200]
```



In [8]:
```
help(plt.legend)
```

```
Help on function legend in module matplotlib.pyplot:

legend(*args, **kwargs)
    Place a legend on the Axes.

    Call signatures::

        legend()
        legend(handles, labels)
        legend(handles=handles)
        legend(labels)

    The call signatures correspond to the following different ways to use
    this method:

    **1. Automatic detection of elements to be shown in the legend**

    The elements to be added to the legend are automatically determined,
    when you do not pass in any extra arguments.

    In this case, the labels are taken from the artist. You can specify
    them either at artist creation or by calling the
    :meth:`~.Artist.set_label` method on the artist::

        ax.plot([1, 2, 3], label='Inline label')
        ax.legend()

    or::

        line, = ax.plot([1, 2, 3])
        line.set_label('Label via method')
        ax.legend()

    .. note::
        Specific artists can be excluded from the automatic legend element
        selection by using a label starting with an underscore, "_".
        A string starting with an underscore is the default label for all
        artists, so calling `.Axes.legend` without any arguments and
        without setting the labels manually will result in no legend being
        drawn.


    **2. Explicitly listing the artists and labels in the legend**

    For full control of which artists have a legend entry, it is possible
    to pass an iterable of legend artists followed by an iterable of
    legend labels respectively::

        ax.legend([line1, line2, line3], ['label1', 'label2', 'label3'])


    **3. Explicitly listing the artists in the legend**

    This is similar to 2, but the labels are taken from the artists'
    label properties. Example::

        line1, = ax.plot([1, 2, 3], label='label1')
        line2, = ax.plot([1, 2, 3], label='label2')
        ax.legend(handles=[line1, line2])


    **4. Labeling existing plot elements**

    .. admonition:: Discouraged
```

```
        This call signature is discouraged, because the relation between
        plot elements and labels is only implicit by their order and can
        easily be mixed up.

    To make a legend for all artists on an Axes, call this function with
    an iterable of strings, one for each legend item. For example::

        ax.plot([1, 2, 3])
        ax.plot([5, 6, 7])
        ax.legend(['First line', 'Second line'])


    Parameters
    ----------
    handles : sequence of `.Artist`, optional
        A list of Artists (lines, patches) to be added to the legend.
        Use this together with *labels*, if you need full control on what
        is shown in the legend and the automatic mechanism described above
        is not sufficient.

        The length of handles and labels should be the same in this
        case. If they are not, they are truncated to the smaller length.

    labels : list of str, optional
        A list of labels to show next to the artists.
        Use this together with *handles*, if you need full control on what
        is shown in the legend and the automatic mechanism described above
        is not sufficient.

    Returns
    -------
    `~matplotlib.legend.Legend`

    Other Parameters
    ----------------

    loc : str or pair of floats, default: :rc:`legend.loc`
        The location of the legend.

        The strings ``'upper left'``, ``'upper right'``, ``'lower left'``,
        ``'lower right'`` place the legend at the corresponding corner of the
        axes.

        The strings ``'upper center'``, ``'lower center'``, ``'center left'``,
        ``'center right'`` place the legend at the center of the corresponding edg
e
        of the axes.

        The string ``'center'`` places the legend at the center of the axes.

        The string ``'best'`` places the legend at the location, among the nine
        locations defined so far, with the minimum overlap with other drawn
        artists.  This option can be quite slow for plots with large amounts of
        data; your plotting speed may benefit from providing a specific location.

        The location can also be a 2-tuple giving the coordinates of the lower-lef
t
        corner of the legend in axes coordinates (in which case *bbox_to_anchor*
        will be ignored).

        For back-compatibility, ``'center right'`` (but no other location) can als
o
        be spelled ``'right'``, and each "string" location can also be given as a
```

```
    numeric value:

        ==================   =============
        Location String      Location Code
        ==================   =============
        'best' (Axes only)   0
        'upper right'        1
        'upper left'         2
        'lower left'         3
        'lower right'        4
        'right'              5
        'center left'        6
        'center right'       7
        'lower center'       8
        'upper center'       9
        'center'             10
        ==================   =============

    bbox_to_anchor : `.BboxBase`, 2-tuple, or 4-tuple of floats
        Box that is used to position the legend in conjunction with *loc*.
        Defaults to `axes.bbox` (if called as a method to `.Axes.legend`) or
        `figure.bbox` (if `.Figure.legend`).  This argument allows arbitrary
        placement of the legend.

        Bbox coordinates are interpreted in the coordinate system given by
        *bbox_transform*, with the default transform
        Axes or Figure coordinates, depending on which ``legend`` is called.

        If a 4-tuple or `.BboxBase` is given, then it specifies the bbox
        ``(x, y, width, height)`` that the legend is placed in.
        To put the legend in the best location in the bottom right
        quadrant of the axes (or figure)::

            loc='best', bbox_to_anchor=(0.5, 0., 0.5, 0.5)

        A 2-tuple ``(x, y)`` places the corner of the legend specified by *loc* at
        x, y.  For example, to put the legend's upper right-hand corner in the
        center of the axes (or figure) the following keywords can be used::

            loc='upper right', bbox_to_anchor=(0.5, 0.5)

    ncols : int, default: 1
        The number of columns that the legend has.

        For backward compatibility, the spelling *ncol* is also supported
        but it is discouraged. If both are given, *ncols* takes precedence.

    prop : None or `matplotlib.font_manager.FontProperties` or dict
        The font properties of the legend. If None (default), the current
        :data:`matplotlib.rcParams` will be used.

    fontsize : int or {'xx-small', 'x-small', 'small', 'medium', 'large', 'x-larg
e', 'xx-large'}
        The font size of the legend. If the value is numeric the size will be the
        absolute font size in points. String values are relative to the current
        default font size. This argument is only used if *prop* is not specified.

    labelcolor : str or list, default: :rc:`legend.labelcolor`
        The color of the text in the legend. Either a valid color string
        (for example, 'red'), or a list of color strings. The labelcolor can
        also be made to match the color of the line or marker using 'linecolor',
        'markerfacecolor' (or 'mfc'), or 'markeredgecolor' (or 'mec').

        Labelcolor can be set globally using :rc:`legend.labelcolor`. If None,
```

```
        use :rc:`text.color`.

    numpoints : int, default: :rc:`legend.numpoints`
        The number of marker points in the legend when creating a legend
        entry for a `.Line2D` (line).

    scatterpoints : int, default: :rc:`legend.scatterpoints`
        The number of marker points in the legend when creating
        a legend entry for a `.PathCollection` (scatter plot).

    scatteryoffsets : iterable of floats, default: ``[0.375, 0.5, 0.3125]``
        The vertical offset (relative to the font size) for the markers
        created for a scatter plot legend entry. 0.0 is at the base the
        legend text, and 1.0 is at the top. To draw all markers at the
        same height, set to ``[0.5]``.

    markerscale : float, default: :rc:`legend.markerscale`
        The relative size of legend markers compared to the originally drawn ones.

    markerfirst : bool, default: True
        If *True*, legend marker is placed to the left of the legend label.
        If *False*, legend marker is placed to the right of the legend label.

    reverse : bool, default: False
        If *True*, the legend labels are displayed in reverse order from the inpu
t.
        If *False*, the legend labels are displayed in the same order as the inpu
t.

        .. versionadded:: 3.7

    frameon : bool, default: :rc:`legend.frameon`
        Whether the legend should be drawn on a patch (frame).

    fancybox : bool, default: :rc:`legend.fancybox`
        Whether round edges should be enabled around the `.FancyBboxPatch` which
        makes up the legend's background.

    shadow : bool, default: :rc:`legend.shadow`
        Whether to draw a shadow behind the legend.

    framealpha : float, default: :rc:`legend.framealpha`
        The alpha transparency of the legend's background.
        If *shadow* is activated and *framealpha* is ``None``, the default value i
s
        ignored.

    facecolor : "inherit" or color, default: :rc:`legend.facecolor`
        The legend's background color.
        If ``"inherit"``, use :rc:`axes.facecolor`.

    edgecolor : "inherit" or color, default: :rc:`legend.edgecolor`
        The legend's background patch edge color.
        If ``"inherit"``, use take :rc:`axes.edgecolor`.

    mode : {"expand", None}
        If *mode* is set to ``"expand"`` the legend will be horizontally
        expanded to fill the axes area (or *bbox_to_anchor* if defines
        the legend's size).

    bbox_transform : None or `matplotlib.transforms.Transform`
        The transform for the bounding box (*bbox_to_anchor*). For a value
        of ``None`` (default) the Axes'
        :data:`~matplotlib.axes.Axes.transAxes` transform will be used.
```

```
    title : str or None
        The legend's title. Default is no title (``None``).

    title_fontproperties : None or `matplotlib.font_manager.FontProperties` or dic
t
        The font properties of the legend's title. If None (default), the
        *title_fontsize* argument will be used if present; if *title_fontsize* is
        also None, the current :rc:`legend.title_fontsize` will be used.

    title_fontsize : int or {'xx-small', 'x-small', 'small', 'medium', 'large', 'x
-large', 'xx-large'}, default: :rc:`legend.title_fontsize`
        The font size of the legend's title.
        Note: This cannot be combined with *title_fontproperties*. If you want
        to set the fontsize alongside other font properties, use the *size*
        parameter in *title_fontproperties*.

    alignment : {'center', 'left', 'right'}, default: 'center'
        The alignment of the legend title and the box of entries. The entries
        are aligned as a single block, so that markers always lined up.

    borderpad : float, default: :rc:`legend.borderpad`
        The fractional whitespace inside the legend border, in font-size units.

    labelspacing : float, default: :rc:`legend.labelspacing`
        The vertical space between the legend entries, in font-size units.

    handlelength : float, default: :rc:`legend.handlelength`
        The length of the legend handles, in font-size units.

    handleheight : float, default: :rc:`legend.handleheight`
        The height of the legend handles, in font-size units.

    handletextpad : float, default: :rc:`legend.handletextpad`
        The pad between the legend handle and text, in font-size units.

    borderaxespad : float, default: :rc:`legend.borderaxespad`
        The pad between the axes and legend border, in font-size units.

    columnspacing : float, default: :rc:`legend.columnspacing`
        The spacing between columns, in font-size units.

    handler_map : dict or None
        The custom dictionary mapping instances or types to a legend
        handler. This *handler_map* updates the default handler map
        found at `matplotlib.legend.Legend.get_legend_handler_map`.

    draggable : bool, default: False
        Whether the legend can be dragged with the mouse.


    See Also
    --------
    .Figure.legend

    Notes
    -----
    Some artists are not supported by this function.  See
    :doc:`/tutorials/intermediate/legend_guide` for details.

    Examples
    --------
    .. plot:: gallery/text_labels_and_annotations/legend.py
```

Q. 748 There is an array of scores of 5 Batsmen in 4 T20 Matches. Which is given below.

Scores= [[31, 12, 19, 53], [67, 48, 95, 83], [59, 67, 13, 59], [62, 29, 99, 88], [87, 91, 69, 76]]

1. Find the maximum score in T_20-3 and print it (use only the numpy module)
2. Find the minimum score of YUVRAJ and print it (use only the numpy module)
3. Add an extra column with the sum of all 4 T20 Matches' scores of each batsman in the array created and print it. (use only the numpy module)

In [12]:
```
import numpy as np

# Scores array
Scores = np.array([
    [31, 12, 19, 53],
    [67, 48, 95, 83],
    [59, 67, 13, 59],
    [62, 29, 99, 88],
    [87, 91, 69, 76]
])

# Task 1: Find the maximum score in T_20-3 and print it
max_score_t20_3 = np.max(Scores[:, 2])
print(f"Maximum score in T_20-3: {max_score_t20_3}")

# Task 2: Find the minimum score of YUVRAJ and print it
yuvraj_scores = Scores[1]  # Assuming YUVRAJ is in the second row
min_score_yuvraj = np.min(yuvraj_scores)
print(f"Minimum score of YUVRAJ: {min_score_yuvraj}")

# Task 3: Add an extra column with the sum of all 4 T20 Matches' scores of each bat
total_scores = np.sum(Scores, axis=1, keepdims=True)
Scores_with_totals = np.concatenate((Scores, total_scores), axis=1)

# Print the array with the extra column
print("Array with total scores:")
print(Scores_with_totals)
```

```
Maximum score in T_20-3: 99
Minimum score of YUVRAJ: 48
Array with total scores:
[[ 31  12  19  53 115]
 [ 67  48  95  83 293]
 [ 59  67  13  59 198]
 [ 62  29  99  88 278]
 [ 87  91  69  76 323]]
```

In [ ]: