Seaborn

(Code: Subhajit Das)

Data Visualization:

- The process of finding trends and correlations in data by representing it pictorially is called Data Visualization.
- To perform data visualization in python, we can use various python data visualization modules such as Matplotlib, Seaborn, Plotly, Ggplot.

What is Matplotlib:

➤ Seaborn is a library for making statistical graphics in Python.

➤ It builds on top of matplotlib and integrates closely with pandas data structures.

➤ Seaborn is a Python library that makes it easy to create statistical graphics. It is built on top of Matplotlib and provides a high-level interface for creating attractive and informative statistical graphics.

Matplotlib vs Seaborn

Sure, here are some key differences between Matplotlib and Seaborn:

1. **Functionality**:
    - Matplotlib is utilized for making basic graphs. Datasets are visualized with the help of bar graphs, histograms, pie charts, scatter plots, lines, and so on.
    - Seaborn contains a number of patterns and plots for data visualization. It uses fascinating themes. It helps in compiling whole data into a single plot. It also provides distribution of data.
2. **Syntax**:
    - Matplotlib uses comparatively complex and lengthy syntax.
    - Seaborn uses comparatively simple syntax which is easier to learn and understand.
3. **Dealing Multiple Figures**:
    - In Matplotlib, we can open and use multiple figures simultaneously.
    - Seaborn sets time for the creation of each figure.
4. **Visualization**:
    - Matplotlib is well connected with Numpy and Pandas and acts as a graphics package for data visualization in Python.
    - Seaborn is much more functional and organized than Matplotlib and treats the whole dataset as a single unit.
5. **Aesthetics**:
    - By default, Matplotlib's plots have a utilitarian look with basic color schemes.
    - Seaborn emphasizes aesthetics and provides pleasant default styles and color palettes.
6. **Statistical Visualizations**:
    - Matplotlib offers a broad range of plot types and customization options but has limited built-in statistical plotting functions.
    - Seaborn specializes in statistical visualizations and offers a wide array of built-in statistical plotting functions.
7. **Ease of Use**:

- Matplotlib follows a more low-level approach, requiring users to write more code to create visualizations.
- Seaborn focuses on simplicity and ease of use.

8. **Integration with Pandas**:
    - Seaborn is more comfortable handling Pandas data frames.
    - Matplotlib works efficiently with data frames and arrays.

Different categories of plot in Seaborn: Plots are basically used for visualizing the relationship between variables. Those variables can be either completely numerical or a category like a group, class, or division.

1. **Relational plots**: This plot is used to understand the relation between two variables.
2. **Categorical plots**: This plot deals with categorical variables and how they can be visualized.
3. **Distribution plots**: This plot is used for examining univariate and bivariate distributions
4. **Regression plots**: The regression plots in Seaborn are primarily intended to add a visual guide that helps to emphasize patterns in a dataset during exploratory data analyses.
5. **Matrix plots**: A matrix plot is an array of scatterplots.
6. **Multi-plot grids**: It is a useful approach to draw multiple instances of the same plot on different subsets of the dataset.

Seaborn Graphs:

1. **Line Plot**: This plot is used to visualize the relationship between two continuous variables. It can be useful in understanding the trend or pattern in the data. **Syntax**: sns.lineplot(x="x_values", y="y_values", data=dataframe)
2. **Scatter Plot**: Scatter plots' primary uses are to observe and show relationships between two numeric variables. **Syntax**: sns.scatterplot(x="x_values", y="y_values", data=dataframe)
3. **Bar Plot**: A bar chart can be used to compare the values of different categories. **Syntax**: sns.barplot(x="x_values", y="y_values", data=dataframe)
4. **Hist Plot**: A histogram can be used to show the distribution of values in a dataset. **Syntax**: sns.histplot(data=dataframe, x="column_name")
5. **Box Plot**: A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables. **Syntax**: sns.boxplot(x="x_values", y="y_values", data=dataframe)
6. **Violin Plot**: A violin plot plays a similar role as a box and whisker plot. It shows the distribution of quantitative data across several levels of one (or more) categorical variables. **Syntax**: sns.violinplot(x="x_values", y="y_values", data=dataframe)
7. **Heat Map Plot**: A heatmap can be used to show the relationship between two categorical variables. **Syntax**: sns.heatmap(data=dataframe)
8. **Point Plot**: This plot provides a similar function as a line plot. However, it plots points along with a line connecting them. **Syntax**: sns.pointplot(x="x_values", y="y_values", data=dataframe)
9. **Count Plot**: This plot shows the counts of observations in each categorical bin using bars. **Syntax**: sns.countplot(x="x_values", data=dataframe)
10. **Strip Plot**: This plot is very similar to a scatter plot, yet the locations of points are adjusted automatically to avoid overlap even if the jitter value is not applied. **Syntax**: sns.stripplot(x="x_values", y="y_values", data=dataframe)
11. **Density (KDE - Kernal Density Estimate) Plot**: A KDE plot can be used to show the density of values in a dataset. **Syntax**: sns.kdeplot(data=dataframe, x="column_name")
12. **Joint Plot**: A joint plot can be used to show the relationship between two variables, with a focus on the marginal distributions. **Syntax**: sns.jointplot(x="x_values", y="y_values", data=dataframe)
13. **Pair Plot**: A pair plot can be used to show the relationship between all pairs of variables in a dataset. **Syntax**: sns.pairplot(data=dataframe)

14. **Factor (Cat) Plot**: The factorplot function in Seaborn has been renamed to catplot. The catplot function in Seaborn is a figure-level function that provides a high-level interface for drawing attractive and informative statistical graphics involving categorical variables. **Syntax:** sns.catplot(x="x", y="y", data=df)
15. **Relational Graph**: This graph is used to understand the relationship between two variables. The major difference between relplot and scatterplot (as well as others) is that it can also segregate data attributes into rows and columns of subplots. **Syntax**: sns.relplot(x="x_values", y="y_values"

Install Seaborn

In [1]:
```
# pip install seaborn
```

Import Seaborn

In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
```

Linking the dataset from Seaborn for Data Visualizations

In [3]:
```
titan = sns.load_dataset("titanic")
titan
```

Out[3]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False	NaN
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN

891 rows × 15 columns

Linking another dataset for visualization

In [4]:
```
file = pd.read_csv('/content/drive/MyDrive/ML and DL DataSets/4_NSCA_DC Seaborn Pract

file
```

Out[4]:

	Not_AvailableME_of_Responder	NUMBER_OF_MEMBERS_IN_HOUSEHOLD	AVG_HOUSEHOLD_INCOME
0	Abhinav Das	8	105000
1	Rohit Kumar	3	75000
2	Debjit Roy	3	40000
3	Malabika Ghosh	10	100000
4	Arpita Chaudhury	4	100000
...	...	...	...
205	Punit Agarwal	3	150000
206	Ashish Chetri	3	50000
207	Ramesh BalakrishNot_Availablen	5	90000
208	Aaditya Varma	7	40000
209	Akhil Sharma	3	25000

210 rows × 25 columns

Line Plot

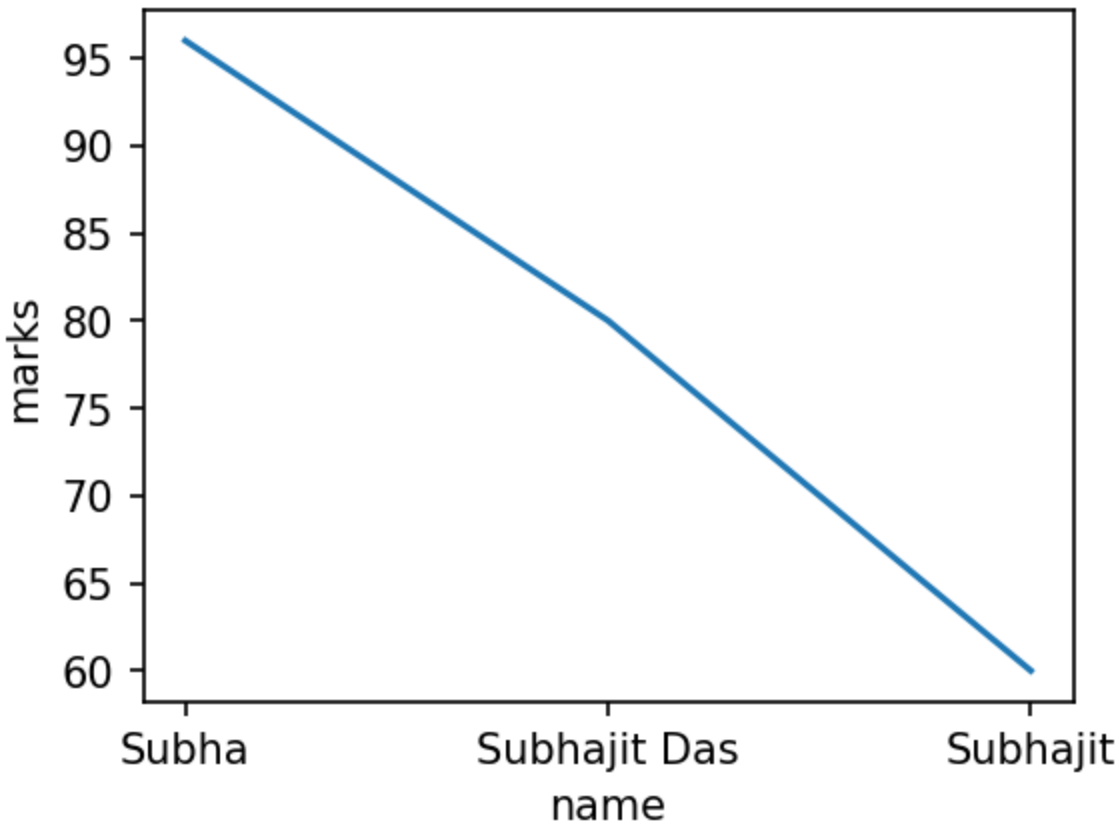
Syntax: sns.lineplot(x="x_values", y="y_values", data=dataframe)

The `sns.lineplot()` function in `seaborn` has the following parameters:

1. **data:** The data to be plotted. This can be a Pandas DataFrame or a NumPy array.
2. **x:** The name of the column or array to use for the x-axis.
3. **y**: The name of the column or array to use for the y-axis.
4. **hue**: The name of the column or array to use for coloring the lines.
5. **size**: The name of the column or array to use for sizing the lines.
6. **style**: The name of the column or array to use for styling the lines.
7. **legend**: A boolean indicating whether to show a legend.
8. **ci**: A string or tuple indicating the type of confidence interval to show.
9. **alpha**: The alpha value for the lines.
10. **markers**: A list of markers to use for the lines.
11. **dashes**: A list of dashes to use for the lines.
12. **err_kws**: Keyword arguments to pass to the errorbars.

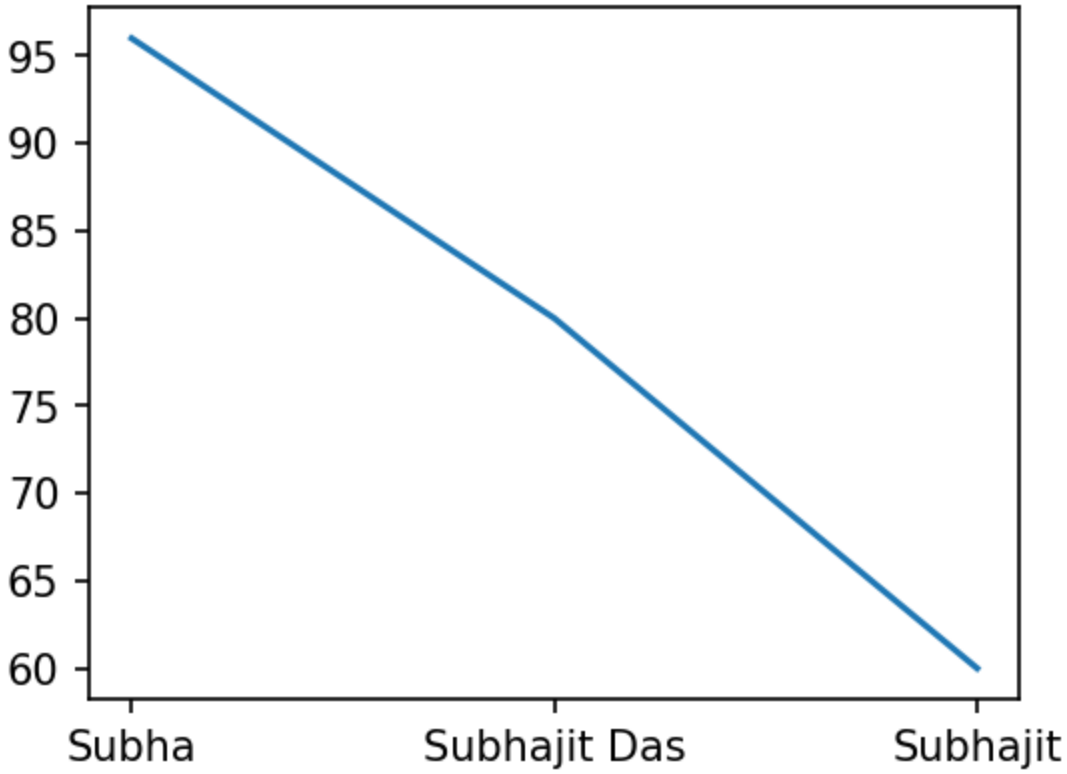
In [5]:
```
fig = plt.figure(figsize=(4, 3), dpi=150)

# Creating line plot with data frame
name = ('Subha', 'Subhajit Das', 'Subhajit')
marks = [96, 80, 60]

sample = pd.DataFrame({"name": name, "marks": marks})
sns.lineplot(x = 'name', y = 'marks', data = sample)
```

Out[5]: <Axes: xlabel='name', ylabel='marks'>



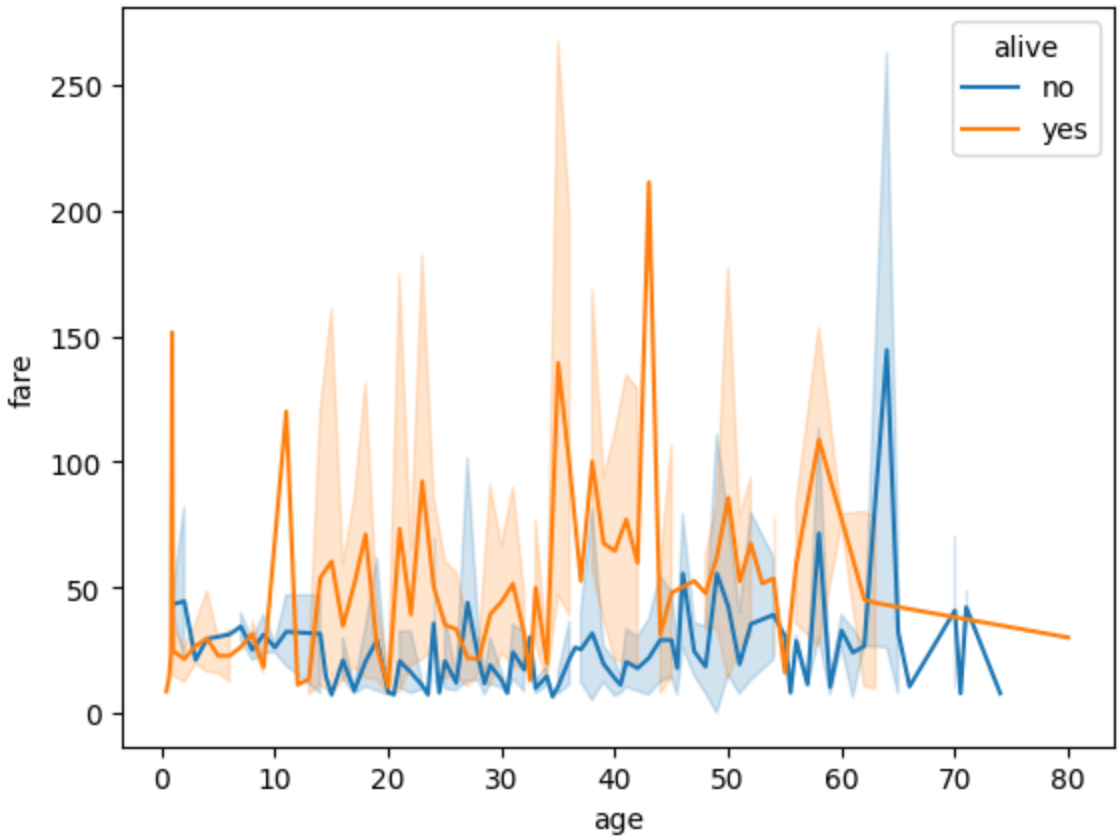
In [6]:
```
fig = plt.figure(figsize=(4, 3), dpi=150)

# Creating line plot without data frame
name = ('Subha', 'Subhajit Das', 'Subhajit')
marks = [96, 80, 60]

sns.lineplot(x = name, y = marks)
```

Out[6]: <Axes: >



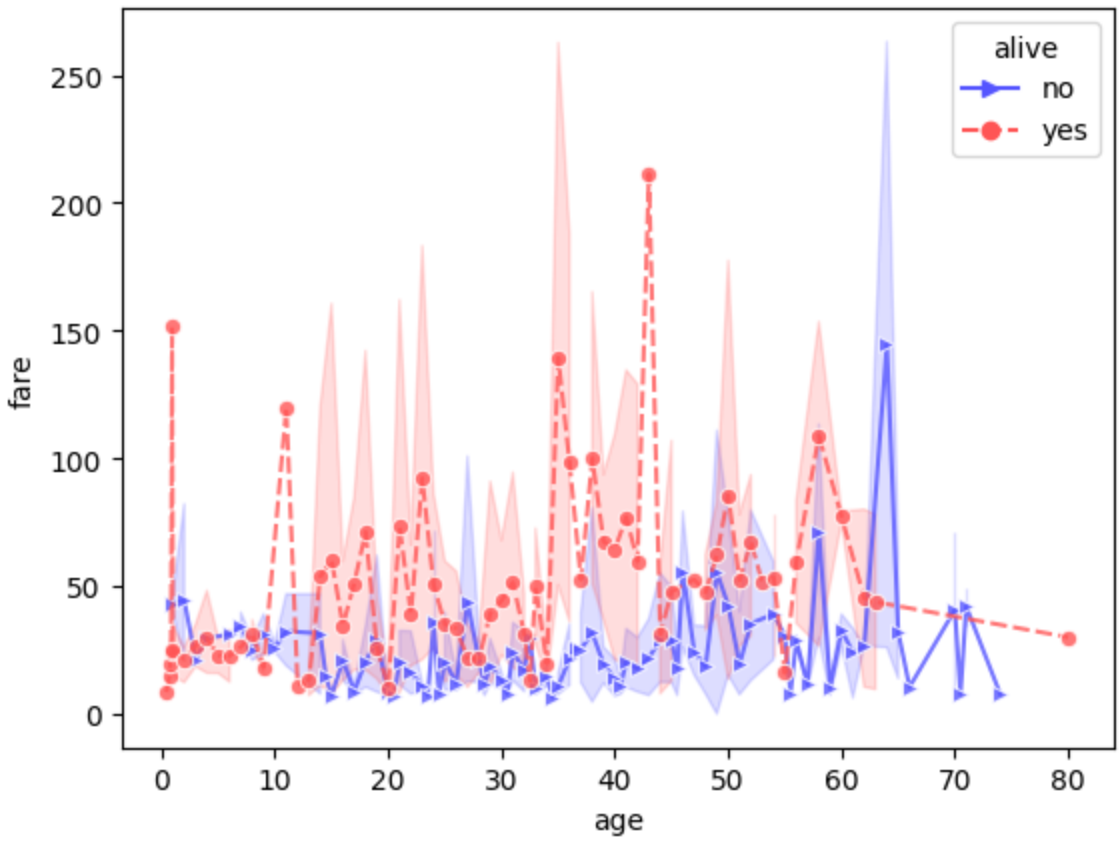
In [7]:
```
# Analyzing alive persons with age and fare
sns.lineplot(x = 'age', y = 'fare', data = titan, hue = 'alive')
```

Out[7]: <Axes: xlabel='age', ylabel='fare'>



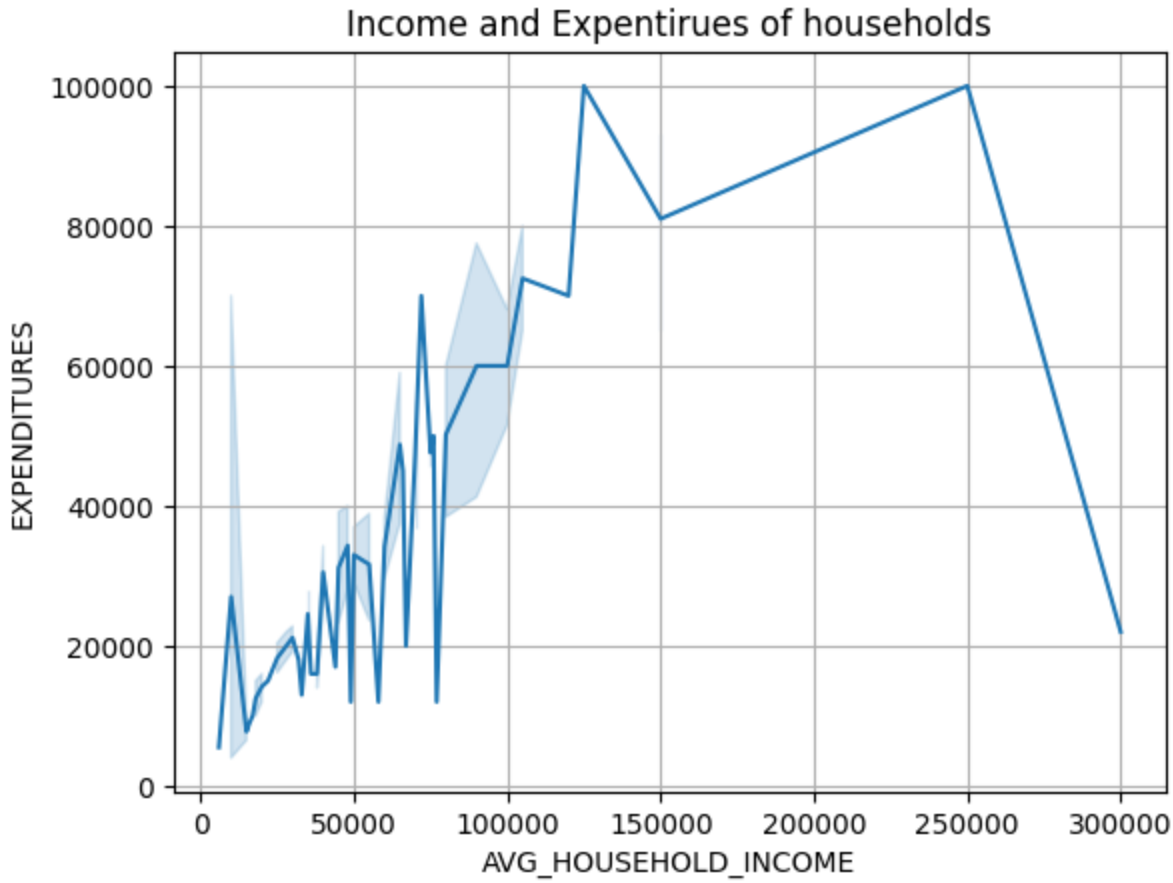
In [8]:
```
# Adding the lineplot parameters in the graph
sns.lineplot(x = 'age', y = 'fare',
             data = titan,
             hue = 'alive',
             style = 'alive', # It will create dotted line
             alpha = 0.8,
             palette = 'seismic', # This is the similar thing of cmap in matplotlib
             markers = ['>', 'o'],
             dashes = True, # If it's false, it will draw straight line instead of do
             legend = 'full') # It can be, 'auto', 'brief', 'full', 'boolean'
```

Out[8]: <Axes: xlabel='age', ylabel='fare'>



In [9]:
```
# Lineplot of Income and Expentirues of households
sns.lineplot(x = 'AVG_HOUSEHOLD_INCOME', y = 'EXPENDITURES', data = file)
plt.grid()
plt.title("Income and Expentirues of households", fontsize = 12)
```

Out[9]:
```
Text(0.5, 1.0, 'Income and Expentirues of households')
```



Scatter Plot

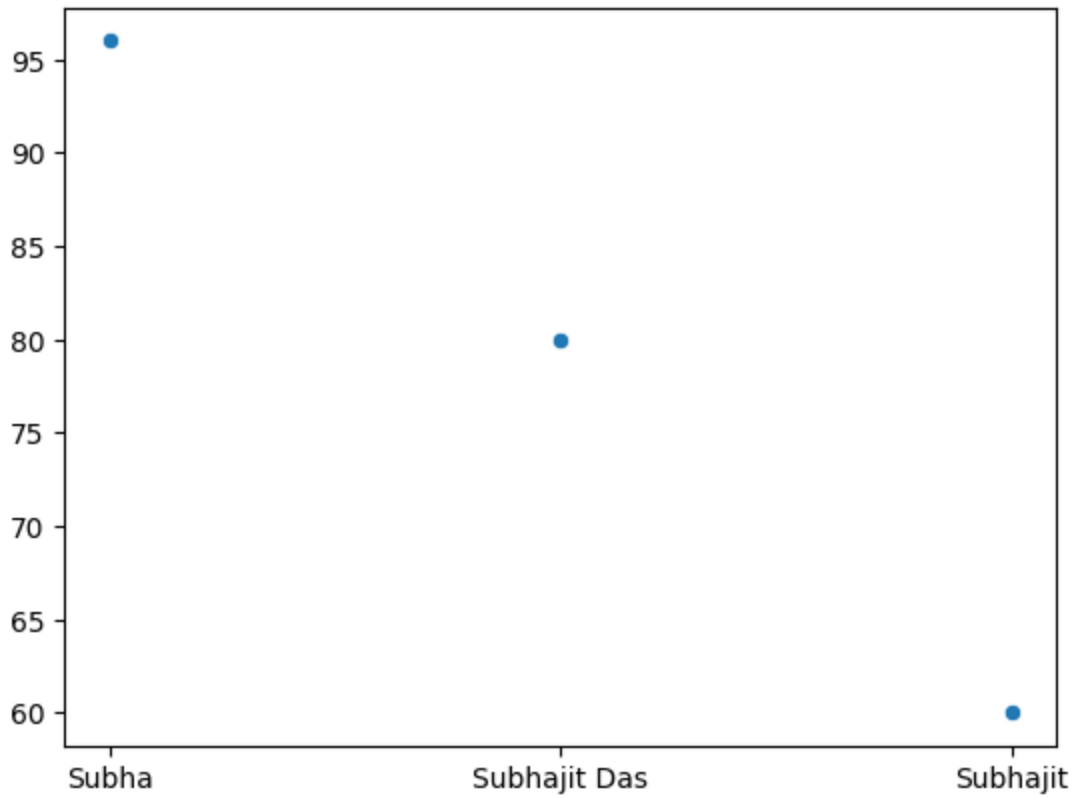
Syntax: sns.scatterplot(x="x_values", y="y_values", data=dataframe)

The `sns.scatterplot()` function in `seaborn` has the following parameters:

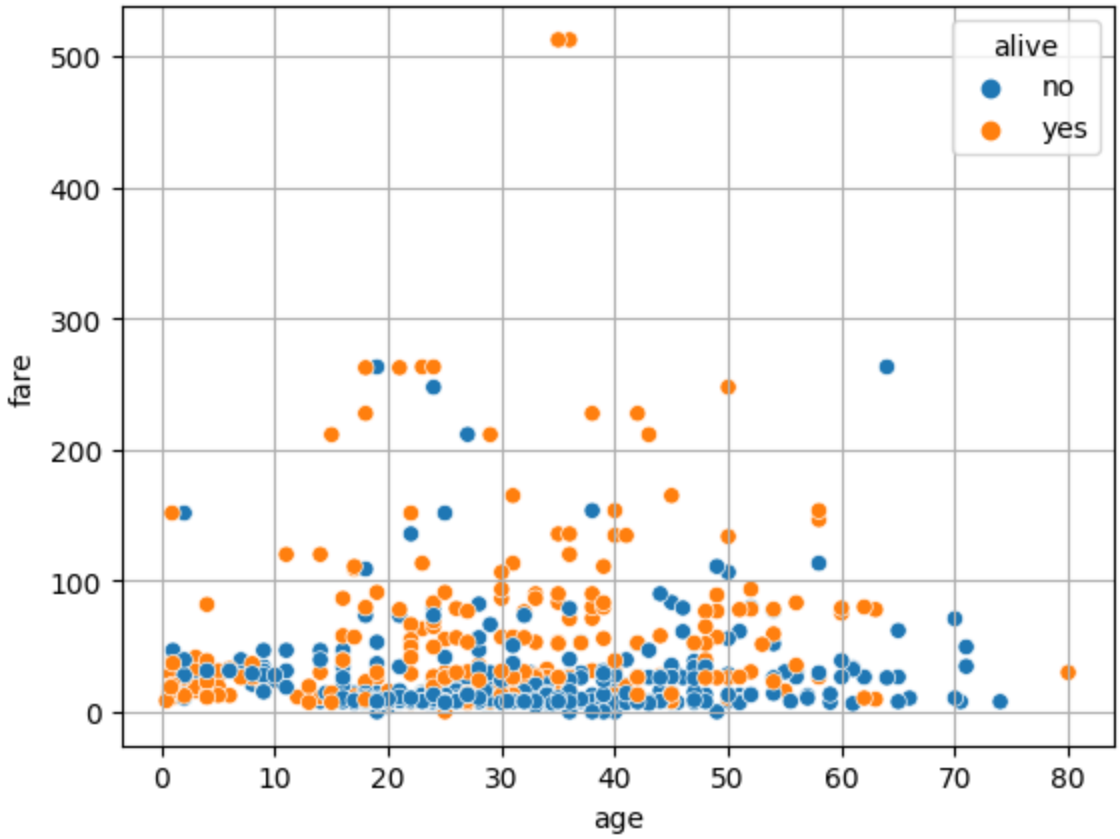
1. **x: and y:** These are the variables that will be plotted on the x and y axes, respectively.
2. **data:** This is the pandas DataFrame or NumPy array that contains the data to be plotted.
3. **hue:** This is the variable that will be used to color the points in the scatter plot.
4. **size:** This is the variable that will be used to determine the size of the points in the scatter plot.
5. **style:** This is the variable that will be used to determine the shape of the points in the scatter plot.
6. **palette:** This is the color palette that will be used to color the points in the scatter plot.
7. **hue_order:** This is the order in which the levels of the hue variable will be plotted.
8. **size_order:** This is the order in which the levels of the size variable will be plotted.
9. **style_order:** This is the order in which the levels of the style variable will be plotted.
10. **markers:** This is a list of markers that will be used to plot the data points.
11. **alpha:** This is the transparency of the points in the scatter plot.
12. **x_jitter:** This is the amount of jitter to apply to the x-coordinates.
13. **y_jitter:** This is the amount of jitter to apply to the y-coordinates.
14. **legend:** This is the legend that will be displayed with the scatter plot.
15. **ax:** This is the Axes object on which to plot the scatter plot.

In [10]:
```
# Creating scatter plot without data frame
name = ('Subha', 'Subhajit Das', 'Subhajit')
marks = [96, 80, 60]

sns.scatterplot(x = name, y = marks)
```

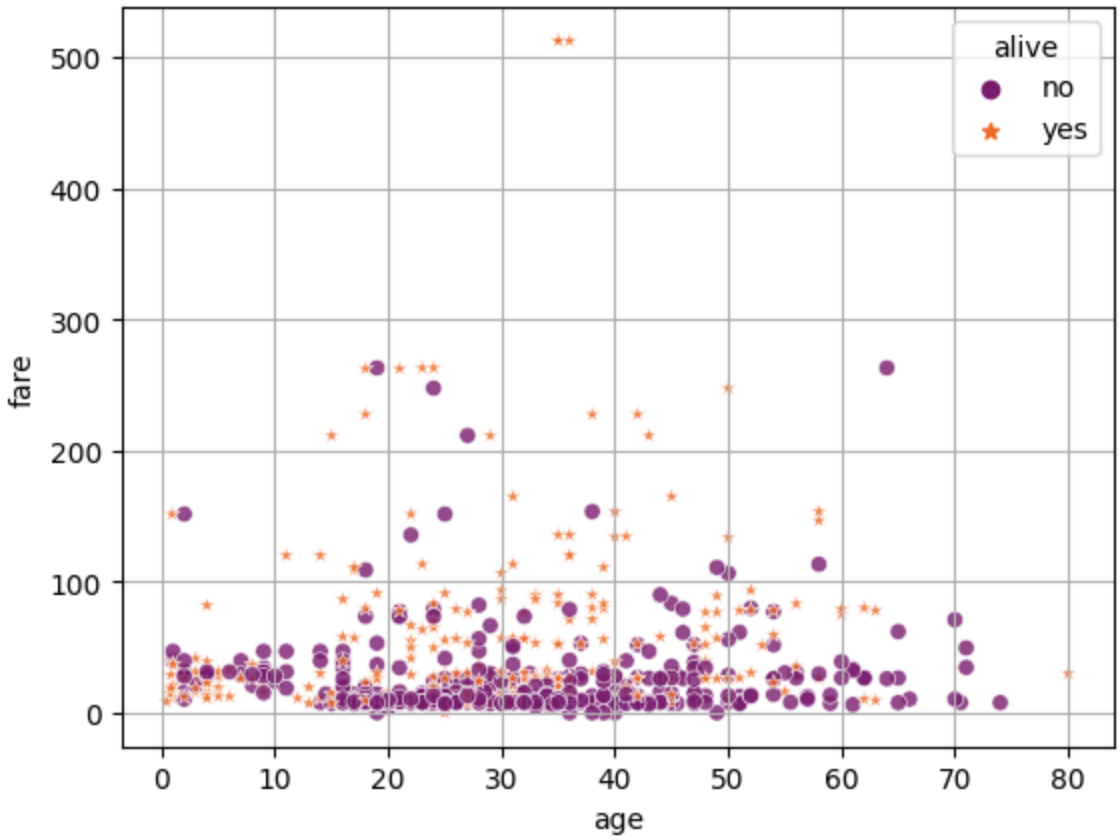
Out[10]: <Axes: >



In [11]:
```
# Analyzing alive persons with age and fare
sns.scatterplot(x = 'age', y = 'fare', data = titan, hue = 'alive')

plt.grid()
```

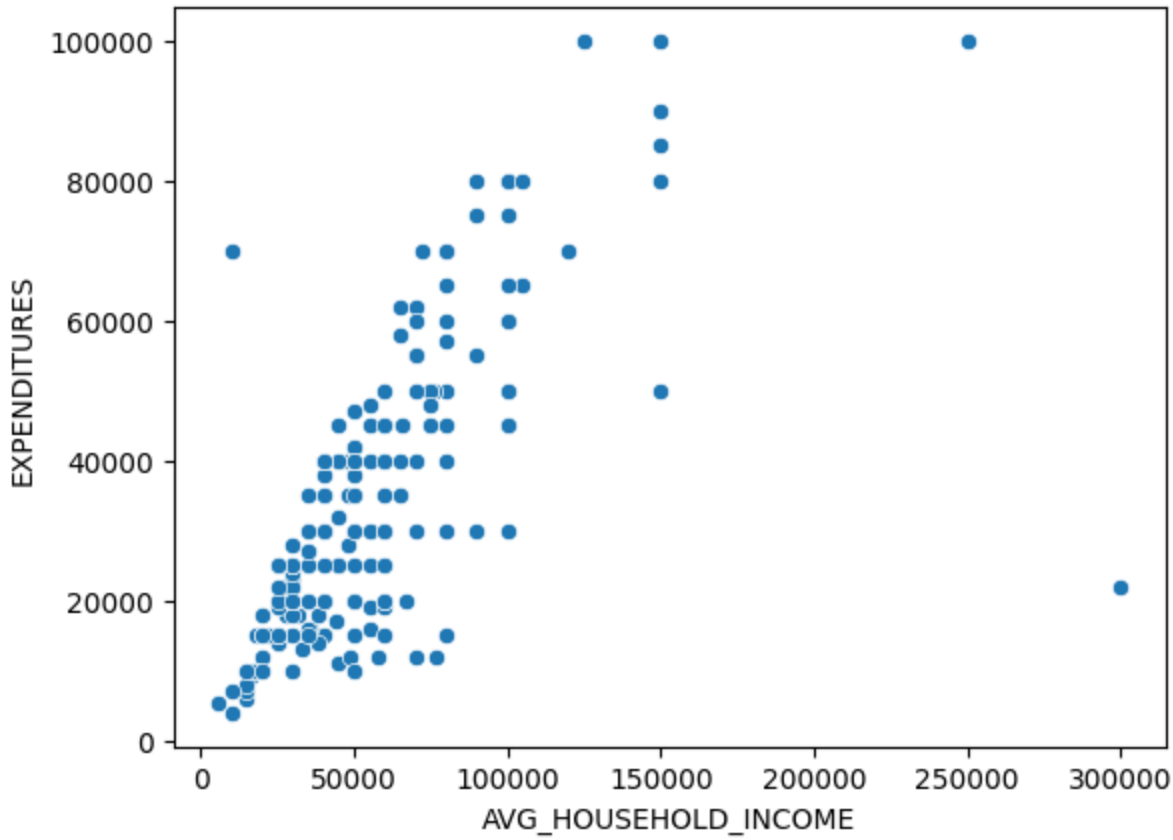


In [12]:
```
# Adding the scatter plot parameters in the graph
m = {'yes': '*', 'no': 'o'}
sns.scatterplot(x = 'age', y = 'fare',
                data = titan,
                hue = 'alive',
                style = 'alive', # It will add style in that column data
                sizes = (40, 20),
                alpha = 0.8,
                palette = 'inferno',
                legend = 'brief', # It can be, 'auto', 'brief', 'full', 'boolean'
                markers = m)
plt.grid()
```



In [13]:
```
# Scatterplot of Income and Expentirues of households
sns.scatterplot(x = 'AVG_HOUSEHOLD_INCOME', y = 'EXPENDITURES', data = file)
```

Out[13]: <Axes: xlabel='AVG_HOUSEHOLD_INCOME', ylabel='EXPENDITURES'>



Bar Plot:

Syntax: sns.barplot(x="x_values", y="y_values", data=dataframe)

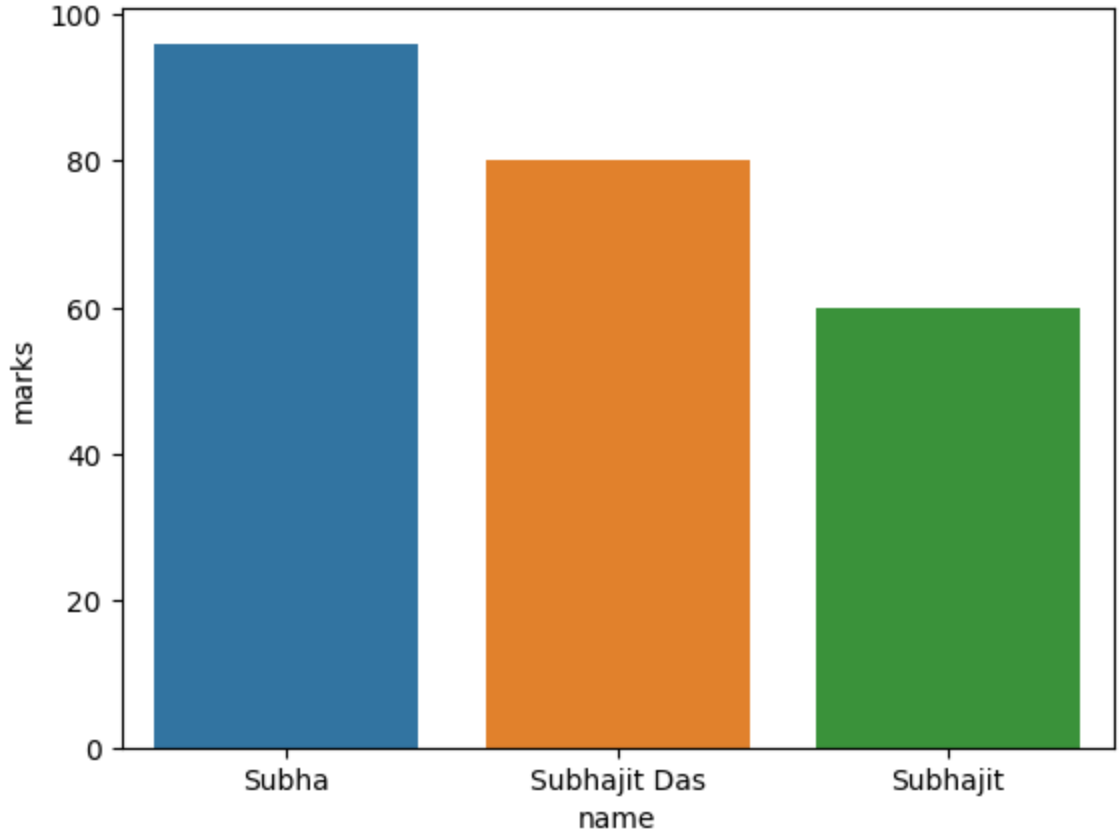
The `sns.barplot()` function in `seaborn` has the following parameters:

1. **x, y, hue:** These parameters take names of variables in data or vector data12. data: This parameter can take a DataFrame, array, or list of arrays for plotting.
2. **order, hue_order:** These parameters take lists of strings and are used to plot the categorical levels in order.
3. **estimator:** This is a string or callable that maps vector -> scalar, and it's used as a statistical function to estimate within each categorical bin.
4. **ci:** This parameter is used for the size of confidence intervals to draw around estimated values.
5. **n_boot:** This parameter is used for the number of bootstrap samples to compute confidence intervals.
6. **units:** This parameter is an identifier of sampling units, which will be used to perform a multilevel bootstrap and account for repeated measures design.
7. **orient:** This parameter can take "v" | "h", and it's used for the orientation of the plot (vertical or horizontal).
8. **color:** This parameter can take a matplotlib color, and it's used for all of the elements, or seed for a gradient palette.
9. **alpha:** This is the transparency of the bar in barplot.
10. **palette:** This parameter can take a seaborn palette or a dictionary mapping hue levels to matplotlib colors. It's used to set the color of the bars for different categories.
11. **dodge:** This is a boolean parameter. If set to True, the bars are dodged, meaning they are placed side-by-side. If set to False, the bars are stacked on top of each other.
12. **saturation:** This is a float parameter that controls the saturation of the colors, from 0 (completely desaturated) to 1 (fully saturated).

13. **errcolor:** This parameter takes a matplotlib color. It's used to set the color of the error bars.
14. **errwidth:** This is a float parameter that controls the linewidth of the error bars.
15. **capsize:** This is also a float parameter. It controls the size of the "caps" on error bars.

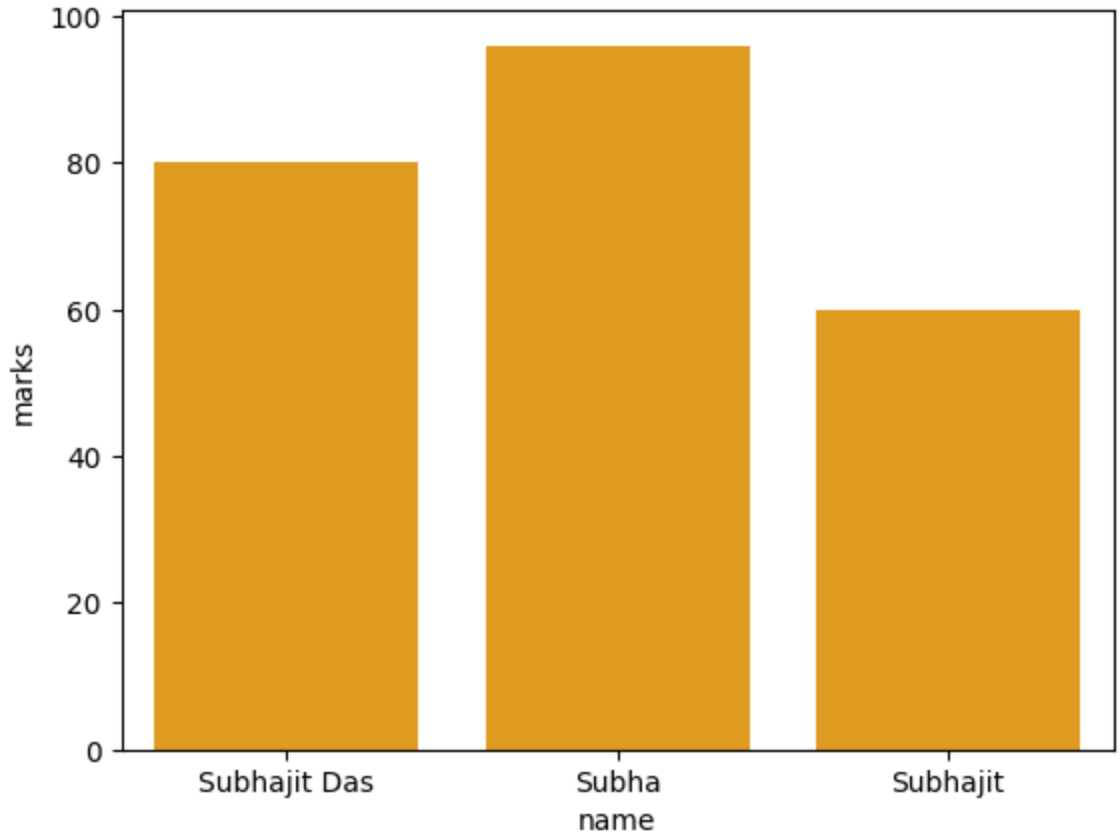
In [14]:
```
# Creating scatter plot with data frame
name = ('Subha', 'Subhajit Das', 'Subhajit')
marks = [96, 80, 60]

sample = pd.DataFrame({"name": name, "marks": marks})
#sns.barplot(x = 'name', y = 'marks', data = sample) # If we create this, without dat

# To create without dataframe
sns.barplot(x = sample.name, y = sample.marks)
```

Out[14]: <Axes: xlabel='name', ylabel='marks'>



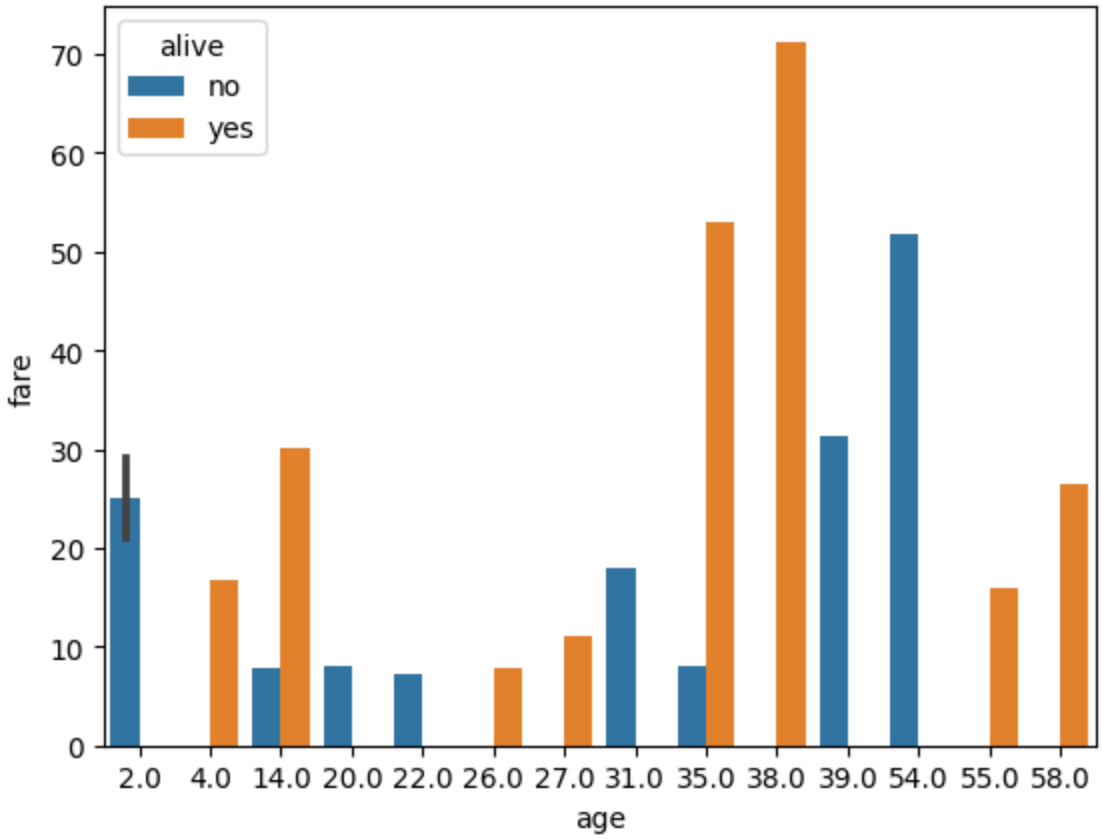
In [15]:
```
# Using order parameter
o = ['Subhajit Das', 'Subha', 'Subhajit']

sns.barplot(x = 'name', y = 'marks',
            data = sample,
            order = o,
            color = 'orange')
```

Out[15]: <Axes: xlabel='name', ylabel='marks'>



In [16]:
```
# Analyzing alive persons with age and fare of first 20 data
titan_20 = sns.load_dataset("titanic").head(20)
titan_20

sns.barplot(x = 'age', y = 'fare', data = titan_20, hue = 'alive')
```

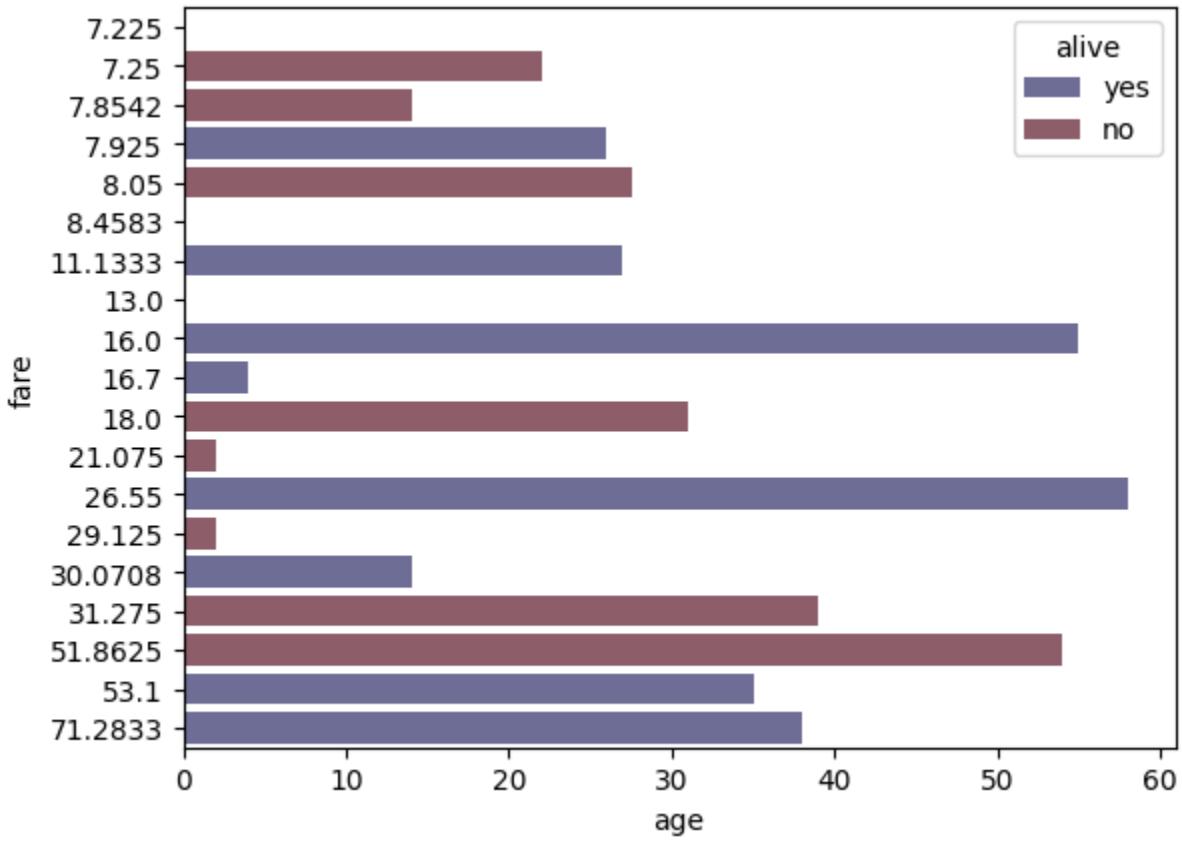
Out[16]: <Axes: xlabel='age', ylabel='fare'>



In [17]:
```
# Using the parameters
titan_20 = sns.load_dataset("titanic").head(20)
titan_20

sns.barplot(x = 'age', y = 'fare',
            data = titan_20,
            hue = 'alive',
            hue_order = ['yes', 'no'], # hue_order will change based on color
            alpha = 0.8,
            ci = 40, # Value can be 0 to 100
            n_boot = 8,
            orient = 'h', # It will not work, if it's a string value
            saturation = 0.8,
            palette = 'icefire',
            dodge = False)
```

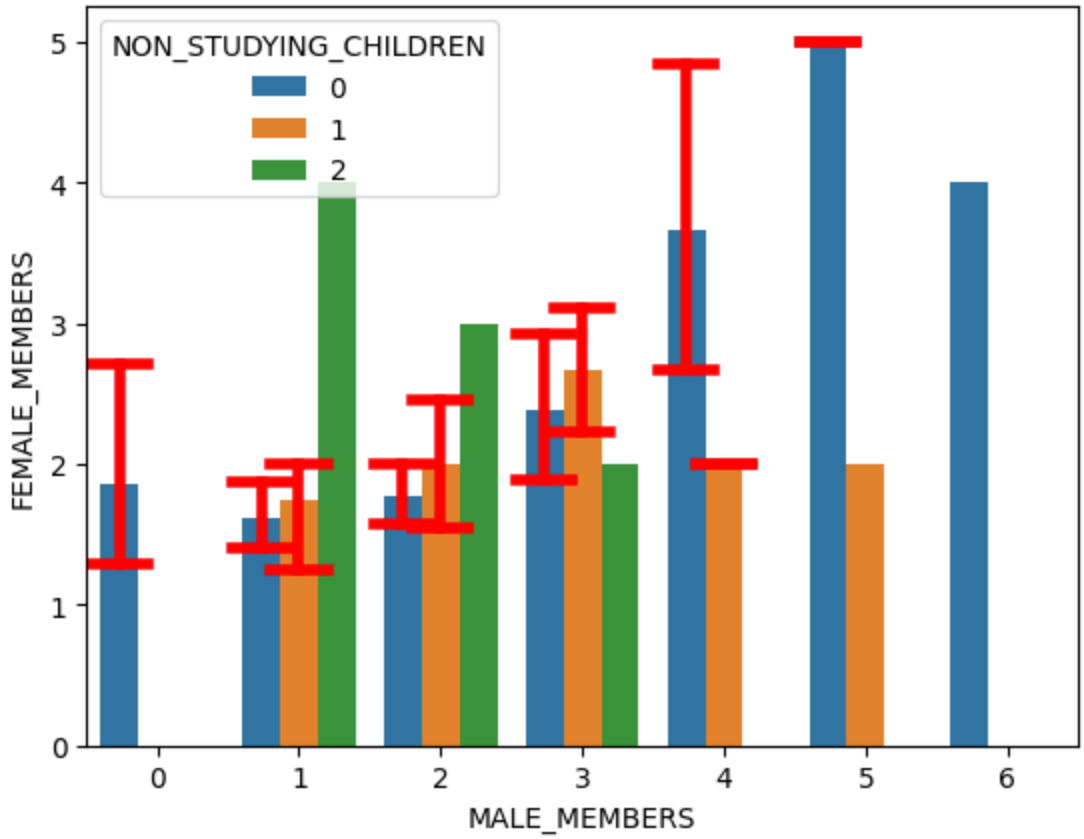
```
<ipython-input-17-fe9ca0b413bd>:5: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', 40)` for the same effect.

  sns.barplot(x = 'age', y = 'fare',
```

Out[17]: <Axes: xlabel='age', ylabel='fare'>



In [18]:
```
# Non-Studying in Male and Female
sns.barplot(x = 'MALE_MEMBERS', y = 'FEMALE_MEMBERS', data = file,
            hue = 'NON_STUDYING_CHILDREN',
            errcolor = 'red',
            errwidth = 4,
            capsize = 0.4)
```

Out[18]: <Axes: xlabel='MALE_MEMBERS', ylabel='FEMALE_MEMBERS'>



Hist Plot

Syntax: sns.histplot(x="column_name", data=dataframe)/

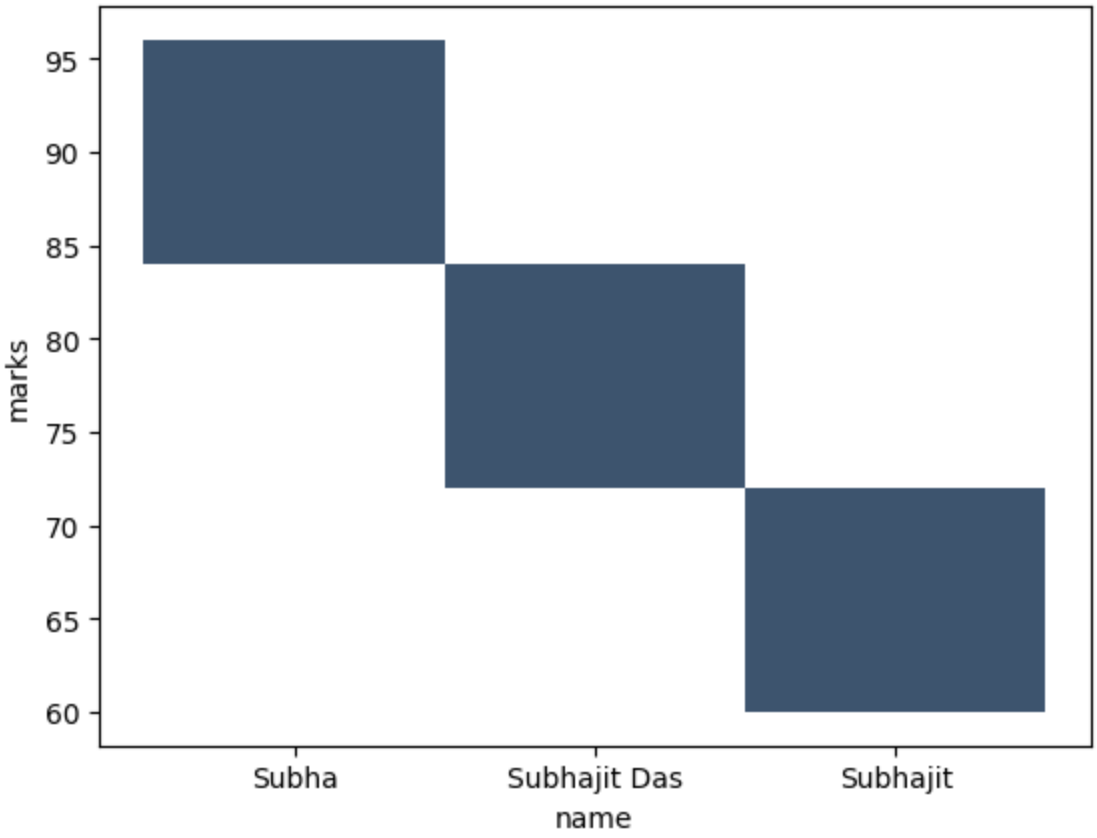
Syntax: sns.displot(x="column_name", data=dataframe), Also we can use y axis also in both syntax

The `sns.histplot()` , `sns.displot()` function in `seaborn` has the following parameters:

1. **data:** The data that you want to plot.
2. **x:** The column that you want to plot on the x-axis.
3. **y:** The column that you want to plot on the y-axis.
4. **hue:** The column that you want to use to color the bars in the histogram.
5. **bins:** The number of bins to use in the histogram.
6. **binwidth:** The width of each bin in the histogram.
7. **density:** Whether to plot the histogram as a density plot.
8. **kde:** Whether to plot the histogram as a kernel density estimate. (Kernal Density Estimator)
9. **stat:** The statistic to use to summarize the data in each bin.
10. **cumulative:** Whether to plot the cumulative distribution function.
11. **log:** Whether to plot the histogram on a logarithmic scale.
12. **p:** The probability value to use for the histogram.
13. **ax:** The axes to plot the histogram on.

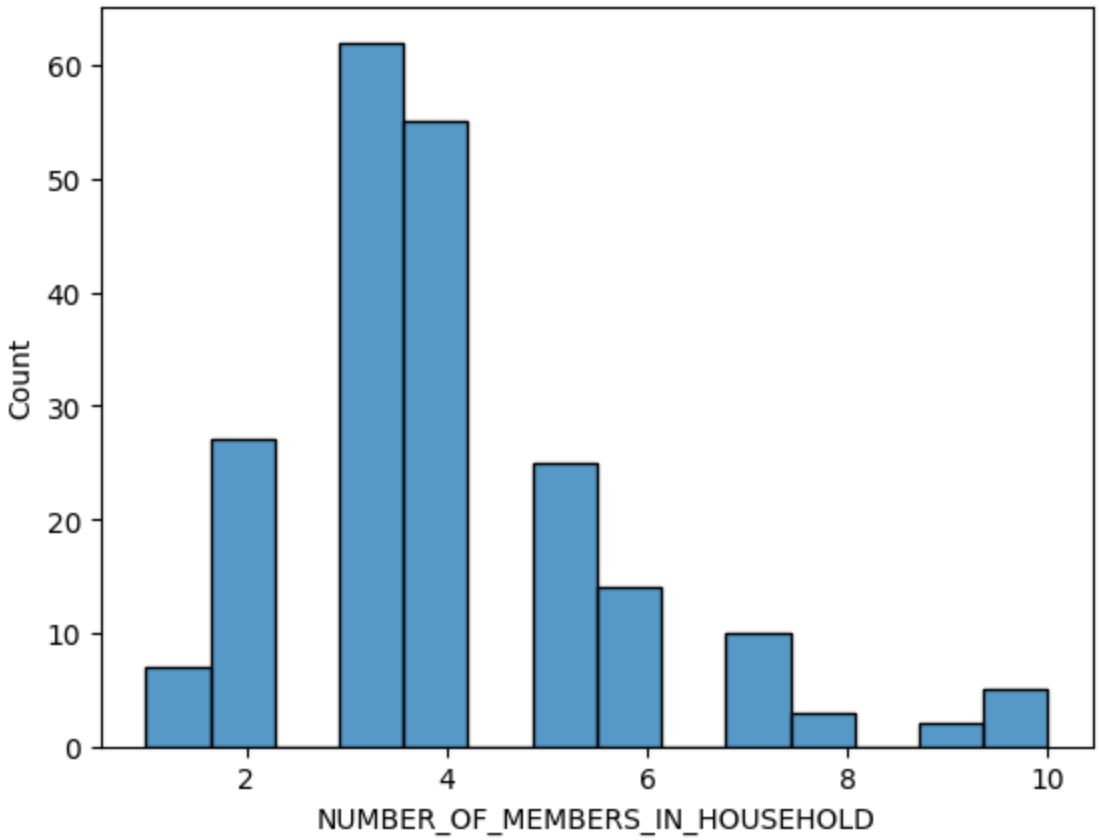
In [19]:
```
# Creating scatter plot with data frame
name = ('Subha', 'Subhajit Das', 'Subhajit')
marks = [96, 80, 60]

sample = pd.DataFrame({"name": name, "marks": marks})
sns.histplot(x = 'name', y = 'marks', data = sample)
```

Out[19]: <Axes: xlabel='name', ylabel='marks'>



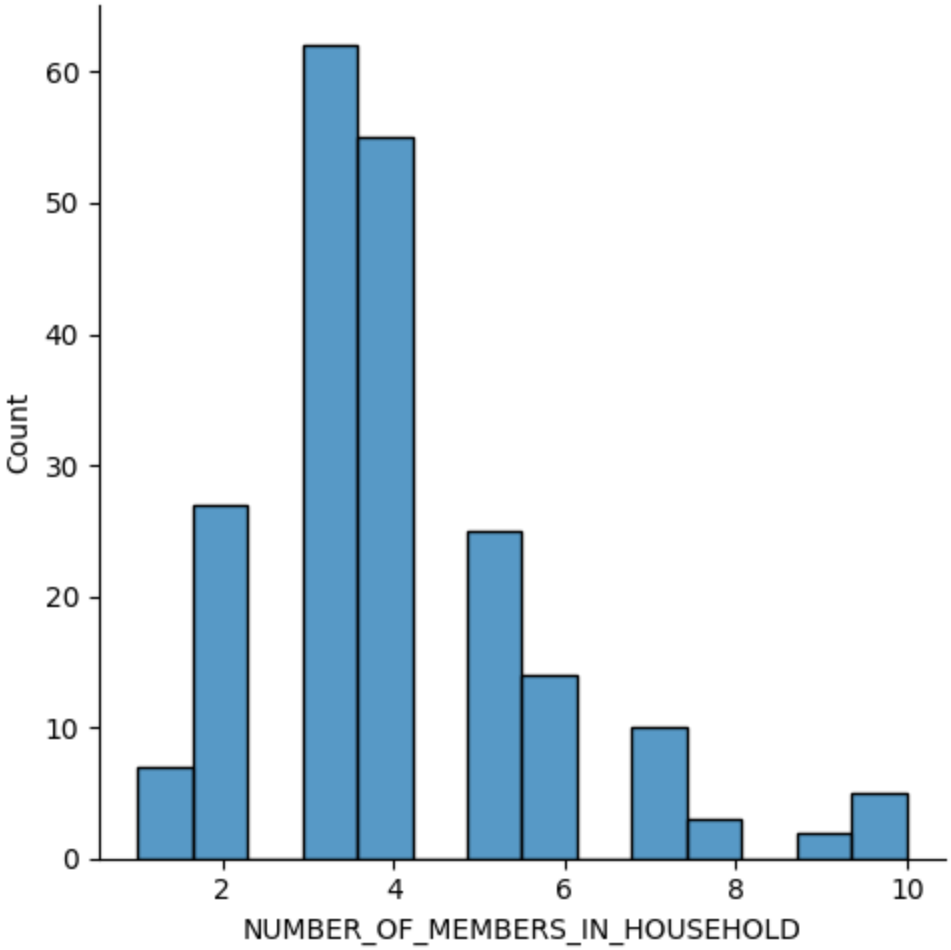
In [20]:
```
# Analyzing the number of members in household (histplot)
sns.histplot(x=file['NUMBER_OF_MEMBERS_IN_HOUSEHOLD'])
```

Out[20]: <Axes: xlabel='NUMBER_OF_MEMBERS_IN_HOUSEHOLD', ylabel='Count'>



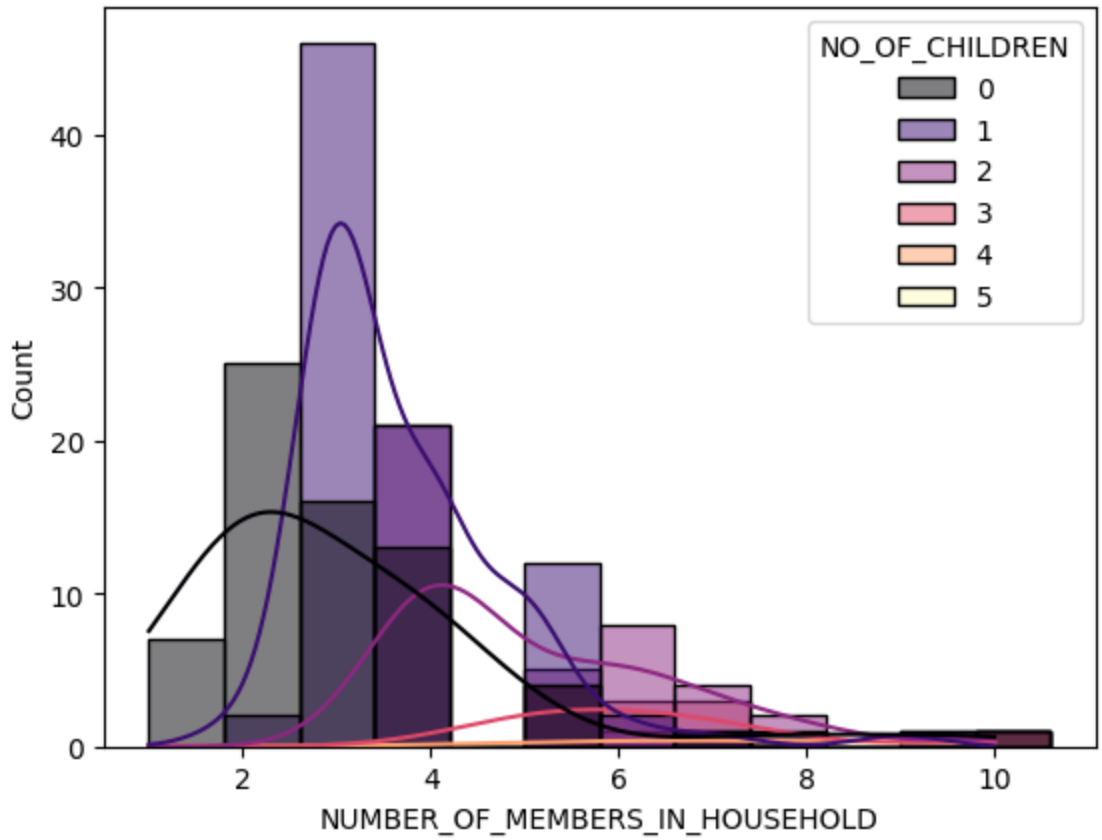
In [21]:
```
# Analyzing the number of members in household (displot)
sns.displot(x=file['NUMBER_OF_MEMBERS_IN_HOUSEHOLD'])
```

Out[21]: <seaborn.axisgrid.FacetGrid at 0x7a5e0ceba830>



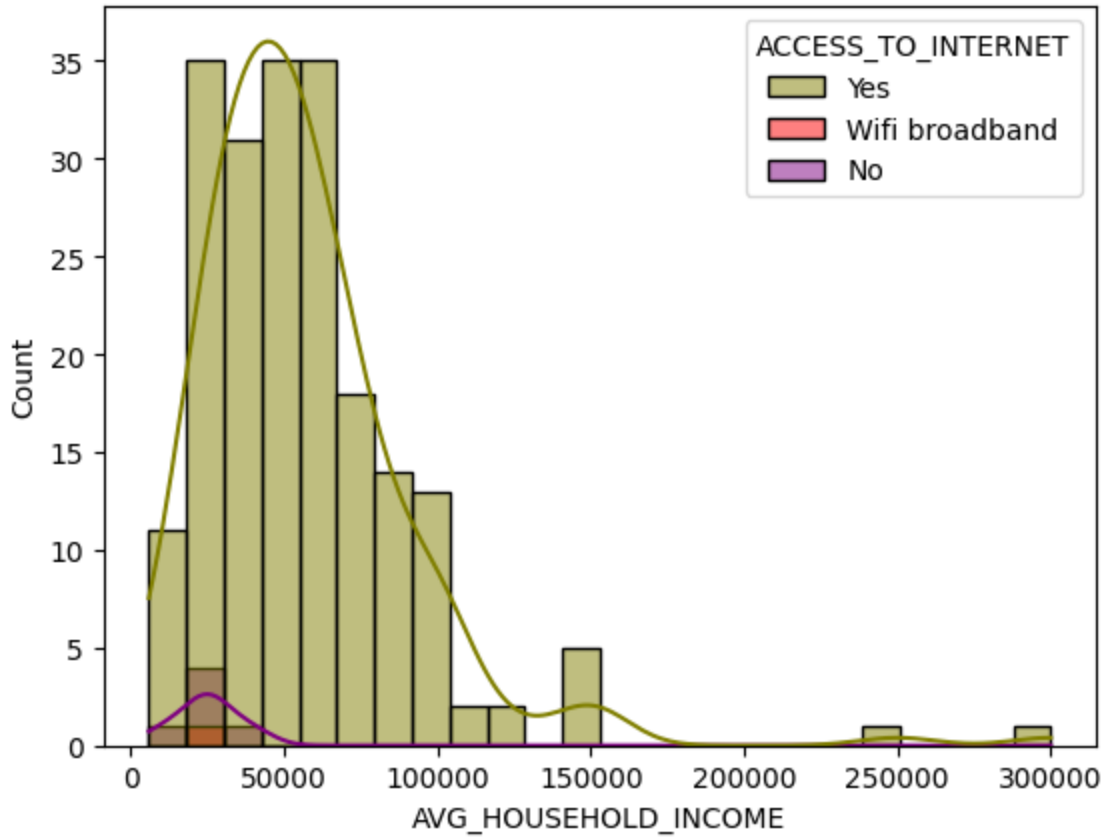
In [22]:
```
# Analyzing the number of members in household

sns.histplot(x = file['NUMBER_OF_MEMBERS_IN_HOUSEHOLD'],
             hue = file['NO_OF_CHILDREN'],
             bins = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
             binwidth = 0.8,
             kde = True,
             palette = "magma")
```

Out[22]: <Axes: xlabel='NUMBER_OF_MEMBERS_IN_HOUSEHOLD', ylabel='Count'>



In [23]:
```
#Histogram showing the availability of internet wrt income
sns.histplot(x=file['AVG_HOUSEHOLD_INCOME'],
             hue=file['ACCESS_TO_INTERNET'],
             kde=True,
             palette="brg_r")
```

Out[23]: <Axes: xlabel='AVG_HOUSEHOLD_INCOME', ylabel='Count'>



Box Plot

Syntax: sns.boxplot(x="x_values", y="y_values", data=dataframe)

The `sns.boxplot()` function in `seaborn` has the following parameters:

1. **x, y, hue:** These are inputs for plotting long-form data. They are names of variables in data or vector data.
2. **data:** This is the dataset for plotting. If x and y are absent, this is interpreted as wide-form.
3. **order, hue_order:** These are lists of strings. They are used to order the plot of the categorical levels; otherwise, the levels are inferred from the data objects.
4. **orient:** This can be either “v” or “h”. It is the orientation of the plot (vertical or horizontal). This is usually inferred based on the type of the input variables, but it can be used to resolve ambiguity when both x and y are numeric or when plotting wide-form data.
5. **color:** This is a matplotlib color. It is a single color for all of the elements in the plot.
6. **palette:** This can be a palette name, list, or dict. It is used for colors to use for the different levels of the hue variable. It should be something that can be interpreted by color_palette(), or a dictionary mapping hue levels to matplotlib colors.
7. **saturation:** This is a float. It is a proportion of the original saturation to draw colors at.
8. **width:** This is a float. It is the width of a full element when not using hue nesting, or width of all the elements for one level of the major grouping variable.
9. **dodge:** This is a boolean. When hue nesting is used, it determines whether elements should be shifted along the categorical axis.
10. **fliersize:** This is a float. It is the size of the markers used to indicate outlier observations.
11. **linewidth:** This is a float. It is the width of the gray lines that frame the plot elements.

12. **whis:** This is a float. It is the maximum length of the plot whiskers as proportion of the interquartile range.
13. **ax:** This can be a matplotlib Axes. It is an Axes object to draw the plot onto; otherwise, it uses the current Axes.
14. **notch:** This parameter specifies whether or not to add notches to the boxes.
15. **showmeans:** This is a boolean parameter. If set to True, it will show the mean of the data. The default value is False.
16. **meanprops:** This is a dictionary of properties to customize the appearance of the mean. For example, you can set it as follows to change the marker style, edge color, and size.

In [24]:
```
df_tips = sns.load_dataset("tips")
df_tips
```

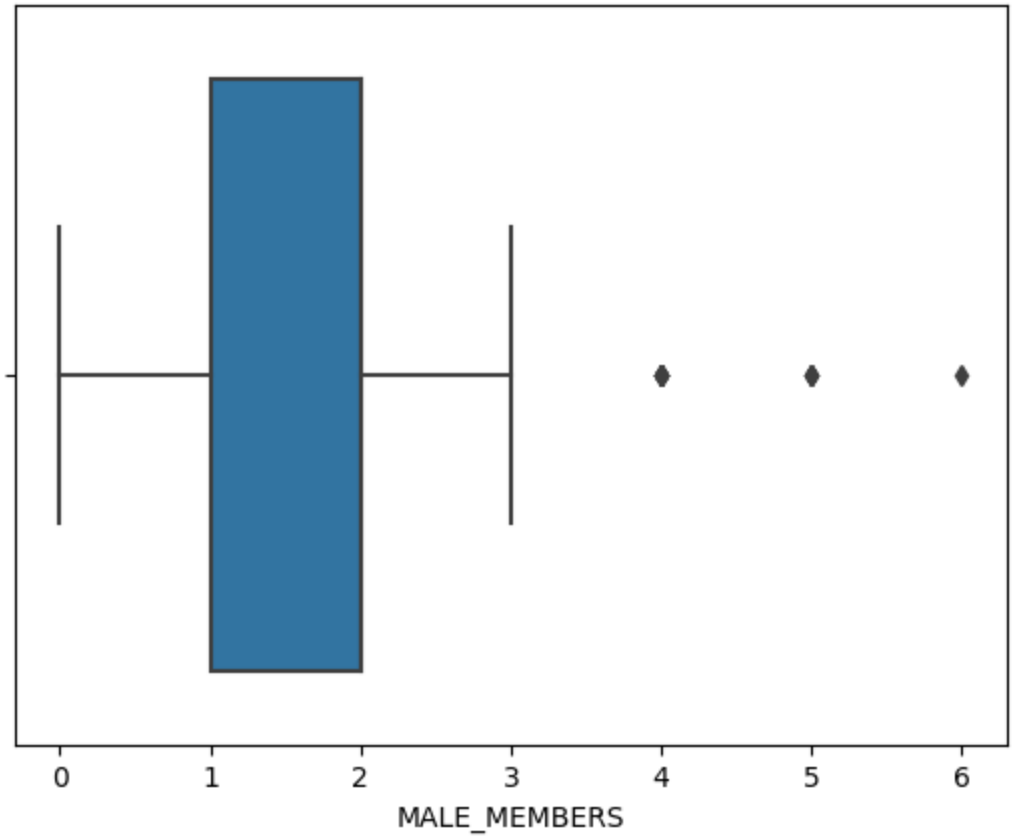
Out[24]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

244 rows × 7 columns

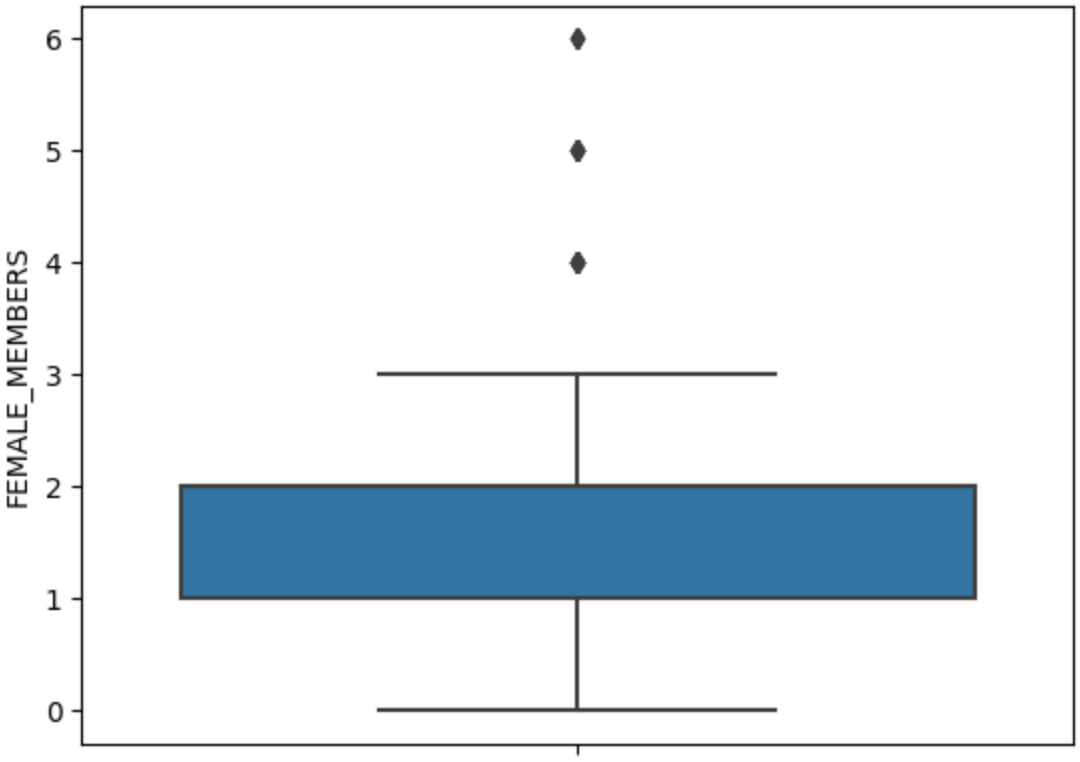
In [25]:
```
# Box plot of a single column
sns.boxplot(x = file.MALE_MEMBERS)
```

Out[25]: <Axes: xlabel='MALE_MEMBERS'>



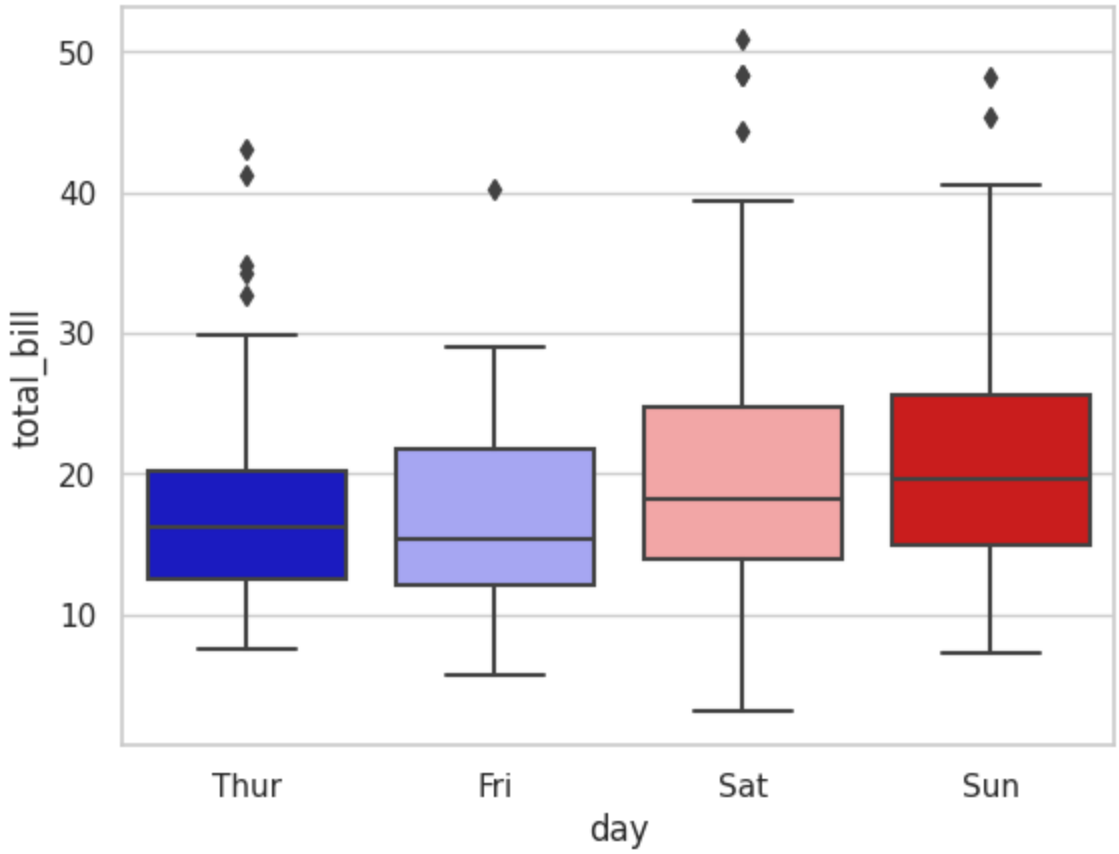
In [26]:
```
# Box plot of a single column
sns.boxplot(y = file['FEMALE_MEMBERS'])
```

Out[26]: <Axes: ylabel='FEMALE_MEMBERS'>



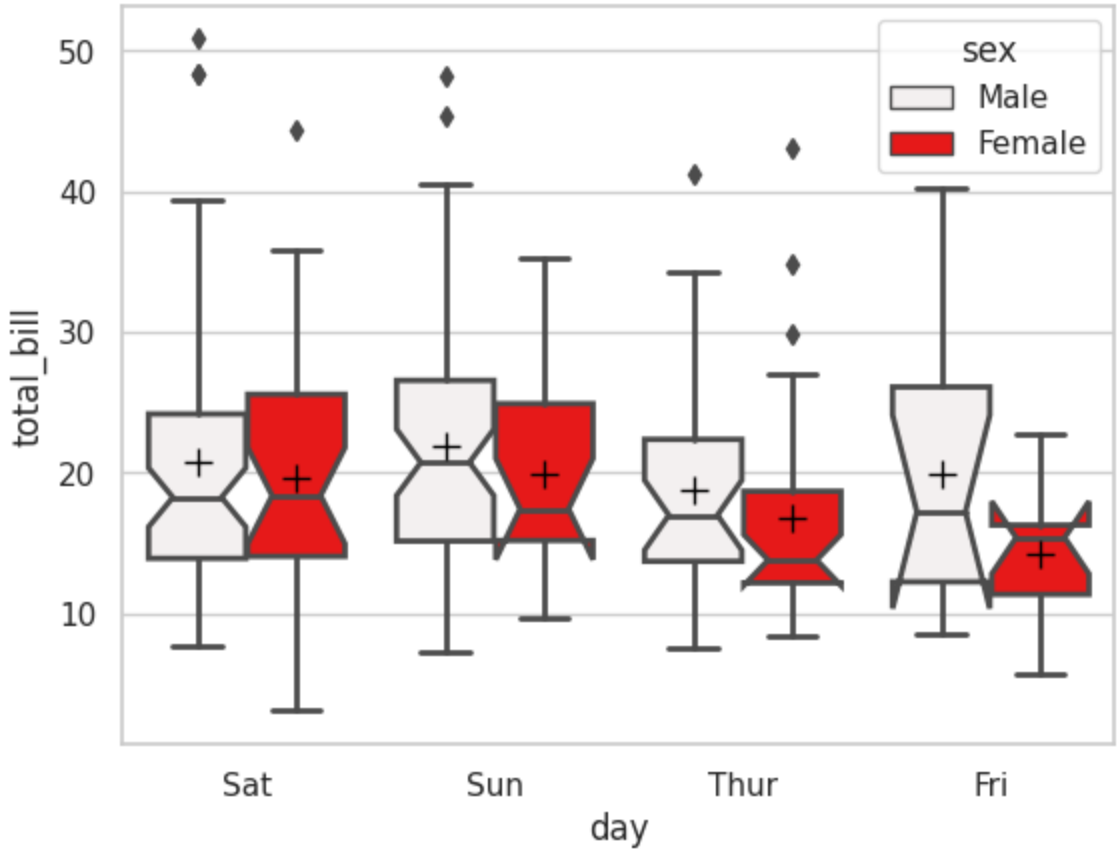
In [27]:
```
sns.set(style = 'whitegrid')
sns.boxplot(x = 'day', y = 'total_bill', data = df_tips, palette="seismic")
```

Out[27]: <Axes: xlabel='day', ylabel='total_bill'>



In [28]:
```
# Parameters used in boxplot
sns.set(style = 'whitegrid')
sns.boxplot(x = 'day', y = 'total_bill', data = df_tips,
            hue = 'sex',
            order = ['Sat', 'Sun', 'Thur', 'Fri'],
            orient = 'v', # By default it's 'v'. If we give 'h' it will throw error o
            showmeans = True,
            meanprops = {"marker": "+", "markeredgecolor": "black", "markersize": "10
            notch = True,
            width = 0.8,
            linewidth = 2,
            color = 'red',
            saturation = 0.8)
```

Out[28]:
```
<Axes: xlabel='day', ylabel='total_bill'>
```



Violin Plot

Syntax: sns.violinplot(x="x_values", y="y_values", data=dataframe)

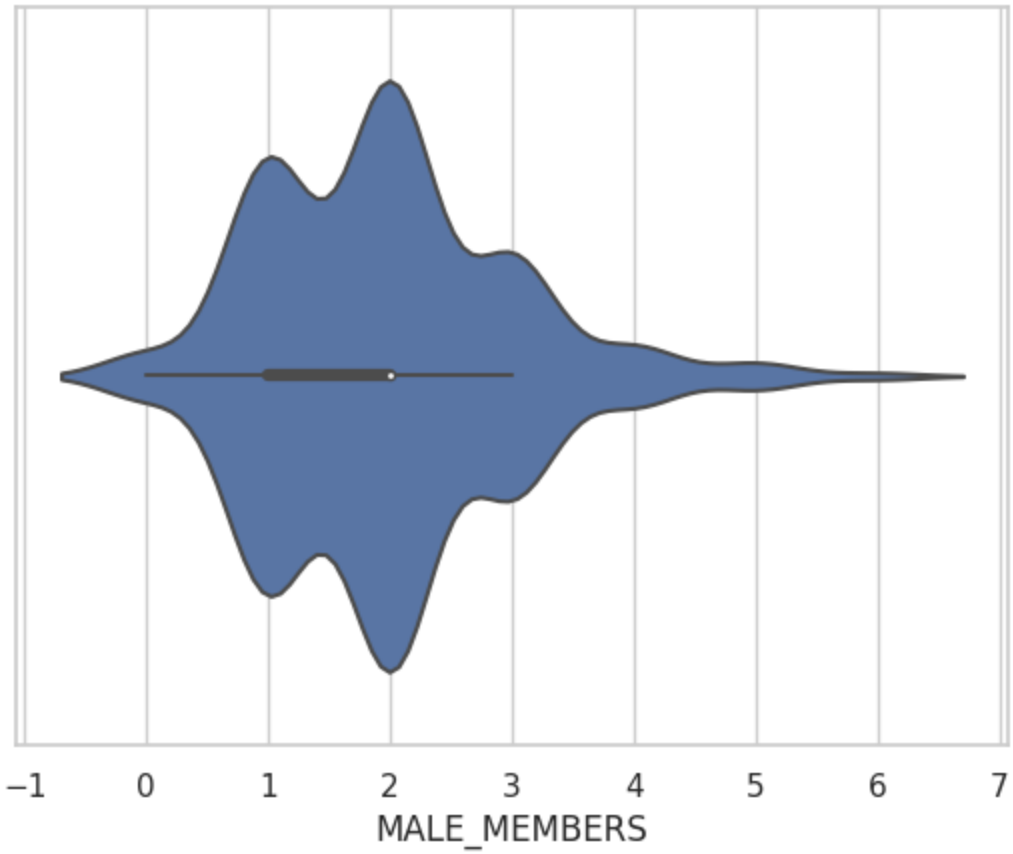
The `sns.violinplot()` function in `seaborn` has the following parameters:

1. **x, y:** These are inputs for plotting long-form data. They are names of variables in data or vector data.
2. **data:** This is the dataset for plotting. If x and y are absent, this is interpreted as wide-form.
3. **hue:** This is used to group variable that will produce violins with different colors.
4. **split:** When using hue nesting with a variable that takes two levels, setting split to True will draw half of a violin for each level. This can make it easier to directly compare the distributions.
5. **inner:** This can be either “box”, “quartile”, “point”, “stick”, or None. It represents what to draw inside the violin.
6. **bw:** This can be either ‘scott’, ‘silverman’, or a float. It is either the name of a reference rule or the scale factor to use when computing the kernel bandwidth.

7. **scale:** This can be either “area”, “count”, or “width”. It is the method used to scale the width of each violin.

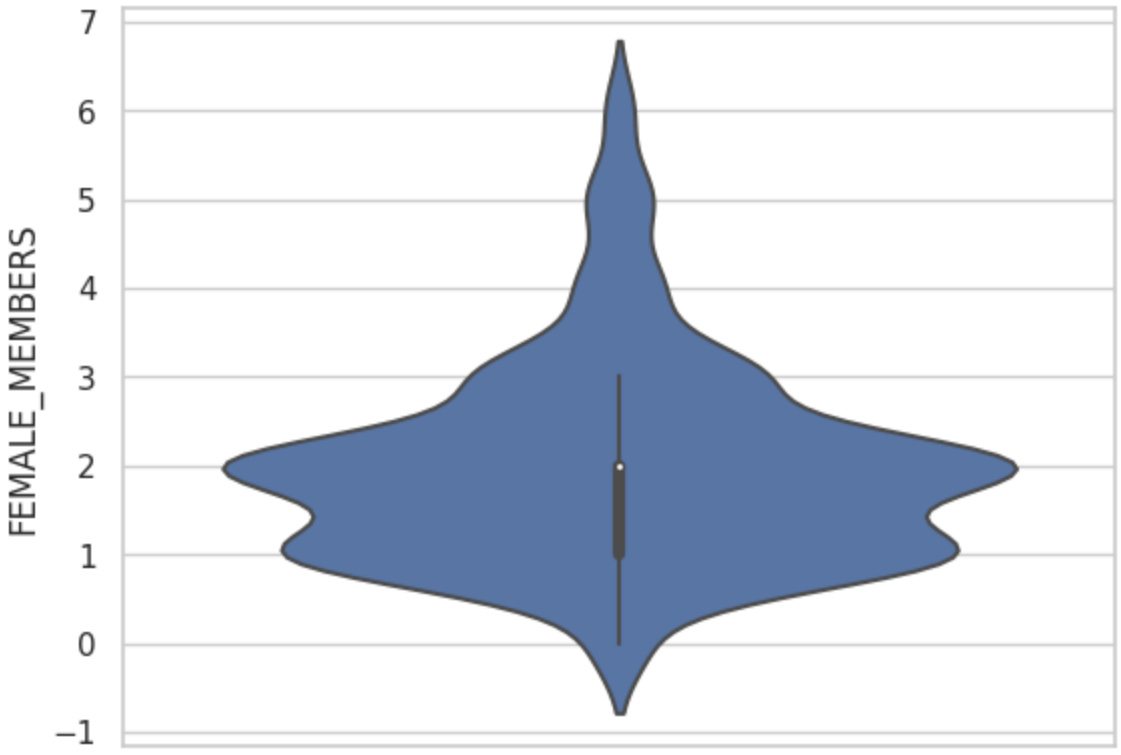
In [29]:
```
# Violin plot of a single column
sns.violinplot(x = file.MALE_MEMBERS)
```

Out[29]: <Axes: xlabel='MALE_MEMBERS'>



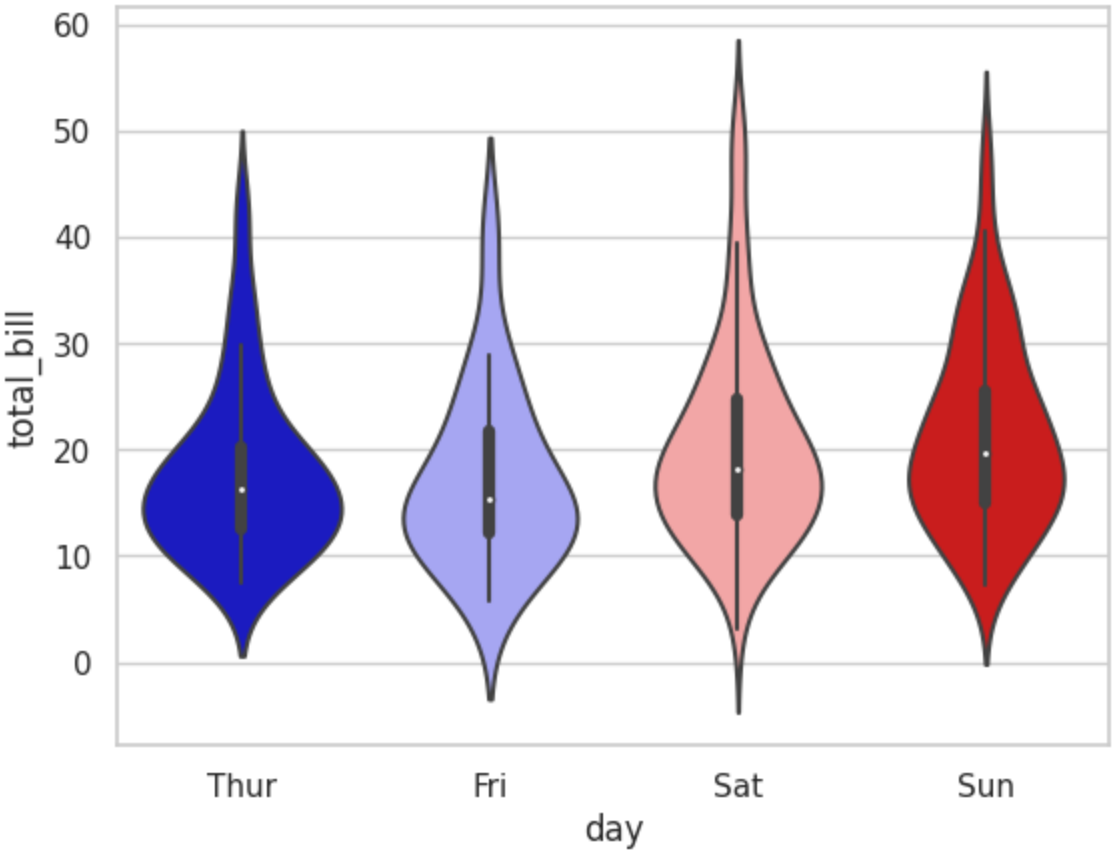
In [30]:
```
# Violin plot of a single column
sns.violinplot(y = file['FEMALE_MEMBERS'])
```

Out[30]: <Axes: ylabel='FEMALE_MEMBERS'>



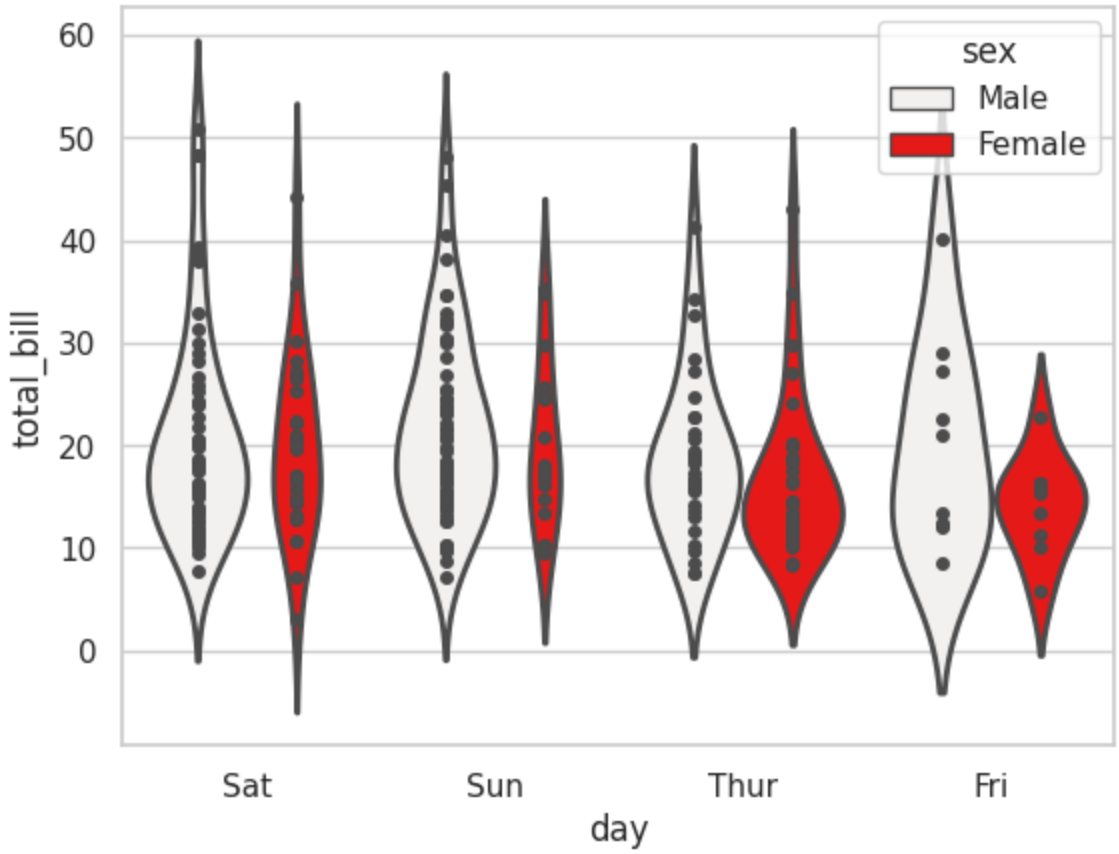
In [31]:
```
sns.set(style = 'whitegrid')
sns.violinplot(x = 'day', y = 'total_bill', data = df_tips, palette="seismic")
```

Out[31]: <Axes: xlabel='day', ylabel='total_bill'>



In [32]:
```
# Parameters used in violinplot
sns.set(style = 'whitegrid')
sns.violinplot(x = 'day', y = 'total_bill', data = df_tips,
               hue = 'sex',
               order = ['Sat', 'Sun', 'Thur', 'Fri'],
               orient = 'v', # By default it's 'v'. If we give 'h' it will throw error o
               showmeans = True,
               meanprops = {"marker": "+", "markeredgecolor": "black", "markersize": "10
               split = False,
               notch = True,
               width = 0.8,
               linewidth = 2,
               color = 'red',
               saturation = 0.8,
               inner = 'point',
               scale = 'count',
               bw = 'scott')
```

Out[32]: <Axes: xlabel='day', ylabel='total_bill'>



Heat Map Plot

Syntax: sns.heatmap(data=dataframe)

The `sns.heatmap()` function in `seaborn` has the following parameters:

1. **data:** 2D dataset that can be coerced into an ndarray.
2. **vmin, vmax:** Values to anchor the colormap, otherwise they are inferred from the data and other keyword arguments.
3. **cmap:** The mapping from data values to color space.
4. **center:** The value at which to center the colormap when plotting divergent data.
5. **annot:** If True, write the data value in each cell.
6. **annot_kwsdict:** Keyword arguments for matplotlib.axes.Axes.text() when annot is True.
7. **fmt:** String formatting code to use when adding annotations.

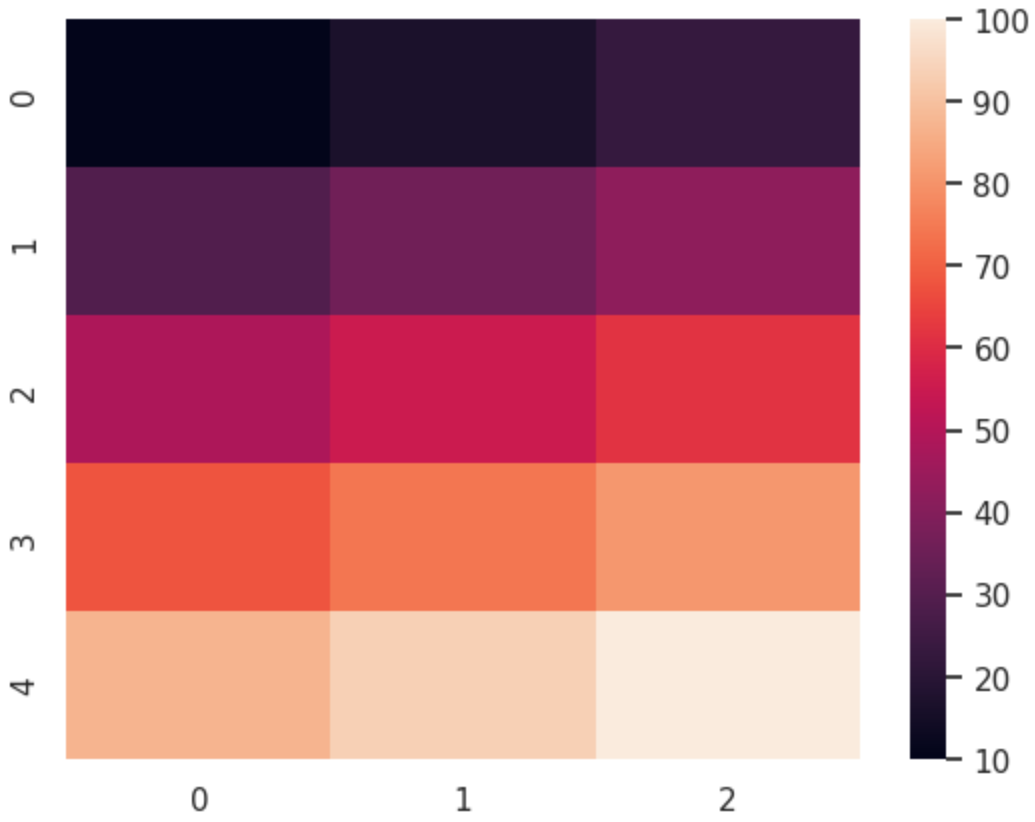
8. **linewidths:** Width of the lines that will divide each cell.
9. **linecolor:** Color of the lines that will divide each cell.
10. **cbar:** Whether to draw a colorbar.
11. **xticklabels, yticklabels:** If True, plot the column names of the dataframe. If False, don't plot the column names.

In [33]:
```
var_heat = np.linspace(10, 100, 15).reshape(5, 3)
var_heat
```

Out[33]:
```
array([[ 10.        ,  16.42857143,  22.85714286],
       [ 29.28571429,  35.71428571,  42.14285714],
       [ 48.57142857,  55.        ,  61.42857143],
       [ 67.85714286,  74.28571429,  80.71428571],
       [ 87.14285714,  93.57142857, 100.        ]])
```

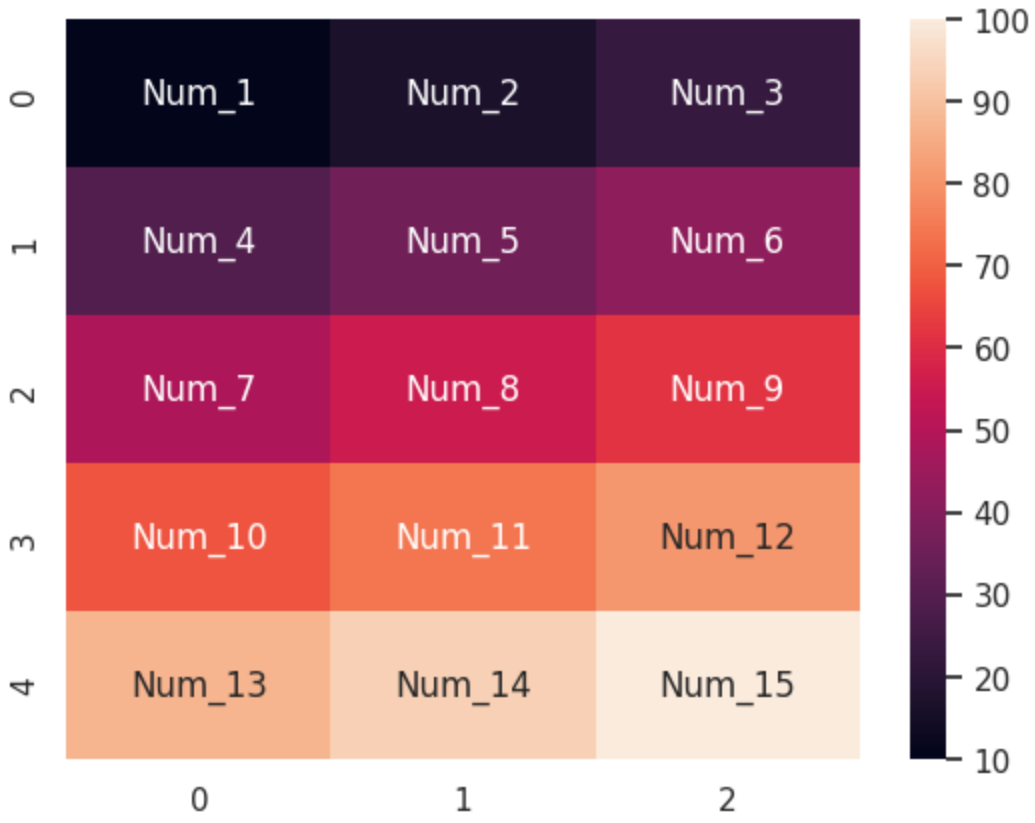
In [34]:
```
sns.heatmap(var_heat)
```

Out[34]: <Axes: >



In [35]:
```
name_change = np.array([['Num_1','Num_2','Num_3'],
                        ['Num_4','Num_5','Num_6'],
                        ['Num_7','Num_8','Num_9'],
                        ['Num_10','Num_11','Num_12'],
                        ['Num_13','Num_14','Num_15']])

sns.heatmap(var_heat,
            annot = name_change, fmt = 's')
```

Out[35]: <Axes: >



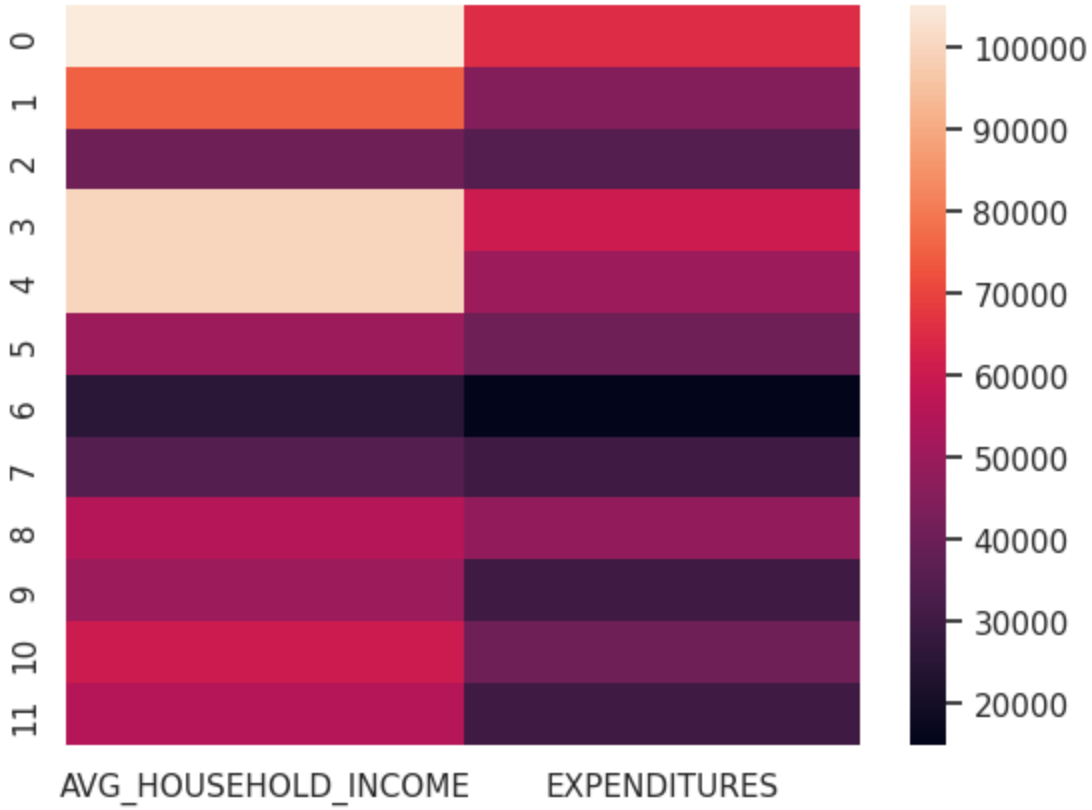
In [36]:
```
file_12 = pd.read_csv('/content/drive/MyDrive/ML and DL DataSets/3.2_NSCA_DC Matplotl

file_12
```

Out[36]:

	AVG_HOUSEHOLD_INCOME	EXPENDITURES
0	105000	65000
1	75000	45000
2	40000	35000
3	100000	60000
4	100000	50000
5	50000	40000
6	25000	15000
7	35000	30000
8	55000	48000
9	50000	30000
10	60000	40000
11	55000	30000

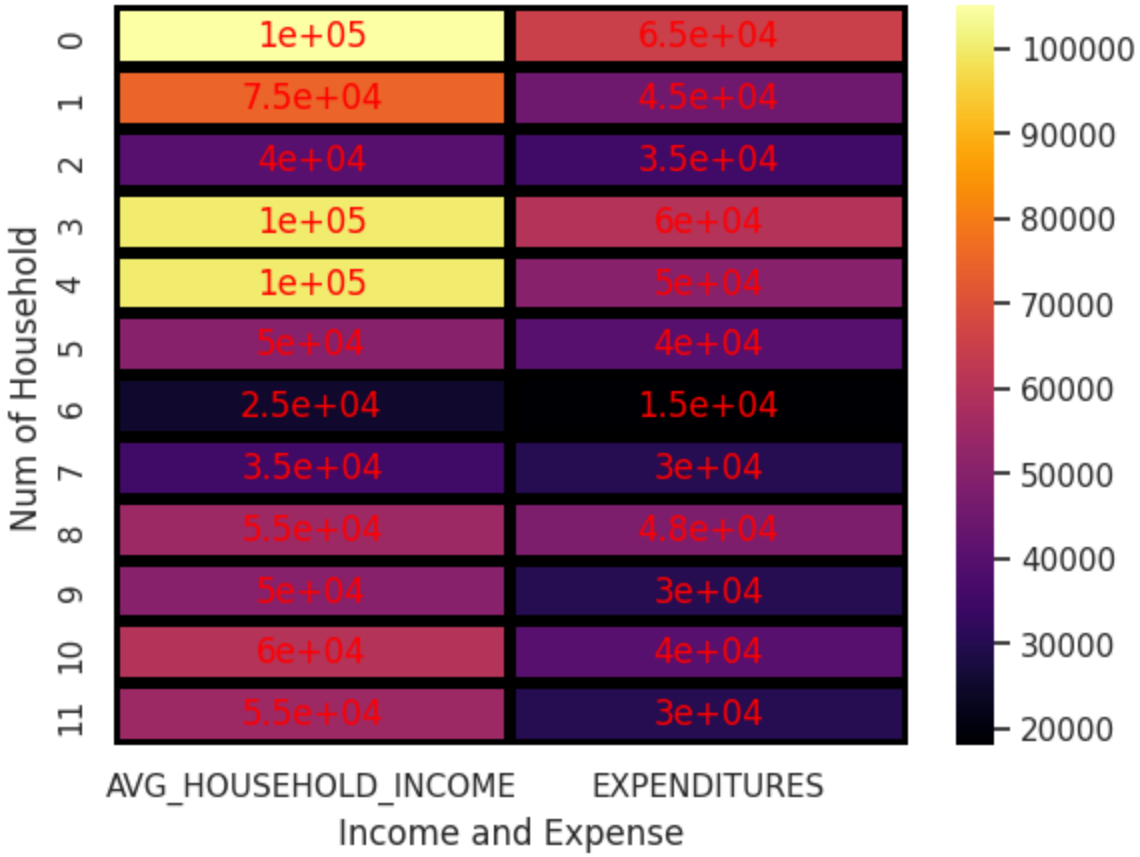
In [37]:
```
# Analysis of income and expenditure of first 12 households
sns.heatmap(file_12)
```

Out[37]: <Axes: >



In [38]:
```
# Parameters used in heatmap
par = sns.heatmap(file_12,
            vmin = 18000, vmax = 105000,
            cmap = 'inferno',
            annot = True,
            annot_kws = {'fontsize': 12, 'color': 'red'},
            linewidth = 4,
            linecolor = 'black',
            cbar = True, # By default it's true. False for remove colorbar
            xticklabels = True, # By default it's true. False for remove x labels
            yticklabels = True) # By default it's true. False for remove y labels

par.set(xlabel = 'Income and Expense', ylabel = 'Num of Household')
# sns.set(font_scale = 4) # To increase the font scale
```

Out[38]:
```
[Text(0.5, 17.846874999999997, 'Income and Expense'),
 Text(46.25, 0.5, 'Num of Household')]
```



Point Plot

Syntax: sns.pointplot(x="x_values", y="y_values", data=dataframe)

The `sns.pointplot()` function in `seaborn` has the following parameters:

1. **x,y:** These parameters take names of variables as input that plot the long form data.
2. **data:** This is the dataframe that is used to plot graphs.
3. **hue:** Names of variables in the dataframe that are needed for plotting the graph.
4. **linestyles:** Line styles to use for each of the hue levels.
5. **dodge:** This parameter takes a boolean value. if we use hue nesting, passing true to this parameter will separate the strips for different hue levels. If False is passed, the points for each level will be plotted on top of each other.
6. **orient:** It takes values “h” or “v” and the orientation of the graph is determined based on this.
7. **color:** matplotlib color is taken as input and this determines the color of all the elements.
8. **palette:** This parameter specifies the colors for different hue mappings.

9. **join:** Takes boolean values and if True, lines between point estimates will be drawn at the same hue level.

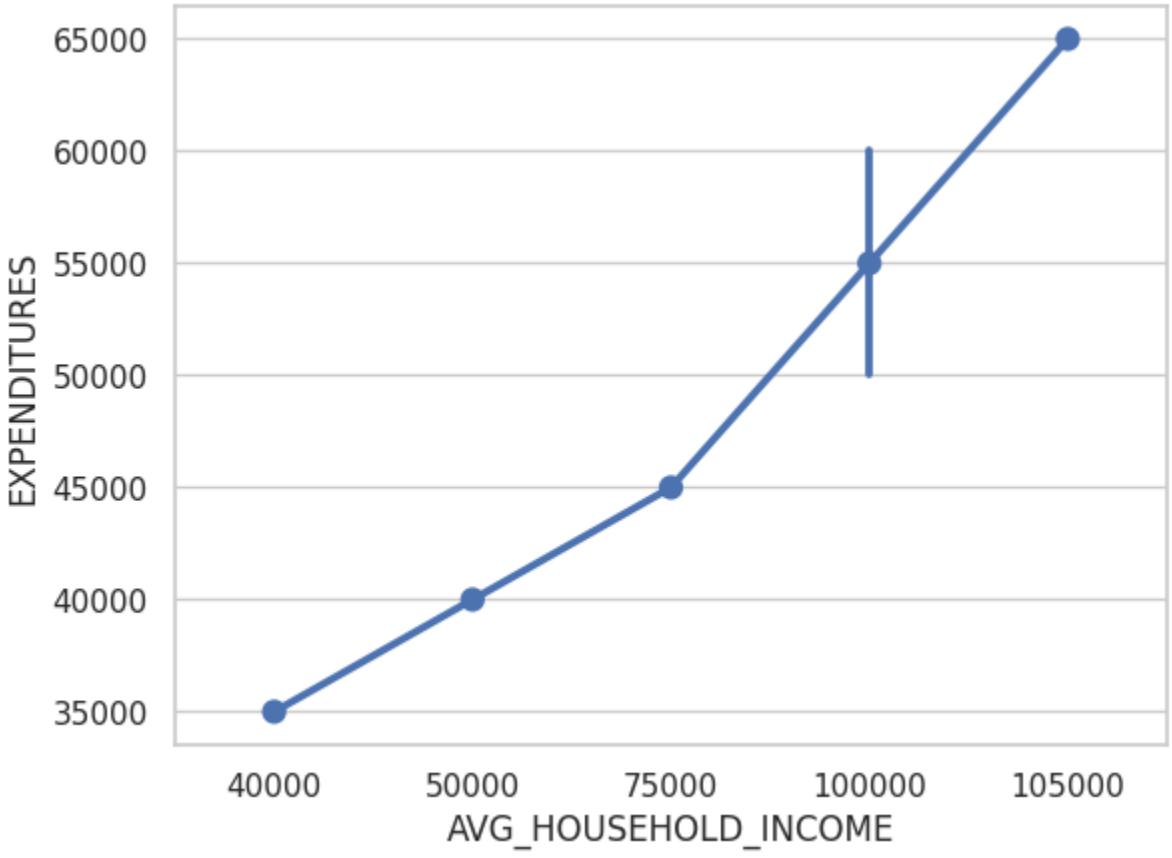
In [39]:
```
# Taking the first 6 datasets for analysis
file_6 = pd.read_csv('/content/drive/MyDrive/ML and DL DataSets/3.2_NSCA_DC Matplotli
                     usecols = ['AVG_HOUSEHOLD_INCOME', 'EXPENDITURES', 'NUMBER_OF_M

file_6
```

Out[39]:

	NUMBER_OF_MEMBERS_IN_HOUSEHOLD	AVG_HOUSEHOLD_INCOME	EXPENDITURES
0	8	105000	65000
1	3	75000	45000
2	3	40000	35000
3	10	100000	60000
4	4	100000	50000
5	3	50000	40000

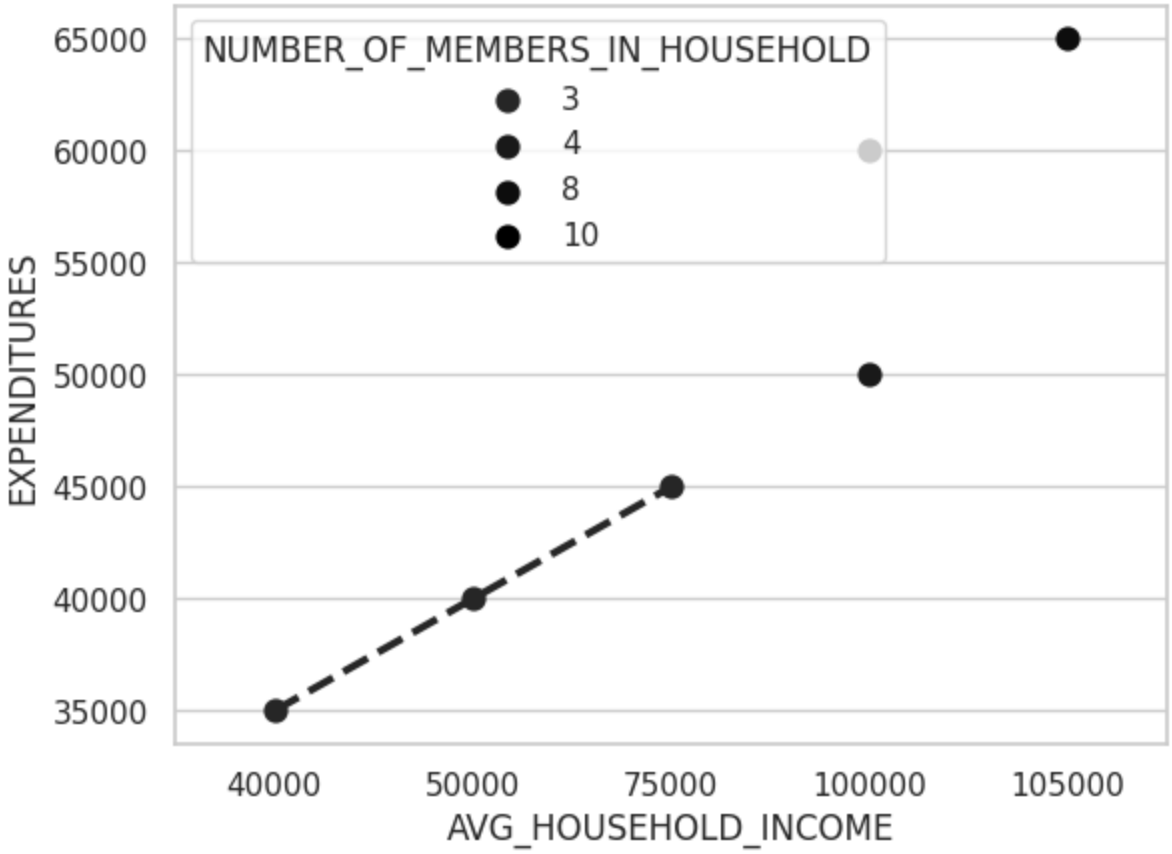
In [40]:
```
# Income and Expenditure in Point plot
sns.pointplot(x= 'AVG_HOUSEHOLD_INCOME',  y= 'EXPENDITURES', data= file_6)
```

Out[40]:
```
<Axes: xlabel='AVG_HOUSEHOLD_INCOME', ylabel='EXPENDITURES'>
```



In [41]:
```
# Using the parameters
sns.pointplot(x= 'AVG_HOUSEHOLD_INCOME',  y= 'EXPENDITURES', data= file_6,
              hue = 'NUMBER_OF_MEMBERS_IN_HOUSEHOLD',
              linestyles = 'dashed',
              color = 'black',
              join = True) # By default is True
```

Out[41]: <Axes: xlabel='AVG_HOUSEHOLD_INCOME', ylabel='EXPENDITURES'>



Count Plot

Syntax: sns.countplot(x="x_values", data=dataframe)

Count Plot vs Bar Plot

Countplot: A categorical plot that displays the count of observations for each category.

Barplot: A numerical plot that displays the mean value of a numerical variable for each category.

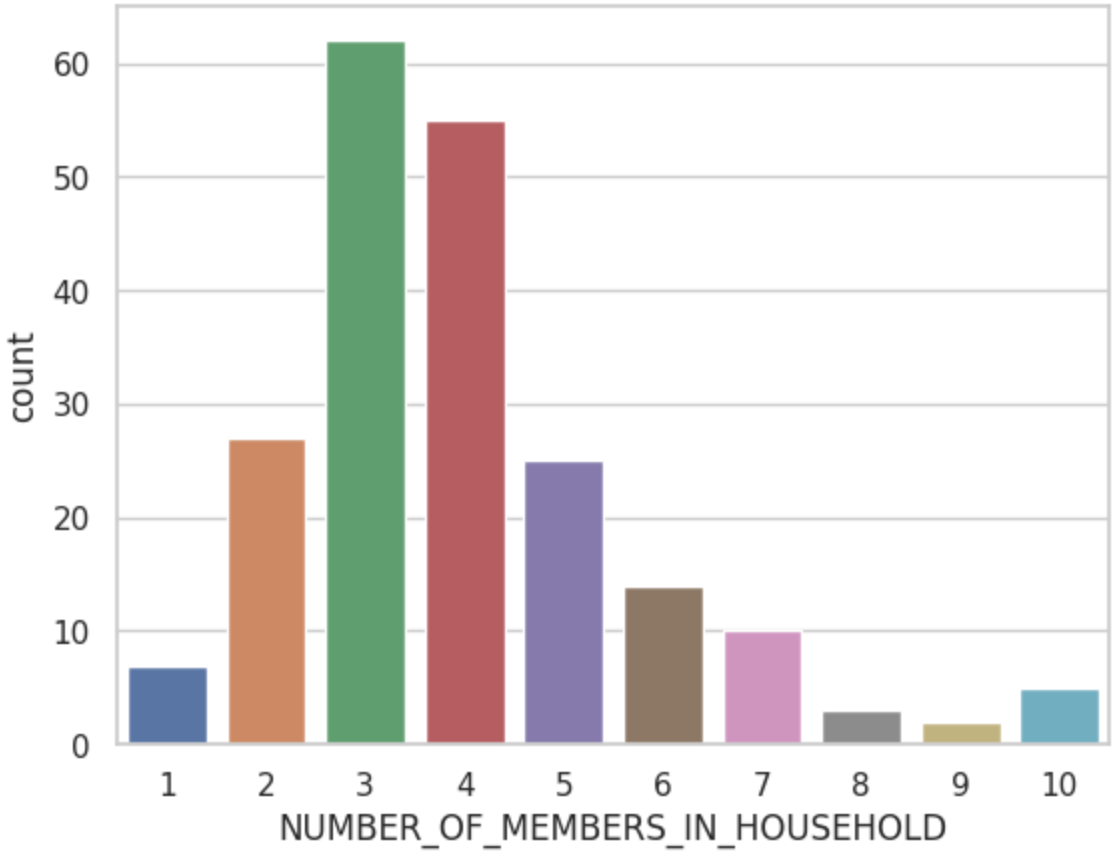
The `sns.countplot()` function in `seaborn` has the following parameters:

1. **x,y:** These parameters take names of variables as input that plot the long form data.
2. **data:** This is the dataframe that is used to plot graphs.
3. **hue:** Names of variables in the dataframe that are needed for plotting the graph.
4. **linewidth:** This parameter takes floating values and determines the width of the gray lines that frame the elements in the plot.
5. **dodge:** This parameter takes a Boolean value. if we use hue nesting, passing true to this parameter will separate the strips for different hue levels. If False is passed, the points for each level will be plotted on top of each other.
6. **orient:** It takes values “h” or “v” and the orientation of the graph is determined based on this.
7. **color:** matplotlib color is taken as input and this determines the color of all the elements.
8. **palette:** This parameter specifies the colors for different hue mappings.

9. **saturation:** Takes a floating value and the proportion of the original saturation to draw colors is

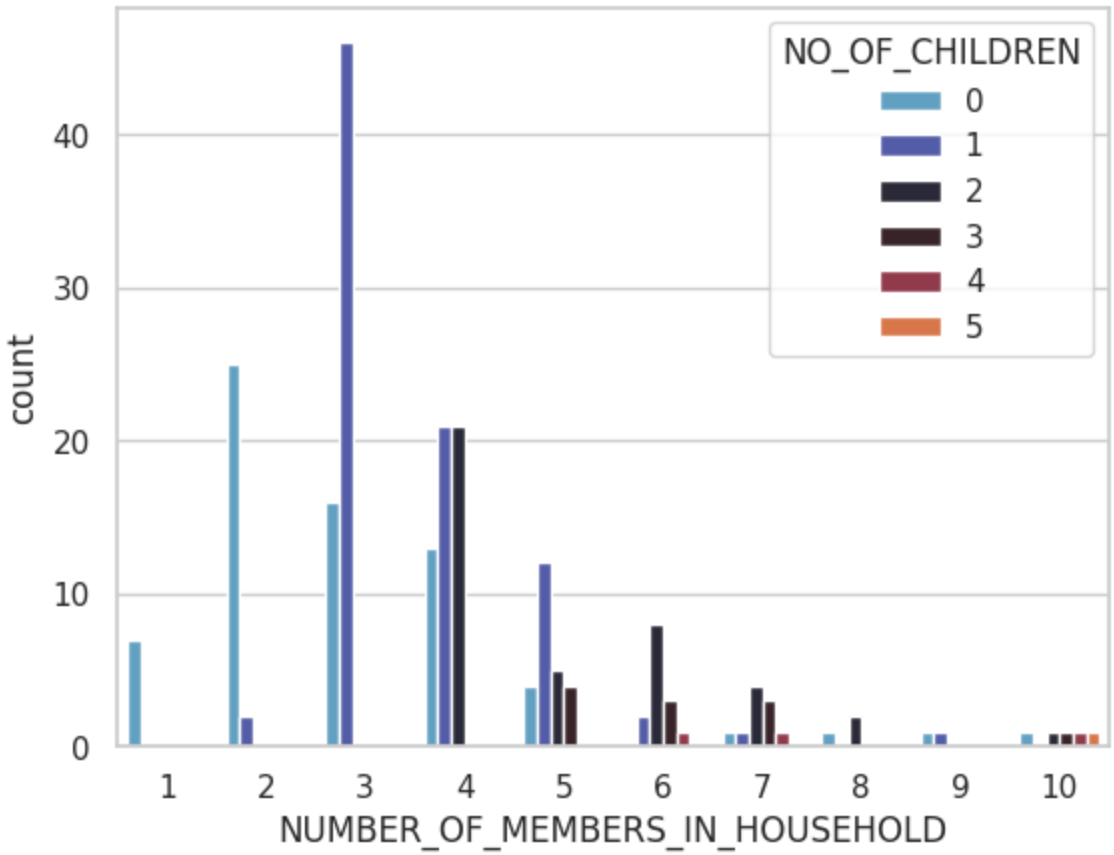
In [42]:
```
# Counting number of members in household

sns.countplot(x= file['NUMBER_OF_MEMBERS_IN_HOUSEHOLD'])
```

Out[42]: <Axes: xlabel='NUMBER_OF_MEMBERS_IN_HOUSEHOLD', ylabel='count'>



In [43]:
```
sns.countplot(x = 'NUMBER_OF_MEMBERS_IN_HOUSEHOLD', data = file,
              hue = 'NO_OF_CHILDREN',
              orient = 'h',
              palette = 'icefire',
              saturation = 0.8)
```

Out[43]: <Axes: xlabel='NUMBER_OF_MEMBERS_IN_HOUSEHOLD', ylabel='count'>



Strip Plot

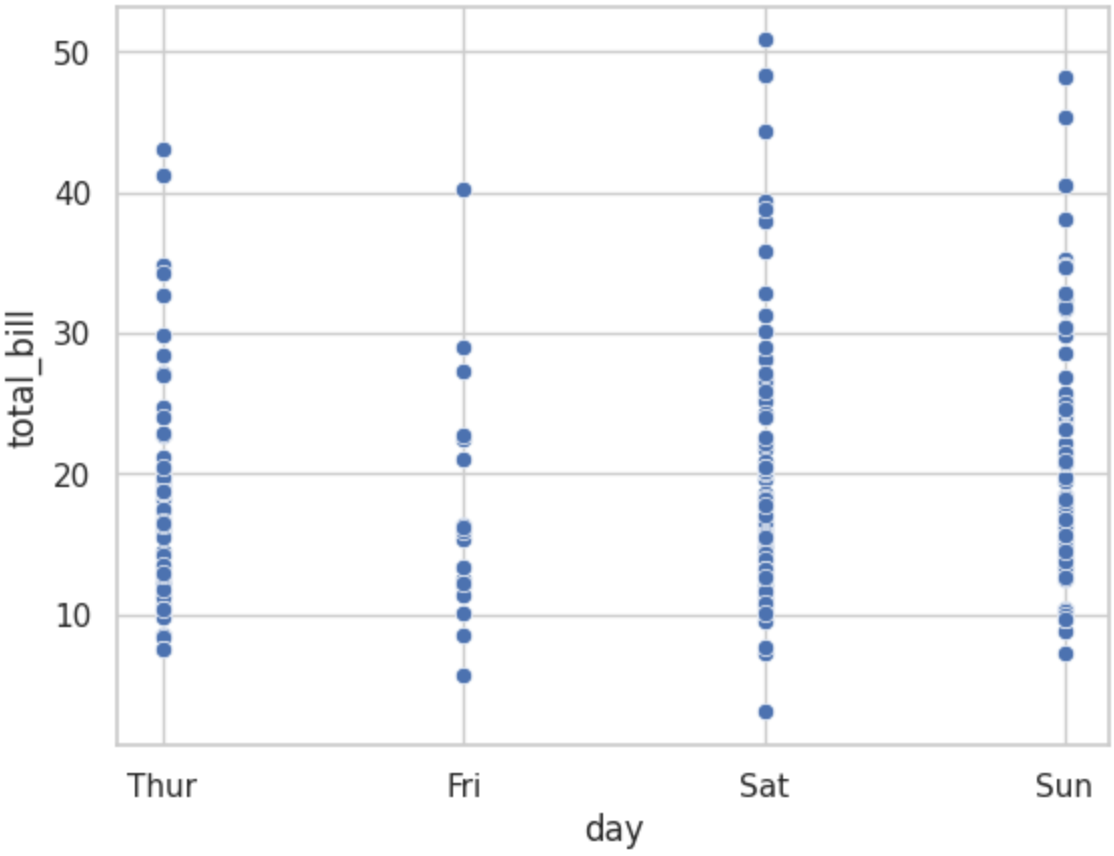
Syntax: sns.stripplot(x="x_values", y="y_values", data=dataframe)

The `sns.stripplot()` function in `seaborn` has the following parameters:

1. **x:** The name of the column to use as the x-axis.
2. **y:** The name of the column to use as the y-axis.
3. **hue:** The name of the column to use for coloring the points.
4. **data:** The Pandas DataFrame to plot.
5. **palette:** The name of the color palette to use for coloring the points.
6. **size:** The size of the points.
7. **marker:** Used to plot the data points.

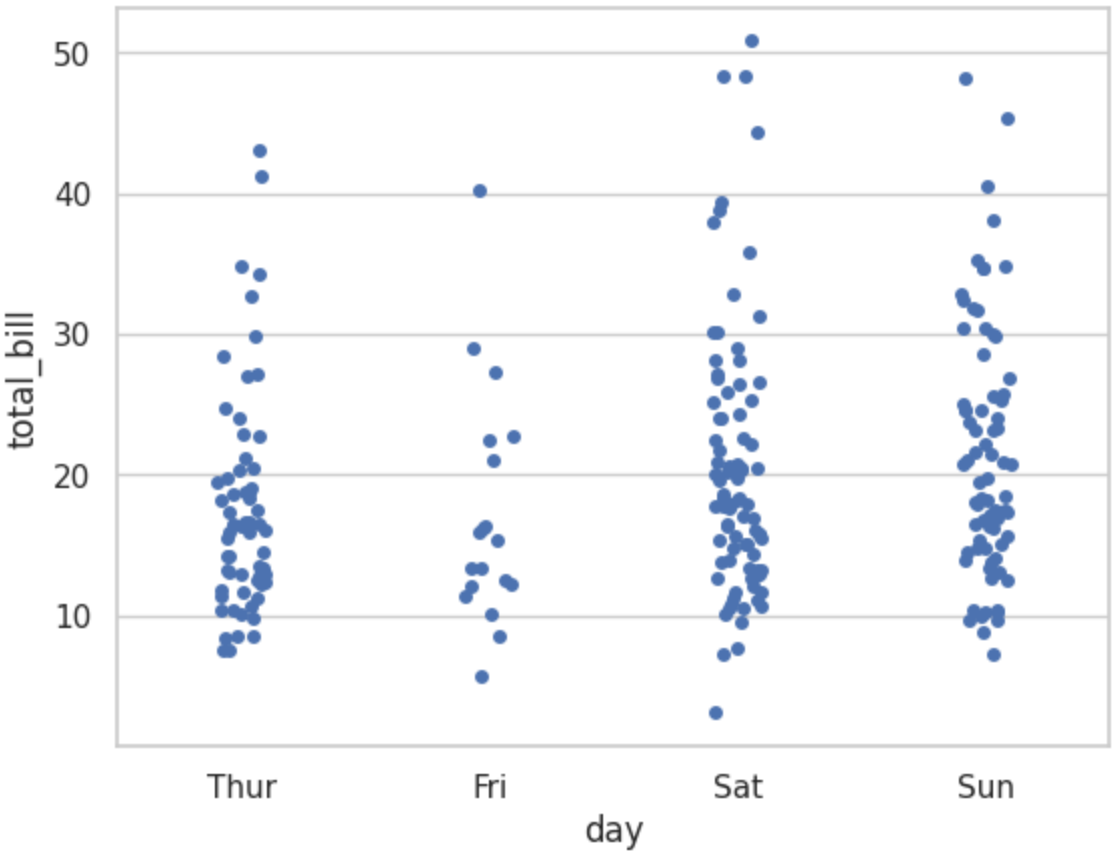
In [44]:
```
# Analyzing total_bill generated in a particular day through scatter plot
sns.scatterplot(x = 'day', y = 'total_bill', data = df_tips)
```

Out[44]: <Axes: xlabel='day', ylabel='total_bill'>



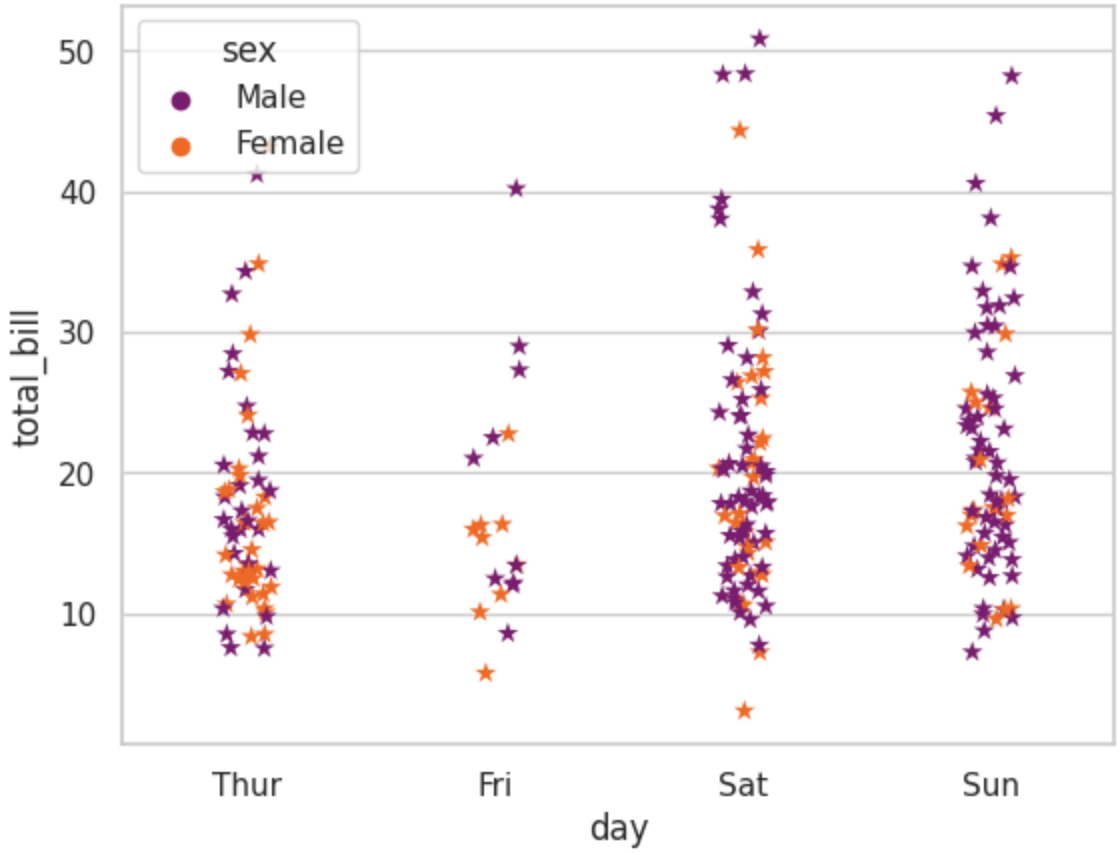
In [45]:
```
# Analyzing total_bill generated in a particular day through strip plot
sns.stripplot(x = 'day', y = 'total_bill', data = df_tips)
```

Out[45]: <Axes: xlabel='day', ylabel='total_bill'>



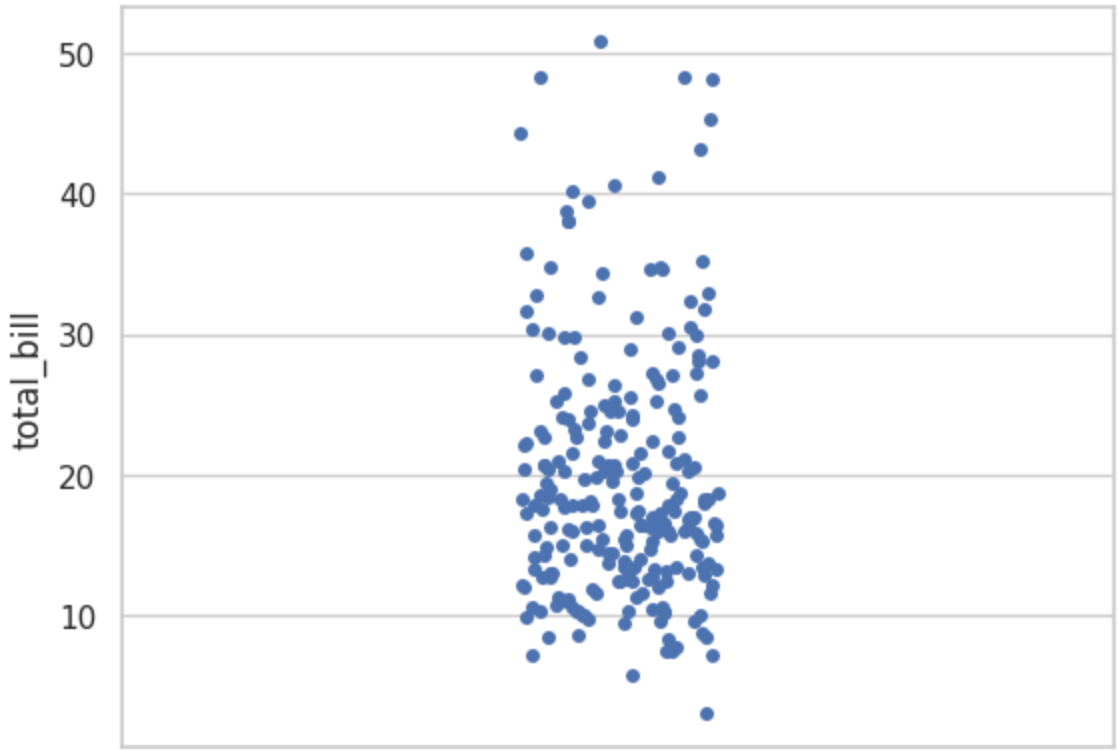
In [46]:
```
# Using the parameters
sns.stripplot(x = 'day', y = 'total_bill', data = df_tips,
              hue = 'sex',
              palette = 'inferno',
              size = 8,
              marker = '*')
```

Out[46]: <Axes: xlabel='day', ylabel='total_bill'>



In [47]:
```
# Strip plot of a single data
sns.stripplot(df_tips['total_bill'])
```

Out[47]: <Axes: ylabel='total_bill'>



Density (KDE - Kernal Density Estimate) Plot:

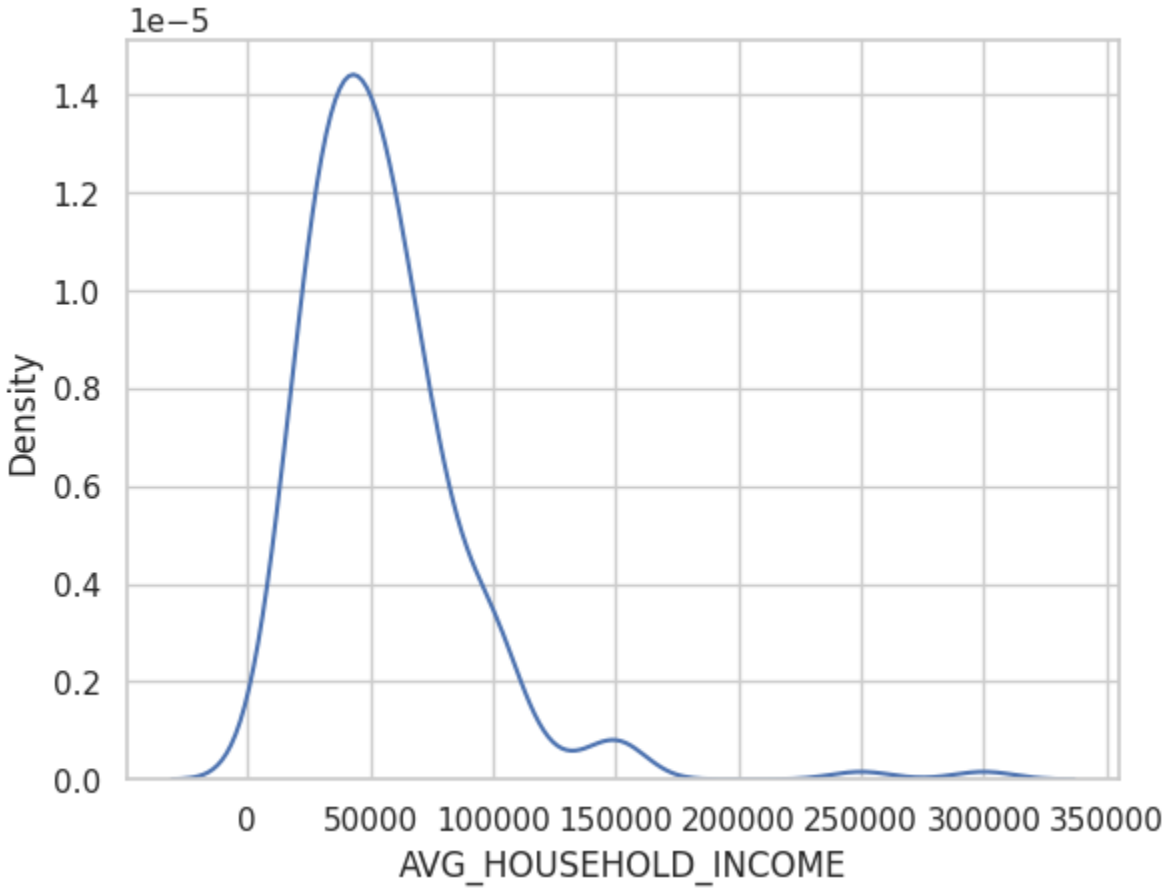
Syntax: sns.kdeplot(data=dataframe, x="column_name")

The `sns.kdeplot()` function in `seaborn` has the following parameters:

1. **x and y:** These are the variables to be plotted. They can be either single variables or a list of variables.
2. **palette:** This parameter specifies the color palette to be used for the plot. The default is None, which means that the default color palette is used.
3. **fill:** This parameter determines whether the plot is filled or not. The default is False, which means that the plot is not shaded.
4. **cbar:** This parameter determines whether a colorbar is displayed or not. The default is False, which means that a colorbar is not displayed.
5. **ax:** This parameter specifies the axes to be used for the plot. The default is None, which means that the current axes are used.

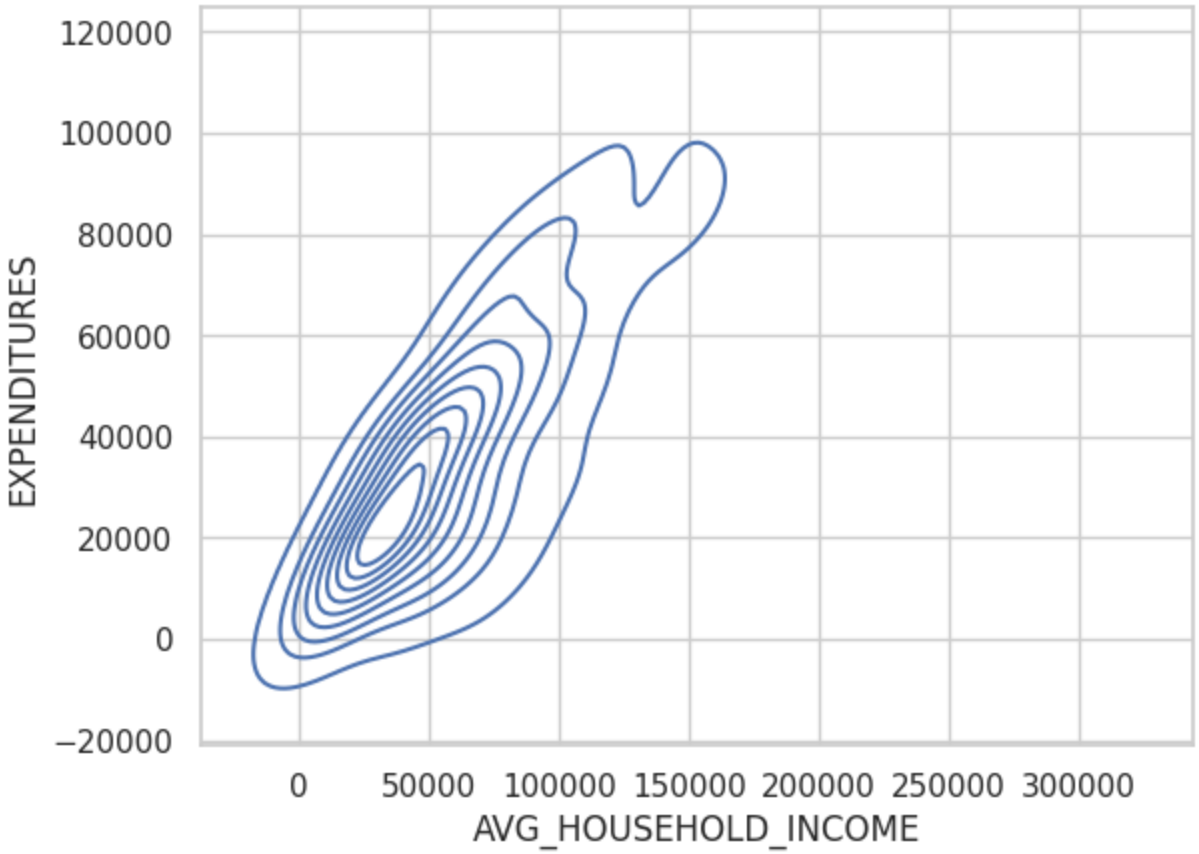
In [48]:
```
# Analysing the average household income (Single column)
sns.kdeplot(x = 'AVG_HOUSEHOLD_INCOME', data = file)
```

Out[48]:
```
<Axes: xlabel='AVG_HOUSEHOLD_INCOME', ylabel='Density'>
```



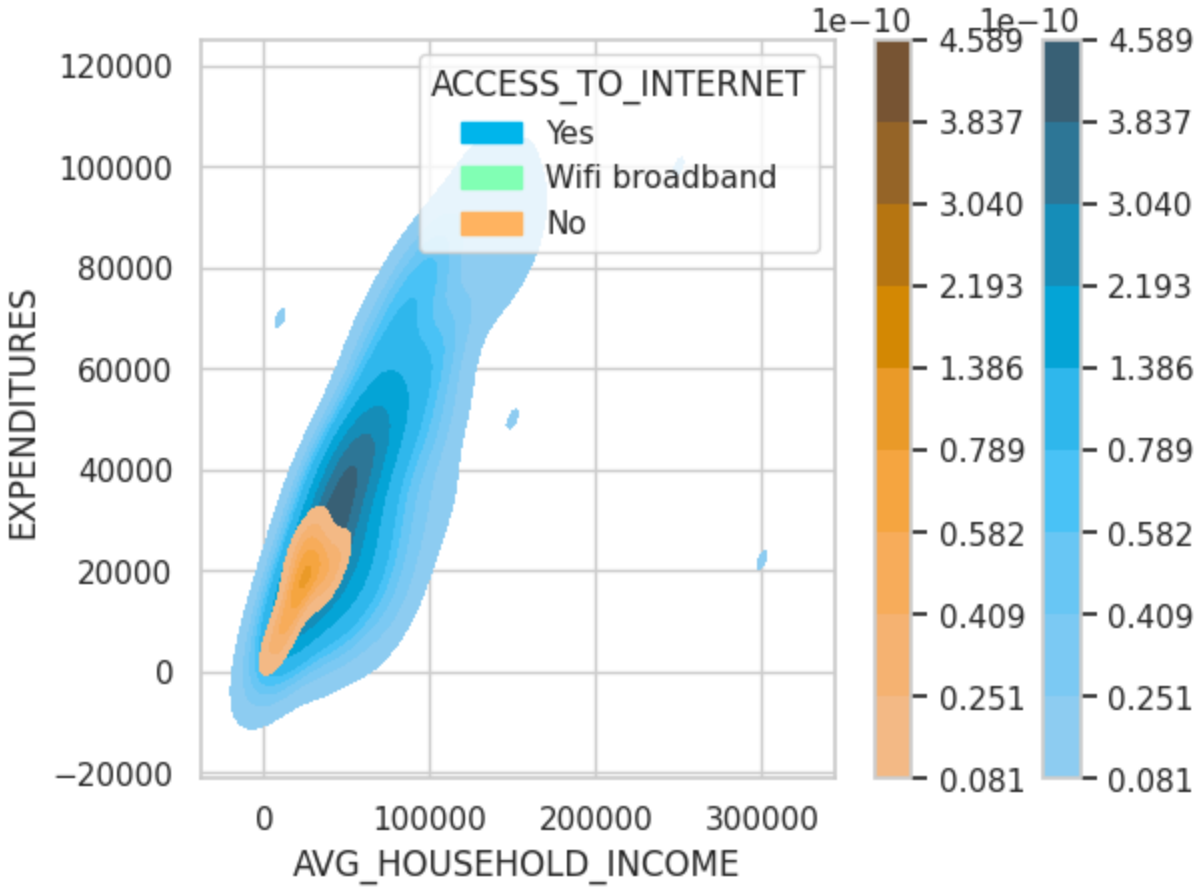
In [49]:
```
# Analysing the average household income and expenditures (Multiple column)
sns.kdeplot(x = 'AVG_HOUSEHOLD_INCOME', y = 'EXPENDITURES', data = file)
```

Out[49]: <Axes: xlabel='AVG_HOUSEHOLD_INCOME', ylabel='EXPENDITURES'>



In [50]:
```
# Using the parameters
sns.kdeplot(x = 'AVG_HOUSEHOLD_INCOME', y = 'EXPENDITURES', data = file,
            hue = 'ACCESS_TO_INTERNET',
            palette = 'rainbow',
            fill = True, # Default: False
            cbar = True, # Default: False
            warn_singular=False) # If it's True, python will show this warning: "KDE
```

Out[50]:
```
<Axes: xlabel='AVG_HOUSEHOLD_INCOME', ylabel='EXPENDITURES'>
```



Joint Plot

Syntax: sns.jointplot(x="x_values", y="y_values", data=dataframe)

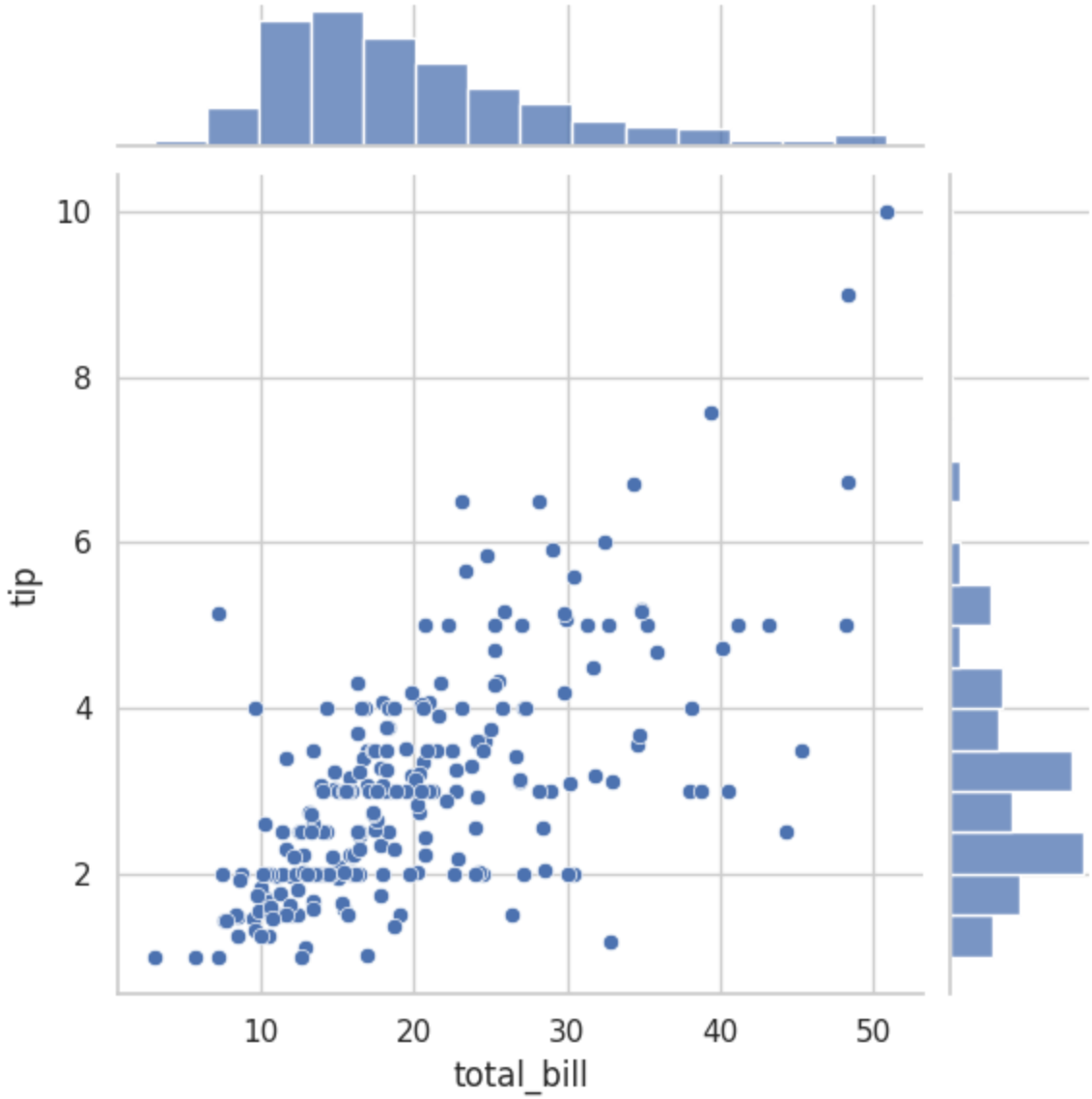
The `sns.jointplot()` function in `seaborn` has the following parameters:

1. **x, y:** These are the names of variables in data, or can be directly provided as data.
2. **data:** This is an optional parameter that takes a DataFrame when x and y are variable names.
3. **kind:** This is the kind of plot to draw. The options are “scatter” | “kde” | “hist” | “hex” | “reg” | “resid”. The default is ‘scatter’.
4. **color:** This is an optional parameter that specifies the color used for the plot elements.
5. **dropna:** This is a boolean value. If True, it removes observations that are missing from x and y.

Other parameters include: **hue, height, ratio, space, xlim, ylim,** etc

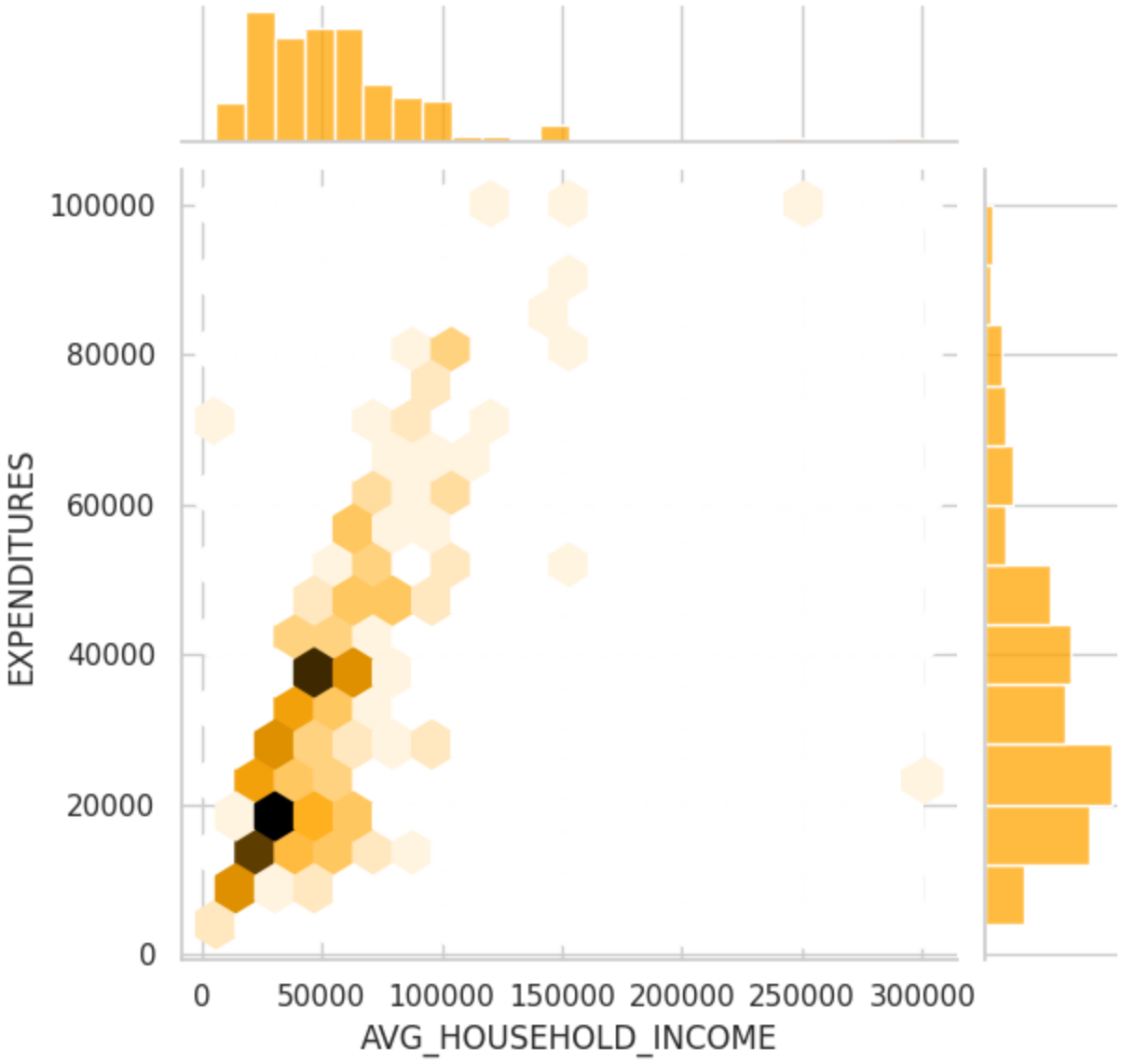
In [51]:
```
# Finding relation between total bill and tip
sns.jointplot(x = 'total_bill', y = 'tip', data = df_tips)
```

Out[51]: <seaborn.axisgrid.JointGrid at 0x7a5e09f3ecb0>



In [52]:
```
# Analysing the average household income and expenditures through joint plot
sns.jointplot(x = 'AVG_HOUSEHOLD_INCOME', y = 'EXPENDITURES',data = file,
              kind = 'hex', # Default: Scatter. Can use “scatter” | “kde” | “hist” |
              dropna = True,
              color = 'orange')
```

Out[52]: <seaborn.axisgrid.JointGrid at 0x7a5e09da9660>



Pair Plot

Syntax: sns.pairplot(data=dataframe)

The `sns.jointplot()` function in `seaborn` has the following parameters:

1. **data:** A Pandas DataFrame containing the data to be plotted.
2. **hue:** The name of the column in data to use to color the plots.
3. **diag_kind:** The type of plot to use for the diagonal plots. Supported values include "hist" for a histogram, "kde" for a kernel density estimate, and "scatter" for a scatter plot.
4. **kind:** The type of plot to use for the off-diagonal plots. Supported values include "scatter", "reg", and "hex".
5. **vars:** To set the limited var, for visualization
6. **height:** Takes a scalar value and determines the height of the facet.
7. `**aspect:** Takes scalar value and Aspect ratio of each facet, so that as pect * height gives the width of each facet in inches.`

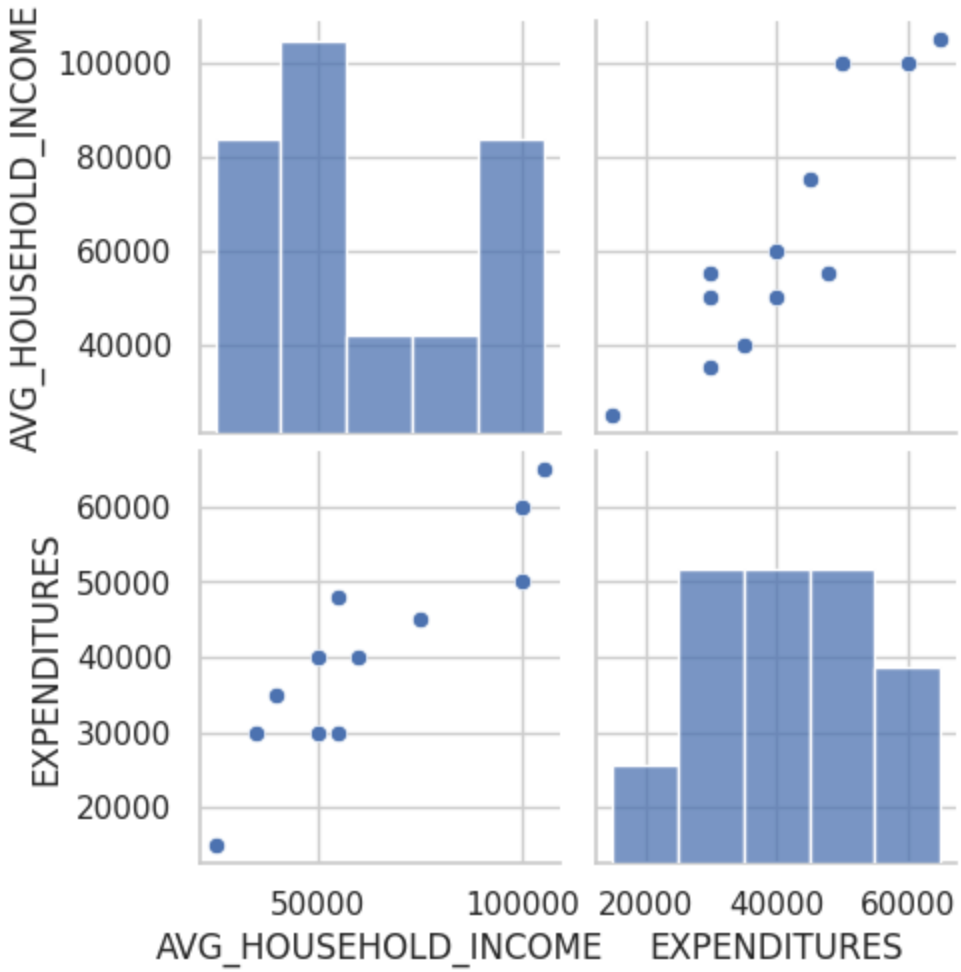
8. **corner:** Takes Boolean value and If True, don't add axes to the upper (off-diagonal) triangle of the grid, making this a "corner" plot.

9. **hue_order:** Takes lists as input and Order for the levels of the facet ing variables is determined by this order.

10. **markers:** A list of markers to use for the off-diagonal plots.

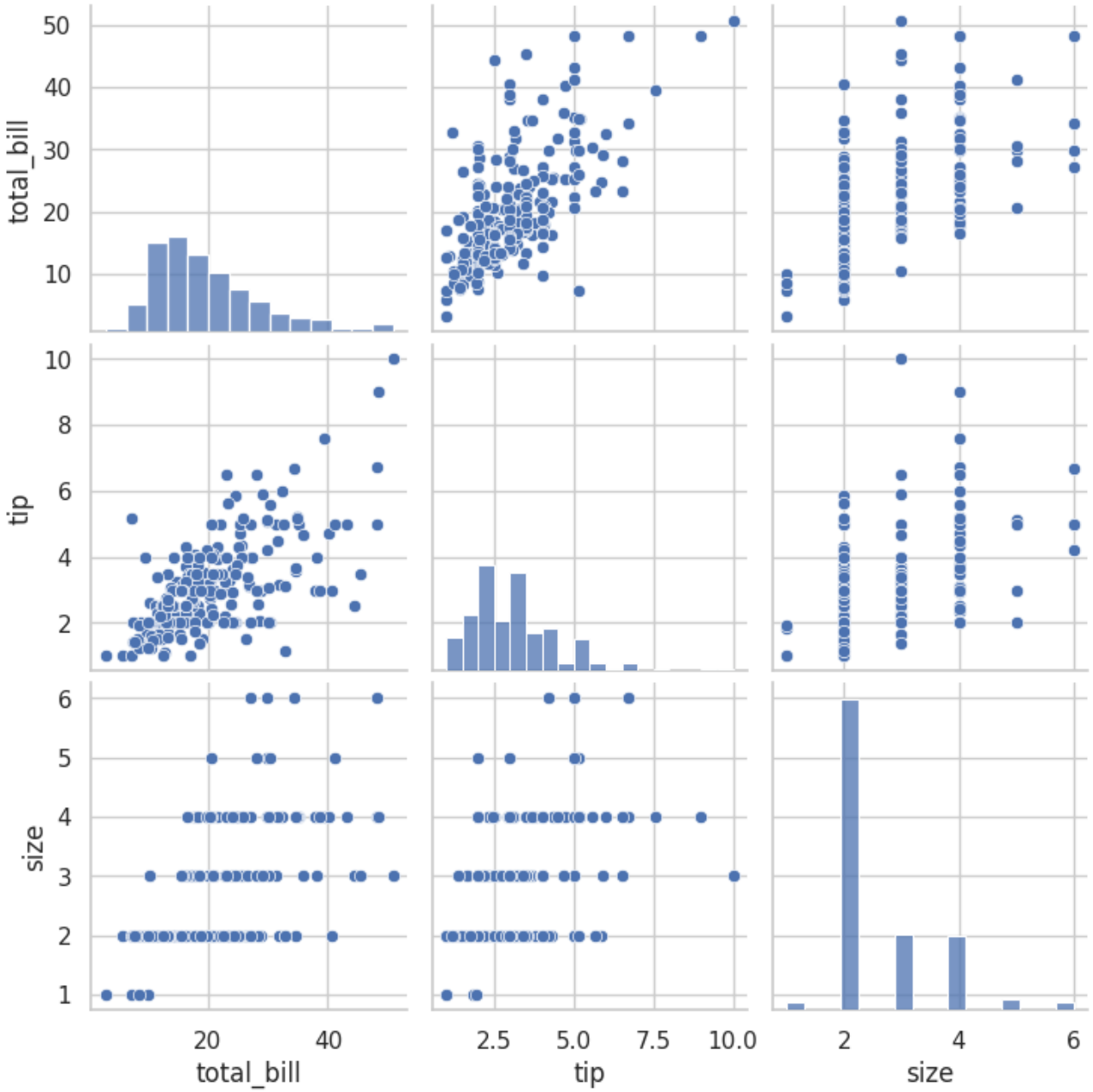
In [53]:
```
# Average household income and expenditure analysis with 12 datasets
sns.pairplot(file_12)
```

Out[53]:
```
<seaborn.axisgrid.PairGrid at 0x7a5e09686ef0>
```



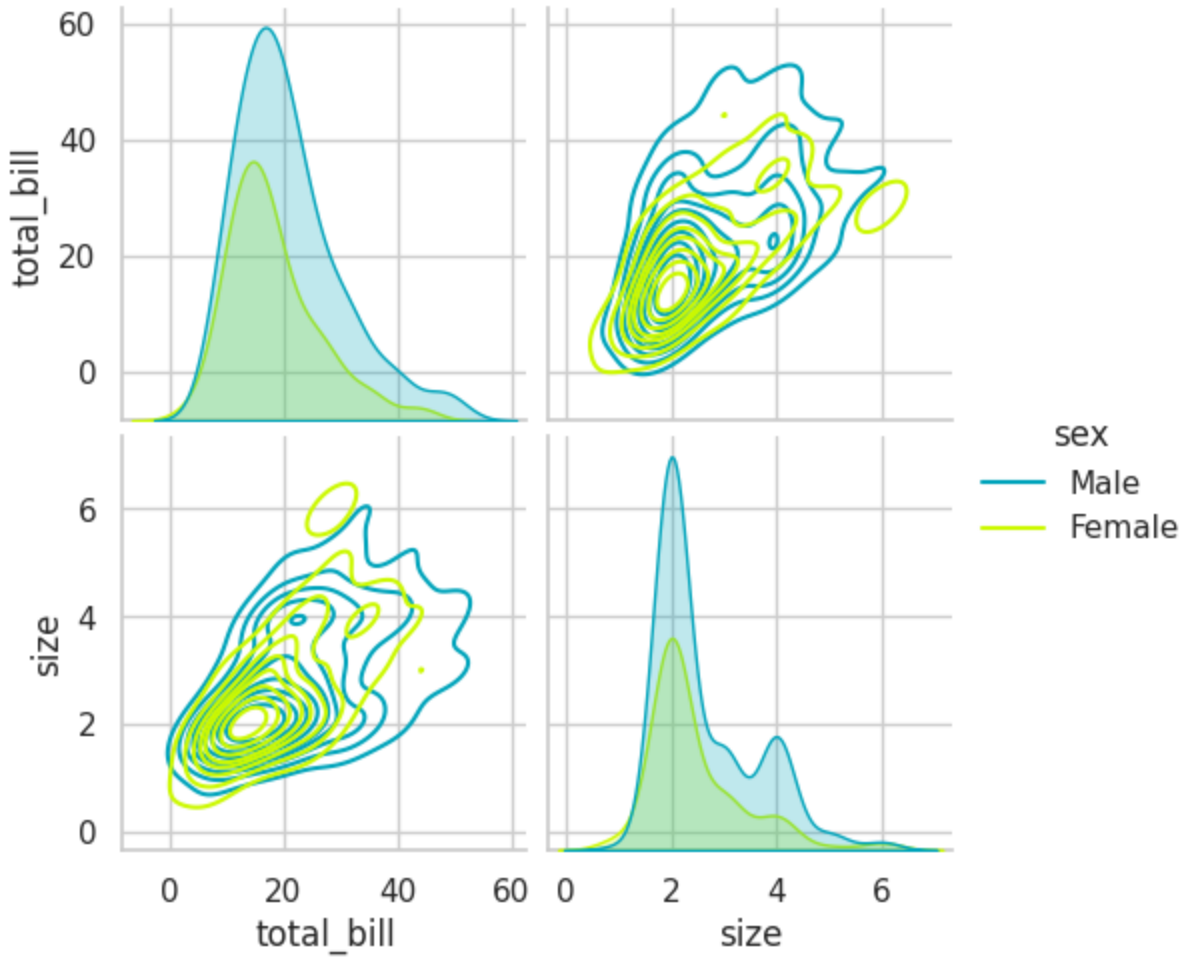
In [54]:
```
# Analysis of tips dataset
sns.pairplot(df_tips) # It will give the visualization of numerical value
```

Out[54]: <seaborn.axisgrid.PairGrid at 0x7a5e0945f460>



In [55]:
```
# Using the parameters, based on (sex)
sns.pairplot(df_tips, hue = 'sex',
             vars = ['total_bill', 'size'],
             kind = 'kde',
             palette = 'nipy_spectral')
```

Out[55]: <seaborn.axisgrid.PairGrid at 0x7a5e08bd5690>



Factor (Cat) Plot

Syntax: sns.catplot(x="x", y="y", data=df)

The `sns.catplot()` function in `seaborn` has the following parameters:

1. **x,y:** These parameters take names of variables as input that plot the long form data.
2. **data:** This is the dataframe that is used to plot graphs.
3. **hue:** Names of variables in the dataframe that are needed for plotting the graph.
4. **kind:** This parameter takes different values such as swarm, strip etc and the nature of the graph depends on this.
5. **height:** These parameter takes scalar values and takes the height of the plot in inches.
6. **aspect:** This parameter takes a scalar value. aspect*height will give the width of the plot.
7. **orient:** It takes values “h” or “v” and the orientation of the graph is determined based on this.
8. **row, col:** These are categorical variables from the dataframe that are to be plotted.

Categorical scatterplots with catplot

- stripplot() – with kind="strip"
- swarmplot() – with kind="swarm"

Categorical estimate plots with catplot

- pointplot() – with kind="point"
- barplot() – with kind="bar"
- countplot() – with kind="count"

Categorical distribution plots with catplot

- boxplot() – with kind="box"
- violinplot() – with kind="violin"
- boxenplot() – with kind="boxen"

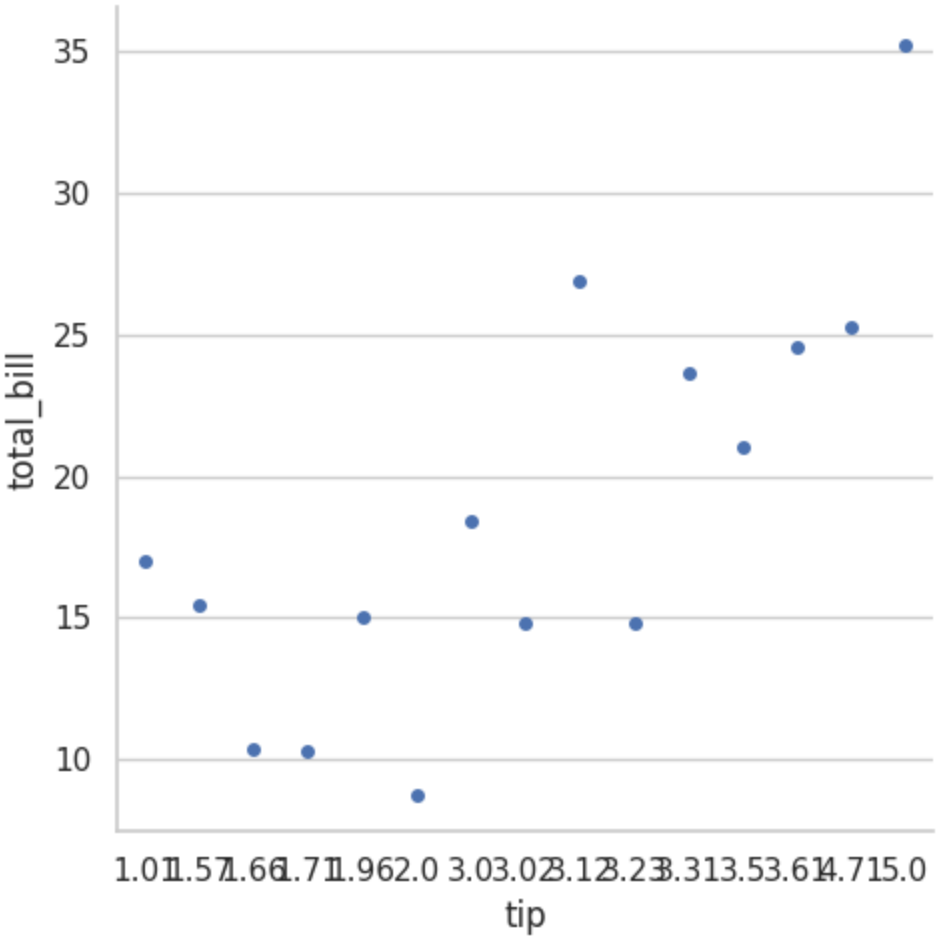
In [56]:
```
# Fetching first 15 data for analysis
df_tips_15 = sns.load_dataset("tips").head(15)
df_tips_15
```

Out[56]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
5	25.29	4.71	Male	No	Sun	Dinner	4
6	8.77	2.00	Male	No	Sun	Dinner	2
7	26.88	3.12	Male	No	Sun	Dinner	4
8	15.04	1.96	Male	No	Sun	Dinner	2
9	14.78	3.23	Male	No	Sun	Dinner	2
10	10.27	1.71	Male	No	Sun	Dinner	2
11	35.26	5.00	Female	No	Sun	Dinner	4
12	15.42	1.57	Male	No	Sun	Dinner	2
13	18.43	3.00	Male	No	Sun	Dinner	4
14	14.83	3.02	Female	No	Sun	Dinner	2

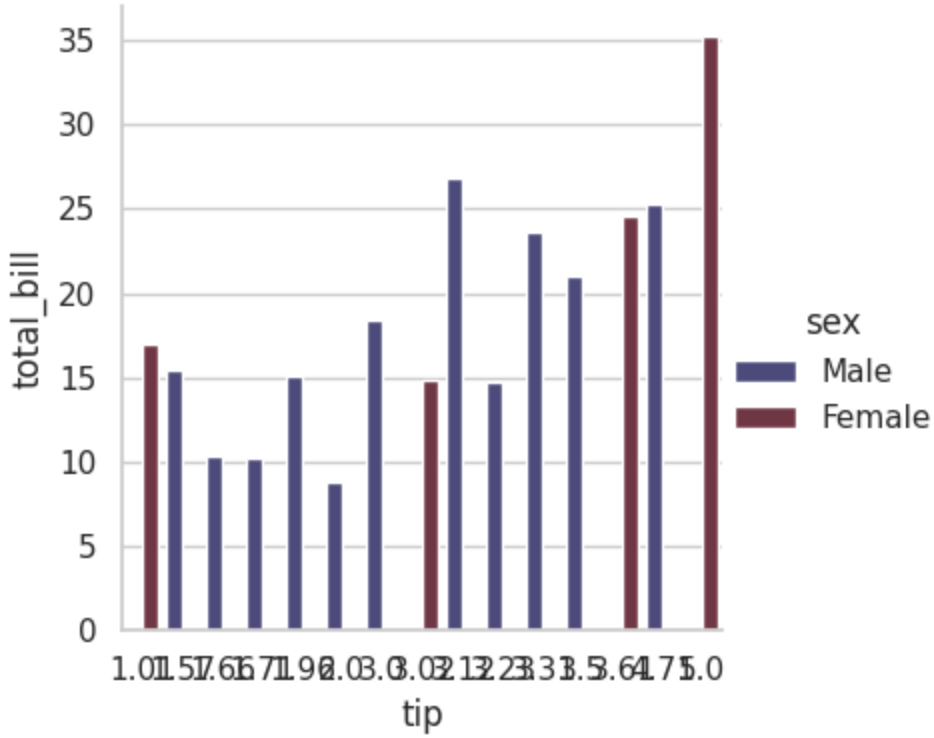
In [57]:
```
# Analyzing tip and total_bill of 15 datasets
sns.catplot(x= 'tip',  y= 'total_bill', data= df_tips_15)
```

Out[57]: <seaborn.axisgrid.FacetGrid at 0x7a5e089038e0>



In [58]:
```
# Using the parameters
sns.catplot(x = 'tip',  y = 'total_bill', data = df_tips_15,
            kind = 'bar',
            hue = 'sex',
            palette = 'icefire',
            height = 4)
```

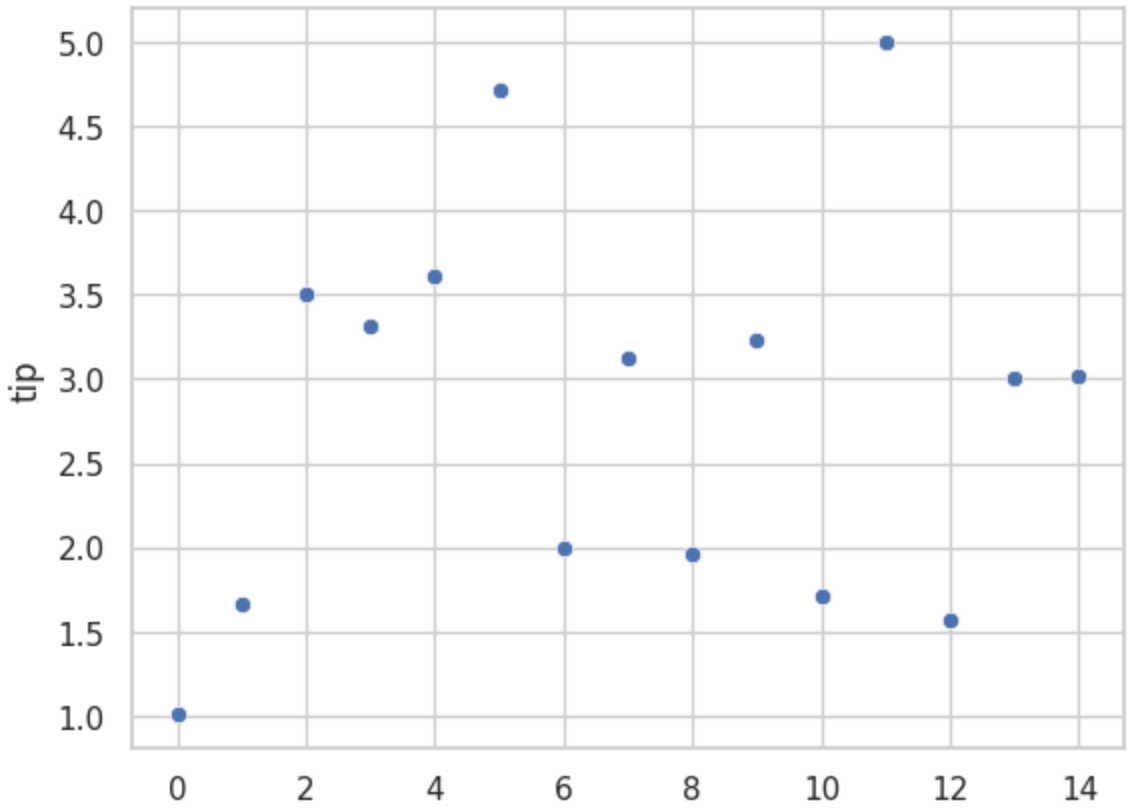
Out[58]: <seaborn.axisgrid.FacetGrid at 0x7a5e08701d20>



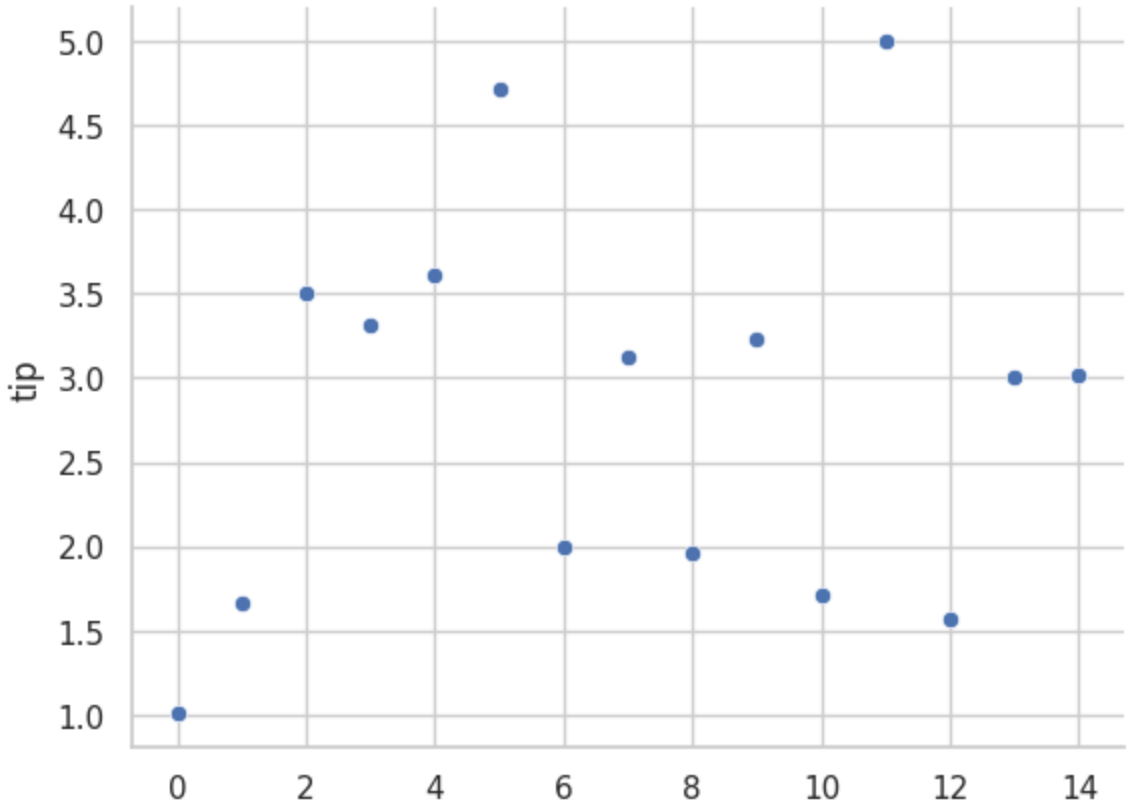
Styling Plot in Seaborn

In [59]:
```
sns.set_style("whitegrid") # white, whitegrid, dark, darkgrid and ticks are avail

sns.scatterplot(df_tips_15['tip'])
```

Out[59]: <Axes: ylabel='tip'>

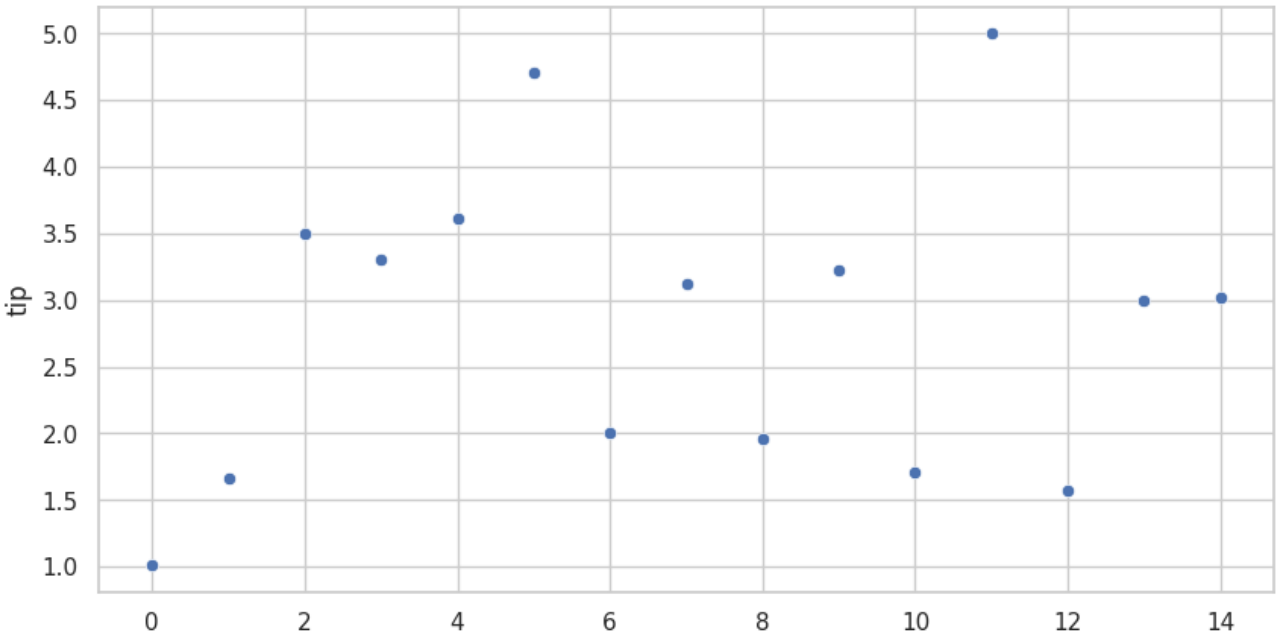


In [60]:
```
sns.set_style("whitegrid")

sns.scatterplot(df_tips_15['tip'])
sns.despine() # It's removing the side axis or axis spines
```



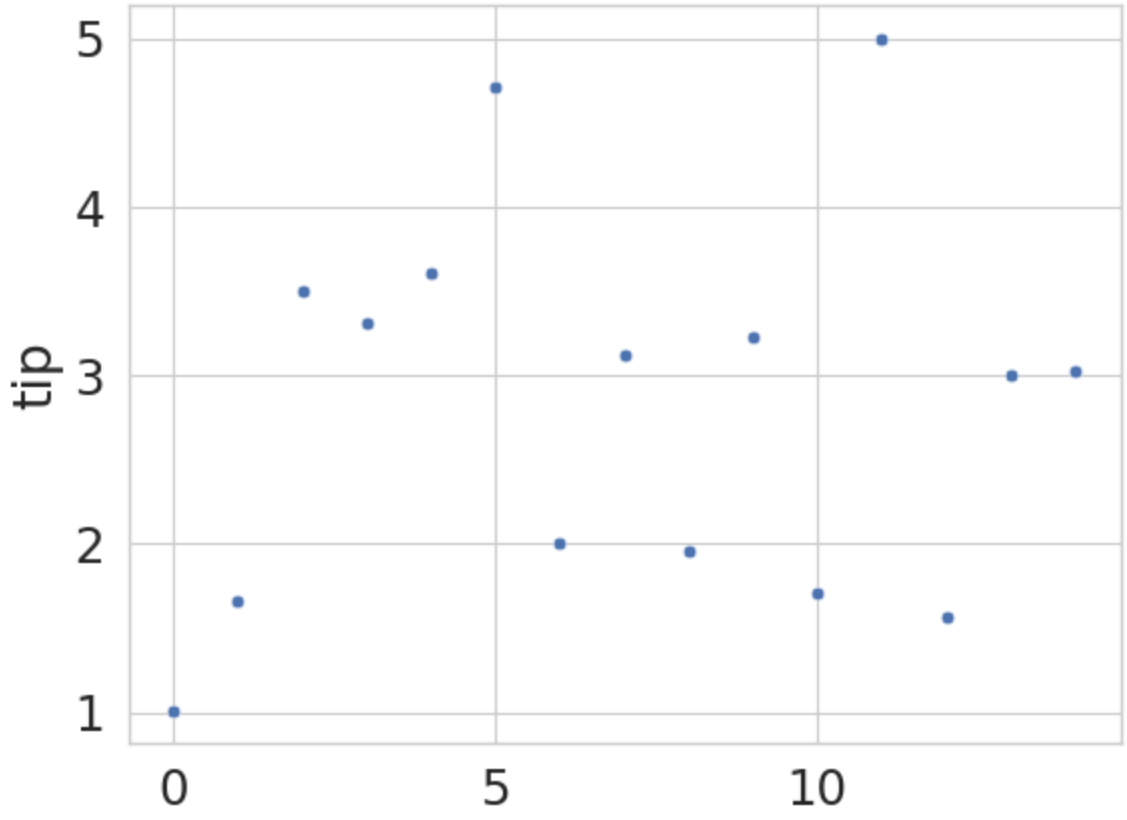
In [61]:
```
plt.figure(figsize = (10,5))

sns.scatterplot(df_tips_15['tip'])
```

Out[61]: <Axes: ylabel='tip'>



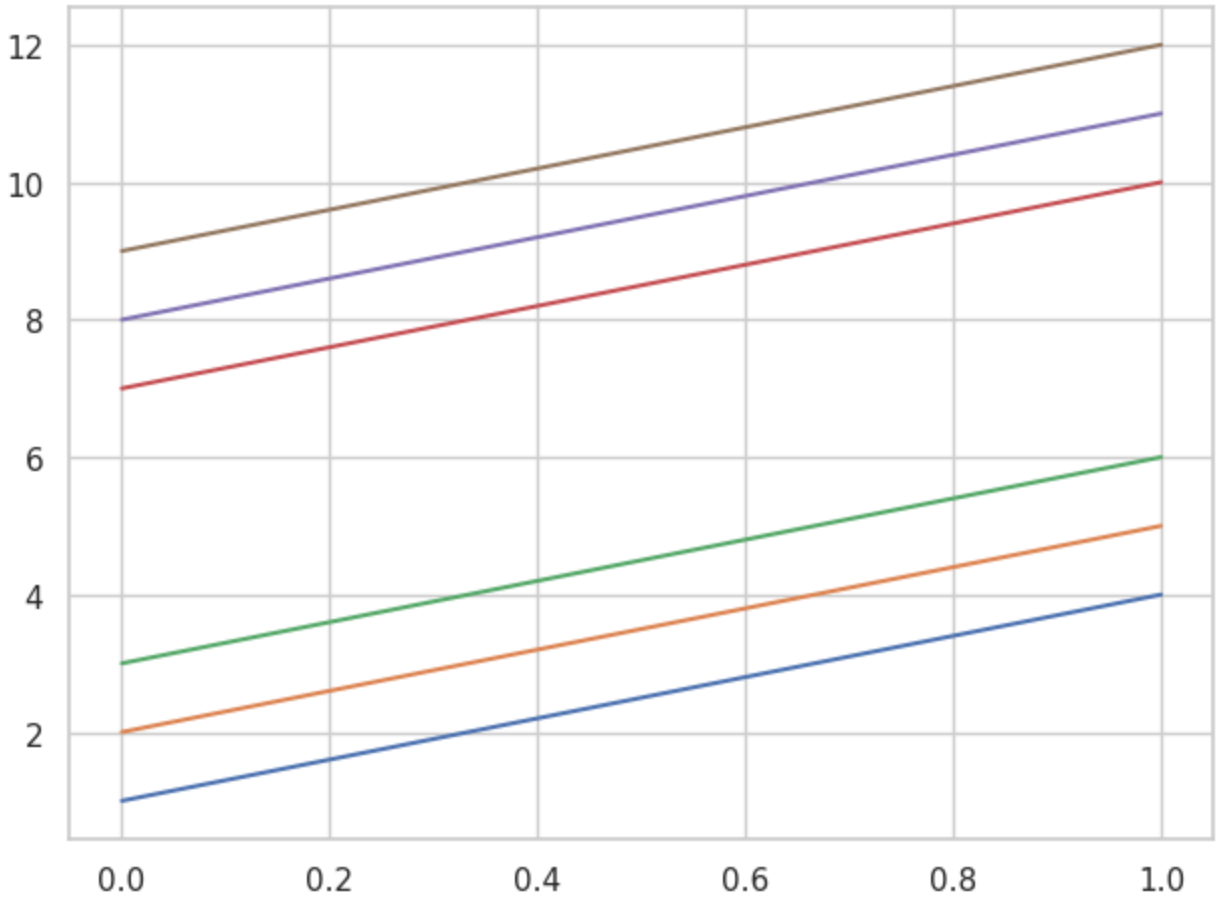
In [68]:
```
sns.set_context('paper', font_scale = 2) # We can use: paper, notebook, talk, poster

sns.scatterplot(df_tips_15['tip'])
```

Out[68]: <Axes: ylabel='tip'>

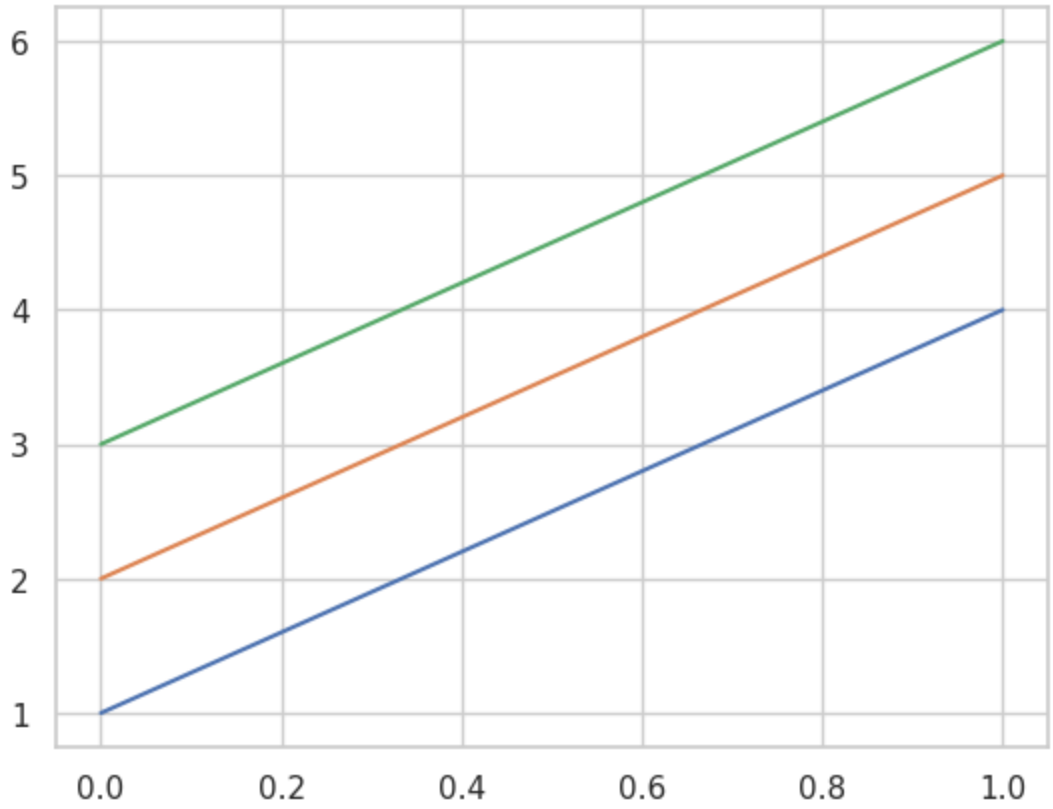


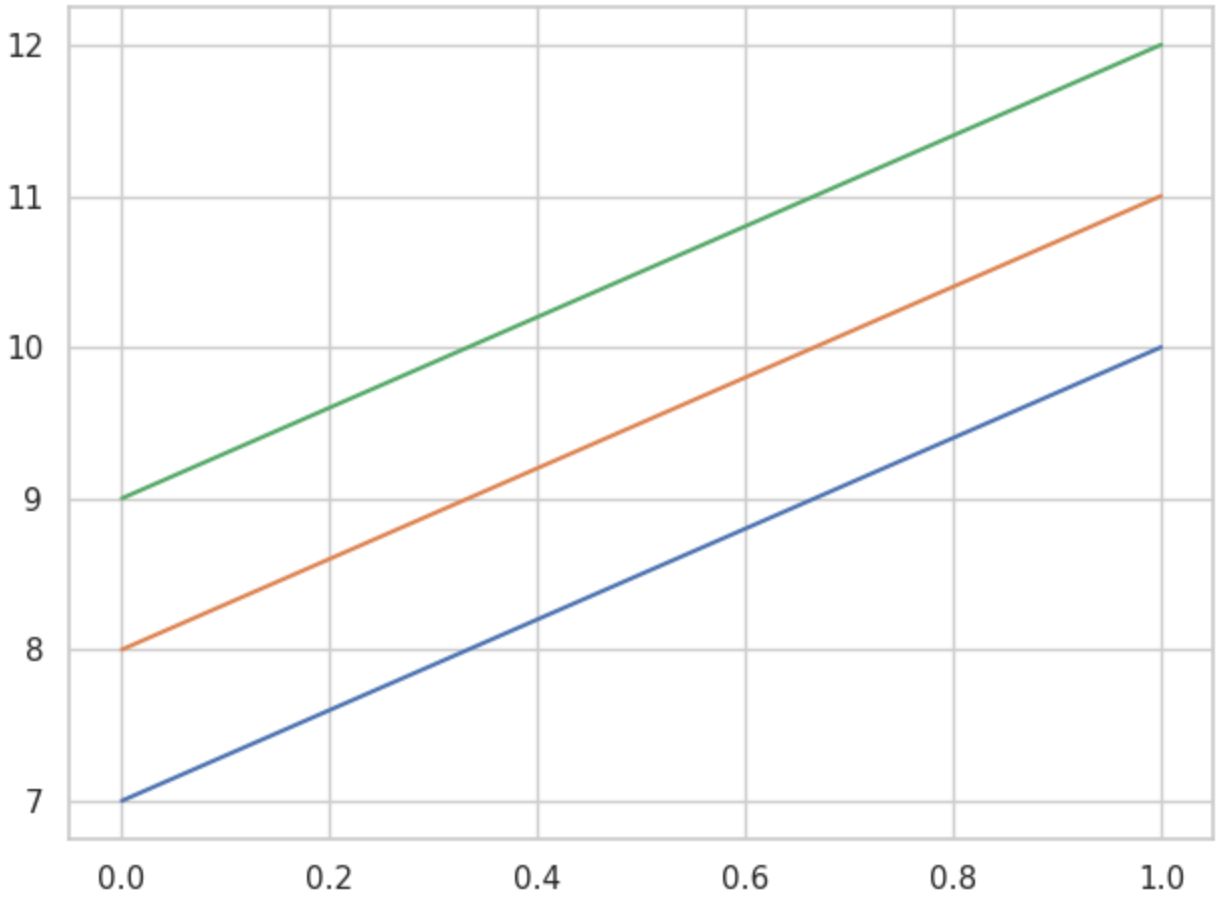
Multiple plot in a graph (Facet Grid)

In [63]:
```
# Ploting 2 graph in a same graph
# Create the first graph
ax1 = ([1, 2, 3], [4, 5, 6])
plt.plot(ax1)

# Create the second graph
ax2 = ([7, 8, 9], [10, 11, 12])
plt.plot(ax2)

# Combine the graphs
plt.tight_layout()
plt.show()
```



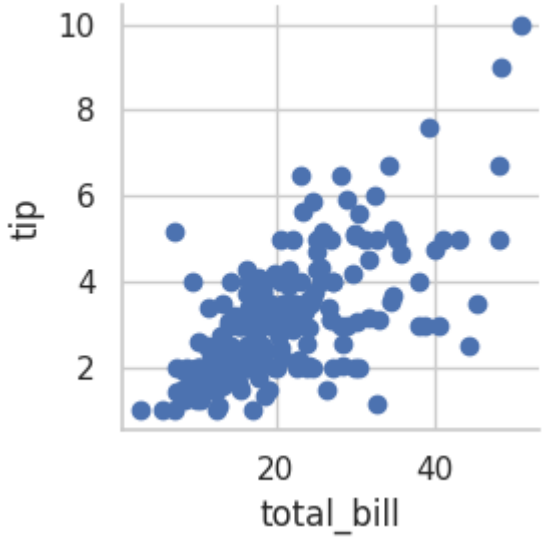
In [64]:
```
# Ploting 2 graph in different graph
# Create the first graph
ax1 = ([1, 2, 3], [4, 5, 6])
plt.plot(ax1)

# Create the second graph
fig = plt.subplots()
ax2 = ([7, 8, 9], [10, 11, 12])
plt.plot(ax2)

# Combine the graphs
plt.tight_layout()
plt.show()
```





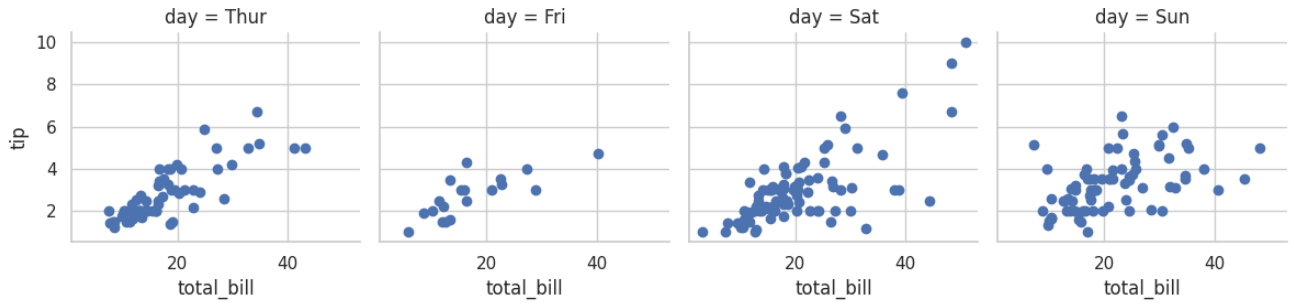
In [65]:
```
# Plotting a graph through Facet
fig = sns.FacetGrid(df_tips)

fig.map(plt.scatter, 'total_bill', 'tip')
```

Out[65]: <seaborn.axisgrid.FacetGrid at 0x7a5e082baa70>



In [66]:
```
# Using col parameter
fig = sns.FacetGrid(df_tips, col = 'day')

fig.map(plt.scatter, 'total_bill', 'tip')
```

Out[66]:
```
<seaborn.axisgrid.FacetGrid at 0x7a5e082719c0>
```



In [67]:
```
# Using the parameters
fig = sns.FacetGrid(df_tips, col = 'day',
                    hue = 'sex',
                    palette = 'hsv')

fig.map(plt.scatter, 'total_bill', 'tip',
        edgecolor = 'black').add_legend()
```

Out[67]:
```
<seaborn.axisgrid.FacetGrid at 0x7a5e07ffffa0>
```