DAY 10

* Today started with data analytics as this is the last module in our training and only 2 more days were left, we were too excited
* So, what do you mean by data analytics and why do we need it, was the first discussion point for us
  + So, to answer the question, I think data analytics simply means visualising the data to get insights in simpler way and attractive way and cleaning data so that extra unwanted things don’t affect the output we are thinking of.
  + Libraries we may require are -numpy
* Wht we did was first install numpy: - pip install numpy
  + Then created 1D-array using numpy library as we imported it
  + And perfomed operations like
    - Print array
    - Find length
    - Ty[pe of it
    - Positive and negative indexing
    - How to find dimension of array
  + Then we created 2D – Array
    - Remember one thing no of elements in first bracket should be SAME AS second array otw it throw error
    - Length of array
    - New way to access index for 2d array: b[0,1] #prints value from 1st bracket 2nd index
    - Then calculate the dimension of array using .ndim
  + Then we created 3D-array
    - Print the array
    - Print length
    - Print dimension
    - Update value : print(a[0,1,1]) #prints 22
  + Then sir took quite hard problem which is no more hard for us due to contant hands-on practice: problem was
    - #Find 13 and update it with
    - a=np.array([[[[[[[[[[[[[[1,2,3],[11,22,33],[12,13,14]]]]]]]]]]]]]]) to try it and let me know if you are able to solve it
* Then we started learning about .shape method
  + .shape method means length of array in each bracket
  + b = np.array([[1,2,3],[11,22,33]]) #so in first bracket there are 2 bracket so length for it is two and in next bracket there ar e 3 elements so 3
* Then we started with .reshape method
  + # .reshape method: first value in it tells us how much part we have to divide and second value tells us how much element should be there in each part
* Copy and view
  + copy mein ek baar values copy ho jaati hai uske baad woh dono independently act karte hai ek dusre pr depend nahi karte
  + but in view both are same so even if we change a it will change v and same for v so we can say that it refers to same address
* Iterate through array:
  + We can use any loop
  + Ex: for i in a
* combines two arrays using concatenate function
* Then we installed matplotlib for visualising data
  + And imported it at the top only
  + Then we created basic line plot and learned what operations we are able to perform they are - :
    - marker: Defines the shape of the marker. Possible values:
      * o → Round
      * d → Diamond
      * h → Hexagon
      * p → Pentagon
      * s → Square
      * \* → Star
    - markerfacecolor: Colors the inner part of the marker.
    - markeredgecolor: Defines the border color of the marker.
    - linestyle: Determines the pattern of the line. Possible values:
      * : → Dotted
      * - → Solid
      * -- → Dashed
    - linewidth: Adjusts the thickness of the line.
    - color or c: Defines the color of the line.
* More properties given by gpt:
  + **Marker Properties:**
    - markersize or ms: Defines the size of the marker.
    - markeredgewidth or mew: Sets the thickness of the marker's border.
    - markeredgecolor or mec: Defines the color of the marker's edge.
    - markerfacecolor or mfc: Fills the inner color of the marker.
  + **Line Properties:**
    - linestyle or ls: Defines the type of line. Additional options:
    - -. → Dash-dot
    - None → No line (only markers will be shown)
    - linewidth or lw: Adjusts the width of the line.
  + **Color Options:**
    - color or c: Defines the color of the line.
    - Colors can be specified in different ways:
    - Named colors (e.g., "red", "blue", "green")
    - Hex codes (e.g., #FF5733)
    - RGB tuples (e.g., (1, 0, 0) for red)
    - Shorthand ('r' for red, 'b' for blue, etc.)
  + **Alpha and Transparency:**
    - alpha: Controls transparency (range: 0 to 1, where 0 is fully transparent and 1 is opaque).
  + **Other Important Customizations:**
    - label: Adds a label for the line (used for legends).
    - zorder: Controls the layering of lines (higher values appear on top).
    - drawstyle: Changes how the line is drawn ('default', 'steps', 'steps-pre', 'steps-post')
* Then we learn how to apply grid and its related options to edit it:
  + **axis**: Defines which axis the grid will be displayed on.
    - 'x' → Shows the grid only on the X-axis.
    - 'y' → Shows the grid only on the Y-axis.
    - 'both' (default) → Displays the grid on both axes.
  + **linestyle (ls)**: Defines the style of the grid lines. Possible values:
    - '-' → Solid line
    - '--' → Dashed line
    - ':' → Dotted line
    - '-. ' → Dash-dot line
  + **color (c)**: Defines the grid color. Can be set using:
    - Color names ('red', 'blue', 'green', etc.)
    - Hex codes (#FF5733, #00FF00, etc.)
    - RGB tuples ((1, 0, 0) for red)
    - Shorthand ('r' for red, 'b' for blue, etc.)
  + **linewidth (lw)**: Defines the thickness of the grid lines.
    - Example: linewidth=1.5
  + **alpha**: Controls the transparency of the grid.
    - 0 → Fully transparent
    - 1 → Fully opaque
    - Example: alpha=0.5
  + **which**: Specifies whether to show major or minor grid lines.
    - 'major' → Displays major grid lines (default).
    - 'minor' → Displays minor grid lines.
    - 'both' → Displays both major and minor grid lines.
  + **visible**: Controls whether the grid is displayed or hidden.
    - True → Shows the grid (default).
    - False → Hides the grid.
* Then we learnt how to add title
  + Add title to graph: plt.title(“Your title here”)
    - We can use property loc to locate its position
      * By default it is center we can change it to left,right
  + Add label to xaxis – plt.xlabel(“X-axis”)
  + Add label to yaxis – plt.ylabel(“Y-axis”)
* Then we started learning Bar graph
  + **1. bar() (Vertical Bar Chart)**
    - Creates vertical bars by default.
    - X-axis represents categories or values, Y-axis represents the height of bars.
    - Can be customized using color, width, edge color, etc.
  + **2. barh() (Horizontal Bar Chart)**
    - Similar to bar(), but bars are drawn horizontally.
    - X-axis represents values, Y-axis represents categories or positions.
  + **3. Customization Options**
    - **color** → Changes the bar colors (single or multiple).
    - **edgecolor** → Defines the border color of bars.
    - **linewidth** → Adjusts the thickness of the border around bars.
    - **alpha** → Controls transparency (0 = fully transparent, 1 = fully visible).
    - **width** → Adjusts bar width in vertical bar charts.
    - **height** → Adjusts bar height in horizontal bar charts.
    - **align** → Controls alignment of bars ('center' for default, 'edge' for edge alignment).
  + **4. Adding Titles and Labels**
    - **title** → Adds a title to the chart.
    - **xlabel** → Labels the X-axis.
    - **ylabel** → Labels the Y-axis.
* Then we learnt scatter plot
  + **1. Purpose**
    - Creates a scatter plot to visualize data points.
    - Can overlay multiple scatter plots.
  + **2. Customization Options**
    - **color** → Sets point colors.
    - **marker** → Changes marker shape (o, \*, s, etc.).
    - **s** → Adjusts marker size.
    - **edgecolor** → Sets marker border color.
    - **linewidths** → Controls border thickness.
    - **alpha** → Adjusts transparency.
    - **cmap** → Uses colormap for color variations.
* Then we learnt to create pie chart
  + Created pie chart
  + Then added legend

Afternoon Session Started

* Started with histogram
  + **Purpose**
    - Creates a histogram to show the distribution of data.
    - Divides data into bins (intervals) and counts occurrences.
  + **Customization Options**
    - **bins** → Defines the number of bins (default is 10).
    - **color** → Sets bar colors.
    - **edgecolor** → Sets border color of bars.
    - **linewidth** → Controls border thickness.
    - **alpha** → Adjusts transparency.
    - **histtype** → Changes histogram style ('bar', 'step', 'stepfilled').
    - **rwidth** → Adjusts bar width relative to bin size.
    - **cumulative** → Displays cumulative frequency if True.
* Installed pandas and imported it
* Then we did some operation on pandas:
  + Create a Pandas Series from a list using pd.Series(a).
  + Default index is numeric (0, 1, 2, ...).
  + Access elements using position (series[0]).
  + Assign custom index using index=['a','b','c',...].
  + Access elements using custom labels (series['a']).
  + Series maintains data order and index mapping.
  + Works like a dictionary but retains order.
* Then learnt about DataFrames:
  + Create a dictionary with column names as keys and lists as values.
  + Convert dictionary to DataFrame using pd.DataFrame(data).
  + Displays tabular data with rows and columns.
  + Access a column using df['column\_name'].
  + Returns all values of the selected column.
  + Columns behave like Pandas Series.
  + Access a specific row-column value using df['column'][index].
  + Indexing starts from 0 by default.
  + DataFrame preserves structure and relationships.
  + print(a.info()) displays all the information
* Then performed further operation on it
  + **.info()** → Displays DataFrame summary, including column names, data types, and non-null counts.
    - Helps check dataset structure and missing values.
  + **.describe()** → Provides statistical summary (count, mean, std, min, max, etc.) for numerical columns.
    - Useful for quick insights into data distribution.
  + **.value\_counts()** → Counts unique values in a column.
    - Helps analyze frequency of categorical data.
  + **.isnull()** → Returns a DataFrame with True for missing values and False otherwise.
    - Used to detect missing data for handling.
* **.dropna()** → Removes NaN values from the selected column.
* **.fillna(value)** → Replaces NaN with the given value.
* **.drop(columns='col\_name')** → Removes the specified column without modifying the original DataFrame.

Then we started working on CSV files

* **pd.read\_csv('file.csv')** → Loads a CSV file into a DataFrame.
* **.head(n)** → Displays the first n rows (default is 5).
* **.tail(n)** → Displays the last n rows (default is 5).
* **df** → Displays the first 5 and last 5 rows.
* **.to\_string()** → Displays all rows in the DataFrame.

Then we performed data analytics operation on dataset: car\_dekho.csv

I have added all the analytics whicbh we performed graphs we build and all the related dataset are available in code section so try yourself and do let me know how you felt about it!!!!

But in short operation we performed where:

* This dataset contains 301 rows and 9 columns
* There were no null values
* And here is description:

**📌 Insights from Operations Performed on car\_dekho.csv**

* **Fuel Type Distribution (value\_counts() + Pie Chart)**
  + Displays the proportion of fuel types (Petrol, Diesel, etc.) in a pie chart.
  + Helps visualize the dominance of a petrol fuel type.
* **Transmission Type Analysis (value\_counts() + Bar Chart)**
  + Shows the count of manual vs. automatic cars.
  + **Green** represents manual, **Red** automatic.
  + Helps understand the market trend for transmission preferences.
* **Year-wise Car Sales (value\_counts() + Bar Chart)**
  + Displays the number of cars sold each year in a **bar chart**.
  + **Different colors** used for different years.
  + Helps track sales trends and peak-selling years.
* **Seller Type Analysis (value\_counts() + Bar Chart + Grid)**
  + Analyzes whether cars were sold by individual sellers or dealers.
  + **Blue** and **Red** represent dealer and individual resp.
  + **Grid on Y-axis** improves readability of car count.
* Then we attempted the test on sql