**Question 1**

**1. Importing Libraries:**

import pandas as pd

from lets\_plot import \*

* **Pandas** (pd): Used for handling and analyzing data, especially data stored in tables (like CSV files).
* **LetsPlot**: A Python library for creating interactive plots.

**2. Setup:**

LetsPlot.setup\_html(isolated\_frame=True)

* This initializes LetsPlot and sets up the environment for creating interactive plots in HTML format.

**3. Loading Data:**

df = pd.read\_csv("https://github.com/byuidatascience/data4names/raw/master/data-raw/names\_year/names\_year.csv")

* The pd.read\_csv() function loads a CSV file (from the URL) into a **Pandas DataFrame** called df. A DataFrame is like a table where rows represent data and columns represent features like name, year, and occurrences.

**4. Filtering Data for Specific Name and Birth Year:**

name = "Michael"

birth\_year = 1967

df\_name = df[df['name'] == name]

df\_name\_birth\_year = df\_name[df\_name['year'] == birth\_year]

name\_birth\_year\_usage = df\_name\_birth\_year['Total'].values[0] if not df\_name\_birth\_year.empty else 0

* **Filtering**: We're focusing on the name "Michael" and birth year 1967.
* df[df['name'] == name] filters the rows in df where the name is "Michael", resulting in df\_name.
* Then, df\_name[df\_name['year'] == birth\_year] filters df\_name for the birth year 1967, storing the result in df\_name\_birth\_year.
* name\_birth\_year\_usage extracts the total number of occurrences for that specific year. If no data is found, it defaults to 0.

**5. Summarizing Historical Usage of the Name:**

historical\_usage = df\_name[['year', 'Total']].sort\_values(by='year').copy()

historical\_usage['year'] = historical\_usage['year'].astype(int)

historical\_usage['Total'] = historical\_usage['Total'].astype(float)

* **Historical Data**: We want to track the usage of the name "Michael" over the years.
* df\_name[['year', 'Total']] extracts the "year" and "Total" columns from the df\_name DataFrame.
* .sort\_values(by='year') sorts the data by year.
* astype(int) and astype(float) ensure that the columns "year" and "Total" are of correct data types.

**6. Preparing Data for Plot:**

historical\_usage\_dict = historical\_usage.to\_dict(orient='list')

if not df\_name\_birth\_year.empty:

df\_name\_birth\_year\_dict = df\_name\_birth\_year.to\_dict(orient='list')

else:

df\_name\_birth\_year\_dict = {'year': [], 'Total': []}

* **Convert to Dictionary**: LetsPlot works better with dictionaries, so the historical\_usage DataFrame is converted into a dictionary (with orient='list').
* The birth year data is also converted similarly if it's not empty.

**7. Creating the Plot:**

p = (

ggplot(historical\_usage\_dict, aes(x='year', y='Total')) +

geom\_line(color='blue', size=1) +

ggtitle(f"Usage of the name '{name}' over the years (Including {birth\_year})") +

xlab("Year") +

ylab("Number of Occurrences") +

scale\_x\_continuous(

format=".0f",

limits=(min\_year, max\_year),

breaks=list(range(min\_year, max\_year+1, 10))

) +

theme\_minimal() +

ggsize(1000, 600)

)

* **ggplot**: Creates a plot using the data historical\_usage\_dict.
  + aes(x='year', y='Total') specifies that the x-axis will represent the years, and the y-axis will represent the total occurrences.
  + geom\_line(color='blue', size=1) draws a blue line to show how the name's popularity changes over time.
  + ggtitle(), xlab(), and ylab() add titles and labels to the plot.
  + scale\_x\_continuous() customizes the x-axis, setting the range and tick marks.
  + theme\_minimal() applies a simple, clean theme.
  + ggsize(1000, 600) sets the size of the plot.

**What Does aes() Do?**

* **aes()** stands for **aesthetic mappings**. It is used to define how variables from your data should be mapped to visual properties (like the x-axis, y-axis, color, size, etc.) in the plot.
* When you use aes(x='year', y='Total'), you are telling the plot:
  + **x-axis**: Use the values from the 'year' column in your data.
  + **y-axis**: Use the values from the 'Total' column in your data.

**8. Highlighting the Birth Year:**

if df\_name\_birth\_year\_dict['year']:

p += geom\_point(

data=df\_name\_birth\_year\_dict,

mapping=aes(x='year', y='Total'),

color='red',

size=5,

tooltips=layer\_tooltips()

.line("Year|@year")

.line("Usage|@Total")

)

* **geom\_point**: Adds a red dot to highlight the birth year (1967) on the plot.
* tooltips provides interactivity, showing the year and number of occurrences when hovering over the point.

The p += syntax is **adding to the original plot** p by appending another layer to it.

In **LetsPlot**, plots are created layer by layer, meaning you can build up a plot incrementally by adding more elements (like points, lines, labels, etc.) on top of the base plot.

**Breakdown of p += geom\_point(...):**

1. **The p Object**:
   * p is the base plot created earlier in the code, where a line (geom\_line()) is drawn to represent the historical usage of the name.
   * Initially, p contains a line plot that tracks the name's popularity over the years.
2. **The += Operator**:
   * The += operator is a shorthand for adding something to the p plot.
   * In this case, it adds a **point layer** (geom\_point()) on top of the existing line plot.
3. **What geom\_point() Does**:
   * geom\_point() adds a **scatter plot** (or points) to the graph. Specifically, it adds a **red point** at the location of the birth year (1967) to highlight that year on the line plot.
   * mapping=aes(x='year', y='Total') specifies that the point's x and y coordinates should correspond to the year and total occurrences from the birth year data (df\_name\_birth\_year\_dict).
   * The point is styled with color='red' (making it red), and its size is set to size=5.
   * **Tooltips** are added, so when you hover over the red point, it will show information about the year and the number of occurrences at that year.

**What geom\_point() Does:**

1. **geom\_point()** is a function from **LetsPlot** (and similar to other plotting libraries like **ggplot2** in R).
2. In **LetsPlot**, **geom\_point()** is specifically used to add **points** to a plot, which are typically displayed as individual markers on a graph. This is essentially a scatter plot.
   * **Scatter plot** refers to a plot where each data point is represented as a point (or marker) in 2D space, based on its x and y values. In this case, each point corresponds to the values of **year** and **Total** for a specific record.

**Why += and Not =?**

* **+=** allows you to **build up** the plot step by step. This is typical in **layered plotting systems**, where each new element (like points, lines, titles, etc.) is added to the existing plot.
* If you used **=** (e.g., p = geom\_point(...)), it would overwrite p with just the point layer, and you would lose the original line plot.

So, with the +=, you're **adding** a layer of points to the already existing line plot, enriching the visualization with more details without losing any previous elements.

This is one of the strengths of layered plotting frameworks like **LetsPlot**: you can build complex plots by adding layers incrementally.

**9. Rendering the Plot:**

p.show()

* Finally, p.show() renders and displays the plot.

**Key Concepts:**

* **Pandas**: Used for handling the CSV data as a table, filtering, and preparing it for visualization.
* **LetsPlot**: Used to create an interactive line plot of the name's popularity over time, with special emphasis on the birth year.

**Question 2**

This code builds a detailed visualization using **Pandas** for data manipulation and **LetsPlot** for plotting. It focuses on analyzing and visualizing the usage of a given name, "Brittany," across various years, with additional analysis to find the peak usage year, guessable age, and years of low usage. Let's go through the code step by step.

**1. Importing Libraries:**

import pandas as pd

from lets\_plot import \*

* **Pandas** (pd): Used for reading, processing, and analyzing data, especially structured data like CSV files.
* **LetsPlot**: Used for plotting and visualizing the data interactively.

**2. Setting Up LetsPlot:**

LetsPlot.setup\_html(isolated\_frame=True)

* This initializes LetsPlot and sets it up for generating plots inside an HTML page. isolated\_frame=True ensures the plot is rendered in a self-contained HTML frame, which is useful for web-based displays.

**3. Loading Data:**

df = pd.read\_csv("https://github.com/byuidatascience/data4names/raw/master/data-raw/names\_year/names\_year.csv")

* The pd.read\_csv() function loads data from a CSV file hosted online into a **Pandas DataFrame** (df).
* This dataset contains the number of occurrences for various names over different years.

**4. Defining the Name and Filtering Data:**

name = "Brittany" # Replace with any name you want to search for

df\_name = df[df['name'] == name]

* Here, the variable name is set to "Brittany." You can change this value to analyze a different name.
* The line df[df['name'] == name] filters the dataset to include only rows where the name is "Brittany," creating a new DataFrame (df\_name).

**5. Summarizing Historical Usage:**

historical\_usage = df\_name[['year', 'Total']].sort\_values(by='year').copy()

* This extracts the columns year and Total from the filtered df\_name DataFrame and sorts the data by year (sort\_values(by='year')).
* historical\_usage is a new DataFrame that contains the name's total occurrences over the years, sorted chronologically.

**6. Finding Peak Usage Year and Guessable Age:**

peak\_usage\_year = historical\_usage.loc[historical\_usage['Total'].idxmax()]['year']

peak\_usage = historical\_usage['Total'].max()

guessable\_age = 2025 - peak\_usage\_year # Assuming current year is 2025

* **Peak Usage**:
  + historical\_usage['Total'].idxmax() finds the index of the maximum value in the Total column (i.e., the year with the highest occurrences).
  + historical\_usage.loc[...]['year'] retrieves the year that corresponds to the peak usage.
  + peak\_usage stores the actual number of occurrences for that peak year.
* **Guessable Age**:
  + Assuming the current year is 2025, guessable\_age estimates the age of someone named "Brittany" by subtracting the peak usage year from 2025. This gives an approximate age for someone born around the peak year.

**7. Finding Years with Low Usage:**

min\_usage\_years = historical\_usage[historical\_usage['Total'] < historical\_usage['Total'].quantile(0.1)]

* This identifies years where the name "Brittany" had low usage by filtering the years where the number of occurrences is in the bottom 10%. This is done by calculating the **10th percentile** of the Total column (quantile(0.1)).
* min\_usage\_years is a DataFrame containing years with particularly low usage.

**8. Converting Data for LetsPlot:**

historical\_usage\_dict = historical\_usage.to\_dict(orient='list')

peak\_usage\_dict = {'year': [peak\_usage\_year], 'Total': [peak\_usage]}

low\_usage\_dict = min\_usage\_years.to\_dict(orient='list')

* The data is converted from Pandas DataFrames into dictionaries using to\_dict(orient='list') to make it compatible with LetsPlot.
  + **historical\_usage\_dict**: Contains the historical usage data for plotting the line.
  + **peak\_usage\_dict**: Contains just the peak usage year and its occurrence for highlighting.
  + **low\_usage\_dict**: Contains the low-usage years for highlighting on the plot.

**9. Creating the Base Plot:**

p = (

ggplot(historical\_usage\_dict, aes(x='year', y='Total')) +

geom\_line(color='blue', size=1, label=f"Usage of '{name}'") +

ggtitle(f"Usage of the name '{name}' over the years") +

xlab("Year") +

ylab("Number of Occurrences") +

scale\_x\_continuous(format=".0f") + # Ensures no commas in x-axis labels

theme\_minimal() +

ggsize(1000, 600) # Set plot size

)

* **Base Plot**:
  + The line plot is created with geom\_line(color='blue', size=1) to visualize the name's usage over time. The x-axis corresponds to the year, and the y-axis corresponds to the Total occurrences.
  + Titles and labels are added (ggtitle(), xlab(), ylab()).
  + The x-axis is customized to display year values without commas using scale\_x\_continuous(format=".0f").
  + theme\_minimal() applies a clean and simple theme, and ggsize(1000, 600) sets the size of the plot.

**10. Highlighting the Peak Usage Year:**

p += geom\_point(

data=peak\_usage\_dict,

mapping=aes(x='year', y='Total'),

color='red',

size=5,

tooltips=layer\_tooltips()

.line("Year|@year")

.line("Usage|@Total")

)

* **Highlighting Peak Usage**:
  + geom\_point() adds a red dot at the year with peak usage (the highest point on the line plot).
  + Tooltips are added to show the year and usage when hovering over the point.

**11. Highlighting Low Usage Years:**

p += geom\_point(

data=low\_usage\_dict,

mapping=aes(x='year', y='Total'),

color='green',

size=6,

shape=16,

alpha=0.8, # Adjust transparency

tooltips=layer\_tooltips()

.line("Year|@year")

.line("Usage|@Total")

)

* **Highlighting Low Usage**:
  + Another geom\_point() adds green dots to highlight the years with minimal usage (bottom 10%).
  + These points are larger (size=6) and more transparent (alpha=0.8).
  + Tooltips are added for these low-usage points as well.

**12. Rendering the Plot:**

p.show()

* **p.show()** renders the plot, displaying the line plot with highlighted points for the peak usage year and low-usage years.

**13. Outputting Additional Information:**

print("")

print(f"Guessing the age of someone named {name}:")

print(f"The peak year for the name {name} was {int(peak\_usage\_year)},\n"

f"which means someone named {name} is most likely around {int(guessable\_age)} years old in 2025.")

print(f"Ages you would not guess for {name} are likely in the years\n"

f"{', '.join(map(str, min\_usage\_years['year'].values))} (years highlighted in green dots).\n"

f"These years have very low usage of the name.")

* This section prints out:
  + **Guessable Age**: Based on the peak usage year, it estimates the age of someone named "Brittany."
  + **Low-Usage Years**: It lists the years with minimal usage (highlighted in green on the plot), which might correspond to ages you wouldn't guess for someone named "Brittany."

**Summary of the Code:**

* The code analyzes the name "Brittany" over time, visualizes its usage using a line plot, and highlights the peak usage year and low-usage years.
* It provides a rough estimate of the age of someone named "Brittany" based on the peak usage year and identifies years with minimal usage for further insights.

This approach allows you to understand both the historical popularity of a name and make educated guesses about the age of people with that name.

**Question 3**

This code analyzes and visualizes the historical usage of four specific names ("Mary," "Martha," "Peter," "Paul") between the years 1920 and 2000. It uses **Pandas** for data filtering and processing and **LetsPlot** for visualization. Here's a detailed explanation of each section:

**1. Importing Libraries:**

import pandas as pd

from lets\_plot import \*

* **Pandas**: A Python library for data manipulation and analysis.
* **LetsPlot**: A library for creating interactive and static visualizations.

**2. Setting Up LetsPlot:**

LetsPlot.setup\_html(isolated\_frame=True)

* Initializes LetsPlot for rendering plots in an HTML page.
* isolated\_frame=True ensures the plots are embedded within a separate frame, useful for interactive visualizations.

**3. Loading Data:**

df = pd.read\_csv("https://github.com/byuidatascience/data4names/raw/master/data-raw/names\_year/names\_year.csv")

* Reads a CSV file hosted online into a Pandas DataFrame (df).
* The dataset contains information on the occurrences of various names over a range of years. It likely has columns such as name, year, and Total.

**4. Defining Names and Filtering Data:**

names\_to\_compare = ["Mary", "Martha", "Peter", "Paul"]

df\_filtered = df[(df['name'].isin(names\_to\_compare)) & (df['year'] >= 1920) & (df['year'] <= 2000)]

* **names\_to\_compare**: A list of names of interest.
* **Filtering**:
  + df['name'].isin(names\_to\_compare): Filters rows where the name column matches any name in the names\_to\_compare list.
  + (df['year'] >= 1920) & (df['year'] <= 2000): Filters rows where the year is between 1920 and 2000, inclusive.
  + The combination of these conditions filters the data to include only the specified names within the given year range.

**5. Converting Data for LetsPlot:**

df\_filtered\_dict = df\_filtered.to\_dict(orient='list')

* Converts the filtered DataFrame (df\_filtered) into a dictionary format with lists as values. This is required for compatibility with LetsPlot.

Example structure of df\_filtered\_dict:

{

'name': ['Mary', 'Mary', 'Peter', 'Paul', ...],

'year': [1920, 1921, 1920, 1920, ...],

'Total': [50000, 48000, 12000, 13000, ...]

}

**6. Creating the Plot:**

p = (

ggplot(df\_filtered\_dict, aes(x='year', y='Total', color='name')) +

geom\_line(size=1) +

ggtitle("Usage of Christian Names (Mary, Martha, Peter, Paul) from 1920 to 2000") +

xlab("Year") +

ylab("Number of Occurrences") +

scale\_x\_continuous(format=".0f") +

theme\_minimal() +

ggsize(1000, 600)

)

* **Base Plot**:
  + **ggplot(df\_filtered\_dict, aes(x='year', y='Total', color='name'))**:
    - Maps the year to the x-axis and Total (name occurrences) to the y-axis.
    - The color='name' argument differentiates the lines by assigning a unique color to each name.
  + **geom\_line(size=1)**: Adds a line plot for each name, with a line thickness of 1.
* **Titles and Labels**:
  + ggtitle(): Adds a title to the plot.
  + xlab() and ylab(): Set labels for the x-axis and y-axis, respectively.
* **Formatting**:
  + scale\_x\_continuous(format=".0f"): Ensures that years on the x-axis are displayed as integers (no commas).
  + theme\_minimal(): Applies a clean, minimalistic theme to the plot.
  + ggsize(1000, 600): Sets the dimensions of the plot to 1000x600 pixels.

**7. Rendering the Plot:**

p.show()

* **p.show()** displays the plot. The resulting visualization includes:
  + A line plot for each name ("Mary," "Martha," "Peter," "Paul").
  + The x-axis represents the years (1920–2000).
  + The y-axis represents the total occurrences of each name.
  + Each line is color-coded to distinguish between the names.

**Key Features of the Plot:**

* **Comparison Across Names**: You can visually compare the popularity trends of the names over time.
* **Color Coding**: Each name is represented by a unique color.
* **Year Range**: The plot focuses only on the years between 1920 and 2000, providing a clear time frame for analysis.

**Summary:**

This code provides a clear and interactive visualization of the historical usage of selected names over a specified period. By filtering the data and using LetsPlot's visualization capabilities, it allows for easy comparison of naming trends across multiple names within a defined range of years.

**Question 4**

This code analyzes the historical usage of the name "Forrest" and visualizes it alongside the release year of the movie *Forrest Gump* (1994). It uses **Pandas** for data processing and **LetsPlot** for creating the visualization. Here’s a detailed breakdown:

**1. Importing Libraries:**

import pandas as pd

from lets\_plot import \*

* **Pandas**: A library for data manipulation and analysis.
* **LetsPlot**: A Python library for creating visualizations.

**2. Setting Up LetsPlot:**

LetsPlot.setup\_html(isolated\_frame=True)

* Initializes LetsPlot for rendering plots in HTML.
* isolated\_frame=True ensures plots are embedded separately, making them easier to view and interact with.

**3. Loading the Data:**

df = pd.read\_csv("https://github.com/byuidatascience/data4names/raw/master/data-raw/names\_year/names\_year.csv")

* Reads the dataset from an online CSV file into a Pandas DataFrame (df).
* This dataset contains columns like:
  + name: The name of a person.
  + year: The year the name was recorded.
  + Total: The number of occurrences of the name in that year.

**4. Setting Up Variables:**

name = "Forrest"

movie\_release\_date = "1994-07-06"

* **name**: The name being analyzed ("Forrest").
* **movie\_release\_date**: The release date of the movie *Forrest Gump*.

**5. Filtering Data for the Name:**

df\_name = df[df['name'] == name]

* Filters the DataFrame (df) to include only rows where the name column matches "Forrest".
* The result (df\_name) contains data about the yearly occurrences of the name "Forrest".

**6. Converting Data for LetsPlot:**

df\_name\_dict = df\_name.to\_dict(orient='list')

* Converts the filtered DataFrame into a dictionary with lists as values for LetsPlot compatibility.

**7. Extracting the Movie Release Year:**

movie\_release\_year = int(movie\_release\_date.split('-')[0])

* Extracts the year (1994) from the movie\_release\_date string by splitting it at the hyphens and converting the first part to an integer.

**8. Finding Year Range for Plot:**

min\_year = df\_name['year'].min()

max\_year = df\_name['year'].max()

* Finds the earliest and latest years in the filtered data for setting the x-axis range.

**9. Creating the Base Plot:**

p = (

ggplot(df\_name\_dict, aes(x='year', y='Total')) +

geom\_line(color='blue', size=1, label=f"Usage of '{name}'") +

ggtitle(f"Usage of the name '{name}' over the years (Including {movie\_release\_year})") +

xlab("Year") +

ylab("Number of Occurrences") +

scale\_x\_continuous(format="d", limits=(min\_year, max\_year), expand=(0, 0.05)) +

theme\_minimal() +

ggsize(1000, 600)

)

* **Base Plot**:
  + **ggplot(df\_name\_dict, aes(x='year', y='Total'))**: Sets up the data mapping for the plot:
    - x='year': Maps the year column to the x-axis.
    - y='Total': Maps the Total column (occurrences of the name) to the y-axis.
  + **geom\_line(color='blue', size=1)**: Adds a blue line representing the historical usage of the name.
* **Titles and Labels**:
  + ggtitle(): Adds a title to the plot.
  + xlab() and ylab(): Set labels for the x-axis and y-axis.
* **X-Axis Range**:
  + scale\_x\_continuous(format="d", limits=(min\_year, max\_year), expand=(0, 0.05)):
    - Ensures years are displayed as integers without commas.
    - Sets the x-axis range to the min and max years in the data.
    - expand=(0, 0.05): Adjusts spacing at the edges of the x-axis to make the plot cleaner.
* **Theme and Size**:
  + theme\_minimal(): Applies a minimalistic theme.
  + ggsize(1000, 600): Sets the plot size to 1000x600 pixels.

**10. Adding the Movie Release Year:**

p += geom\_vline(

xintercept=movie\_release\_year,

color='red',

linetype='dashed',

size=1,

label=f"Movie Release ({movie\_release\_year})"

)

* **geom\_vline**: Adds a vertical line at xintercept=movie\_release\_year (1994).
* **Styling**:
  + color='red': Makes the line red.
  + linetype='dashed': Uses a dashed line style.
  + size=1: Sets the line thickness.
  + The vertical line highlights the year *Forrest Gump* was released, providing a visual reference for potential influences on the name's popularity.

**11. Adjusting Theme Settings:**

p += theme(

axis\_title\_x=element\_text(margin=15),

axis\_title\_y=element\_text(margin=15),

axis\_text\_x=element\_text(margin=10),

axis\_text\_y=element\_text(margin=10),

plot\_margin=15

)

* Customizes the spacing and appearance of axis titles, labels, and margins:
  + Adds margins to avoid overlap between labels and other elements.
  + Enhances plot readability.

**12. Rendering the Plot:**

p.show()

* Displays the plot, which includes:
  + A blue line showing the historical usage of "Forrest".
  + A red dashed line marking the movie release year (1994).
  + Clear axis labels and a title.

**Key Takeaways:**

* **Historical Trend**: The plot reveals the popularity trend of the name "Forrest" over the years.
* **Impact of Pop Culture**: The vertical line indicates the release of *Forrest Gump*, allowing for visual analysis of its influence on the name's usage.
* **Customizations**: The code ensures clarity through styling, axis adjustments, and meaningful labels.

**Question 5**

This code visualizes the historical usage of the name "Elliot" over the years (1950–2020), with vertical dashed lines indicating the release dates of three versions of the movie *E.T.*. It combines **Pandas** for data processing and **LetsPlot** for visualization. Here’s a detailed breakdown:

**1. Importing Libraries:**

import pandas as pd

from lets\_plot import \*

* **Pandas**: A library for data manipulation and analysis.
* **LetsPlot**: A Python visualization library.

**2. Setting Up LetsPlot:**

LetsPlot.setup\_html(isolated\_frame=True)

* Initializes LetsPlot to render plots in HTML format.
* isolated\_frame=True: Embeds the plot in an isolated frame for better interaction and display.

**3. Loading the Data:**

df = pd.read\_csv("https://github.com/byuidatascience/data4names/raw/master/data-raw/names\_year/names\_year.csv")

* Loads a CSV file from an online source into a Pandas DataFrame (df).
* **Columns in the dataset**:
  + name: The name of a person.
  + year: The year the name was recorded.
  + Total: The number of occurrences of the name in that year.

**4. Setting Up Variables:**

name = "Elliot" # Name to analyze

movie\_release\_dates = [

("1982-06-11", "ET Release 1 (1982)"),

("1985-07-19", "ET Release 2 (1985)"),

("2002-03-22", "ET Release 3 (2002)")

]

* **name**: The name being analyzed ("Elliot").
* **movie\_release\_dates**: A list of tuples containing:
  + Release dates of three versions of *E.T.*.
  + Labels to describe each release (e.g., "ET Release 1 (1982)").

**5. Filtering the Data:**

df\_name = df[(df['name'] == name) & (df['year'] >= 1950)]

* Filters the dataset to include only rows where:
  + The name column matches "Elliot".
  + The year column is 1950 or later.
* Result: A subset of the data containing information about the name "Elliot" from 1950 onward.

**6. Converting Data for LetsPlot:**

df\_name\_dict = df\_name.to\_dict(orient='list')

* Converts the filtered DataFrame (df\_name) into a dictionary with lists as values. This format is required for LetsPlot.

**7. Setting Plot Limits:**

min\_year = 1950

max\_year = 2020

* Sets the minimum (min\_year) and maximum (max\_year) years for the x-axis.

**8. Creating the Base Plot:**

p = (

ggplot(df\_name\_dict, aes(x='year', y='Total')) +

geom\_line(color='blue', size=1, label=f"Usage of '{name}'") +

ggtitle(f"Usage of the name '{name}' over the years (1950-2020)") +

xlab("Year") +

ylab("Number of Occurrences") +

scale\_x\_continuous(

format=".0f",

limits=(min\_year, max\_year),

breaks=list(range(min\_year, max\_year+1, 10))

) +

theme\_minimal() +

ggsize(1000, 600)

)

* **Base Plot**:
  + **ggplot(df\_name\_dict, aes(x='year', y='Total'))**: Maps:
    - year to the x-axis.
    - Total (number of occurrences) to the y-axis.
  + **geom\_line(color='blue', size=1)**: Adds a blue line to represent the historical usage of "Elliot".
* **Titles and Labels**:
  + ggtitle(): Adds a title to the plot.
  + xlab() and ylab(): Set labels for the x-axis and y-axis.
* **X-Axis Customization**:
  + scale\_x\_continuous():
    - format=".0f": Ensures years are displayed as integers without commas.
    - limits=(min\_year, max\_year): Sets the range of the x-axis to 1950–2020.
    - breaks=list(range(min\_year, max\_year+1, 10)): Places ticks every 10 years.
* **Theme and Size**:
  + theme\_minimal(): Applies a simple, clean theme.
  + ggsize(1000, 600): Sets the plot size.

**9. Adding Vertical Lines for Movie Releases:**

for release\_year, label in movie\_release\_dates:

p += geom\_vline(

xintercept=int(release\_year.split('-')[0]),

color='red',

linetype='dashed',

size=1,

label=label

)

* Loops through each movie release date in movie\_release\_dates:
  + **release\_year**: The full release date (e.g., "1982-06-11").
  + **label**: The corresponding label (e.g., "ET Release 1 (1982)").
* **Vertical Lines**:
  + **geom\_vline**: Adds a vertical line at the specified year.
    - xintercept=int(release\_year.split('-')[0]): Extracts the release year as an integer.
    - color='red': Sets the line color to red.
    - linetype='dashed': Makes the line dashed.
    - size=1: Sets the line thickness.
    - label: Adds a label to the line (e.g., "ET Release 1 (1982)").

**10. Rendering the Plot:**

p.show()

* Displays the plot, which includes:
  + A blue line showing the historical usage of "Elliot" from 1950–2020.
  + Three red dashed vertical lines marking the release years of *E.T.*.

**Key Takeaways:**

* **Historical Trend**: The plot shows how the popularity of "Elliot" has changed over time.
* **Pop Culture Influence**: Red dashed lines indicate the years of *E.T.* releases, making it easy to analyze potential impacts of the movie on the name's popularity.
* **Customization**: The plot is highly customized, with clean axis labels, tick marks every 10 years, and well-labeled reference points (movie releases).

Let's explore places where **numpy** could have been helpful in your 5 examples:

### 1. **Example 1: Guessable Age and Peak Usage Year**

#### Current Code:

* Calculations like:

guessable\_age = 2025 - peak\_usage\_year

min\_usage\_years = historical\_usage[historical\_usage['Total'] < historical\_usage['Total'].quantile(0.1)]

#### Where **numpy** could help:

* **Quantile Calculation**: Instead of relying on Pandas .quantile():

import numpy as np

quantile\_10 = np.percentile(historical\_usage['Total'], 10)

min\_usage\_years = historical\_usage[historical\_usage['Total'] < quantile\_10]

* **Age Calculation**: For operations involving arrays of years (e.g., when analyzing multiple names), **numpy** arrays allow efficient vectorized calculations:

ages = 2025 - np.array(historical\_usage['year'])

### 2. **Example 2: Comparing Names (Mary, Martha, Peter, Paul)**

#### Current Code:

* No heavy numerical computation; just filtering and plotting.

#### Where **numpy** could help:

* **Slicing and Aggregation**: If you were calculating statistics (e.g., mean, sum) across names:

import numpy as np

totals = np.array(df\_filtered['Total'])

avg\_usage = np.mean(totals)

* **Efficient Filtering**: While Pandas handles filtering well, **numpy** could be used for conditions like:

filtered\_indices = np.where((df['name'].isin(names\_to\_compare)) & (df['year'] >= 1920) & (df['year'] <= 2000))

df\_filtered\_numpy = df.iloc[filtered\_indices]

### 3. **Example 3: Forrest Gump and the Movie Release Year**

#### Current Code:

* A single movie release year (movie\_release\_year) was highlighted with a vertical line.

#### Where **numpy** could help:

* **Batch Operations**: If you analyzed the impact of multiple movie release years, **numpy** arrays could simplify handling dates:

release\_years = np.array([1994, 1995, 1996])

adjusted\_years = release\_years + 1 # Example operation

* **Min/Max Calculations**: Instead of using Pandas:

min\_year = np.min(df\_name['year'])

max\_year = np.max(df\_name['year'])

### 4. **Example 4: Elliot and E.T. Releases**

#### Current Code:

* Looped through multiple movie release years to add vertical lines.

#### Where **numpy** could help:

* **Vectorized Handling of Release Years**: Instead of looping:

release\_years = np.array([1982, 1985, 2002])

for year in release\_years:

p += geom\_vline(xintercept=year, color='red', linetype='dashed', size=1)

* **Dynamic Tick Calculations**: If tick intervals were derived from data:

tick\_intervals = np.arange(min\_year, max\_year + 1, 10)

### 5. **General Usage for Numerical Computation**:

* **Data Smoothing or Aggregation**: For plots with noisy data, **numpy** could smooth the data using moving averages:

smoothed\_totals = np.convolve(df\_name['Total'], np.ones(5)/5, mode='valid') # 5-year moving average

* **Correlation or Statistical Analysis**: If you analyzed correlations between names or trends:

correlation = np.corrcoef(df\_name['year'], df\_name['Total'])

### Summary:

In your specific examples:

* **numpy** wasn't strictly necessary because Pandas and Lets-Plot handled most tasks efficiently.
* However, **numpy** could enhance performance and clarity for:
  + Heavy numerical calculations (e.g., smoothing, correlations).
  + Array-based operations on large datasets (e.g., filtering, aggregations).
  + Vectorized alternatives to loops (e.g., batch processing).