**Homework 1 ChatGPT Conversation**

Year-to-Year Crime Plot

User

# Group crimes by year and count incidents

yearly\_crime\_count = df.groupby(df['CrimeDate'].dt.year).size()

# Create a plot for year-to-year crime incidence

plt.figure(figsize=(12, 6))

plt.plot(yearly\_crime\_count.index, yearly\_crime\_count.values, marker='o', linestyle='-', color='b')

plt.title('Year-to-Year Crime Incidence')

plt.xlabel('Year')

plt.ylabel('Total Incidents')

plt.grid(True)

plt.show()

x axis should be more detailed

ChatGPT

To make the x-axis more detailed, you can increase the number of ticks or choose a different type of date formatting. You can use the `matplotlib` library's `DateFormatter` and `AutoDateLocator` to achieve this. Here's an updated version of your code with a more detailed x-axis:

```python

import matplotlib.pyplot as plt

import matplotlib.dates as mdates

# Assuming df['CrimeDate'] is a datetime column

yearly\_crime\_count = df.groupby(df['CrimeDate'].dt.year).size()

# Create a plot for year-to-year crime incidence

plt.figure(figsize=(12, 6))

plt.plot(yearly\_crime\_count.index, yearly\_crime\_count.values, marker='o', linestyle='-', color='b')

plt.title('Year-to-Year Crime Incidence')

plt.xlabel('Year')

plt.ylabel('Total Incidents')

plt.grid(True)

# Configure the x-axis for better date formatting

plt.gca().xaxis.set\_major\_locator(mdates.YearLocator())

plt.gca().xaxis.set\_major\_formatter(mdates.DateFormatter('%Y'))

plt.gca().xaxis.set\_minor\_locator(mdates.AutoDateLocator())

plt.xticks(rotation=45) # Rotate x-axis labels for readability

plt.tight\_layout()

plt.show()

```

This code uses `mdates.YearLocator()` to set major tick marks at yearly intervals and `mdates.DateFormatter('%Y')` to format the labels as years. It also automatically adjusts minor tick marks for better readability using `mdates.AutoDateLocator()`. The `plt.xticks(rotation=45)` line rotates the x-axis labels by 45 degrees for better visibility if needed.

User

include more years in x labeling

# Group crimes by year and count incidents

yearly\_crime\_count = df.groupby(df['CrimeDate'].dt.year).size()

# Create a plot for year-to-year crime incidence

plt.figure(figsize=(12, 6))

plt.plot(yearly\_crime\_count.index, yearly\_crime\_count.values, marker='o', linestyle='-', color='b')

plt.title('Year-to-Year Crime Incidence')

plt.xlabel('Year')

plt.ylabel('Total Incidents')

plt.grid(True)

plt.show()

ChatGPT

To include more years in the x-axis labeling, you can specify a larger range of years to display as x-ticks. Here's an updated version of your code that includes more years in the x-axis labeling:

```python

import matplotlib.pyplot as plt

# Group crimes by year and count incidents

yearly\_crime\_count = df.groupby(df['CrimeDate'].dt.year).size()

# Create a plot for year-to-year crime incidence

plt.figure(figsize=(12, 6))

plt.plot(yearly\_crime\_count.index, yearly\_crime\_count.values, marker='o', linestyle='-', color='b')

plt.title('Year-to-Year Crime Incidence')

plt.xlabel('Year')

plt.ylabel('Total Incidents')

plt.grid(True)

# Specify a larger range of years for x-axis labeling (e.g., every 2 years)

plt.xticks(range(min(yearly\_crime\_count.index), max(yearly\_crime\_count.index) + 1, 2), rotation=45)

plt.tight\_layout()

plt.show()

```

In this updated code, I've used the `plt.xticks` function to specify a range of years for x-axis labeling. You can adjust the range as needed, and the `rotation` argument is used to rotate the x-axis labels for better readability.

User

# Extract the year from the 'CrimeDate' column

df['Year'] = df['CrimeDate'].dt.year

# Create a facet grid for crime type distribution year-to-year

g = sns.FacetGrid(data=df, col='Description', col\_wrap=3, height=3, aspect=2, palette='viridis')

g.map(sns.countplot, 'Year', order=df['Year'].unique(), color='b')

g.set\_axis\_labels('Year', 'Count')

g.set\_titles(col\_template="{col\_name}")

g.tight\_layout()

plt.show()

rotate the xticks with 60

ChatGPT

To rotate the x-axis labels with a 60-degree angle in your seaborn facet grid, you can use the `set\_xticklabels` method and pass the `rotation` parameter. Here's how you can modify your code:

```python

import seaborn as sns

import matplotlib.pyplot as plt

# Extract the year from the 'CrimeDate' column

df['Year'] = df['CrimeDate'].dt.year

# Create a facet grid for crime type distribution year-to-year

g = sns.FacetGrid(data=df, col='Description', col\_wrap=3, height=3, aspect=2, palette='viridis')

g.map(sns.countplot, 'Year', order=df['Year'].unique(), color='b')

g.set\_axis\_labels('Year', 'Count')

g.set\_titles(col\_template="{col\_name}")

# Rotate x-axis labels with 60-degree angle

for ax in g.axes.flat:

ax.set\_xticklabels(ax.get\_xticklabels(), rotation=60)

g.tight\_layout()

plt.show()

```

In this modified code, the `for ax in g.axes.flat:` loop iterates through each subplot in the facet grid and rotates the x-axis labels using `ax.set\_xticklabels`.

Dataset Profile Summary

User

Create a dataset profile table that gives an overview of the dataset. You may refer to profile tools such as https://oralytics.com/2022/04/04/python-data-profiling-libraries/

At a minimum, you will need to gather the following information from the dataset via writing Python scripts where appropriate.

Total number of call instances/rows

Total number of call features/columns

For each column:

Are they numeric/real/continuous or symbolic/discrete types of attributes or belong to temporal or spatial categories?

If it is a numeric attribute, what are the min, max, mean, median, and standard deviation of the values?

If it is a discrete attribute, what is the total number of unique values? What are the three attribute values with the largest count?

What is the level of missingness (% of rows with missing values)?

Comment on anything surprising/strange/unique you noted from the data profile.

df= pd.read\_csv('Baltimore911.csv')

df

df= pd.read\_csv('Baltimore911.csv')

df

CrimeDate CrimeTime CrimeCode Location Description Inside/Outside Weapon Post District Neighborhood Longitude Latitude Location 1 Premise vri\_name1 Total Incidents

0 02/08/2020 2:18:00 5D 1700 CLIFTVIEW AVE BURGLARY I NaN 312.0 EASTERN DARLEY PARK -76.593870 39.315341 NaN GROCERY/CONFECTION Eastern 1 1

1 02/08/2020 6:00:00 6G 700 N STREEPER ST LARCENY I NaN 333.0 EASTERN MADISON EASTEND -76.577425 39.299094 NaN ROW/TOWNHOUSE-OCC NaN 1

2 02/08/2020 9:50:00 3JF 1400 N BROADWAY ROBBERY - RESIDENCE I FIREARM 314.0 EASTERN OLIVER -76.594992 39.306941 NaN ROW/TOWNHOUSE-OCC NaN 1

3 02/08/2020 9:50:00 3JF 1400 N BROADWAY ROBBERY - RESIDENCE I FIREARM 314.0 EASTERN OLIVER -76.594992 39.306941 NaN ROW/TOWNHOUSE-OCC NaN 1

4 02/08/2020 17:30:00 5A 1600 E OLIVER ST BURGLARY I NaN 314.0 EASTERN OLIVER -76.595946 39.307350 NaN ROW/TOWNHOUSE-OCC Eastern 1 1

... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...

292756 01/01/1978 10:30:00 2A 1900 ARGONNE DR RAPE I OTHER 421.0 NORTHEAST MORGAN STATE UNIVERS -76.582056 39.340492 NaN ROW/TOWNHOUSE-OCC NaN 1

292757 05/01/1977 0:01:00 2A 600 W 34TH ST RAPE I OTHER 531.0 NORTHERN WYMAN PARK -76.626880 39.328817 NaN ROW/TOWNHOUSE-OCC NaN 1

292758 06/01/1975 0:00:00 2A 4400 OLD FREDERICK RD RAPE I OTHER 822.0 SOUTHWEST UPLANDS -76.691297 39.289582 NaN OTHER - INSIDE NaN 1

292759 07/20/1969 21:00:00 2A 5400 ROLAND AVE RAPE NaN OTHER 534.0 NORTHERN ROLAND PARK -76.635300 39.358898 NaN NaN NaN 1

292760 10/30/1963 0:00:00 2A 3100 FERNDALE AVE RAPE I OTHER 622.0 NORTHWEST HOWARD PARK -76.702636 39.326875 NaN ROW/T

ChatGPT

To create a dataset profile table for the given dataset 'Baltimore911.csv,' you can use Python and pandas library along with some data profiling techniques. Here's a Python script that profiles the dataset and creates a table with the requested information:

```python

import pandas as pd

# Load the dataset

df = pd.read\_csv('Baltimore911.csv')

# Initialize an empty DataFrame for the profile summary

profile\_summary = pd.DataFrame(columns=['Column Name', 'Data Type', 'Min', 'Max', 'Mean', 'Median', 'Std Dev', 'Unique Values', 'Top 3 Values', 'Missing %'])

# Get the total number of rows and columns

total\_rows, total\_columns = df.shape

# Loop through each column in the dataset

for column in df.columns:

# Get data type

data\_type = df[column].dtype

# Initialize variables to store column statistics

min\_val, max\_val, mean\_val, median\_val, std\_dev = None, None, None, None, None

unique\_vals, top\_3\_vals = None, None

missing\_percentage = df[column].isnull().mean() \* 100

# Calculate statistics based on data type

if data\_type in ['int64', 'float64']:

min\_val = df[column].min()

max\_val = df[column].max()

mean\_val = df[column].mean()

median\_val = df[column].median()

std\_dev = df[column].std()

elif data\_type == 'object':

unique\_vals = df[column].nunique()

top\_3\_vals = df[column].value\_counts().head(3)

# Append the row to the profile summary

profile\_summary = profile\_summary.append({

'Column Name': column,

'Data Type': data\_type,

'Min': min\_val,

'Max': max\_val,

'Mean': mean\_val,

'Median': median\_val,

'Std Dev': std\_dev,

'Unique Values': unique\_vals,

'Top 3 Values': top\_3\_vals,

'Missing %': missing\_percentage

}, ignore\_index=True)

# Print the profile summary

print("Dataset Profile Summary:")

print(f"Total Rows: {total\_rows}")

print(f"Total Columns: {total\_columns}")

print(profile\_summary)

```

This script will load the dataset, iterate through each column, and calculate the requested statistics and profiling information. It will then print the dataset profile summary, including data type, min, max, mean, median, standard deviation, unique values, top 3 values, and missing percentage for each column.

Please note that you need to have the 'Baltimore911.csv' file in the same directory as this script for it to work. You can adjust the script to save the profile summary as a CSV or Excel file if needed.

User

Comment on anything surprising/strange/unique you noted from the data profile.

ChatGPT

To comment on anything surprising or unique from the dataset profile, I would need to see the actual output of the profile summary. However, I can provide some general observations that you might look for when reviewing the dataset profile:

1. Missing Values: Check for columns with a high percentage of missing values. High missing percentages could indicate data quality issues or columns that may not be useful for analysis.

2. Data Types: Ensure that the data types of columns match your expectations. For example, numeric columns should have data types like 'int64' or 'float64,' while categorical columns should have the 'object' data type.

3. Numeric Statistics: Pay attention to the summary statistics of numeric columns (min, max, mean, median, and standard deviation). Unusually large or small values may warrant further investigation.

4. Categorical Columns: Examine the unique values and their frequencies in categorical columns. Look for unexpected or rare categories that may require data cleaning or consolidation.

5. Top 3 Values: In categorical columns, the top 3 most frequent values can provide insights into common categories or classes within the dataset.

6. Data Consistency: Check for consistency in data. For example, if a column represents dates, ensure that the date format is consistent throughout the dataset.

7. Outliers: Identify any potential outliers in numeric columns. These could be values that are significantly different from the mean and median.

8. Data Distribution: Analyze the distribution of numeric columns. A normal distribution is common, but other distributions (e.g., skewed or bimodal) may be indicative of specific data patterns.

9. Unique Values: Look at the number of unique values in each column. Extremely high cardinality in a categorical column might require special handling.

10. Column Names: Review column names to ensure they are descriptive and easy to understand.

To provide more specific comments on surprises or anomalies, it would be helpful to see the actual output of the profile summary generated by the script.

User

Comment on anything surprising/strange/unique you noted from the data profile.

Column Name Data Type Min Max Mean Median Std Dev Unique Values Top 3 Values Missing %

0 CrimeDate object NaN NaN NaN NaN NaN 2300 04/27/2015 421 06/05/2016 255 12/20/2018... 0.000000

1 CrimeTime object NaN NaN NaN NaN NaN 1475 18:00:00 6772 17:00:00 6463 12:00:00 ... 0.008881

2 CrimeCode object NaN NaN NaN NaN NaN 81 4E 48061 6D 38977 5A 25351 Name: Crim... 0.000000

3 Location object NaN NaN NaN NaN NaN 27064 1500 RUSSELL ST 847 3500 BOSTON ST ... 0.456345

4 Description object NaN NaN NaN NaN NaN 14 LARCENY 65508 COMMON ASSAULT 4806... 0.000000

5 Inside/Outside object NaN NaN NaN NaN NaN 4 O 128776 I 124454 Outside ... 11.531591

6 Weapon object NaN NaN NaN NaN NaN 5 FIREARM 29036 OTHER 17313 KNIFE 1... 78.982173

7 Post object NaN NaN NaN NaN NaN 259 114.0 4171 913.0 4109 111.0 3950 Name... 0.811584

8 District object NaN NaN NaN NaN NaN 10 NORTHEAST 44207 SOUTHEAST 41816 CENTRAL ... 0.000000

9 Neighborhood object NaN NaN NaN NaN NaN 278 DOWNTOWN 9312 FRANKFORD 6922 BE... 3.587910

10 Longitude float64 -81.529189 -76.383228 -76.617452 -76.613977 0.043734 None None 0.558476

11 Latitude float64 37.577262 39.661332 39.306746 39.302672 0.029658 None None 0.558476

12 Location 1 float64 NaN NaN NaN NaN NaN None None 100.000000

13 Premise object None None None None None 125 STREET 104827 ROW/TOWNHOUSE-O... 11.617668

14 vri\_name1 object None None None None None 8 Tri-District 6607 Western 5825 Cent... 88.186951

15 Total Incidents int64 1 1 1.0 1.0 0.0 None None 0.000000

ChatGPT

Here are some observations and comments based on the provided data profile:

1. \*\*Missing Values\*\*: Several columns have missing values. The "Location" column has a relatively high missing percentage of approximately 45.6%, and the "Weapon" column has a very high missing percentage of about 78.98%. The "Location 1" column is entirely missing (100% missing).

2. \*\*Data Types\*\*: The data types of the columns seem appropriate. Categorical columns like "CrimeDate," "CrimeTime," "CrimeCode," "Description," "Inside/Outside," "Weapon," "District," "Neighborhood," "Premise," and "vri\_name1" are correctly identified as objects, while numeric columns like "Longitude," "Latitude," and "Total Incidents" have the expected data types.

3. \*\*Categorical Columns\*\*: The "CrimeCode," "Description," and "District" columns have a reasonably high number of unique values. The "Description" column has 14 unique values, suggesting a limited number of crime types.

4. \*\*Location Data\*\*: The "Longitude" and "Latitude" columns appear to represent geographical coordinates, which is expected. However, there is a high percentage of missing values (approximately 55.85%) in these columns.

5. \*\*Crime Time\*\*: The "CrimeTime" column is of data type object, which might be expected to be in time format. It also has a very small percentage of missing values (approximately 0.00888%).

6. \*\*Total Incidents\*\*: The "Total Incidents" column appears to have constant values (all 1s) and a data type of int64. This might indicate that this column does not provide much useful information, as it does not vary.

7. \*\*Unique Values\*\*: The "Weapon" column has 5 unique values, which could be categories of weapons used in crimes. Similarly, the "Inside/Outside" column has 4 unique values, indicating whether the crime occurred indoors or outdoors.

8. \*\*Post and Neighborhood\*\*: The "Post" and "Neighborhood" columns have a significant number of unique values, indicating diversity in these categories.

9. \*\*Location 1\*\*: The "Location 1" column is entirely missing, which might imply that it does not contain any useful information or that it wasn't included in the dataset.

10. \*\*Longitude and Latitude Ranges\*\*: The "Longitude" column has a minimum value of approximately -81.53 and a maximum value of -76.38, while the "Latitude" column ranges from approximately 37.58 to 39.66. These values seem to be within the expected range for Baltimore, MD.

Overall, this dataset appears to contain information about crimes, including details about the date and time of the crime, location, crime type, and other related attributes. The missing values in the "Location" and "Weapon" columns may require handling if these attributes are crucial for analysis. Additionally, the "Total Incidents" column seems to have little variation and may not be informative for most analyses. Further exploration and data preprocessing may be necessary based on the specific analytical goals and use cases.

User

drop columns

ChatGPT

To drop columns from your DataFrame in Python using pandas, you can use the `drop` method. Here's how you can drop specific columns from your DataFrame based on their names:

```python

# List of column names to be dropped

columns\_to\_drop = ['Location 1', 'Total Incidents']

# Drop the specified columns

df = df.drop(columns=columns\_to\_drop)

# Alternatively, you can use the following one-liner to achieve the same result:

# df.drop(columns=columns\_to\_drop, inplace=True)

# Print the updated DataFrame

print(df)

```

In this example, `columns\_to\_drop` is a list containing the names of the columns you want to remove from the DataFrame. You can add or remove column names in the `columns\_to\_drop` list as needed. The `drop` method removes the specified columns, and the `inplace=True` argument updates the DataFrame in place without the need to reassign it.

Make sure to run this code after you've loaded your dataset and performed any initial data profiling or analysis you need.

User

RangeIndex: 292761 entries, 0 to 292760

Data columns (total 12 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 CrimeDate 292761 non-null string

1 CrimeTime 292735 non-null string

2 CrimeCode 292761 non-null string

3 Location 291425 non-null string

4 Description 292761 non-null string

5 Inside/Outside 259001 non-null string

6 Post 290385 non-null object

7 District 292761 non-null string

8 Neighborhood 282257 non-null string

9 Longitude 291126 non-null Float64

10 Latitude 291126 non-null Float64

11 Premise 258749 non-null string

dtypes: Float64(2), object(1), string(9)

Convert the below columns to categorical datatypes

Inside/Outside, District, Description

ChatGPT

You can convert the specified columns to categorical data types in your pandas DataFrame using the `.astype()` method. Here's how you can do it for the "Inside/Outside," "District," and "Description" columns:

```python

# Convert specified columns to categorical data types

df['Inside/Outside'] = df['Inside/Outside'].astype('category')

df['District'] = df['District'].astype('category')

df['Description'] = df['Description'].astype('category')

# Print the updated DataFrame info to verify the data type changes

print(df.info())

```

This code will change the data types of the specified columns to categorical. Make sure to run this code after you have loaded your dataset and performed any initial data profiling or cleaning.

User

convert the column CrimeTime to datetime format

ChatGPT

To convert the "CrimeTime" column to datetime format in your pandas DataFrame, you can use the `pd.to\_datetime()` function. Here's how you can do it:

```python

# Convert the "CrimeTime" column to datetime format

df['CrimeTime'] = pd.to\_datetime(df['CrimeTime'], format='%H:%M:%S', errors='coerce')

# Print the updated DataFrame info to verify the data type change

print(df.info())

```

In the code above:

- `pd.to\_datetime()` is used to convert the "CrimeTime" column to datetime format.

- The `format='%H:%M:%S'` argument specifies the format of the time in the "CrimeTime" column, which is expected to be in hours, minutes, and seconds (HH:MM:SS).

- The `errors='coerce'` argument handles any parsing errors by converting them to NaN values, ensuring that the conversion process does not raise errors for invalid time values.

After running this code, the "CrimeTime" column in your DataFrame will be converted to datetime format, allowing you to perform date and time-related operations on it.

User

drop rows which contain null values

ChatGPT

To drop rows containing null (NaN) values from your pandas DataFrame, you can use the `dropna()` method. Here's how you can do it:

```python

# Drop rows with null values

df = df.dropna()

# Reset the index to reflect the changes

df = df.reset\_index(drop=True)

# Print the updated DataFrame

print(df)

```

In this code:

1. `df.dropna()` is used to drop all rows containing any null values.

2. `reset\_index(drop=True)` is used to reset the index of the DataFrame so that it reflects the changes after dropping rows. The `drop=True` argument ensures that the old index is not added as a new column in the DataFrame.

After running this code, your DataFrame will only contain rows that do not have any null values. Make sure to run this code after you have loaded your dataset and performed any initial data cleaning or conversion operations.

User

Generate a series of plots to describe the temporal pattern (year-to-year, monthly, and day-of-week) of the overall crime incidence aggregating from all geo-locations for above dataset

ChatGPT

To generate a series of plots to describe the temporal pattern of overall crime incidence aggregated from all geo-locations, you can follow these steps using Python's matplotlib and pandas libraries. This will include year-to-year, monthly, and day-of-week analyses:

1. First, ensure you have the necessary libraries installed:

```bash

pip install matplotlib pandas

```

2. Load your dataset (assuming it's already loaded as `df`).

3. Convert the "CrimeDate" column to a datetime object if it's not already in datetime format:

```python

df['CrimeDate'] = pd.to\_datetime(df['CrimeDate'])

```

Now, let's create the temporal plots:

### Year-to-Year Analysis:

```python

import matplotlib.pyplot as plt

# Group crimes by year and count incidents

yearly\_crime\_count = df.groupby(df['CrimeDate'].dt.year).size()

# Create a plot for year-to-year crime incidence

plt.figure(figsize=(12, 6))

plt.plot(yearly\_crime\_count.index, yearly\_crime\_count.values, marker='o', linestyle='-', color='b')

plt.title('Year-to-Year Crime Incidence')

plt.xlabel('Year')

plt.ylabel('Total Incidents')

plt.grid(True)

plt.show()

```

### Monthly Analysis:

```python

# Group crimes by month and count incidents

monthly\_crime\_count = df.groupby(df['CrimeDate'].dt.month).size()

# Create a plot for monthly crime incidence

plt.figure(figsize=(12, 6))

plt.plot(monthly\_crime\_count.index, monthly\_crime\_count.values, marker='o', linestyle='-', color='g')

plt.title('Monthly Crime Incidence')

plt.xlabel('Month')

plt.ylabel('Total Incidents')

plt.grid(True)

plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])

plt.show()

```

### Day-of-Week Analysis:

```python

# Group crimes by day of the week and count incidents

day\_of\_week\_crime\_count = df.groupby(df['CrimeDate'].dt.day\_name()).size()

# Reorder the days of the week for plotting

day\_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']

day\_of\_week\_crime\_count = day\_of\_week\_crime\_count.reindex(day\_order)

# Create a plot for day-of-week crime incidence

plt.figure(figsize=(12, 6))

plt.bar(day\_of\_week\_crime\_count.index, day\_of\_week\_crime\_count.values, color='r')

plt.title('Day-of-Week Crime Incidence')

plt.xlabel('Day of the Week')

plt.ylabel('Total Incidents')

plt.grid(True)

plt.show()

```

These code snippets will generate three separate plots: one showing the year-to-year crime incidence, another showing the monthly crime incidence, and the last one displaying the day-of-week crime incidence. You can run each code snippet individually to visualize the temporal patterns of overall crime incidence in your dataset.

User

in seaborn

ChatGPT

Certainly! You can create the same temporal plots using Seaborn, a Python data visualization library that works well with pandas DataFrames. Make sure you have Seaborn installed:

```bash

pip install seaborn

```

Here's how you can create the temporal plots using Seaborn:

```python

import seaborn as sns

import matplotlib.pyplot as plt

# Set the style of the plots

sns.set\_style("whitegrid")

# Convert the "CrimeDate" column to a datetime object if it's not already in datetime format

df['CrimeDate'] = pd.to\_datetime(df['CrimeDate'])

# Year-to-Year Analysis

plt.figure(figsize=(12, 6))

sns.lineplot(data=df, x=df['CrimeDate'].dt.year, y='Total Incidents', estimator='count', ci=None, color='b')

plt.title('Year-to-Year Crime Incidence')

plt.xlabel('Year')

plt.ylabel('Total Incidents')

plt.grid(True)

plt.show()

# Monthly Analysis

plt.figure(figsize=(12, 6))

sns.lineplot(data=df, x=df['CrimeDate'].dt.month, y='Total Incidents', estimator='count', ci=None, color='g')

plt.title('Monthly Crime Incidence')

plt.xlabel('Month')

plt.ylabel('Total Incidents')

plt.grid(True)

plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])

plt.show()

# Day-of-Week Analysis

plt.figure(figsize=(12, 6))

sns.barplot(data=df, x=df['CrimeDate'].dt.day\_name(), y='Total Incidents', estimator=sum, ci=None, color='r',

order=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])

plt.title('Day-of-Week Crime Incidence')

plt.xlabel('Day of the Week')

plt.ylabel('Total Incidents')

plt.grid(True)

plt.show()

```

These Seaborn-based code snippets will create the same year-to-year, monthly, and day-of-week temporal plots as before, but with Seaborn's aesthetics and style. You can run each code snippet individually to visualize the temporal patterns of overall crime incidence using Seaborn.

User

# Group crimes by year and count incidents

yearly\_crime\_count = df.groupby(df['CrimeDate'].dt.year).size()

# Create a plot for year-to-year crime incidence

plt.figure(figsize=(12, 6))

plt.plot(yearly\_crime\_count.index, yearly\_crime\_count.values, marker='o', linestyle='-', color='b')

plt.title('Year-to-Year Crime Incidence')

plt.xlabel('Year')

plt.ylabel('Total Incidents')

plt.grid(True)

plt.show()

improve the above with some more detailing

ChatGPT

To improve the year-to-year crime incidence plot with more detailing, you can add labels to data points and include a trendline. You can use Seaborn for this purpose, which provides a simpler way to add trendlines. Here's an improved version of the year-to-year plot:

```python

import seaborn as sns

import matplotlib.pyplot as plt

# Set the style of the plot

sns.set\_style("whitegrid")

# Group crimes by year and count incidents

yearly\_crime\_count = df.groupby(df['CrimeDate'].dt.year).size().reset\_index(name='Total Incidents')

# Create a plot for year-to-year crime incidence with trendline

plt.figure(figsize=(12, 6))

sns.lineplot(data=yearly\_crime\_count, x='CrimeDate', y='Total Incidents', marker='o', color='b')

sns.regplot(data=yearly\_crime\_count, x='CrimeDate', y='Total Incidents', scatter=False, color='r', label='Trendline')

plt.title('Year-to-Year Crime Incidence')

plt.xlabel('Year')

plt.ylabel('Total Incidents')

plt.grid(True)

plt.legend()

plt.show()

```

In this improved version:

- We reset the index of the grouped DataFrame to have "Year" and "Total Incidents" as separate columns for better readability.

- We used Seaborn's `lineplot` and `regplot` functions to create the plot. `lineplot` is used for the line plot, and `regplot` adds a trendline to the plot.

- The legend is added to differentiate the trendline from the actual data points.

This will give you a more detailed year-to-year crime incidence plot with a clear trendline to visualize the overall trend in crime incidents over the years.

User

Generate a plot describing the distribution of crime type aggregating from all geo-locations and all time periods;

ChatGPT

To generate a plot describing the distribution of crime types aggregated from all geo-locations and all time periods, you can use a bar plot or countplot in Seaborn. Here's how to create such a plot:

```python

import seaborn as sns

import matplotlib.pyplot as plt

# Set the style of the plot

sns.set\_style("whitegrid")

# Create a countplot of crime types

plt.figure(figsize=(12, 6))

sns.countplot(data=df, x='Description', order=df['Description'].value\_counts().index, palette='viridis')

plt.title('Distribution of Crime Types')

plt.xlabel('Crime Type')

plt.ylabel('Count')

plt.xticks(rotation=90) # Rotate x-axis labels for better readability

plt.tight\_layout()

plt.show()

```

In this code:

- We use Seaborn's `countplot` to create a bar plot of crime types.

- The `x` parameter is set to 'Description' to plot the distribution of crime types.

- We use `order=df['Description'].value\_counts().index` to order the bars by the count of each crime type in descending order.

- The `palette='viridis'` argument sets the color palette for the plot. You can change it to other color palettes if you prefer.

- We rotate the x-axis labels for better readability using `plt.xticks(rotation=90)`.

- Finally, `plt.tight\_layout()` is used to ensure that all elements of the plot are properly displayed.

This code will generate a bar plot that shows the distribution of crime types aggregated from all geo-locations and all time periods, allowing you to visualize the frequency of each crime type.

User

Generate a series of plots to illustrate how crime type distributions might vary from year-to-year, month-to-month, or by day-of-week aggregating across all locations.

ChatGPT

To illustrate how crime type distributions might vary from year-to-year, month-to-month, and by day-of-week aggregating across all locations, you can create a series of bar plots or countplots using Seaborn. Here's how you can generate these plots:

### Year-to-Year Variation:

```python

import seaborn as sns

import matplotlib.pyplot as plt

# Set the style of the plots

sns.set\_style("whitegrid")

# Create a countplot of crime types for each year

plt.figure(figsize=(12, 6))

sns.countplot(data=df, x='CrimeDate', hue='Description', palette='viridis')

plt.title('Crime Type Distribution Year-to-Year')

plt.xlabel('Year')

plt.ylabel('Count')

plt.xticks(rotation=90)

plt.legend(bbox\_to\_anchor=(1.05, 1), loc='upper left') # Move the legend outside the plot

plt.tight\_layout()

plt.show()

```

### Month-to-Month Variation:

```python

# Create a countplot of crime types for each month

plt.figure(figsize=(12, 6))

sns.countplot(data=df, x=df['CrimeDate'].dt.month, hue='Description', palette='viridis')

plt.title('Crime Type Distribution Month-to-Month')

plt.xlabel('Month')

plt.ylabel('Count')

plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])

plt.legend(bbox\_to\_anchor=(1.05, 1), loc='upper left')

plt.tight\_layout()

plt.show()

```

### Day-of-Week Variation:

```python

# Create a countplot of crime types for each day of the week

plt.figure(figsize=(12, 6))

sns.countplot(data=df, x=df['CrimeDate'].dt.day\_name(), hue='Description', palette='viridis',

order=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])

plt.title('Crime Type Distribution by Day of the Week')

plt.xlabel('Day of the Week')

plt.ylabel('Count')

plt.legend(bbox\_to\_anchor=(1.05, 1), loc='upper left')

plt.tight\_layout()

plt.show()

```

In these code snippets:

- We use Seaborn's `countplot` to create countplots for crime type distributions for each year, each month, and each day of the week.

- The `hue='Description'` parameter is used to differentiate crime types by color.

- For the month-to-month variation, we set the x-axis labels to the names of the months for better readability.

- The legend is placed outside the plot to prevent overlap with the data.

These plots will help you visualize how the distribution of crime types varies over different time periods (year-to-year, month-to-month, and by day-of-week) aggregating across all locations.

User

the plot is not clear

ChatGPT

I apologize for the clarity issue. When dealing with a large number of crime types, the plots may become crowded and difficult to interpret. To improve clarity, you can consider the following options:

1. \*\*Aggregate and Simplify Crime Types\*\*: Group similar crime types into broader categories to reduce the number of unique crime types. This can make the plots more manageable.

2. \*\*Select a Subset of Data\*\*: If there are too many years, months, or days of the week, consider selecting a specific time range or specific months/days to focus on.

3. \*\*Use a Facet Grid\*\*: Instead of overlaying all crime types in a single plot, you can create a facet grid to display each crime type in a separate subplot. This can make it easier to compare distributions.

Here's an example using a facet grid for the year-to-year variation:

```python

import seaborn as sns

import matplotlib.pyplot as plt

# Set the style of the plots

sns.set\_style("whitegrid")

# Create a facet grid for crime type distribution year-to-year

g = sns.FacetGrid(data=df, col='Description', col\_wrap=3, height=3, aspect=2, palette='viridis')

g.map(sns.countplot, 'CrimeDate', order=df['CrimeDate'].dt.year.unique(), color='b')

g.set\_axis\_labels('Year', 'Count')

g.set\_titles(col\_template="{col\_name}")

g.tight\_layout()

plt.show()

```

In this code, we create a facet grid with each crime type in a separate subplot, which can improve clarity when visualizing the year-to-year variation.

You can apply similar techniques to month-to-month and day-of-week variations as well, adapting the code to your specific dataset and visualization needs.

User

CrimeDate CrimeTime CrimeCode Location Description Inside/Outside Post District Neighborhood Longitude Latitude Premise Total Incidents

0 2020-02-08 1900-01-01 02:18:00 5D 1700 CLIFTVIEW AVE BURGLARY I 312.0 EASTERN DARLEY PARK -76.59387 39.315341 GROCERY/CONFECTION 1

1 2020-02-08 1900-01-01 06:00:00 6G 700 N STREEPER ST LARCENY I 333.0 EASTERN MADISON EASTEND -76.577425 39.299094 ROW/TOWNHOUSE-OCC 1

2 2020-02-08 1900-01-01 09:50:00 3JF 1400 N BROADWAY ROBBERY - RESIDENCE I 314.0 EASTERN OLIVER -76.594992 39.306941 ROW/TOWNHOUSE-OCC 1

3 2020-02-08 1900-01-01 09:50:00 3JF 1400 N BROADWAY ROBBERY - RESIDENCE I 314.0 EASTERN OLIVER -76.594992 39.306941 ROW/TOWNHOUSE-OCC 1

4 2020-02-08 1900-01-01 17:30:00 5A 1600 E OLIVER ST BURGLARY I 314.0 EASTERN OLIVER -76.595946 39.30735 ROW/TOWNHOUSE-OCC 1

... ... ... ... ... ... ... ... ... ... ... ... ... ...

247866 1980-07-01 1900-01-01 00:01:00 2A 600 E FAYETTE ST RAPE O 111.0 CENTRAL DOWNTOWN -76.607691 39.290562 STREET 1

247867 1978-01-01 1900-01-01 10:30:00 2A 1900 ARGONNE DR RAPE I 421.0 NORTHEAST MORGAN STATE UNIVERS -76.582056 39.340492 ROW/TOWNHOUSE-OCC 1

247868 1977-05-01 1900-01-01 00:01:00 2A 600 W 34TH ST RAPE I 531.0 NORTHERN WYMAN PARK -76.62688 39.328817 ROW/TOWNHOUSE-OCC 1

247869 1975-06-01 1900-01-01 00:00:00 2A 4400 OLD FREDERICK RD RAPE I 822.0 SOUTHWEST UPLANDS -76.691297 39.289582 OTHER - INSIDE 1

247870 1963-10-30 1900-01-01 00:00:00 2A 3100 FERNDALE AVE RAPE I 622.0 NORTHWEST HOWARD PARK -76.702636 39.326875 ROW/TOWNHOUSE-OCC 1

Generate a series of plots to illustrate how crime type distributions might vary from year-to-year, month-to-month, or by day-of-week aggregating across all locations.

ChatGPT

To illustrate how crime type distributions might vary from year-to-year, month-to-month, or by day-of-week while aggregating across all locations, you can create a series of plots using Seaborn. Here are the steps and code to generate these plots:

### Year-to-Year Variation:

```python

import seaborn as sns

import matplotlib.pyplot as plt

# Set the style of the plots

sns.set\_style("whitegrid")

# Extract the year from the 'CrimeDate' column

df['Year'] = df['CrimeDate'].dt.year

# Create a countplot of crime types for each year

plt.figure(figsize=(12, 6))

sns.countplot(data=df, x='Year', hue='Description', palette='viridis')

plt.title('Crime Type Distribution Year-to-Year')

plt.xlabel('Year')

plt.ylabel('Count')

plt.xticks(rotation=90)

plt.legend(bbox\_to\_anchor=(1.05, 1), loc='upper left') # Move the legend outside the plot

plt.tight\_layout()

plt.show()

```

### Month-to-Month Variation:

```python

# Extract the month from the 'CrimeDate' column

df['Month'] = df['CrimeDate'].dt.month

# Create a countplot of crime types for each month

plt.figure(figsize=(12, 6))

sns.countplot(data=df, x='Month', hue='Description', palette='viridis')

plt.title('Crime Type Distribution Month-to-Month')

plt.xlabel('Month')

plt.ylabel('Count')

plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])

plt.legend(bbox\_to\_anchor=(1.05, 1), loc='upper left')

plt.tight\_layout()

plt.show()

```

### Day-of-Week Variation:

```python

# Extract the day of the week from the 'CrimeDate' column

df['DayOfWeek'] = df['CrimeDate'].dt.day\_name()

# Create a countplot of crime types for each day of the week

plt.figure(figsize=(12, 6))

sns.countplot(data=df, x='DayOfWeek', hue='Description', palette='viridis',

order=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])

plt.title('Crime Type Distribution by Day of the Week')

plt.xlabel('Day of the Week')

plt.ylabel('Count')

plt.legend(bbox\_to\_anchor=(1.05, 1), loc='upper left')

plt.tight\_layout()

plt.show()

```

In these code snippets:

- We first extract the year, month, and day of the week from the 'CrimeDate' column.

- Then, we create countplots for each time period (year, month, and day of the week) to visualize the variation in crime type distributions while aggregating across all locations.

- Legends are placed outside the plots for better clarity.

These plots will help you understand how crime type distributions vary over different time periods while considering all locations.

User

change the above code for each description plot a series plot

ChatGPT

If you want to create a separate series plot for each crime description, illustrating how their distributions vary from year to year, month to month, and by day of the week, you can use a facet grid in Seaborn. Here's the code to generate these separate plots:

```python

import seaborn as sns

import matplotlib.pyplot as plt

# Set the style of the plots

sns.set\_style("whitegrid")

# Create a facet grid for crime type distribution year-to-year

g = sns.FacetGrid(data=df, col='Description', col\_wrap=3, height=3, aspect=2, palette='viridis')

g.map(sns.countplot, 'Year', order=df['Year'].unique(), color='b')

g.set\_axis\_labels('Year', 'Count')

g.set\_titles(col\_template="{col\_name}")

g.tight\_layout()

plt.show()

# Create a facet grid for crime type distribution month-to-month

g = sns.FacetGrid(data=df, col='Description', col\_wrap=3, height=3, aspect=2, palette='viridis')

g.map(sns.countplot, 'Month', order=range(1, 13), color='g')

g.set\_axis\_labels('Month', 'Count')

g.set\_titles(col\_template="{col\_name}")

g.tight\_layout()

plt.show()

# Create a facet grid for crime type distribution by day of the week

g = sns.FacetGrid(data=df, col='Description', col\_wrap=3, height=3, aspect=2, palette='viridis')

g.map(sns.countplot, 'DayOfWeek', order=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'], color='r')

g.set\_axis\_labels('Day of the Week', 'Count')

g.set\_titles(col\_template="{col\_name}")

g.tight\_layout()

plt.show()

```

In this code:

- We create a separate series plot (facet grid) for each crime description using Seaborn's `FacetGrid`.

- For each facet grid, we use `map` to create countplots for year-to-year, month-to-month, and day-of-week distributions.

- Each facet grid corresponds to a unique crime description.

- The `col\_wrap` parameter controls how many plots are displayed in each row.

- We set appropriate axis labels and titles for each facet grid.

This code will generate a series of separate series plots, one for each crime description, illustrating how their distributions vary over different time periods (year-to-year, month-to-month, and by day of the week).

User

---------------------------------------------------------------------------

KeyError Traceback (most recent call last)

~\AppData\Local\Temp\ipykernel\_8916\1482360429.py in <module>

1 # Create a facet grid for crime type distribution month-to-month

2 g = sns.FacetGrid(data=df, col='Description', col\_wrap=3, height=3, aspect=2, palette='viridis')

----> 3 g.map(sns.countplot, 'Month', order=range(1, 13), color='g')

4 g.set\_axis\_labels('Month', 'Count')

5 g.set\_titles(col\_template="{col\_name}")

~\anaconda3\lib\site-packages\seaborn\axisgrid.py in map(self, func, \*args, \*\*kwargs)

698

699 # Get the actual data we are going to plot with

--> 700 plot\_data = data\_ijk[list(args)]

701 if self.\_dropna:

702 plot\_data = plot\_data.dropna()

~\anaconda3\lib\site-packages\pandas\core\frame.py in \_\_getitem\_\_(self, key)

3509 if is\_iterator(key):

3510 key = list(key)

-> 3511 indexer = self.columns.\_get\_indexer\_strict(key, "columns")[1]

3512

3513 # take() does not accept boolean indexers

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in \_get\_indexer\_strict(self, key, axis\_name)

5794 keyarr, indexer, new\_indexer = self.\_reindex\_non\_unique(keyarr)

5795

-> 5796 self.\_raise\_if\_missing(keyarr, indexer, axis\_name)

5797

5798 keyarr = self.take(indexer)

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in \_raise\_if\_missing(self, key, indexer, axis\_name)

5854 if use\_interval\_msg:

5855 key = list(key)

-> 5856 raise KeyError(f"None of [{key}] are in the [{axis\_name}]")

5857

5858 not\_found = list(ensure\_index(key)[missing\_mask.nonzero()[0]].unique())

KeyError: "None of [Index(['Month'], dtype='object')] are in the [columns]"

ChatGPT

I apologize for the error. It seems that the 'Month' column is not present in your DataFrame. To create plots based on the month-to-month variation, you need to first extract the month from the 'CrimeDate' column. Here's the updated code:

```python

import seaborn as sns

import matplotlib.pyplot as plt

# Set the style of the plots

sns.set\_style("whitegrid")

# Extract the month from the 'CrimeDate' column

df['Month'] = df['CrimeDate'].dt.month

# Create a facet grid for crime type distribution month-to-month

g = sns.FacetGrid(data=df, col='Description', col\_wrap=3, height=3, aspect=2, palette='viridis')

g.map(sns.countplot, 'Month', order=range(1, 13), color='g')

g.set\_axis\_labels('Month', 'Count')

g.set\_titles(col\_template="{col\_name}")

g.tight\_layout()

plt.show()

```

Make sure you've added the line `df['Month'] = df['CrimeDate'].dt.month` before creating the facet grid for the month-to-month variation. This code will extract the month and then create the plots as intended.

User

---------------------------------------------------------------------------

KeyError Traceback (most recent call last)

~\AppData\Local\Temp\ipykernel\_8916\697794996.py in <module>

1 # Create a facet grid for crime type distribution by day of the week

2 g = sns.FacetGrid(data=df, col='Description', col\_wrap=3, height=3, aspect=2, palette='viridis')

----> 3 g.map(sns.countplot, 'DayOfWeek', order=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'], color='r')

4 g.set\_axis\_labels('Day of the Week', 'Count')

5 g.set\_titles(col\_template="{col\_name}")

~\anaconda3\lib\site-packages\seaborn\axisgrid.py in map(self, func, \*args, \*\*kwargs)

698

699 # Get the actual data we are going to plot with

--> 700 plot\_data = data\_ijk[list(args)]

701 if self.\_dropna:

702 plot\_data = plot\_data.dropna()

~\anaconda3\lib\site-packages\pandas\core\frame.py in \_\_getitem\_\_(self, key)

3509 if is\_iterator(key):

3510 key = list(key)

-> 3511 indexer = self.columns.\_get\_indexer\_strict(key, "columns")[1]

3512

3513 # take() does not accept boolean indexers

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in \_get\_indexer\_strict(self, key, axis\_name)

5794 keyarr, indexer, new\_indexer = self.\_reindex\_non\_unique(keyarr)

5795

-> 5796 self.\_raise\_if\_missing(keyarr, indexer, axis\_name)

5797

5798 keyarr = self.take(indexer)

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in \_raise\_if\_missing(self, key, indexer, axis\_name)

5854 if use\_interval\_msg:

5855 key = list(key)

-> 5856 raise KeyError(f"None of [{key}] are in the [{axis\_name}]")

5857

5858 not\_found = list(ensure\_index(key)[missing\_mask.nonzero()[0]].unique())

KeyError: "None of [Index(['DayOfWeek'], dtype='object')] are in the [columns]"

ChatGPT

I apologize for the oversight. It appears that the 'DayOfWeek' column is not present in your DataFrame. To create plots based on the day-of-week variation, you need to first extract the day of the week from the 'CrimeDate' column. Here's the updated code:

```python

import seaborn as sns

import matplotlib.pyplot as plt

# Set the style of the plots

sns.set\_style("whitegrid")

# Extract the day of the week from the 'CrimeDate' column

df['DayOfWeek'] = df['CrimeDate'].dt.day\_name()

# Create a facet grid for crime type distribution by day of the week

g = sns.FacetGrid(data=df, col='Description', col\_wrap=3, height=3, aspect=2, palette='viridis')

g.map(sns.countplot, 'DayOfWeek', order=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'], color='r')

g.set\_axis\_labels('Day of the Week', 'Count')

g.set\_titles(col\_template="{col\_name}")

g.tight\_layout()

plt.show()

```

Make sure you've added the line `df['DayOfWeek'] = df['CrimeDate'].dt.day\_name()` before creating the facet grid for the day-of-week variation. This code will extract the day of the week and then create the plots as intended.

User

Design a dashboard that allows users like Gary to explore the spatial and temporal patterns of crime. You may get inspiration from tasks 2-4, but feel free to add insights. Bonus: Create a real dynamic dashboard that allows users to change parameters such as location and time period.

ChatGPT

Creating a dynamic dashboard for exploring the spatial and temporal patterns of crime requires web development tools and frameworks like HTML, CSS, JavaScript, and libraries such as Plotly or Dash for Python. Below, I'll provide you with a conceptual design for such a dashboard. Please note that building a fully functional dynamic dashboard goes beyond the scope of this text-based platform, but I'll outline the key components and features you can include.

\*\*Dashboard Components:\*\*

1. \*\*Map Component:\*\* Display an interactive map where users can visualize crime incidents based on location. Use markers or heatmaps to represent the spatial distribution of crimes. Users should be able to zoom in/out and click on markers for more details.

2. \*\*Time Slider:\*\* Include a time slider or date picker that allows users to filter crime data by a specific time period. Users can select a date range or use a slider to change the temporal focus.

3. \*\*Crime Type Filter:\*\* Provide a dropdown or checkbox list that lets users filter crime incidents by type (e.g., burglary, robbery, larceny) to explore specific crime categories.

4. \*\*Charts and Plots:\*\* Display various charts and plots to show insights. For example:

- \*\*Yearly Trends:\*\* Line chart showing the overall trend in crime incidents over the years.

- \*\*Monthly Patterns:\*\* Bar chart displaying the monthly distribution of crimes.

- \*\*Day-of-Week Analysis:\*\* Bar chart showing how crime varies by day of the week.

5. \*\*Location Selector:\*\* Allow users to select a specific neighborhood or district from a dropdown menu or type in an address to center the map on that location.

6. \*\*Data Table:\*\* Include a table that displays detailed information about selected crime incidents, including date, time, type, location, and description. This table can update based on user interactions.

\*\*Dynamic Features:\*\*

1. \*\*Interactive Map:\*\* Users can pan, zoom, and click on markers for more information about specific incidents.

2. \*\*Time Filter:\*\* The map and charts should update dynamically as users adjust the time slider or select a specific time range.

3. \*\*Crime Type Filtering:\*\* Charts and the map should respond to changes in crime type selections.

4. \*\*Location Selector:\*\* Users can select different neighborhoods or districts to see crime patterns in specific areas.

5. \*\*Data Table Interaction:\*\* Clicking on a chart or map marker should update the data table with relevant details.

\*\*Bonus Features:\*\*

1. \*\*Data Export:\*\* Allow users to export filtered data or charts for further analysis.

2. \*\*User Authentication:\*\* Implement user authentication to save preferences and provide personalized experiences.

3. \*\*Geospatial Analysis:\*\* Add geospatial analytics features like clustering to help users identify high-crime areas.

4. \*\*Notifications:\*\* Notify users about recent crime incidents in their selected area or based on their preferences.

To create this dynamic dashboard, you can use web development frameworks like Dash by Plotly for Python or JavaScript libraries like Leaflet for maps. You'll also need a backend server to handle data retrieval and processing if the dataset is extensive.

Creating a fully functional dynamic dashboard would require expertise in web development, data visualization, and potentially database management, so you may need to collaborate with a developer to bring this dashboard to life.

User

CrimeDate CrimeTime CrimeCode Location Description Inside/Outside Weapon Post District Neighborhood Longitude Latitude Location 1 Premise vri\_name1 Total Incidents

0 02/08/2020 2:18:00 5D 1700 CLIFTVIEW AVE BURGLARY I NaN 312.0 EASTERN DARLEY PARK -76.593870 39.315341 NaN GROCERY/CONFECTION Eastern 1 1

1 02/08/2020 6:00:00 6G 700 N STREEPER ST LARCENY I NaN 333.0 EASTERN MADISON EASTEND -76.577425 39.299094 NaN ROW/TOWNHOUSE-OCC NaN 1

2 02/08/2020 9:50:00 3JF 1400 N BROADWAY ROBBERY - RESIDENCE I FIREARM 314.0 EASTERN OLIVER -76.594992 39.306941 NaN ROW/TOWNHOUSE-OCC NaN 1

3 02/08/2020 9:50:00 3JF 1400 N BROADWAY ROBBERY - RESIDENCE I FIREARM 314.0 EASTERN OLIVER -76.594992 39.306941 NaN ROW/TOWNHOUSE-OCC NaN 1

4 02/08/2020 17:30:00 5A 1600 E OLIVER ST BURGLARY I NaN 314.0 EASTERN OLIVER -76.595946 39.307350 NaN ROW/TOWNHOUSE-OCC Eastern 1 1

... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...

292756 01/01/1978 10:30:00 2A 1900 ARGONNE DR RAPE I OTHER 421.0 NORTHEAST MORGAN STATE UNIVERS -76.582056 39.340492 NaN ROW/TOWNHOUSE-OCC NaN 1

292757 05/01/1977 0:01:00 2A 600 W 34TH ST RAPE I OTHER 531.0 NORTHERN WYMAN PARK -76.626880 39.328817 NaN ROW/TOWNHOUSE-OCC NaN 1

292758 06/01/1975 0:00:00 2A 4400 OLD FREDERICK RD RAPE I OTHER 822.0 SOUTHWEST UPLANDS -76.691297 39.289582 NaN OTHER - INSIDE NaN 1

292759 07/20/1969 21:00:00 2A 5400 ROLAND AVE RAPE NaN OTHER 534.0 NORTHERN ROLAND PARK -76.635300 39.358898 NaN NaN NaN 1

292760 10/30/1963 0:00:00 2A 3100 FERNDALE AVE RAPE I OTHER 622.0 NORTHWEST HOWARD PARK -76.702636 39.326875 NaN ROW/TOWNHOUSE-OCC NaN 1

ChatGPT

It appears you have provided a sample of your dataset again. If you have any specific questions or tasks related to this dataset or if you need further assistance, please let me know how I can help you.

User

Design a dashboard that allows users like Gary to explore the spatial and temporal patterns of crime. You may get inspiration from tasks 2-4, but feel free to add insights. Bonus: Create a real dynamic dashboard that allows users to change parameters such as location and time period.

ChatGPT

Designing a dynamic dashboard for exploring spatial and temporal patterns of crime involves combining data visualization, user interaction, and web development. Below is a conceptual design for such a dashboard, and I'll also mention the bonus feature of making it interactive.

\*\*Dashboard Components:\*\*

1. \*\*Map Component:\*\*

- An interactive map at the center of the dashboard displaying crime incidents.

- Use markers or heatmaps to represent the spatial distribution of crimes.

- Allow users to zoom in/out and pan to explore different areas.

- Clicking on a marker displays additional details about the crime incident.

2. \*\*Time Filter:\*\*

- Include a date range selector or a time slider that allows users to filter data based on a specific time period.

- Users can choose a date range or slide through time to focus on a particular timeframe.

3. \*\*Crime Type Filter:\*\*

- Provide a dropdown or checkbox list for users to filter crime incidents by type (e.g., burglary, robbery, vandalism).

- Enable users to select multiple crime types simultaneously.

4. \*\*Location Selector:\*\*

- Include a location search bar or dropdown that allows users to specify a neighborhood, district, or address.

- The map should update to center on the selected location.

5. \*\*Data Table:\*\*

- Display a table beneath the map that lists detailed information about the selected crime incidents, including date, time, location, description, and more.

- Users can sort and filter the table.

\*\*Interactive Features (Bonus):\*\*

1. \*\*Real-Time Data Updates:\*\*

- Fetch and display the latest crime data from a live data source.

- Allow users to set the data refresh interval.

2. \*\*Customized Insights:\*\*

- Include predefined insights or statistics, such as the total number of incidents, most common crime type, and recent incidents.

3. \*\*User-Defined Layers:\*\*

- Allow users to toggle different layers on the map, such as crime density, police stations, or schools, to gain more context.

4. \*\*Export and Share:\*\*

- Provide options for users to export filtered data, charts, or maps for further analysis or sharing.

5. \*\*Heatmap Option:\*\*

- Enable users to switch between a marker-based map and a heatmap representation of crime incidents.

6. \*\*User Authentication:\*\*

- Implement user accounts to save preferences, favorite locations, and customized views.

7. \*\*Mobile Responsiveness:\*\*

- Ensure that the dashboard is mobile-friendly for users who access it on smartphones or tablets.

8. \*\*Feedback and Reporting:\*\*

- Include a feedback mechanism that allows users to report inaccuracies or provide suggestions for improvement.

9. \*\*Help and Tutorials:\*\*

- Offer tooltips, tutorials, or help guides to assist users in navigating the dashboard.

Building a dynamic dashboard like this involves web development skills using frameworks like React, Angular, or Vue.js for the frontend and a backend server for data retrieval and processing. You can use mapping libraries like Leaflet or Google Maps API for the map component.

This is a high-level design concept, and creating a fully functional dashboard would require collaboration between web developers, data analysts, and designers.

User

give me code

ChatGPT

Creating a fully functional dynamic dashboard like the one described is a complex task that requires web development skills and tools. I can provide you with a basic example using Python and Dash by Plotly for creating an interactive map with a date range filter. However, please note that this is a simplified example, and building a complete dashboard with all the features mentioned earlier would require a more comprehensive development effort.

To get started, make sure you have Dash and Plotly installed:

```bash

pip install dash

pip install plotly

```

Here's a basic example of a dynamic crime dashboard:

```python

import dash

from dash import dcc, html

from dash.dependencies import Input, Output

import pandas as pd

import plotly.express as px

# Sample dataset (replace with your own data)

df = pd.read\_csv('crime\_data.csv')

# Initialize the Dash app

app = dash.Dash(\_\_name\_\_)

# Layout of the dashboard

app.layout = html.Div([

html.H1("Crime Dashboard"),

dcc.DatePickerRange(

id='date-range-picker',

start\_date=df['CrimeDate'].min(),

end\_date=df['CrimeDate'].max(),

display\_format='YYYY-MM-DD',

),

dcc.Graph(id='crime-map'),

])

# Callback to update the map based on the selected date range

@app.callback(

Output('crime-map', 'figure'),

[Input('date-range-picker', 'start\_date'),

Input('date-range-picker', 'end\_date')]

)

def update\_map(start\_date, end\_date):

filtered\_df = df[(df['CrimeDate'] >= start\_date) & (df['CrimeDate'] <= end\_date)]

fig = px.scatter\_mapbox(

filtered\_df,

lat='Latitude',

lon='Longitude',

hover\_name='Description',

color='Description',

zoom=10,

)

fig.update\_layout(mapbox\_style="carto-positron")

return fig

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

```

In this example:

1. We import the necessary libraries and create a Dash app.

2. The app layout consists of a date range picker and a map component.

3. We define a callback function that filters the dataset based on the selected date range and updates the map accordingly.

This is a simple example, and you would need to expand upon it to include additional features, such as crime type filtering, location selection, and more. Building a complete dynamic dashboard with advanced features would require significant development effort and possibly collaboration with web developers.

User

File "C:\Users\siddh\AppData\Local\Temp\ipykernel\_25064\4065854645.py", line 5

Input('date-range-picker', 'end\_date')])

^

SyntaxError: unexpected EOF while parsing

ChatGPT

I apologize for the confusion. It appears that there is an error in the code I provided. It seems like the code was cut off prematurely. Here's the complete code with the missing part:

```python

import dash

from dash import dcc, html

from dash.dependencies import Input, Output

import pandas as pd

import plotly.express as px

# Sample dataset (replace with your own data)

df = pd.read\_csv('crime\_data.csv')

# Initialize the Dash app

app = dash.Dash(\_\_name\_\_)

# Layout of the dashboard

app.layout = html.Div([

html.H1("Crime Dashboard"),

dcc.DatePickerRange(

id='date-range-picker',

start\_date=df['CrimeDate'].min(),

end\_date=df['CrimeDate'].max(),

display\_format='YYYY-MM-DD',

),

dcc.Graph(id='crime-map'),

])

# Callback to update the map based on the selected date range

@app.callback(

Output('crime-map', 'figure'),

[Input('date-range-picker', 'start\_date'),

Input('date-range-picker', 'end\_date')]

)

def update\_map(start\_date, end\_date):

filtered\_df = df[(df['CrimeDate'] >= start\_date) & (df['CrimeDate'] <= end\_date)]

fig = px.scatter\_mapbox(

filtered\_df,

lat='Latitude',

lon='Longitude',

hover\_name='Description',

color='Description',

zoom=10,

)

fig.update\_layout(mapbox\_style="carto-positron")

return fig

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

```

Please copy and paste this complete code into your Python environment, and it should work as intended. If you have any more questions or encounter any issues, feel free to ask.

User

Create a real dynamic dashboard that allows users to change parameters such as location and time period.

can you please modify the above as real dynamic dashboard

ChatGPT

Creating a fully dynamic dashboard with the capability to change parameters like location and time period requires a more extensive development effort and typically involves creating a web application. Below is a modified version of the code that converts the previous example into a more interactive web application using Dash by Plotly.

First, make sure you have Dash and Plotly installed:

```bash

pip install dash

pip install plotly

```

Here's the code for a dynamic crime dashboard web application:

```python

import dash

from dash import dcc, html

from dash.dependencies import Input, Output

import pandas as pd

import plotly.express as px

# Sample dataset (replace with your own data)

df = pd.read\_csv('crime\_data.csv')

# Initialize the Dash app

app = dash.Dash(\_\_name\_\_)

# Layout of the dashboard

app.layout = html.Div([

html.H1("Crime Dashboard"),

dcc.DatePickerRange(

id='date-range-picker',

start\_date=df['CrimeDate'].min(),

end\_date=df['CrimeDate'].max(),

display\_format='YYYY-MM-DD',

),

dcc.Dropdown(

id='location-dropdown',

options=[

{'label': location, 'value': location}

for location in df['Location'].unique()

],

multi=True,

placeholder="Select Location(s)",

),

dcc.Graph(id='crime-map'),

])

# Callback to update the map based on selected date range and location(s)

@app.callback(

Output('crime-map', 'figure'),

[Input('date-range-picker', 'start\_date'),

Input('date-range-picker', 'end\_date'),

Input('location-dropdown', 'value')]

)

def update\_map(start\_date, end\_date, selected\_locations):

filtered\_df = df[(df['CrimeDate'] >= start\_date) & (df['CrimeDate'] <= end\_date)]

if selected\_locations:

filtered\_df = filtered\_df[filtered\_df['Location'].isin(selected\_locations)]

fig = px.scatter\_mapbox(

filtered\_df,

lat='Latitude',

lon='Longitude',

hover\_name='Description',

color='Description',

zoom=10,

)

fig.update\_layout(mapbox\_style="carto-positron")

return fig

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

```

In this code:

1. We've added a dropdown component (`location-dropdown`) that allows users to select one or more locations.

2. We've updated the callback to filter data based on the selected date range and location(s).

3. The resulting map (`crime-map`) displays crime incidents based on the selected parameters.

To run this code as a web application, save it to a Python file (e.g., `crime\_dashboard.py`) and run it. You can access the dashboard by opening a web browser and going to the provided URL (usually `http://127.0.0.1:8050/`).

User

KeyError Traceback (most recent call last)

~\anaconda3\lib\site-packages\plotly\express\\_chart\_types.py in scatter\_mapbox(

data\_frame= CrimeDate CrimeTime CrimeCode ...hursday

229209 Monday

244967 Monday ,

lat='Latitude',

lon='Longitude',

color='Description',

text=None,

hover\_name='Description',

hover\_data=None,

custom\_data=None,

size=None,

animation\_frame=None,

animation\_group=None,

category\_orders=None,

labels=None,

color\_discrete\_sequence=None,

color\_discrete\_map=None,

color\_continuous\_scale=None,

range\_color=None,

color\_continuous\_midpoint=None,

opacity=None,

size\_max=None,

zoom=10,

center=None,

mapbox\_style=None,

title=None,

template=None,

width=None,

height=None

)

1245 symbol mark on a Mapbox map.

1246 """

-> 1247 return make\_figure(args=locals(), constructor=go.Scattermapbox)

global make\_figure = <function make\_figure at 0x000001B047FA23A0>

global args = undefined

global locals = undefined

global constructor = undefined

global go.Scattermapbox = <class 'plotly.graph\_objs.\_scattermapbox.Scattermapbox'>

1248

1249

~\anaconda3\lib\site-packages\plotly\express\\_core.py in make\_figure(

args={'animation\_frame': None, 'animation\_group': None, 'category\_orders': None, 'center': None, 'color': 'Description', 'color\_continuous\_midpoint': None, 'color\_continuous\_scale': None, 'color\_discrete\_map': None, 'color\_discrete\_sequence': None, 'custom\_data': None, ...},

constructor=<class 'plotly.graph\_objs.\_scattermapbox.Scattermapbox'>,

trace\_patch={'mode': 'markers'},

layout\_patch={}

)

2001 )

2002 grouper = [x.grouper or one\_group for x in grouped\_mappings] or [one\_group]

-> 2003 groups, orders = get\_groups\_and\_orders(args, grouper)

groups = undefined

global orders = undefined

global get\_groups\_and\_orders = <function get\_groups\_and\_orders at 0x000001B047FA2310>

args = {'data\_frame': CrimeDate CrimeTime CrimeCode Location \

1 2020-02-08 1900-01-01 06:00:00 6G 700 N STREEPER ST

182 2020-02-05 1900-01-01 19:00:00 6G 700 N STREEPER ST

183 2020-02-05 1900-01-01 19:00:00 6G 700 N STREEPER ST

289 2020-02-03 1900-01-01 12:45:00 4E 700 N STREEPER ST

291 2020-02-03 1900-01-01 12:45:00 4C 700 N STREEPER ST

8650 2019-10-27 1900-01-01 22:30:00 4B 700 N STREEPER ST

14236 2019-08-31 1900-01-01 09:00:00 6G 700 N STREEPER ST

18026 2019-07-27 1900-01-01 10:15:00 4E 700 N STREEPER ST

19347 2019-07-15 1900-01-01 22:30:00 4E 700 N STREEPER ST

28084 2019-04-24 1900-01-01 19:30:00 7A 700 N STREEPER ST

68820 2018-03-16 1900-01-01 07:30:00 5C 700 N STREEPER ST

89061 2017-09-03 1900-01-01 05:30:00 7A 700 N STREEPER ST

95285 2017-07-12 1900-01-01 19:00:00 6D 700 N STREEPER ST

97222 2017-06-25 1900-01-01 11:30:00 4E 700 N STREEPER ST

97223 2017-06-25 1900-01-01 11:30:00 4E 700 N STREEPER ST

101780 2017-05-19 1900-01-01 23:00:00 6D 700 N STREEPER ST

111908 2017-02-16 1900-01-01 10:20:00 5A 700 N STREEPER ST

113781 2017-01-29 1900-01-01 20:30:00 5A 700 N STREEPER ST

155908 2016-02-16 1900-01-01 19:00:00 6E 700 N STREEPER ST

199541 2015-02-23 1900-01-01 10:00:00 5B 700 N STREEPER ST

207566 2014-12-08 1900-01-01 14:59:00 5A 700 N STREEPER ST

219128 2014-09-04 1900-01-01 08:00:00 4E 700 N STREEPER ST

229209 2014-06-16 1900-01-01 11:00:00 4E 700 N STREEPER ST

244967 2014-01-27 1900-01-01 11:00:00 5A 700 N STREEPER ST

Description Inside/Outside Post District Neighborhood \

1 LARCENY I 333.0 EASTERN MADISON EASTEND

182 LARCENY I 333.0 EASTERN MADISON EASTEND

183 LARCENY I 333.0 EASTERN MADISON EASTEND

289 COMMON ASSAULT I 333.0 EASTERN MADISON EASTEND

291 AGG. ASSAULT I 333.0 EASTERN MADISON EASTEND

8650 AGG. ASSAULT I 333.0 EASTERN MADISON EASTEND

14236 LARCENY I 333.0 EASTERN MADISON EASTEND

18026 COMMON ASSAULT I 333.0 EASTERN MADISON EASTEND

19347 COMMON ASSAULT I 333.0 EASTERN MADISON EASTEND

28084 AUTO THEFT O 333.0 EASTERN MADISON EASTEND

68820 BURGLARY O 333.0 EASTERN MADISON EASTEND

89061 AUTO THEFT O 333.0 EASTERN MADISON EASTEND

95285 LARCENY FROM AUTO O 333.0 EASTERN MADISON EASTEND

97222 COMMON ASSAULT I 333.0 EASTERN MADISON EASTEND

97223 COMMON ASSAULT I 333.0 EASTERN MADISON EASTEND

101780 LARCENY FROM AUTO O 333.0 EASTERN MADISON EASTEND

111908 BURGLARY I 333.0 EASTERN MADISON EASTEND

113781 BURGLARY I 333.0 EASTERN MADISON EASTEND

155908 LARCENY O 333.0 EASTERN MADISON EASTEND

199541 BURGLARY I 333 EASTERN MADISON EASTEND

207566 BURGLARY I 333 EASTERN MADISON EASTEND

219128 COMMON ASSAULT I 333.0 EASTERN MADISON EASTEND

229209 COMMON ASSAULT I 333.0 EASTERN MADISON EASTEND

244967 BURGLARY I 333.0 EASTERN MADISON EASTEND

Longitude Latitude Premise Total Incidents Year Month \

1 -76.577425 39.299094 ROW/TOWNHOUSE-OCC 1 2020 2

182 -76.577449 39.299529 ROW/TOWNHOUSE-OCC 1 2020 2

183 -76.577449 39.299529 ROW/TOWNHOUSE-OCC 1 2020 2

289 -76.577449 39.299529 ROW/TOWNHOUSE-OCC 1 2020 2

291 -76.577449 39.299529 ROW/TOWNHOUSE-OCC 1 2020 2

8650 -76.577444 39.299426 ROW/TOWNHOUSE-OCC 1 2019 10

14236 -76.577444 39.299426 ROW/TOWNHOUSE-OCC 1 2019 8

18026 -76.577449 39.299529 ROW/TOWNHOUSE-OCC 1 2019 7

19347 -76.577444 39.299426 ROW/TOWNHOUSE-OCC 1 2019 7

28084 -76.577425 39.299094 STREET 1 2019 4

68820 -76.577342 39.299641 YARD 1 2018 3

89061 -76.577425 39.299094 STREET 1 2017 9

95285 -76.577425 39.299094 STREET 1 2017 7

97222 -76.57731 39.299101 ROW/TOWNHOUSE-OCC 1 2017 6

97223 -76.57731 39.299101 ROW/TOWNHOUSE-OCC 1 2017 6

101780 -76.577449 39.299529 STREET 1 2017 5

111908 -76.577453 39.299596 ROW/TOWNHOUSE-OCC 1 2017 2

113781 -76.577333 39.299501 ROW/TOWNHOUSE-OCC 1 2017 1

155908 -76.577348 39.29971 STREET 1 2016 2

199541 -76.577459 39.299665 ROW/TOWNHOUSE-OCC 1 2015 2

207566 -76.577348 39.29971 ROW/TOWNHOUSE-OCC 1 2014 12

219128 -76.577345 39.299675 ROW/TOWNHOUSE-OCC 1 2014 9

229209 -76.577444 39.299426 ROW/TOWNHOUSE-OCC 1 2014 6

244967 -76.577466 39.299734 VACANT BUILDING 1 2014 1

DayOfWeek

1 Saturday

182 Wednesday

183 Wednesday

289 Monday

291 Monday

8650 Sunday

14236 Saturday

18026 Saturday

19347 Monday

28084 Wednesday

68820 Friday

89061 Sunday

95285 Wednesday

97222 Sunday

97223 Sunday

101780 Friday

111908 Thursday

113781 Sunday

155908 Tuesday

199541 Monday

207566 Monday

219128 Thursday

229209 Monday

244967 Monday , 'lat': 'Latitude', 'lon': 'Longitude', 'color': 'Description', 'text': None, 'hover\_name': 'Description', 'hover\_data': None, 'custom\_data': None, 'size': None, 'animation\_frame': None, 'animation\_group': None, 'category\_orders': None, 'labels': None, 'color\_discrete\_sequence': None, 'color\_discrete\_map': None, 'color\_continuous\_scale': None, 'range\_color': None, 'color\_continuous\_midpoint': None, 'opacity': None, 'size\_max': None, 'zoom': 10, 'center': None, 'mapbox\_style': None, 'title': None, 'template': None, 'width': None, 'height': None}

grouper = ['Description', <function one\_group at 0x000001B047FA1A60>]

2004

2005 col\_labels = []

~\anaconda3\lib\site-packages\plotly\express\\_core.py in get\_groups\_and\_orders(

args={'animation\_frame': None, 'animation\_group': None, 'category\_orders': None, 'center': None, 'color': 'Description', 'color\_continuous\_midpoint': None, 'color\_continuous\_scale': None, 'color\_discrete\_map': None, 'color\_discrete\_sequence': None, 'custom\_data': None, ...},

grouper=['Description', <function one\_group>]

)

1976 full\_sorted\_group\_names = [tuple(g) for g in full\_sorted\_group\_names]

1977

-> 1978 groups = {

groups = undefined

global sf = undefined

global grouped.get\_group = undefined

global s = undefined

global len = undefined

global zip = undefined

full\_sorted\_group\_names = [('ARSON', ''), ('HOMICIDE', ''), ('RAPE', ''), ('ROBBERY - CARJACKING', ''), ('ROBBERY - COMMERCIAL', ''), ('ROBBERY - RESIDENCE', ''), ('ROBBERY - STREET', ''), ('SHOOTING', ''), ('LARCENY', ''), ('COMMON ASSAULT', ''), ('AGG. ASSAULT', ''), ('AUTO THEFT', ''), ('BURGLARY', ''), ('LARCENY FROM AUTO', '')]

sorted\_group\_names = [('ARSON',), ('HOMICIDE',), ('RAPE',), ('ROBBERY - CARJACKING',), ('ROBBERY - COMMERCIAL',), ('ROBBERY - RESIDENCE',), ('ROBBERY - STREET',), ('SHOOTING',), ('LARCENY',), ('COMMON ASSAULT',), ('AGG. ASSAULT',), ('AUTO THEFT',), ('BURGLARY',), ('LARCENY FROM AUTO',)]

1979 sf: grouped.get\_group(s if len(s) > 1 else s[0])

1980 for sf, s in zip(full\_sorted\_group\_names, sorted\_group\_names)

~\anaconda3\lib\site-packages\plotly\express\\_core.py in <dictcomp>(.0=<zip object>)

1977

1978 groups = {

-> 1979 sf: grouped.get\_group(s if len(s) > 1 else s[0])

sf = ('ARSON', '')

global grouped.get\_group = undefined

s = ('ARSON',)

global len = undefined

1980 for sf, s in zip(full\_sorted\_group\_names, sorted\_group\_names)

1981 }

~\anaconda3\lib\site-packages\pandas\core\groupby\groupby.py in get\_group(

self=<pandas.core.groupby.generic.DataFrameGroupBy object>,

name='ARSON',

obj= Description Latitude Longitude

0 ...77444

23 BURGLARY 39.299734 -76.577466

)

745 inds = self.\_get\_index(name)

746 if not len(inds):

--> 747 raise KeyError(name)

global KeyError = undefined

name = 'ARSON'

748

749 return obj.\_take\_with\_is\_copy(inds, axis=self.axis)

KeyError: 'ARSON'

---------------------------------------------------------------------------

KeyError Traceback (most recent call last)

~\anaconda3\lib\site-packages\plotly\express\\_chart\_types.py in scatter\_mapbox(

data\_frame= CrimeDate CrimeTime CrimeCode ...43 6 Saturday

236880 4 Tuesday ,

lat='Latitude',

lon='Longitude',

color='Description',

text=None,

hover\_name='Description',

hover\_data=None,

custom\_data=None,

size=None,

animation\_frame=None,

animation\_group=None,

category\_orders=None,

labels=None,

color\_discrete\_sequence=None,

color\_discrete\_map=None,

color\_continuous\_scale=None,

range\_color=None,

color\_continuous\_midpoint=None,

opacity=None,

size\_max=None,

zoom=10,

center=None,

mapbox\_style=None,

title=None,

template=None,

width=None,

height=None

)

1245 symbol mark on a Mapbox map.

1246 """

-> 1247 return make\_figure(args=locals(), constructor=go.Scattermapbox)

global make\_figure = <function make\_figure at 0x000001B047FA23A0>

global args = undefined

global locals = undefined

global constructor = undefined

global go.Scattermapbox = <class 'plotly.graph\_objs.\_scattermapbox.Scattermapbox'>

1248

1249

~\anaconda3\lib\site-packages\plotly\express\\_core.py in make\_figure(

args={'animation\_frame': None, 'animation\_group': None, 'category\_orders': None, 'center': None, 'color': 'Description', 'color\_continuous\_midpoint': None, 'color\_continuous\_scale': None, 'color\_discrete\_map': None, 'color\_discrete\_sequence': None, 'custom\_data': None, ...},

constructor=<class 'plotly.graph\_objs.\_scattermapbox.Scattermapbox'>,

trace\_patch={'mode': 'markers'},

layout\_patch={}

)

2001 )

2002 grouper = [x.grouper or one\_group for x in grouped\_mappings] or [one\_group]

-> 2003 groups, orders = get\_groups\_and\_orders(args, grouper)

groups = undefined

global orders = undefined

global get\_groups\_and\_orders = <function get\_groups\_and\_orders at 0x000001B047FA2310>

args = {'data\_frame': CrimeDate CrimeTime CrimeCode Location \

4 2020-02-08 1900-01-01 17:30:00 5A 1600 E OLIVER ST

655 2020-01-30 1900-01-01 15:00:00 3JO 1600 E OLIVER ST

2940 2020-01-01 1900-01-01 17:20:00 4A 1600 E OLIVER ST

3930 2019-12-19 1900-01-01 23:00:00 5A 1600 E OLIVER ST

4166 2019-12-15 1900-01-01 01:30:00 6D 1600 E OLIVER ST

15851 2019-08-15 1900-01-01 22:00:00 6D 1600 E OLIVER ST

19141 2019-07-17 1900-01-01 14:00:00 5A 1600 E OLIVER ST

21559 2019-06-25 1900-01-01 14:00:00 7A 1600 E OLIVER ST

25248 2019-05-21 1900-01-01 02:07:00 3AF 1600 E OLIVER ST

32232 2019-03-10 1900-01-01 03:10:00 3AF 1600 E OLIVER ST

36009 2019-01-23 1900-01-01 15:30:00 6D 1600 E OLIVER ST

37417 2019-01-08 1900-01-01 20:25:00 3AJF 1600 E OLIVER ST

41145 2018-12-06 1900-01-01 16:45:00 3K 1600 E OLIVER ST

41978 2018-11-29 1900-01-01 23:23:00 3AF 1600 E OLIVER ST

50193 2018-09-22 1900-01-01 20:00:00 6E 1600 E OLIVER ST

71004 2018-02-19 1900-01-01 13:50:00 7A 1600 E OLIVER ST

82018 2017-11-02 1900-01-01 16:45:00 7A 1600 E OLIVER ST

87837 2017-09-15 1900-01-01 18:15:00 3AF 1600 E OLIVER ST

87957 2017-09-14 1900-01-01 18:15:00 3AF 1600 E OLIVER ST

93057 2017-07-31 1900-01-01 10:37:00 4E 1600 E OLIVER ST

93477 2017-07-27 1900-01-01 08:50:00 4D 1600 E OLIVER ST

119020 2016-12-02 1900-01-01 15:00:00 6D 1600 E OLIVER ST

127017 2016-10-06 1900-01-01 22:08:00 5A 1600 E OLIVER ST

130655 2016-09-09 1900-01-01 12:32:00 5C 1600 E OLIVER ST

145033 2016-05-21 1900-01-01 09:19:00 5A 1600 E OLIVER ST

165078 2015-11-22 1900-01-01 01:17:00 4E 1600 E OLIVER ST

190871 2015-05-13 1900-01-01 12:00:00 6E 1600 E OLIVER ST

215615 2014-10-02 1900-01-01 19:00:00 6F 1600 E OLIVER ST

221418 2014-08-17 1900-01-01 00:35:00 4B 1600 E OLIVER ST

230343 2014-06-07 1900-01-01 23:00:00 3AK 1600 E OLIVER ST

236880 2014-04-15 1900-01-01 11:00:00 5A 1600 E OLIVER ST

Description Inside/Outside Post District Neighborhood \

4 BURGLARY I 314.0 EASTERN OLIVER

655 ROBBERY - RESIDENCE I 314.0 EASTERN OLIVER

2940 AGG. ASSAULT O 314.0 EASTERN OLIVER

3930 BURGLARY I 314.0 EASTERN OLIVER

4166 LARCENY FROM AUTO O 314.0 EASTERN OLIVER

15851 LARCENY FROM AUTO O 314.0 EASTERN OLIVER

19141 BURGLARY I 314.0 EASTERN OLIVER

21559 AUTO THEFT O 314.0 EASTERN OLIVER

25248 ROBBERY - STREET O 314.0 EASTERN OLIVER

32232 ROBBERY - STREET O 314.0 EASTERN OLIVER

36009 LARCENY FROM AUTO O 314.0 EASTERN OLIVER

37417 ROBBERY - CARJACKING O 314.0 EASTERN OLIVER

41145 ROBBERY - RESIDENCE I 314.0 EASTERN OLIVER

41978 ROBBERY - STREET O 314.0 EASTERN OLIVER

50193 LARCENY O 314.0 EASTERN OLIVER

71004 AUTO THEFT O 314.0 EASTERN OLIVER

82018 AUTO THEFT O 314.0 EASTERN OLIVER

87837 ROBBERY - STREET I 314.0 EASTERN OLIVER

87957 ROBBERY - STREET I 314.0 EASTERN OLIVER

93057 COMMON ASSAULT I 314.0 EASTERN OLIVER

93477 AGG. ASSAULT O 314.0 EASTERN OLIVER

119020 LARCENY FROM AUTO O 314.0 EASTERN OLIVER

127017 BURGLARY I 314.0 EASTERN OLIVER

130655 BURGLARY I 314.0 EASTERN OLIVER

145033 BURGLARY I 314.0 EASTERN OLIVER

165078 COMMON ASSAULT I 314 EASTERN OLIVER

190871 LARCENY O 314 EASTERN OLIVER

215615 LARCENY O 314 EASTERN OLIVER

221418 AGG. ASSAULT O 314.0 EASTERN OLIVER

230343 ROBBERY - STREET O 314.0 EASTERN OLIVER

236880 BURGLARY I 314.0 EASTERN OLIVER

Longitude Latitude Premise Total Incidents Year \

4 -76.595946 39.30735 ROW/TOWNHOUSE-OCC 1 2020

655 -76.595946 39.30735 ROW/TOWNHOUSE-OCC 1 2020

2940 -76.595553 39.307366 STREET 1 2020

3930 -76.596706 39.307469 ROW/TOWNHOUSE-OCC 1 2019

4166 -76.596706 39.307469 STREET 1 2019

15851 -76.595946 39.30735 STREET 1 2019

19141 -76.595481 39.307517 ROW/TOWNHOUSE-OCC 1 2019

21559 -76.596706 39.307469 STREET 1 2019

25248 -76.595481 39.307517 STREET 1 2019

32232 -76.596706 39.307469 STREET 1 2019

36009 -76.596706 39.307469 STREET 1 2019

37417 -76.596706 39.307469 STREET 1 2019

41145 -76.596706 39.307469 ROW/TOWNHOUSE-OCC 1 2018

41978 -76.596706 39.307469 STREET 1 2018

50193 -76.596706 39.307469 STREET 1 2018

71004 -76.596706 39.307469 STREET 1 2018

82018 -76.596706 39.307469 STREET 1 2017

87837 -76.595481 39.307517 ROW/TOWNHOUSE-OCC 1 2017

87957 -76.595514 39.307368 ROW/TOWNHOUSE-OCC 1 2017

93057 -76.595978 39.307496 SINGLE HOUSE 1 2017

93477 -76.595592 39.307365 STREET 1 2017

119020 -76.595978 39.307496 STREET 1 2016

127017 -76.595946 39.30735 ROW/TOWNHOUSE-OCC 1 2016

130655 -76.595958 39.307349 ROW/TOWNHOUSE-OCC 1 2016

145033 -76.595592 39.307365 OTHER - INSIDE 1 2016

165078 -76.595946 39.30735 APT/CONDO - OCCUPIED 1 2015

190871 -76.596706 39.307469 STREET 1 2015

215615 -76.596706 39.307469 STREET 1 2014

221418 -76.596706 39.307469 STREET 1 2014

230343 -76.596706 39.307469 STREET 1 2014

236880 -76.59567 39.307362 ROW/TOWNHOUSE-OCC 1 2014

Month DayOfWeek

4 2 Saturday

655 1 Thursday

2940 1 Wednesday

3930 12 Thursday

4166 12 Sunday

15851 8 Thursday

19141 7 Wednesday

21559 6 Tuesday

25248 5 Tuesday

32232 3 Sunday

36009 1 Wednesday

37417 1 Tuesday

41145 12 Thursday

41978 11 Thursday

50193 9 Saturday

71004 2 Monday

82018 11 Thursday

87837 9 Friday

87957 9 Thursday

93057 7 Monday

93477 7 Thursday

119020 12 Friday

127017 10 Thursday

130655 9 Friday

145033 5 Saturday

165078 11 Sunday

190871 5 Wednesday

215615 10 Thursday

221418 8 Sunday

230343 6 Saturday

236880 4 Tuesday , 'lat': 'Latitude', 'lon': 'Longitude', 'color': 'Description', 'text': None, 'hover\_name': 'Description', 'hover\_data': None, 'custom\_data': None, 'size': None, 'animation\_frame': None, 'animation\_group': None, 'category\_orders': None, 'labels': None, 'color\_discrete\_sequence': None, 'color\_discrete\_map': None, 'color\_continuous\_scale': None, 'range\_color': None, 'color\_continuous\_midpoint': None, 'opacity': None, 'size\_max': None, 'zoom': 10, 'center': None, 'mapbox\_style': None, 'title': None, 'template': None, 'width': None, 'height': None}

grouper = ['Description', <function one\_group at 0x000001B047FA1A60>]

2004

2005 col\_labels = []

~\anaconda3\lib\site-packages\plotly\express\\_core.py in get\_groups\_and\_orders(

args={'animation\_frame': None, 'animation\_group': None, 'category\_orders': None, 'center': None, 'color': 'Description', 'color\_continuous\_midpoint': None, 'color\_continuous\_scale': None, 'color\_discrete\_map': None, 'color\_discrete\_sequence': None, 'custom\_data': None, ...},

grouper=['Description', <function one\_group>]

)

1976 full\_sorted\_group\_names = [tuple(g) for g in full\_sorted\_group\_names]

1977

-> 1978 groups = {

groups = undefined

global sf = undefined

global grouped.get\_group = undefined

global s = undefined

global len = undefined

global zip = undefined

full\_sorted\_group\_names = [('ARSON', ''), ('HOMICIDE', ''), ('RAPE', ''), ('ROBBERY - COMMERCIAL', ''), ('SHOOTING', ''), ('BURGLARY', ''), ('ROBBERY - RESIDENCE', ''), ('AGG. ASSAULT', ''), ('LARCENY FROM AUTO', ''), ('AUTO THEFT', ''), ('ROBBERY - STREET', ''), ('ROBBERY - CARJACKING', ''), ('LARCENY', ''), ('COMMON ASSAULT', '')]

sorted\_group\_names = [('ARSON',), ('HOMICIDE',), ('RAPE',), ('ROBBERY - COMMERCIAL',), ('SHOOTING',), ('BURGLARY',), ('ROBBERY - RESIDENCE',), ('AGG. ASSAULT',), ('LARCENY FROM AUTO',), ('AUTO THEFT',), ('ROBBERY - STREET',), ('ROBBERY - CARJACKING',), ('LARCENY',), ('COMMON ASSAULT',)]

1979 sf: grouped.get\_group(s if len(s) > 1 else s[0])

1980 for sf, s in zip(full\_sorted\_group\_names, sorted\_group\_names)

~\anaconda3\lib\site-packages\plotly\express\\_core.py in <dictcomp>(.0=<zip object>)

1977

1978 groups = {

-> 1979 sf: grouped.get\_group(s if len(s) > 1 else s[0])

sf = ('ARSON', '')

global grouped.get\_group = undefined

s = ('ARSON',)

global len = undefined

1980 for sf, s in zip(full\_sorted\_group\_names, sorted\_group\_names)

1981 }

~\anaconda3\lib\site-packages\pandas\core\groupby\groupby.py in get\_group(

self=<pandas.core.groupby.generic.DataFrameGroupBy object>,

name='ARSON',

obj= Description Latitude Longitude

0...06

30 BURGLARY 39.307362 -76.59567

)

745 inds = self.\_get\_index(name)

746 if not len(inds):

--> 747 raise KeyError(name)

global KeyError = undefined

name = 'ARSON'

748

749 return obj.\_take\_with\_is\_copy(inds, axis=self.axis)

KeyError: 'ARSON'

ChatGPT

It looks like you're trying to create a scatter map using Plotly Express with a `Description` column as the color parameter. However, you encountered a KeyError with the value 'ARSON' when trying to group your data by 'Description'. This error is likely because there is no 'ARSON' value in your 'Description' column.

To resolve this issue, you can do the following:

1. Check your data: Verify that your dataset indeed contains a 'Description' column and check for any typos or inconsistencies in the values.

2. Update the color parameter: If you want to use the 'Description' column as the color parameter, make sure the values in that column are valid and exist in your dataset.

3. Grouping: If you intend to group your data by 'Description', ensure that there are no missing or unexpected values in the 'Description' column.

Here's an example of how you can create a scatter map using Plotly Express:

```python

import plotly.express as px

# Assuming df is your DataFrame

fig = px.scatter\_mapbox(df, lat='Latitude', lon='Longitude', color='Description', hover\_name='Description', zoom=10)

fig.update\_layout(mapbox\_style="open-street-map")

fig.show()

```

Make sure your data is clean and that the 'Description' column contains valid values to resolve this issue.

User

write the full updated code including above code

ChatGPT

Certainly! Below is the full updated code, including the previous code for loading data and the code for creating the scatter mapbox plot:

```python

# Import necessary libraries

import pandas as pd

import plotly.express as px

# Load the dataset

data = pd.read\_csv('crime\_data.csv')

# Ensure the 'CrimeDate' column is in datetime format

data['CrimeDate'] = pd.to\_datetime(data['CrimeDate'])

# Extract the year, month, and day of the week from the 'CrimeDate' column

data['Year'] = data['CrimeDate'].dt.year

data['Month'] = data['CrimeDate'].dt.month

data['DayOfWeek'] = data['CrimeDate'].dt.strftime('%A')

# Create a scatter mapbox plot

fig = px.scatter\_mapbox(data,

lat='Latitude',

lon='Longitude',

color='Description',

text=None,

hover\_name='Description',

hover\_data=None,

custom\_data=None,

size=None,

animation\_frame=None,

animation\_group=None,

category\_orders=None,

labels=None,

color\_discrete\_sequence=None,

color\_discrete\_map=None,

color\_continuous\_scale=None,

range\_color=None,

color\_continuous\_midpoint=None,

opacity=None,

size\_max=None,

zoom=10,

center=None,

mapbox\_style=None,

title='Crime Data in Baltimore',

template=None,

width=None,

height=None)

# Show the plot

fig.show()

```

Make sure to replace `'crime\_data.csv'` with the actual file path or URL of your dataset. This code will load the dataset, preprocess it by converting the 'CrimeDate' column to datetime, extract additional date-related information, and create a scatter mapbox plot using Plotly Express.

User

import dash

from dash import dcc, html

from dash.dependencies import Input, Output

import pandas as pd

import plotly.express as px

# Sample dataset (replace with your own data)

df = pd.read\_csv('crime\_data.csv')

# Initialize the Dash app

app = dash.Dash(\_\_name\_\_)

# Layout of the dashboard

app.layout = html.Div([

html.H1("Crime Dashboard"),

dcc.DatePickerRange(

id='date-range-picker',

start\_date=df['CrimeDate'].min(),

end\_date=df['CrimeDate'].max(),

display\_format='YYYY-MM-DD',

),

dcc.Dropdown(

id='location-dropdown',

options=[

{'label': location, 'value': location}

for location in df['Location'].unique()

],

multi=True,

placeholder="Select Location(s)",

),

dcc.Graph(id='crime-map'),

])

# Callback to update the map based on selected date range and location(s)

@app.callback(

Output('crime-map', 'figure'),

[Input('date-range-picker', 'start\_date'),

Input('date-range-picker', 'end\_date'),

Input('location-dropdown', 'value')]

)

def update\_map(start\_date, end\_date, selected\_locations):

filtered\_df = df[(df['CrimeDate'] >= start\_date) & (df['CrimeDate'] <= end\_date)]

if selected\_locations:

filtered\_df = filtered\_df[filtered\_df['Location'].isin(selected\_locations)]

fig = px.scatter\_mapbox(

filtered\_df,

lat='Latitude',

lon='Longitude',

hover\_name='Description',

color='Description',

zoom=10,

)

fig.update\_layout(mapbox\_style="carto-positron")

return fig

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

update this code while including above code

ChatGPT

To include the code for creating a scatter mapbox plot in your Dash app, you can integrate the provided scatter mapbox code into your existing Dash app code. Here's the complete code:

```python

import dash

from dash import dcc, html

from dash.dependencies import Input, Output

import pandas as pd

import plotly.express as px

# Sample dataset (replace with your own data)

df = pd.read\_csv('crime\_data.csv')

# Initialize the Dash app

app = dash.Dash(\_\_name\_\_)

# Layout of the dashboard

app.layout = html.Div([

html.H1("Crime Dashboard"),

dcc.DatePickerRange(

id='date-range-picker',

start\_date=df['CrimeDate'].min(),

end\_date=df['CrimeDate'].max(),

display\_format='YYYY-MM-DD',

),

dcc.Dropdown(

id='location-dropdown',

options=[

{'label': location, 'value': location}

for location in df['Location'].unique()

],

multi=True,

placeholder="Select Location(s)",

),

dcc.Graph(id='crime-map'),

])

# Callback to update the map based on selected date range and location(s)

@app.callback(

Output('crime-map', 'figure'),

[Input('date-range-picker', 'start\_date'),

Input('date-range-picker', 'end\_date'),

Input('location-dropdown', 'value')]

)

def update\_map(start\_date, end\_date, selected\_locations):

filtered\_df = df[(df['CrimeDate'] >= start\_date) & (df['CrimeDate'] <= end\_date)]

if selected\_locations:

filtered\_df = filtered\_df[filtered\_df['Location'].isin(selected\_locations)]

fig = px.scatter\_mapbox(

filtered\_df,

lat='Latitude',

lon='Longitude',

hover\_name='Description',

color='Description',

zoom=10,

)

fig.update\_layout(mapbox\_style="carto-positron")

return fig

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

```

This code combines the existing Dash app with the scatter mapbox plot. Make sure to replace `'crime\_data.csv'` with the actual file path or URL of your dataset. The map will update based on the selected date range and location(s) using the provided callback function.

User

there is callback error occuring for above code

153888 3 Monday

155015 2 Thursday

156230 2 Thursday

158599 1 Saturday

168769 10 Saturday

181393 7 Tuesday

191195 5 Sunday

199201 2 Friday

204049 1 Friday

218209 9 Friday

225598 7 Tuesday

230051 6 Monday , 'lat': 'Latitude', 'lon': 'Longitude', 'color': 'Description', 'text': None, 'hover\_name': 'Description', 'hover\_data': None, 'custom\_data': None, 'size': None, 'animation\_frame': None, 'animation\_group': None, 'category\_orders': None, 'labels': None, 'color\_discrete\_sequence': None, 'color\_discrete\_map': None, 'color\_continuous\_scale': None, 'range\_color': None, 'color\_continuous\_midpoint': None, 'opacity': None, 'size\_max': None, 'zoom': 10, 'center': None, 'mapbox\_style': None, 'title': None, 'template': None, 'width': None, 'height': None}

grouper = ['Description', <function one\_group at 0x000002030CACBD30>]

2004

2005 col\_labels = []

~\anaconda3\lib\site-packages\plotly\express\\_core.py in get\_groups\_and\_orders(

args={'animation\_frame': None, 'animation\_group': None, 'category\_orders': None, 'center': None, 'color': 'Description', 'color\_continuous\_midpoint': None, 'color\_continuous\_scale': None, 'color\_discrete\_map': None, 'color\_discrete\_sequence': None, 'custom\_data': None, ...},

grouper=['Description', <function one\_group>]

)

1976 full\_sorted\_group\_names = [tuple(g) for g in full\_sorted\_group\_names]

1977

-> 1978 groups = {

groups = undefined

global sf = undefined

global grouped.get\_group = undefined

global s = undefined

global len = undefined

global zip = undefined

full\_sorted\_group\_names = [('ARSON', ''), ('HOMICIDE', ''), ('RAPE', ''), ('ROBBERY - CARJACKING', ''), ('ROBBERY - RESIDENCE', ''), ('SHOOTING', ''), ('LARCENY', ''), ('BURGLARY', ''), ('AUTO THEFT', ''), ('ROBBERY - STREET', ''), ('AGG. ASSAULT', ''), ('COMMON ASSAULT', ''), ('LARCENY FROM AUTO', ''), ('ROBBERY - COMMERCIAL', '')]

sorted\_group\_names = [('ARSON',), ('HOMICIDE',), ('RAPE',), ('ROBBERY - CARJACKING',), ('ROBBERY - RESIDENCE',), ('SHOOTING',), ('LARCENY',), ('BURGLARY',), ('AUTO THEFT',), ('ROBBERY - STREET',), ('AGG. ASSAULT',), ('COMMON ASSAULT',), ('LARCENY FROM AUTO',), ('ROBBERY - COMMERCIAL',)]

1979 sf: grouped.get\_group(s if len(s) > 1 else s[0])

1980 for sf, s in zip(full\_sorted\_group\_names, sorted\_group\_names)

~\anaconda3\lib\site-packages\plotly\express\\_core.py in <dictcomp>(.0=<zip object>)

1977

1978 groups = {

-> 1979 sf: grouped.get\_group(s if len(s) > 1 else s[0])

sf = ('ARSON', '')

global grouped.get\_group = undefined

s = ('ARSON',)

global len = undefined

1980 for sf, s in zip(full\_sorted\_group\_names, sorted\_group\_names)

1981 }

~\anaconda3\lib\site-packages\pandas\core\groupby\groupby.py in get\_group(

self=<pandas.core.groupby.generic.DataFrameGroupBy object>,

name='ARSON',

obj= Description Latitude Longitude

0...42

41 ROBBERY - STREET 39.288746 -76.614193

)

745 inds = self.\_get\_index(name)

746 if not len(inds):

--> 747 raise KeyError(name)

global KeyError = undefined

name = 'ARSON'

748

749 return obj.\_take\_with\_is\_copy(inds, axis=self.axis)

KeyError: 'ARSON'

ChatGPT

The error message you're encountering is a KeyError, indicating that the 'ARSON' value is not found in the 'Description' column of your filtered dataset when trying to create the scatter mapbox plot. This issue may occur if the selected date range or location(s) filter results in an empty dataset for certain categories.

To handle this error gracefully and prevent the app from crashing, you can modify the code as follows:

```python

@app.callback(

Output('crime-map', 'figure'),

[Input('date-range-picker', 'start\_date'),

Input('date-range-picker', 'end\_date'),

Input('location-dropdown', 'value')]

)

def update\_map(start\_date, end\_date, selected\_locations):

filtered\_df = df[(df['CrimeDate'] >= start\_date) & (df['CrimeDate'] <= end\_date)]

if selected\_locations:

filtered\_df = filtered\_df[filtered\_df['Location'].isin(selected\_locations)]

# Check if the filtered\_df is empty

if filtered\_df.empty:

# Return an empty map or a message indicating no data

fig = px.scatter()

else:

fig = px.scatter\_mapbox(

filtered\_df,

lat='Latitude',

lon='Longitude',

hover\_name='Description',

color='Description',

zoom=10,

)

fig.update\_layout(mapbox\_style="carto-positron")

return fig

```

In this modified code, we first check if the filtered dataset `filtered\_df` is empty. If it's empty (meaning there is no data that matches the selected date range and location(s)), we return an empty scatter plot by calling `px.scatter()`. This change prevents the KeyError and provides a more user-friendly response when there's no data to display on the map.

User

please give full modified code

ChatGPT

Certainly, here's the full modified code that includes error handling to prevent a KeyError when there's no data to display:

```python

import dash

from dash import dcc, html

from dash.dependencies import Input, Output

import pandas as pd

import plotly.express as px

# Sample dataset (replace with your own data)

df = pd.read\_csv('crime\_data.csv')

# Initialize the Dash app

app = dash.Dash(\_\_name\_\_)

# Layout of the dashboard

app.layout = html.Div([

html.H1("Crime Dashboard"),

dcc.DatePickerRange(

id='date-range-picker',

start\_date=df['CrimeDate'].min(),

end\_date=df['CrimeDate'].max(),

display\_format='YYYY-MM-DD',

),

dcc.Dropdown(

id='location-dropdown',

options=[

{'label': location, 'value': location}

for location in df['Location'].unique()

],

multi=True,

placeholder="Select Location(s)",

),

dcc.Graph(id='crime-map'),

])

# Callback to update the map based on selected date range and location(s)

@app.callback(

Output('crime-map', 'figure'),

[Input('date-range-picker', 'start\_date'),

Input('date-range-picker', 'end\_date'),

Input('location-dropdown', 'value')]

)

def update\_map(start\_date, end\_date, selected\_locations):

filtered\_df = df[(df['CrimeDate'] >= start\_date) & (df['CrimeDate'] <= end\_date)]

if selected\_locations:

filtered\_df = filtered\_df[filtered\_df['Location'].isin(selected\_locations)]

# Check if the filtered\_df is empty

if filtered\_df.empty:

# Return an empty map or a message indicating no data

fig = px.scatter()

else:

fig = px.scatter\_mapbox(

filtered\_df,

lat='Latitude',

lon='Longitude',

hover\_name='Description',

color='Description',

zoom=10,

)

fig.update\_layout(mapbox\_style="carto-positron")

return fig

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

```

This code should prevent the KeyError and display an empty scatter plot or message when there's no data to display based on the selected date range and location(s).

User

plot the map more detailed and realistic for above code

ChatGPT

To create a more detailed and realistic map in Dash using Plotly Express, you can customize the `scatter\_mapbox` chart with additional map-related parameters. Here's an updated version of your code with a more detailed and realistic map:

```python

import dash

from dash import dcc, html

from dash.dependencies import Input, Output

import pandas as pd

import plotly.express as px

# Sample dataset (replace with your own data)

df = pd.read\_csv('crime\_data.csv')

# Initialize the Dash app

app = dash.Dash(\_\_name\_\_)

# Layout of the dashboard

app.layout = html.Div([

html.H1("Crime Dashboard"),

dcc.DatePickerRange(

id='date-range-picker',

start\_date=df['CrimeDate'].min(),

end\_date=df['CrimeDate'].max(),

display\_format='YYYY-MM-DD',

),

dcc.Dropdown(

id='location-dropdown',

options=[

{'label': location, 'value': location}

for location in df['Location'].unique()

],

multi=True,

placeholder="Select Location(s)",

),

dcc.Graph(id='crime-map'),

])

# Callback to update the map based on selected date range and location(s)

@app.callback(

Output('crime-map', 'figure'),

[Input('date-range-picker', 'start\_date'),

Input('date-range-picker', 'end\_date'),

Input('location-dropdown', 'value')]

)

def update\_map(start\_date, end\_date, selected\_locations):

filtered\_df = df[(df['CrimeDate'] >= start\_date) & (df['CrimeDate'] <= end\_date)]

if selected\_locations:

filtered\_df = filtered\_df[filtered\_df['Location'].isin(selected\_locations)]

# Check if the filtered\_df is empty

if filtered\_df.empty:

# Return an empty map or a message indicating no data

fig = px.scatter()

else:

fig = px.scatter\_mapbox(

filtered\_df,

lat='Latitude',

lon='Longitude',

hover\_name='Description',

color='Description',

zoom=12, # Adjust the zoom level for a more detailed map

)

fig.update\_layout(

mapbox\_style="carto-positron",

mapbox\_zoom=10, # You can adjust this value for initial zoom level

mapbox\_center={"lat": filtered\_df['Latitude'].mean(), "lon": filtered\_df['Longitude'].mean()}, # Center the map on data

)

return fig

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

```

In this updated code:

- I increased the `zoom` level in `px.scatter\_mapbox` to make the map more detailed.

- I set the initial `mapbox\_zoom` level to 10 and centered the map on the average latitude and longitude of the filtered data for a more realistic view of the data.

- You can further customize the map's appearance by adjusting the `mapbox\_style`, `mapbox\_center`, and other map-related parameters as needed.

Make sure to replace the sample dataset and adjust the map settings to fit your specific data and requirements.

User

# Group crimes by year and count incidents

yearly\_crime\_count = df.groupby(df['CrimeDate'].dt.year).size()

# Create a plot for year-to-year crime incidence

plt.figure(figsize=(12, 6))

plt.plot(yearly\_crime\_count.index, yearly\_crime\_count.values, marker='o', linestyle='-', color='b')

plt.title('Year-to-Year Crime Incidence')

plt.xlabel('Year')

plt.ylabel('Total Incidents')

plt.grid(True)

plt.show()

x axis should be more detail

ChatGPT

To make the x-axis more detailed for year-to-year crime incidence, you can use a `Locator` and a `Formatter` from the `matplotlib` library to control the tick placement and format. Here's an updated version of your code with a more detailed x-axis:

```python

import dash

from dash import dcc, html

from dash.dependencies import Input, Output

import pandas as pd

import plotly.express as px

import matplotlib.pyplot as plt

import matplotlib.dates as mdates # Import date-related functionality

# Sample dataset (replace with your own data)

df = pd.read\_csv('crime\_data.csv')

# Initialize the Dash app

app = dash.Dash(\_\_name\_\_)

# Layout of the dashboard

app.layout = html.Div([

html.H1("Crime Dashboard"),

dcc.DatePickerRange(

id='date-range-picker',

start\_date=df['CrimeDate'].min(),

end\_date=df['CrimeDate'].max(),

display\_format='YYYY-MM-DD',

),

dcc.Dropdown(

id='location-dropdown',

options=[

{'label': location, 'value': location}

for location in df['Location'].unique()

],

multi=True,

placeholder="Select Location(s)",

),

dcc.Graph(id='crime-map'),

dcc.Graph(id='yearly-crime-count') # Add a separate graph for yearly crime count

])

# Callback to update the map based on selected date range and location(s)

@app.callback(

Output('crime-map', 'figure'),

[Input('date-range-picker', 'start\_date'),

Input('date-range-picker', 'end\_date'),

Input('location-dropdown', 'value')]

)

def update\_map(start\_date, end\_date, selected\_locations):

filtered\_df = df[(df['CrimeDate'] >= start\_date) & (df['CrimeDate'] <= end\_date)]

if selected\_locations:

filtered\_df = filtered\_df[filtered\_df['Location'].isin(selected\_locations)]

# Check if the filtered\_df is empty

if filtered\_df.empty:

# Return an empty map or a message indicating no data

fig = px.scatter()

else:

fig = px.scatter\_mapbox(

filtered\_df,

lat='Latitude',

lon='Longitude',

hover\_name='Description',

color='Description',

zoom=12, # Adjust the zoom level for a more detailed map

)

fig.update\_layout(

mapbox\_style="carto-positron",

mapbox\_zoom=10, # You can adjust this value for initial zoom level

mapbox\_center={"lat": filtered\_df['Latitude'].mean(), "lon": filtered\_df['Longitude'].mean()}, # Center the map on data

)

return fig

# Callback to update the yearly crime count plot

@app.callback(

Output('yearly-crime-count', 'figure'),

[Input('date-range-picker', 'start\_date'),

Input('date-range-picker', 'end\_date')]

)

def update\_yearly\_crime\_count(start\_date, end\_date):

filtered\_df = df[(df['CrimeDate'] >= start\_date) & (df['CrimeDate'] <= end\_date)]

yearly\_crime\_count = filtered\_df.groupby(filtered\_df['CrimeDate'].dt.year).size()

# Create a plot for year-to-year crime incidence

plt.figure(figsize=(12, 6))

plt.plot(yearly\_crime\_count.index, yearly\_crime\_count.values, marker='o', linestyle='-', color='b')

plt.title('Year-to-Year Crime Incidence')

plt.xlabel('Year')

plt.ylabel('Total Incidents')

plt.grid(True)

# Format the x-axis with a more detailed date format

ax = plt.gca()

ax.xaxis.set\_major\_locator(mdates.YearLocator()) # Major tick every year

ax.xaxis.set\_major\_formatter(mdates.DateFormatter('%Y')) # Format as year (e.g., 2022)

plt.xticks(rotation=45) # Rotate x-axis labels for better readability

return plt

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

```

In this updated code:

- I added a new `dcc.Graph` component named `'yearly-crime-count'` to display the year-to-year crime incidence.

- I created a separate callback (`update\_yearly\_crime\_count`) to generate the yearly crime count plot.

- Inside the `update\_yearly\_crime\_count` callback, I used the `mdates` module to format the x-axis with more detailed year labels, and I rotated the x-axis labels for better readability.

This should provide a more detailed x-axis for your year-to-year crime incidence plot.