

CREATING EFFICIENCY WITH GOOGLE COLAB

HOUR ONE

**SUZANNA HOWARD
WEST JORDAN (UT) POLICE DEPARTMENT**

```
// life motto
if (sad() === true) {
  sad().stop();
  beAwesome();
}
```

Google Colaboratory, commonly known as Google Colab, is a free, cloud-based Jupyter Notebook environment provided by Google. It enables users to write and execute Python code directly in their web browser, making it particularly useful for machine learning, data analysis, and educational purposes.

Key features and aspects of Google Colab include:

Cloud-based Environment:

It runs entirely in the cloud, eliminating the need for local setup or powerful hardware.

Free Access to Resources:

Users can access computing resources, including GPUs and TPUs, which are beneficial for computationally intensive tasks like deep learning.

Jupyter Notebook Compatibility:

Colab notebooks are compatible with Jupyter notebooks, allowing for the combination of executable code, rich text, images, and other media within a single document.

Integration with Google Drive:

It seamlessly integrates with Google Drive for easy saving, sharing, and collaboration on notebooks.

AI-First Features:

Recent advancements have introduced AI-powered capabilities, such as iterative querying for code generation and explanation, a Next-Generation Data Science Agent for autonomous analysis, and effortless code transformation using natural language.

Data Import and Analysis:

Users can easily import data from various sources like Google Drive, GitHub, and spreadsheets, and leverage popular Python libraries for data analysis and visualization.

Collaboration:

Notebooks can be shared and collaborated on with multiple users, with access controlled through Identity and Access Management (IAM) for Colab Enterprise.

Paid Services:

While a free tier is available, Google also offers paid services like Colab Pro, Colab Pro+, and Colab Enterprise, which provide enhanced resources and features like faster

Use Google Colab online to build Jupyter Notebooks.

- 1. Drag and drop an Excel file into Google Colab**
- 2. Use AI-generated Python code to write the steps we would do in Excel (sort, insert columns, formulas, clean data, etc)**
- 3. Export files**
- 4. Save/download Jupyter Notebook for future use**

In Excel terms, it is similar to macros.



HOUR ONE

INTRODUCTION TO GOOGLE COLAB

CAVEATS

ACTIVITIES:

DANGEROUS INTERSECTIONS

GRAPHS / CHARTS

UNPIVOT

HOUR TWO

INTRODUCTION TO GOOGLE COLAB

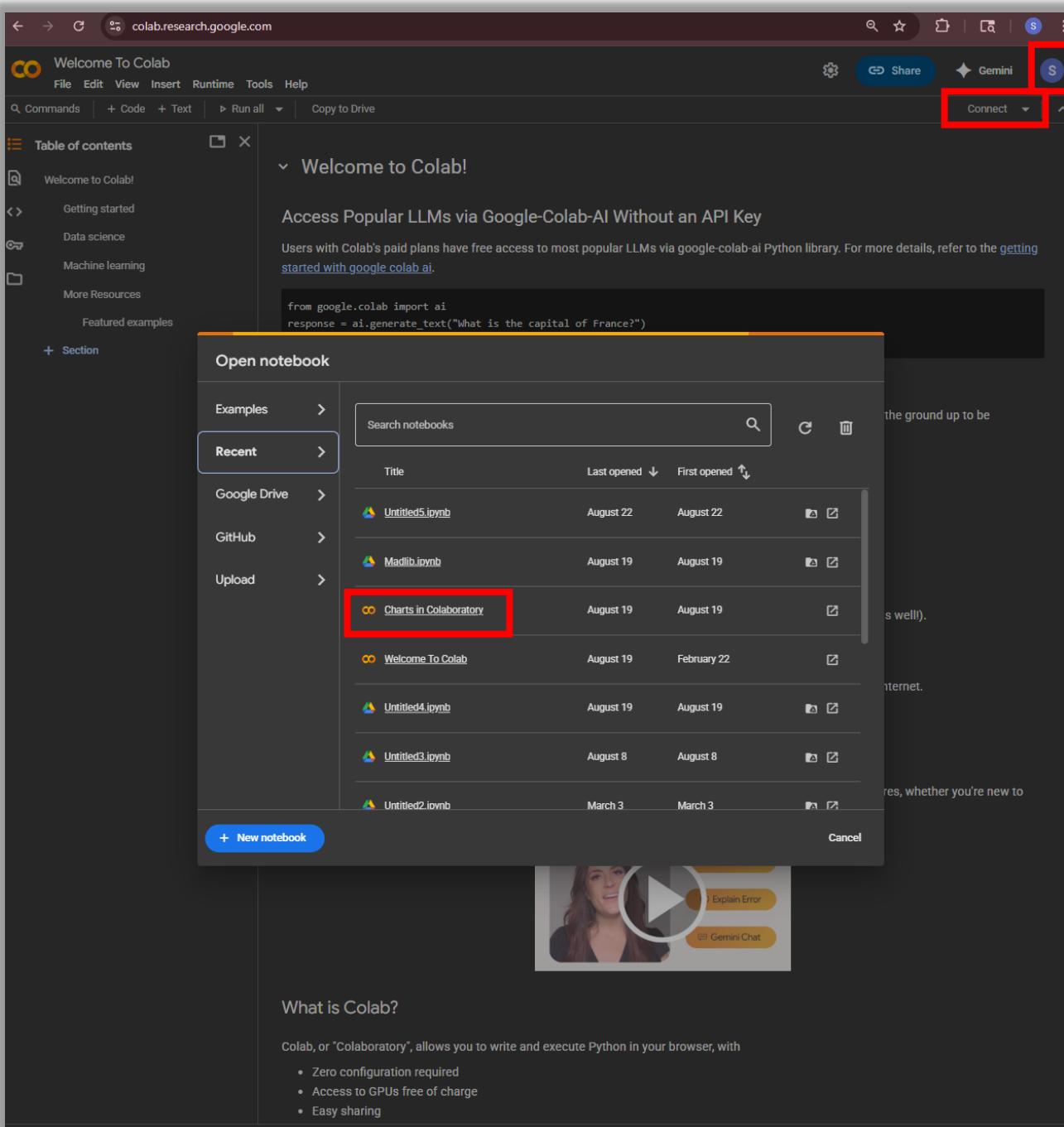
CAVEATS

ACTIVITIES:

OFFICER AVAILABILITY

FORECASTING / EDUCATED GUESSING

GRAPHS / CHARTS



Google “Google Colab”
<https://colab.research.google.com/>
Type: Colab.new

GMAIL ACCOUNT REQUIRED

FREE ONLINE TOOL

NO INSTALLATION/CONFIGURATION

JUPYTER NOTEBOOKS

**AI GENERATED PYTHON CODE
GEMINI
CHAT GPT
COMMENT**



Google what does google colab do with my data

AI Mode All Videos Images Short videos Shopping Forums Other Tools

◆ AI Overview

Google Colab handles user data in several ways, primarily related to the execution of Python code and the storage of notebooks.

Data Handling:

Notebook Storage:

Colab notebooks are, by default, stored in your Google Drive, specifically in a "Colab Notebooks" folder. This means the notebook content itself, including code and markdown cells, is saved within your Google Drive storage.

Execution Environment:

When you run code in a Colab notebook, it executes on Google's cloud servers. Any data that your code processes or generates during execution (e.g., loaded datasets, generated outputs, model weights) resides within this cloud environment.

Google Drive Integration:

You can explicitly mount your Google Drive within a Colab notebook to access files stored there. This allows your code to read from and write to your Google Drive, but this access is granted only when you explicitly authorize it.

Generative AI Features:

If you utilize generative AI features within Colab, Google collects data related to prompts, generated code, output, feature usage information, and feedback. This data is used to improve and develop Google products and services, including machine learning technologies. Google states this data is stored in a way that prevents personal identification and is retained for a limited period. ☀

Security and Privacy Considerations:

Shared Notebooks:

When sharing a Colab notebook, be aware that if the notebook contains malicious code and you execute it, it could potentially access your Google Drive data if you have mounted it and granted permissions.

Data Access:

Colab AI features do not have access to your Google Drive files or user secrets by default. Access is only granted if you explicitly request it within your code. ☀

Encryption and Transparency (Colab Enterprise):

For enterprise users, Colab Enterprise offers features like customer-managed encryption keys (CMEK) and Access Transparency, providing more control and visibility over data handling and access by Google personnel.

People also ask :

Does Google Colab use my data?

By default, Colab AI does not have access to your Google Drive files or user secrets.

However, it can generate code that accesses them at your explicit request.

Google Research

<https://research.google.com> > colaboratory > faq

Google Colab

Does Google Colab have access to my files?

Since a Colab notebook is hosted on Google's cloud servers, [there's no direct access to files on your local drive](#) (unlike a notebook hosted on your machine) or any other environment by default. However, Colab provides various options to connect to almost any data source you can imagine. Mar 14, 2025

Does Google keep sell my data?

We never sell your personal information, and give you controls over who has access. We are committed to protecting your data from third parties. That's why it's our strict policy to never sell your personal information to anyone.

Google Safety Center

<https://safety.google> > privacy > data

Your privacy is protected by responsible data practices.

Can Google see everything I do?

If You Use Google Products

If you do use Google products, they try to track even more. In addition to tracking everything you've ever searched for on Google (e.g. "weird rash"), Google also tracks every video you've ever watched on YouTube. Many people actually don't know that Google owns YouTube; now you know.

What does Google do with all your data?

Google uses the information shared by sites and apps to deliver our services, maintain and improve them, develop new services, measure the effectiveness of advertising, protect against fraud and abuse, and personalize content and ads you see on Google and on our partners' sites and apps.

Google Policies

<https://policies.google.com> > technologies > partner-sites

How Google uses information from sites or apps that use our services

Does Google access your files?

We access your private content only when we have your permission or are required to by law. With the Google Transparency Report, we share data about how the policies and actions of governments and corporations affect privacy, security, and access to information.

Does Google take your personal data?

First, we make sure that it's securely processed and stored as stated in the Google Privacy Policy. We use your personal information solely for the purpose of running your account and communicating account information to you.

Google Help

<https://support.google.com> > adsense > answer

What does Google do with my personal information?

Does Google keep your data safe?

Data is encrypted in-transit and at-rest. If you choose to access these files offline, we store this info on your device. Your Google Account comes with built-in security designed to detect and block threats like spam, phishing and malware. Your activity is stored using strong industry standards and practices.

ChatGPT

what information should i avoid uploading to google colab?

When using **Google Colab**, you should avoid uploading any sensitive, private, or regulated data. While Google Colab is a secure platform, it's still a cloud-based environment, and you should follow best practices to protect yourself and others. Specifically, **do not upload**:

🚫 Personal Identifiable Information (PII):

- Full names linked with other personal data
- Social Security Numbers (SSNs)
- Driver's license numbers
- Passport numbers
- Home addresses, email addresses, or phone numbers (if sensitive)

🚫 Financial Information:

- Credit card or bank account numbers
- Financial transaction records tied to individuals
- Tax documents

🚫 Login Credentials or Authentication Tokens:

- Passwords (even hashed ones)
- API keys or OAuth tokens
- SSH private keys
- Firebase or AWS credentials
- Google Cloud service account keys

🚫 Medical/Health Data (HIPAA-covered info):

- Patient health records
- Medical history
- Lab/test results tied to a person

🚫 Confidential or Proprietary Business Information:

- Internal strategy documents
- Trade secrets
- Non-public client or company data

🚫 Government Classified or Export-Controlled Data (e.g., ITAR, EAR):

- Data subject to national security or export restrictions



✅ Safe to Upload:

- Public datasets
- Open-source code
- Your own hobby/educational project data (that doesn't contain sensitive info)
- Anonymized or synthetic data for experiments

⚠️ Tips to Stay Safe:

- Use `.gitignore` and avoid committing secrets to notebooks
- Use environment variables for secrets (e.g., `os.environ`)
- Use Colab's "secrets" management with Google Drive or GCP, if needed
- Revoke API keys if you accidentally upload them

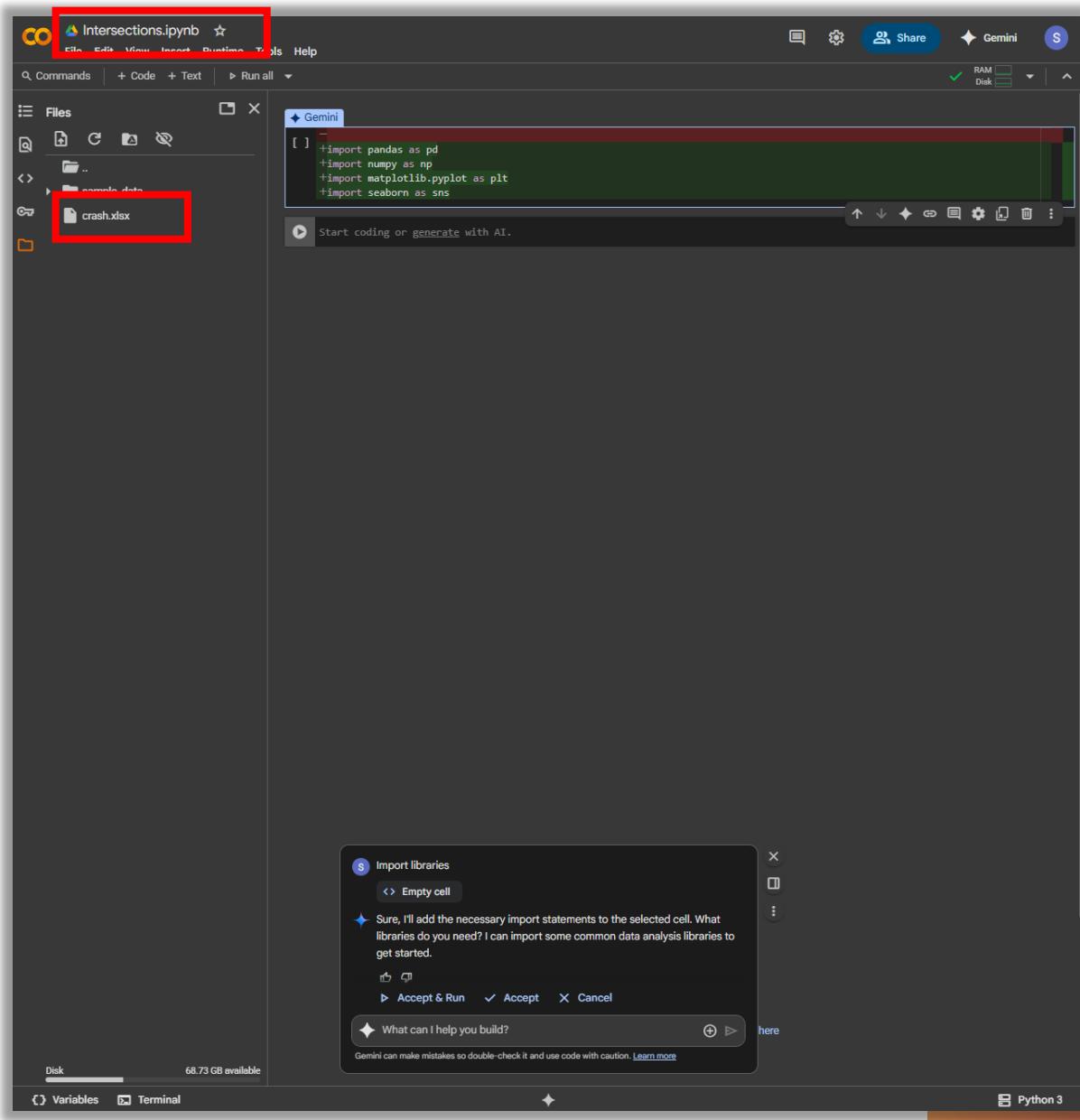
Would you like help scanning a Colab notebook or script for sensitive data before uploading it?

Law Enforcement Specific

- Avoid Victim/Suspect/Officer Information
- Use data that would be on a public facing website
- Get permission from Admin
- Anonymize, if possible

DANGEROUS INTERSECTIONS

VERSATERM



Drag and Drop File – crash.xlsx (Excel, CSV, Text)

Rename Untitled Notebook

IMPORT LIBRARIES

pandas – data

manipulation/analysis

numpy – numerical calculations

matplotlib.pyplot – graphs/charts

seaborn – graphs/charts

plotly.express – graphs/charts

Import Excel file, df.head(), df.tail(), df.shape, df.info()

Intersections.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Files sample_data crash.xlsx

```
[2]: #import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

#import crash.xlsx
df = pd.read_excel('crash.xlsx')
df.head()

First five rows

	Case Number	DOW	Hour	Year	Primary Offense	Location
0	WJ 2021-10GE4	Friday	10.0	2021.0	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	9000 S TEMPLE DR
1	WJ 2021-10EDF	Friday	16.0	2021.0	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	1888 W 7800 S
2	WJ 2021-10EG1	Friday	17.0	2021.0	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	7000 S TEMPLE DR
3	WJ 2021-10EEF	Friday	18.0	2021.0	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	7900 S OLD BINGHAM HWY
4	WJ 2021-10EEG	Friday	18.0	2021.0	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	4850 W TICKLEGASS RD

Next steps: Generate code with df View recommended plots New interactive sheet

View recommended plots (highlighted with a red box)

Plots:

- Hour
- Year
- Primary Offense
- Hour vs Year
- Year vs Hour
- Year vs count()
- Hour
- Year
- Primary Offense vs Hour
- Primary Offense vs Year

df.info() Column Format Category

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9408 entries, 0 to 9407
Data columns (total 6 columns):
 #   Column      Non-Null Count Dtype  
 --- 
 0   Case Number 9408 non-null   object  
 1   DOW          9408 non-null   object  
 2   Hour         9405 non-null   float64 
 3   Year         9405 non-null   float64 
 4   Primary Offense 9408 non-null   object  
 5   Location     9408 non-null   object  
dtypes: float64(2), object(4)
memory usage: 441.1+ KB
```

Rows, Columns

df.shape

```
(9408, 6)
```

df.tail() Last five rows

	Case Number	DOW	Hour	Year	Primary Offense	Location
9403	WJ 2025-G32G	Wednesday	17.0	2025.0	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	7647 S 2700 W
9404	WJ 2025-G343	Wednesday	17.0	2025.0	TCNR-0 TRAFFIC CRASH, NON REPORTABLE ...	4799 W 6200 S
9405	WJ 2025-E432	Wednesday	7.0	2025.0	TCNR-0 TRAFFIC CRASH, NON REPORTABLE ...	6200 S AIRPORT RD
9406	WJ 2025-E444	Wednesday	9.0	2025.0	TCNR-0 TRAFFIC CRASH, NON REPORTABLE ...	2655 W 9000 S
9407	WJ 2025-E513	Wednesday	16.0	2025.0	TCNR-0 TRAFFIC CRASH, NON REPORTABLE ...	4464 W NEW BINGHAM HWY

Object = Text
Float64 = Decimal Number
Int64 = Whole Number
Datetime64 = Date/Time

Convert Hour and Year to Integer

```
[11] df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9408 entries, 0 to 9407
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Case Number    9408 non-null   object  
 1   DOW           9408 non-null   object  
 2   Hour          9405 non-null   float64 
 3   Year          9405 non-null   float64 
 4   Primary Offense 9408 non-null object  
 5   Location       9408 non-null   object  
dtypes: float64(2), object(4)
memory usage: 441.1+ KB

[14] #convert format on Hour and Year from float64 to integer
df['Hour'] = df['Hour'].fillna(0).astype(int)
df['Year'] = df['Year'].fillna(0).astype(int)

[15] df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9408 entries, 0 to 9407
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Case Number    9408 non-null   object  
 1   DOW           9408 non-null   object  
 2   Hour          9408 non-null   int64  
 3   Year          9408 non-null   int64  
 4   Primary Offense 9408 non-null object  
 5   Location       9408 non-null   object  
dtypes: int64(2), object(4)
memory usage: 441.1+ KB
```

◆ It looks like there are missing values in the 'Hour' or 'Year' columns, which is causing an error when converting them to integers. I will fill the missing values with 0 before converting the columns to integer type.

Great! Now that we have the `Hour` and `Year` columns as integers, we can start exploring the data. Would you like to visualize the crash data by hour?



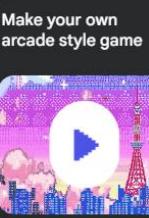
◆ What can I help you build?



Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Hello, Suzanna
Want to try out a few things?

Upload
your
homework
for instant
help



Turn your town
into an edible
world

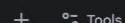
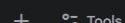


Plan an
adventurous
getaway

Welcome to Gemini, your personal AI assistant

Google Terms and the [Gemini Apps Privacy Notice](#) apply. Chats are reviewed and used to improve Google AI. [Learn about your choices](#). Gemini can make mistakes, so double-check it. Info [about your location](#) is also stored with your Gemini Apps activity.

Ask Gemini



Show me the first 20 lines of my data

A Jupyter Notebook interface displaying a table of 20 traffic crash records. The columns are Case Number, DOW, Hour, Year, Primary Offense, and Location. The data shows various types of crashes (TCPD-0, TCPI-0, TCNR-0) occurring on different dates and times at specific locations like Old Bingham Hwy, Temple Dr, and Starlite Dr.

Case Number	DOW	Hour	Year	Primary Offense	Location
WJ 2021-10GE4	Friday	10	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	9000 S TEMPLE DR
WJ 2021-10EDF	Friday	16	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	1888 W 7800 S
WJ 2021-10EG1	Friday	17	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	7000 S TEMPLE DR
WJ 2021-10EEF	Friday	18	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	7900 S OLD BINGHAM HWY
WJ 2021-10EEG	Friday	18	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	4850 W TICKLEGRASS RD
WJ 2021-11020	Friday	19	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	9000 S BANGERTER NB HWY
WJ 2021-110DG	Friday	23	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	S OLD BINGHAM HWY / W STARLITE DR
WJ 2021-11503	Friday	15	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	6655 S ADVENTURE WAY
WJ 2021-1230G	Friday	8	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	8008 S LISMORE LN
WJ 2021-12335	Friday	12	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	8236 S PIN OAK DR
WJ 2021-1234F	Friday	13	2021	TCNR-0 TRAFFIC CRASH, NON REPORTABLE ...	3963 W 9000 S
WJ 2021-123F0	Friday	14	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	7200 S (CENTER PARK DRIVE) PLAZA CENTER DR
WJ 2021-123E2	Friday	16	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	3600 W 7800 S
WJ 2021-12400	Friday	17	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	9000 S BANGERTER SB HWY
WJ 2021-1240E	Friday	18	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	8200 S REDWOOD RD
WJ 2021-124D0	Friday	22	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	7800 S TEMPLE DR
WJ 2021-13F44	Friday	5	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	1750 S 8200 S
WJ 2021-13F5F	Friday	8	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	8400 S 4000 W
WJ 2021-13F5F	Friday	8	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	W 9000 S / W OLD BINGHAM HWY
WJ 2021-13F5F	Friday	8	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	8350 S 4800 W

In the bottom left cell, the user has asked for the first 20 lines of the data, and Gemini has responded with the requested information.

Delete data in parenthesis

A Jupyter Notebook interface showing the modification of the same crash dataset. In cell [20], the user runs code to replace any text within parentheses in the 'Primary Offense' and 'Location' columns. In cell [21], the user checks the updated data, which now shows clean entries without parentheses. In cell [22], the user asks to remove data in parentheses from the 'Location' column, and Gemini responds by removing the text within parentheses from both columns.

Case Number	DOW	Hour	Year	Primary Offense	Location
WJ 2021-10GE4	Friday	10	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	9000 S TEMPLE DR
WJ 2021-10EDF	Friday	16	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	1888 W 7800 S
WJ 2021-10EG1	Friday	17	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	7000 S TEMPLE DR
WJ 2021-10EEF	Friday	18	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	7900 S OLD BINGHAM HWY
WJ 2021-10EEG	Friday	18	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	4850 W TICKLEGRASS RD
WJ 2021-11020	Friday	19	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	9000 S BANGERTER NB HWY
WJ 2021-110DG	Friday	23	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	S OLD BINGHAM HWY / W STARLITE DR
WJ 2021-11503	Friday	15	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	6655 S ADVENTURE WAY
WJ 2021-1230G	Friday	8	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	8008 S LISMORE LN
WJ 2021-12335	Friday	12	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	8236 S PIN OAK DR
WJ 2021-1234F	Friday	13	2021	TCNR-0 TRAFFIC CRASH, NON REPORTABLE ...	3963 W 9000 S
WJ 2021-123F0	Friday	14	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	7200 S PLAZA CENTER DR
WJ 2021-123E2	Friday	16	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	3600 W 7800 S
WJ 2021-12400	Friday	17	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	9000 S BANGERTER SB HWY
WJ 2021-1240E	Friday	18	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	8200 S REDWOOD RD
WJ 2021-124D0	Friday	22	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	7800 S TEMPLE DR
WJ 2021-13F44	Friday	5	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	1750 S 8200 S
WJ 2021-13F5F	Friday	8	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	8400 S 4000 W
WJ 2021-13F5F	Friday	8	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	W 9000 S / W OLD BINGHAM HWY
WJ 2021-13F5F	Friday	8	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	8350 S 4800 W

GEMINI:

1. show me the first 20 lines of my data
2. In the location column, delete any data in ()

Alphabetize the intersections delimited by “/”

s in the location column, alphabetize the intersections delimited by a /

★ I understand you want to alphabetize the intersections in the 'Location' column where they are separated by a '/'. Here's a plan to achieve that:

- **Identify intersections:** Locate the rows in the 'Location' column that contain the '/' delimiter.
- **Split and sort:** For each of the identified rows, split the string by '/', sort the resulting list of location parts alphabetically, and then join them back together with a '/'.

- **Update dataframe:** Replace the original location string with the alphabetized string in the 'Location' column.
- **Verify changes:** Display the head of the DataFrame to confirm the changes have been applied correctly.
- **Finish task:** Summarize the steps taken and the result.

Task

In the `Location` column, alphabetize the intersections delimited by a `/`.

Identify intersections

Subtask:

Locate the rows in the 'Location' column that contain the '/' delimiter.

Reasoning: Create a boolean mask to identify rows in the 'Location' column containing the '/' delimiter and then filter the DataFrame using this mask to display the relevant rows.

```
[24] mask = df['Location'].str.contains('/')
df_intersections = df[mask]
display(df_intersections.head())
```

	Case Number	DOW	Hour	Year	Primary Offense	Location
6	WJ 2021-110DG	Friday	23	2021	TCPD-0 TRAFFIC CRASH, PERSONAL INJURY ...	OLD BINGHAM HWY / STARLITE DR
18	WJ 2021-13G0F	Friday	13	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	W 9000 S / OLD BINGHAM HWY
26	WJ 2021-15342	Friday	21	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	CAMPUS VIEW DR / JORDAN LANDING BLVD
53	WJ 2021-204EF	Friday	20	2021	TCPD-0 TRAFFIC CRASH, PERSONAL INJURY ...	MORNING LAUREL LN / MOUNTAIN LAUREL LN
55	WJ 2021-2050E	Friday	22	2021	TCPD-0 TRAFFIC CRASH, PERSONAL INJURY ...	HIGHLANDS LOOP RD / 5600 ROUNDABOUT W



GEMINI:

in the location column, alphabetize the intersections
delimited by a /

Alphabetize the intersections delimited by “/”

Subtask:

For each of the identified rows, split the string by ‘/’, sort the resulting list of location parts alphabetically, and then join them back together with a ‘/’.

Reasoning: Define a function to sort location parts and apply it to the ‘Location’ column of the `df_intersections` DataFrame.

```
[25] def sort_location_parts(location):
    if '/' in location:
        parts = location.split('/')
        sorted_parts = sorted([part.strip() for part in parts])
        return ' / '.join(sorted_parts)
    return location

df_intersections['Sorted Location'] = df_intersections['Location'].apply(sort_location_parts)
display(df_intersections.head(20))

/tmp/ipython-input-639675129.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy
df_intersections['Sorted Location'] = df_intersections['Location'].apply(sort_location_parts)

Case Number DOW Hour Year Primary Offense Location Sorted Location
6 WJ 2021-110DG Friday 23 2021 TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ... OLD BINGHAM HWY / STARLITE DR OLD BINGHAM HWY / STARLITE DR
18 WJ 2021-13G0F Friday 13 2021 TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ... W 9000 S / OLD BINGHAM HWY OLD BINGHAM HWY / W 9000 S
26 WJ 2021-15342 Friday 21 2021 TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ... CAMPUS VIEW DR / JORDAN LANDING BLVD CAMPUS VIEW DR / JORDAN LANDING BLVD
53 WJ 2021-204EF Friday 20 2021 TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ... MORNING LAUREL LN / MOUNTAIN LAUREL LN MORNING LAUREL LN / MOUNTAIN LAUREL LN
55 WJ 2021-2050E Friday 22 2021 TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ... HIGHLANDS LOOP RD / 5600 ROUNDABOUT W / HIGHLANDS LOOP RD 5600 ROUNDABOUT W / HIGHLANDS LOOP RD
```

Subtask:

Replace the original location string with the alphabetized string in the ‘Location’ column.

Reasoning: Update the ‘Location’ column in the original DataFrame `df` with the alphabetized locations from `df_intersections`.

```
[26] for index in df_intersections.index:
    df.loc[index, 'Location'] = df_intersections.loc[index, 'Sorted Location']
display(df.head(20))

Case Number DOW Hour Year Primary Offense Location
0 WJ 2021-10GE4 Friday 10 2021 TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ... 9000 TEMPLE DR
1 WJ 2021-10EDF Friday 16 2021 TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ... 1888 W 7800 S
2 WJ 2021-10EG1 Friday 17 2021 TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ... 7000 TEMPLE DR
3 WJ 2021-10EEF Friday 18 2021 TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ... 7900 OLD BINGHAM HWY
4 WJ 2021-10EEG Friday 18 2021 TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ... 4850 TICKLEGRASS RD
5 WJ 2021-11020 Friday 19 2021 TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ... 9000 BANGERTER NB HWY
6 WJ 2021-110DG Friday 23 2021 TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ... OLD BINGHAM HWY / STARLITE DR
7 WJ 2021-11503 Friday 15 2021 TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ... 6655 ADVENTURE WAY
8 WJ 2021-1230G Friday 8 2021 TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ... 8008 LISMORE LN
9 WJ 2021-12335 Friday 12 2021 TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ... 8236 PIN OAK DR
10 WJ 2021-1234F Friday 13 2021 TCNR-0 TRAFFIC CRASH, NON REPORTABLE ... 3963 W 9000 S
11 WJ 2021-123F0 Friday 14 2021 TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ... 7200 PLAZA CENTER DR
12 WJ 2021-123E2 Friday 16 2021 TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ... 3600 W 7800 S
13 WJ 2021-12400 Friday 17 2021 TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ... 9000 BANGERTER SB HWY
14 WJ 2021-1240E Friday 18 2021 TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ... 8200 REDWOOD RD
15 WJ 2021-124D0 Friday 22 2021 TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ... 7800 TEMPLE DR
16 WJ 2021-13F44 Friday 5 2021 TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ... 1750 S 8200 S
17 WJ 2021-13F5F Friday 8 2021 TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ... 8400 S 4000 W
18 WJ 2021-13G0F Friday 13 2021 TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ... OLD BINGHAM HWY / W 9000 S
19 WJ 2021-13G32 Friday 16 2021 TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ... 8350 S 4800 W
```

Alphabetize the intersections delimited by “/”

Verify changes

Subtask:

Display the head of the DataFrame to confirm the changes have been applied correctly.

Reasoning: Display the head of the DataFrame to visually inspect the 'Location' column and confirm that the intersections have been alphabetized.

```
[27] display(df.head(20))
```

	Case Number	DOW	Hour	Year	Primary Offense	Location
0	WJ 2021-10GE4	Friday	10	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	9000 TEMPLE DR
1	WJ 2021-10EDF	Friday	16	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	1888 W 7800 S
2	WJ 2021-10EG1	Friday	17	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	7000 TEMPLE DR
3	WJ 2021-10EEF	Friday	18	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	7900 OLD BINGHAM HWY
4	WJ 2021-10EEG	Friday	18	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	4850 TICKLEGRASS RD
5	WJ 2021-11020	Friday	19	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	9000 BANGERTER NB HWY
6	WJ 2021-110DG	Friday	23	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	OLD BINGHAM HWY / STARLITE DR
7	WJ 2021-11503	Friday	15	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	6655 ADVENTURE WAY
8	WJ 2021-1230G	Friday	8	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	8008 LISMORE LN
9	WJ 2021-12335	Friday	12	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	8236 PIN OAK DR
10	WJ 2021-1234F	Friday	13	2021	TCNR-0 TRAFFIC CRASH, NON REPORTABLE ...	3963 W 9000 S
11	WJ 2021-123F0	Friday	14	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	7200 PLAZA CENTER DR
12	WJ 2021-123E2	Friday	16	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	3600 W 7800 S
13	WJ 2021-12400	Friday	17	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	9000 BANGERTER SB HWY
14	WJ 2021-1240E	Friday	18	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	8200 REDWOOD RD
15	WJ 2021-124D0	Friday	22	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	7800 TEMPLE DR
16	WJ 2021-13F44	Friday	5	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	1750 S 8200 S
17	WJ 2021-13F5F	Friday	8	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	8400 S 4000 W
18	WJ 2021-13G0F	Friday	13	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	OLD BINGHAM HWY / W 9000 S
19	WJ 2021-13G32	Friday	16	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	8350 S 4800 W

Summary:

Data Analysis Key Findings

- The 'Location' column was successfully updated to alphabetize the street names for entries containing a '/'.
- Entries with intersections are now represented with street names in alphabetical order, such as 'OLD BINGHAM HWY / STARLITE DR' and 'OLD BINGHAM HWY / W 9000 S'.

Insights or Next Steps

- This pre-processing step ensures consistency in location data, which could improve the accuracy of any subsequent analysis or modeling based on location.

Generating...

- Preparing
- Identify intersections
- Split and sort
- Update dataframe
- Verify changes
- Finish task

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Reorder the numeric streets (ex. 8400 S 4000 W)

numbers followed by a W or an S, then four numbers followed by a W or an S. I want the lower number first, but the W or S needs to go with it.

💡 I understand. You want to sort locations in the format '#### W/S #### W/S' based on the numerical value, keeping the direction with the number. Here's a plan to address this:

- **Identify locations with the specified format:** Create a mask to identify rows in the 'Location' column that match the pattern of '#### W/S #### W/S'.
- **Extract and sort parts:** For each identified location, extract the two

◆ What can I help you build? ⊕ ▶

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Summary:

Data Analysis Key Findings

- Several attempts were made to create a regular expression mask to identify locations in the format '#### W/S / #### W/S'.
- All tested regular expression patterns failed to match any rows in the 'Location' column, resulting in empty dataframes.
- Inspection of the data revealed that locations with a '/' separator do not follow the '#### W/S / #### W/S' pattern, but rather formats like 'STREET NAME / STREET NAME' or 'NUMBER W/S STREET NAME'.

Insights or Next Steps

- The specified format '#### W/S / #### W/S' does not appear to exist in the dataset.
- Future analysis requiring numerical sorting of locations should consider the actual formats present in the data, such as extracting numbers and directions from patterns like 'NUMBER W/S STREET NAME'.

S There is no "/" in that format of location.

💡 You are absolutely right! My apologies, I misinterpreted the format. The locations you want to sort are in the format '#### W/S #### W/S' without a '/'.

Here is the updated plan to sort these locations numerically:

- **Identify locations with the specified format:** Create a mask to
- ▷ Accept & auto-run ▶ Run step-by-step X Cancel

◆ What can I help you build? ⊕ ▶

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Took three tries to tell Gemini there was no "/" in this location format but it finally worked.

GEMINI:

I would like to do the same with the locations that are formatted with four numbers followed by a W or an S, then four numbers followed by a W or an S. I want the lower number first, but the W or S needs to go with it.

Reorder the numeric streets (ex. 8400 S 4000 W)

Extract and sort parts

Subtask:

For each identified location, extract the two number-direction parts, convert the numbers to integers, sort the parts based on the numerical value, and then rejoin them with a space.

```
[@] import re

def sort_location_numerical(location):
    # Extract number-direction parts
    parts = re.findall(r'\d+\s*[NSsw]', location)
    if len(parts) == 2:
        # Extract number and direction, and convert number to int
        parsed_parts = []
        for part in parts:
            match = re.match(r'(\d+)\s*([NSsw])', part)
            if match:
                number, direction = match.groups()
                parsed_parts.append((int(number), direction.upper(), part))
        # Sort based on the numerical value
        sorted_parts = sorted(parsed_parts)
        # Rejoin the original parts in sorted order
        return f"{sorted_parts[0][2]} {sorted_parts[1][2]}".strip()
    return location

df_specific_intersections['Sorted Location'] = df_specific_intersections['Location'].apply(sort_location_numerical)
display(df_specific_intersections.head())
```

```
/tmp/ipython-input-2829213209.py:20: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-vers
df_specific_intersections['Sorted Location'] = df_specific_intersections['Location'].apply(sort_location_numerical)

Case Number DOW Hour Year Primary Offense Location Sorted Location
1 WJ 2021-10EDF Friday 16 2021 TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ... 1888 W 7800 S 1888 W 7800 S
10 WJ 2021-1234F Friday 13 2021 TCNR-0 TRAFFIC CRASH, NON REPORTABLE ... 3963 W 9000 S 3963 W 9000 S
12 WJ 2021-123E2 Friday 16 2021 TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ... 3600 W 7800 S 3600 W 7800 S
16 WJ 2021-13F44 Friday 5 2021 TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ... 1750 S 8200 S 1750 S 8200 S
17 WJ 2021-13F5F Friday 8 2021 TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ... 8400 S 4000 W 4000 W 8400 S
```

Update dataframe

Subtask:

Replace the original location string with the sorted string in the 'Location' column.

```
[@] for index, row in df_specific_intersections.iterrows():
    df.loc[index, 'Location'] = row['Sorted Location']

display(df.head(20))
```

	Case Number	DOW	Hour	Year	Primary Offense	Location
0	WJ 2021-10GE4	Friday	10	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	9000 TEMPLE DR
1	WJ 2021-10EDF	Friday	16	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	1888 W 7800 S
2	WJ 2021-10EG1	Friday	17	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	7000 TEMPLE DR
3	WJ 2021-10EEF	Friday	18	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	7900 OLD BINGHAM HWY
4	WJ 2021-10EEG	Friday	18	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	4850 TICKLEGRASS RD
5	WJ 2021-11020	Friday	19	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	9000 BANGERTER NB HWY
6	WJ 2021-110DG	Friday	23	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	OLD BINGHAM HWY / STARLITE DR
7	WJ 2021-11503	Friday	15	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	6655 ADVENTURE WAY
8	WJ 2021-1230G	Friday	8	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	8008 LISMORE LN
9	WJ 2021-12335	Friday	12	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	8236 PIN OAK DR
10	WJ 2021-1234F	Friday	13	2021	TCNR-0 TRAFFIC CRASH, NON REPORTABLE ...	3963 W 9000 S
11	WJ 2021-123F0	Friday	14	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	7200 PLAZA CENTER DR
12	WJ 2021-123E2	Friday	16	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	3600 W 7800 S
13	WJ 2021-12400	Friday	17	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	9000 BANGERTER SB HWY
14	WJ 2021-1240E	Friday	18	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	8200 REDWOOD RD
15	WJ 2021-124D0	Friday	22	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	7800 TEMPLE DR
16	WJ 2021-13F44	Friday	5	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	1750 S 8200 S
17	WJ 2021-13F5F	Friday	8	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	4000 W 8400 S
18	WJ 2021-13G0F	Friday	13	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	OLD BINGHAM HWY / W 9000 S
19	WJ 2021-13G32	Friday	16	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	4800 W 8350 S

Add a new column called Score, values based on Primary Offense

```
conditions = [
    df['Primary Offense'].str.contains('FATALITY', case=False) | df['Primary Offense'].str.contains('HOMICIDE', case=False),
    df['Primary Offense'].str.contains('INJURY', case=False)
]
choices = [3, 2]
df['score'] = np.select(conditions, choices, default=1)

display(df.head(10))
```

	Case Number	DOW	Hour	Year	Primary Offense	Location	score
0	WJ 2021-10GE4	Friday	10	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	9000 TEMPLE DR	1
1	WJ 2021-10EDF	Friday	16	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	1888 W 7800 S	1
2	WJ 2021-10EG1	Friday	17	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	7000 TEMPLE DR	1
3	WJ 2021-10EEF	Friday	18	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	7900 OLD BINGHAM HWY	2
4	WJ 2021-10EEG	Friday	18	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	4850 TICKLEGRASS RD	2
5	WJ 2021-11020	Friday	19	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	9000 BANGERTER NB HWY	1
6	WJ 2021-110DG	Friday	23	2021	TCPI-0 TRAFFIC CRASH, PERSONAL INJURY ...	OLD BINGHAM HWY / STARLITE DR	2
7	WJ 2021-11503	Friday	15	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	6655 ADVENTURE WAY	1
8	WJ 2021-1230G	Friday	8	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	8008 LISMORE LN	1
9	WJ 2021-12335	Friday	12	2021	TCPD-0 TRAFFIC CRASH, PROPERTY DAMAGE ...	8236 PIN OAK DR	1

GEMINI:

Insert a new column called score - base the value on the Primary Offense column. If Primary Offense contains "FATALITY" OR "HOMICIDE", put a 3 in the score column. If the Primary Offense column contains "INJURY", put a 2 in the score column, else 1.

Top Location

Subtask:

Display the top 15 locations and their scores.

Reasoning: Display the top 15 locations and their scores.

```
[50] display(top_15_locations)
```

	Location	score
2227	7800 REDWOOD RD	197
2604	9000 REDWOOD RD	185
1827	7000 REDWOOD RD	150
1011	4000 W 9000 S	149
2222	7800 MOUNTAIN VIEW SB HWY	144
672	3200 W 7800 S	142
2221	7800 MOUNTAIN VIEW NB HWY	128
989	4000 W 7800 S	114
1360	5600 W 7800 S	106
901	3800 W 7800 S	102
547	2700 W 9000 S	96
2599	9000 MOUNTAIN VIEW NB HWY	86
2126	7671 CAMPUS VIEW DR	84
2224	7800 OLD BINGHAM HWY	66
2233	7800 TEMPLE DR	66

Next steps: [Generate code with top_15_locations](#) [View recommended plots](#) [New interactive sheet](#)

Summary:

Data Analysis Key Findings

- The locations were successfully ranked based on the sum of their scores in descending order.
- The top 15 locations were identified and displayed with their corresponding scores.

GEMINI:

Based on the score column, rank the top 15 locations descending

(+ Code) (+ Text)

```
def categorize_score(score):
    if score == 3:
        return 'Fatality'
    elif score == 2:
        return 'Injury'
    else:
        return 'Non-Injury'

top_15_locations['Score Category'] = top_15_locations['score'].apply(categorize_score)
display(top_15_locations)
```

/tmp/ipython-input-3290506742.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

	Location	score	Score Category
2227	7800 REDWOOD RD	197	Non-Injury
2604	9000 REDWOOD RD	185	Non-Injury
1827	7000 REDWOOD RD	150	Non-Injury
1011	4000 W 9000 S	149	Non-Injury
2222	7800 MOUNTAIN VIEW SB HWY	144	Non-Injury
672	3200 W 7800 S	142	Non-Injury
2221	7800 MOUNTAIN VIEW NB HWY	128	Non-Injury
989	4000 W 7800 S	114	Non-Injury
1360	5600 W 7800 S	106	Non-Injury
901	3800 W 7800 S	102	Non-Injury
547	2700 W 9000 S	96	Non-Injury
2599	9000 MOUNTAIN VIEW NB HWY	86	Non-Injury
2126	7671 CAMPUS VIEW DR	84	Non-Injury
2224	7800 OLD BINGHAM HWY	66	Non-Injury
2233	7800 TEMPLE DR	66	Non-Injury

GEMINI:

Display the top 15 locations, grouped by score (if 3 call it Fatality, if 2 call it Injury, if 1 call it Non-Injury)

Top Location

G: Would you like to see the distribution of the individual crash scores (Fatality, Injury, Non-Injury) within these top 15 locations, or would you like to categorize the scores in the original DataFrame before calculating the total score for each location?

S: Yes, top 15 within these categories

G: Okay, I understand you want to see the top locations within each category (Fatality, Injury, and Non-Injury). Here's a plan to achieve that:

- Categorize scores:** Create a new column in the original DataFrame to categorize the scores as 'Fatality', 'Injury', or 'Non-Injury'.
- Group by location and category:** Group the DataFrame by both 'Location' and the new 'Score Category' column and calculate the sum of the scores for each group.
- Rank within categories:** For each 'Score Category', rank the locations based on their summed scores in descending order.
- Select top 15 per category:** Select the top 15 locations within each 'Score Category'.
- Display results:** Display the top 15 locations for each category.
- Finish task:** Summarize the findings for each category.

Reasoning: Display the top 15 locations for each category.

```
[59] display(top_15_ranked_within_categories)
```

	Location	Score Category	score	Rank	
0	8660 REDWOOD RD	Fatality	6	1	
1	9000 MOUNTAIN VIEW NB HWY	Fatality	6	2	
2	1437 W 7800 S	Fatality	3	3	
3	1510 W 9000 S	Fatality	3	4	
4	1995 W 9000 S	Fatality	3	5	
5	1820 W 9000 S	Fatality	3	6	
6	3200 W 9000 S	Fatality	3	7	
7	4800 W 7800 S	Fatality	3	8	
8	7000 TEMPLE DR	Fatality	3	9	
9	7292 BANGERTER SB HWY	Fatality	3	10	
10	5250 W 7000 S	Fatality	3	11	
11	5600 W 6363 S	Fatality	3	12	
12	7800 MOUNTAIN VIEW NB HWY	Fatality	3	13	
13	7700 BANGERTER NB HWY	Fatality	3	14	
14	8100 GRIZZLY WAY	Fatality	3	15	
15	7800 REDWOOD RD	Injury	78	1	
16	4000 W 9000 S	Injury	74	2	
17	3200 W 7800 S	Injury	72	3	
18	9000 REDWOOD RD	Injury	64	4	
19	2700 W 9000 S	Injury	60	5	
20	3800 W 7800 S	Injury	60	6	
21	7800 MOUNTAIN VIEW NB HWY	Injury	60	7	
22	7800 MOUNTAIN VIEW SB HWY	Injury	60	8	
23	5600 W 7800 S	Injury	48	9	
24	7000 REDWOOD RD	Injury	46	10	
25	9000 MOUNTAIN VIEW NB HWY	Injury	40	11	
26	7800 TEMPLE DR	Injury	36	12	
27	1300 W 9000 S	Injury	36	13	
28	MOUNTAIN VIEW SB HWY / OLD BINGHAM HWY	Injury	34	14	
29	2200 W 7800 S	Injury	32	15	

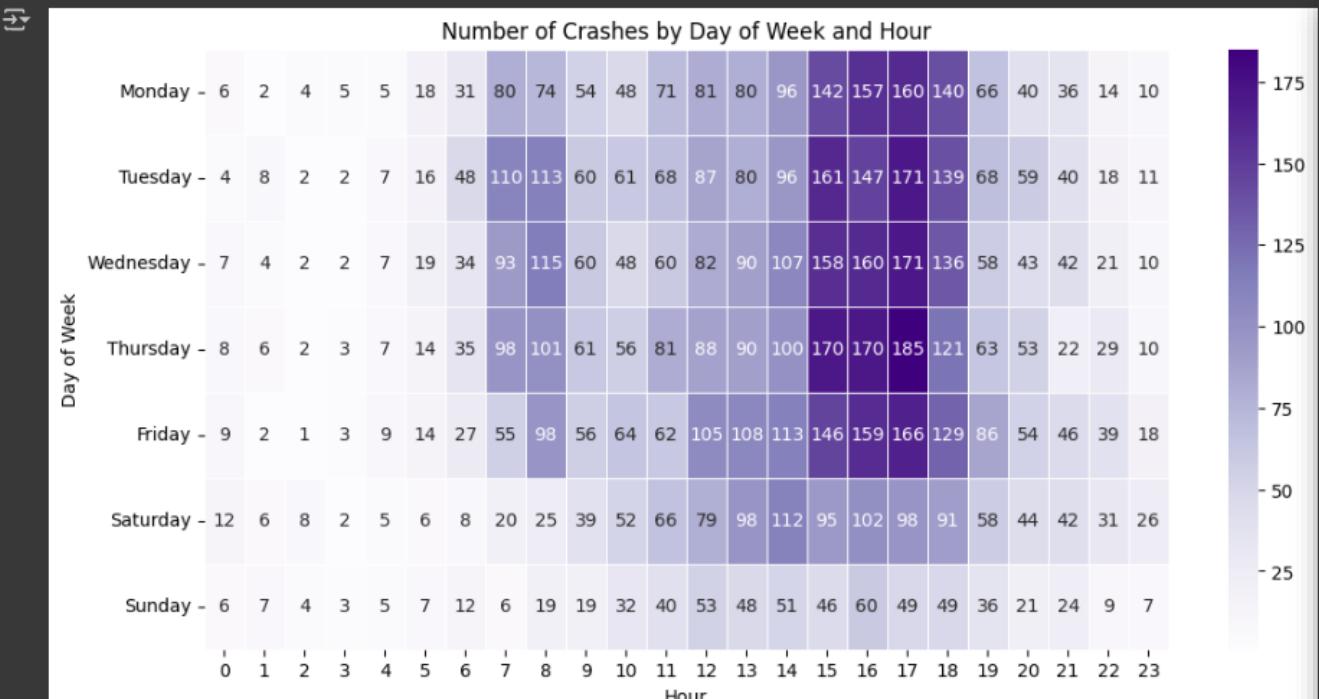
30	9000 REDWOOD RD	Non-Injury	121	1
31	7800 REDWOOD RD	Non-Injury	119	2
32	7000 REDWOOD RD	Non-Injury	104	3
33	4000 W 7800 S	Non-Injury	84	4
34	7800 MOUNTAIN VIEW SB HWY	Non-Injury	84	5
35	7671 CAMPUS VIEW DR	Non-Injury	76	6
36	4000 W 9000 S	Non-Injury	75	7
37	3200 W 7800 S	Non-Injury	70	8
38	7800 MOUNTAIN VIEW NB HWY	Non-Injury	65	9
39	5600 W 7800 S	Non-Injury	58	10
40	8136 JAGUAR DR	Non-Injury	55	11
41	7800 OLD BINGHAM HWY	Non-Injury	52	12
42	5445 COPPER HILLS PKY	Non-Injury	48	13
43	7061 REDWOOD RD	Non-Injury	45	14
44	3800 W 7800 S	Non-Injury	42	15

Matrix

```
# Group by DOW and Hour and count the occurrences
heatmap_data = df.groupby(['DOW', 'Hour']).size().unstack(fill_value=0)

# Reorder the rows (days of the week)
days_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
heatmap_data = heatmap_data.reindex(days_order)

# Create the heatmap
plt.figure(figsize=(12, 6))
sns.heatmap(heatmap_data, cmap='Purples', annot=True, fmt='d', linewidths=.5)
plt.title('Number of Crashes by Day of Week and Hour')
plt.xlabel('Hour')
plt.ylabel('Day of Week')
plt.show()
```



GEMINI:

on the total dataset, create a matrix with DOW in the rows and Hour in the columns, use purple.

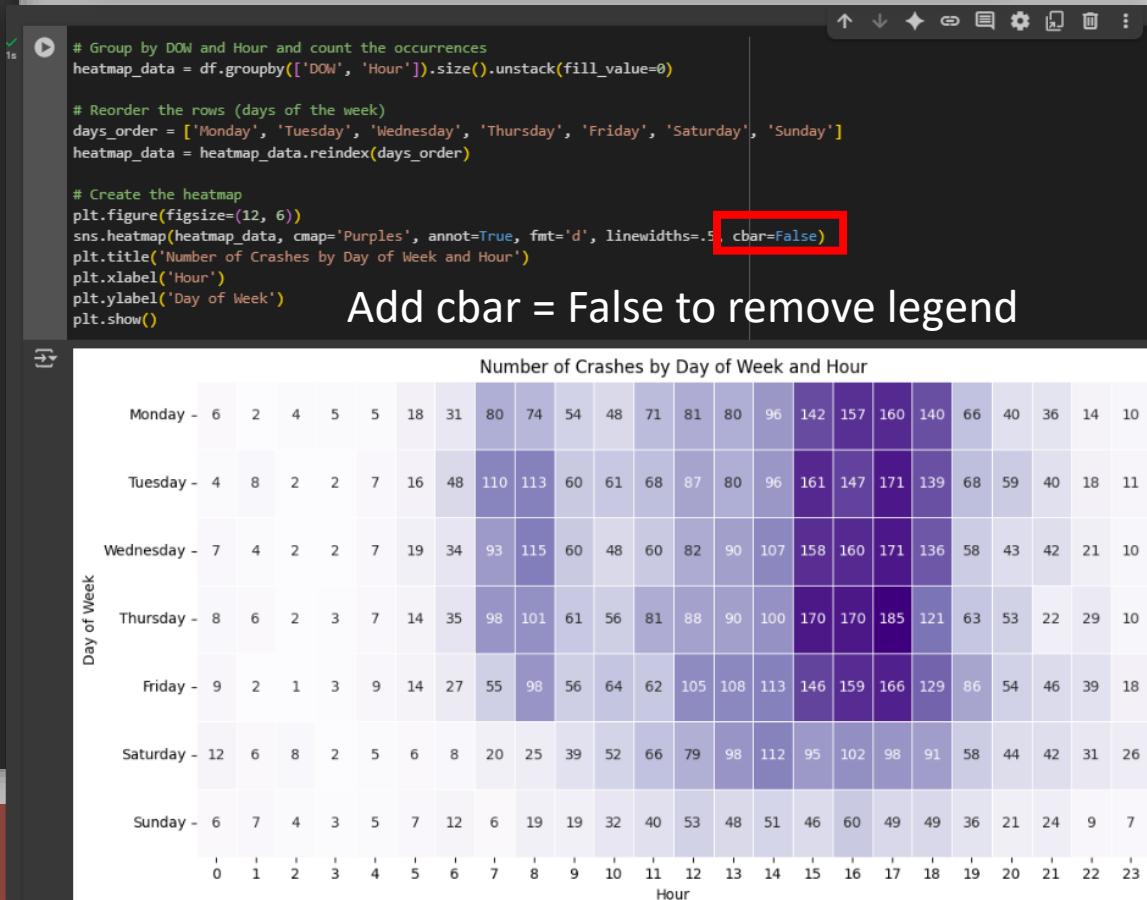
The heatmap showing the number of crashes by day of the week and hour has been generated successfully.

Here are some key findings from the heatmap:

- The highest number of crashes occur during weekday afternoons and evenings (around 3 PM to 6 PM).
- Fridays and Saturdays appear to have higher crash counts in the later hours compared to other weekdays.
- Sundays generally show lower crash counts throughout the day.

What can I help you build? + ▶

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)



1 Built-in Seaborn Palettes

These palettes can be used with `palette="name"` in Seaborn plots.

Palette Name	Description
"deep"	Default Seaborn colors, vibrant and distinct
"muted"	Softer colors, easier on the eyes
"bright"	Highly saturated, bold colors
"pastel"	Light, soft pastel shades
"dark"	Darker shades for contrast
"colorblind"	Optimized for color vision deficiency

2 Perceptually Uniform Sequential Palettes (Good for heatmaps & gradients)

These are single-hue palettes that transition from light to dark.

Palette Name	Description
"Blues"	Shades of blue (light to dark)
"Greens"	Shades of green
"Oranges"	Shades of orange
"Purples"	Shades of purple
"Reds"	Shades of red

3 Diverging Palettes (Good for highlighting positive/negative values)

These palettes balance two opposing colors.

Palette Name	Color Transition
"coolwarm"	Blue → Red
"RdBu"	Red → Blue
"PuOr"	Purple → Orange
"RdYlGn"	Red → Yellow → Green

4 Qualitative Palettes (For categorical data)

These palettes provide **distinct colors** for different categories.

Palette Name	Style
"Set1"	Bright and bold colors
"Set2"	Softer pastel shades
"Set3"	A mix of 12 varied colors
"Paired"	Matched color pairs
"Accent"	Soft, accentuated colors
"tab10"	10 categorical colors (default in Matplotlib)

5 Custom Palettes

You can also **create your own palette** using:

- Manually defined colors (hex codes, RGB)
- Continuous color gradients
- Using `sns.color_palette()`

Example: Custom Hex Colors

python Copy Edit

```
custom_colors = ["#FF5733", "#33FF57", "#3357FF"] # Red, Green, Blue  
sns.barplot(x="DOW", y="Case Number", data=df, palette=custom_colors)
```

Example: Continuous Gradient

python Copy Edit

```
sns.barplot(x="DOW", y="Case Number", data=df, palette=sns.light_palette("blue", as_cmap=True)
```

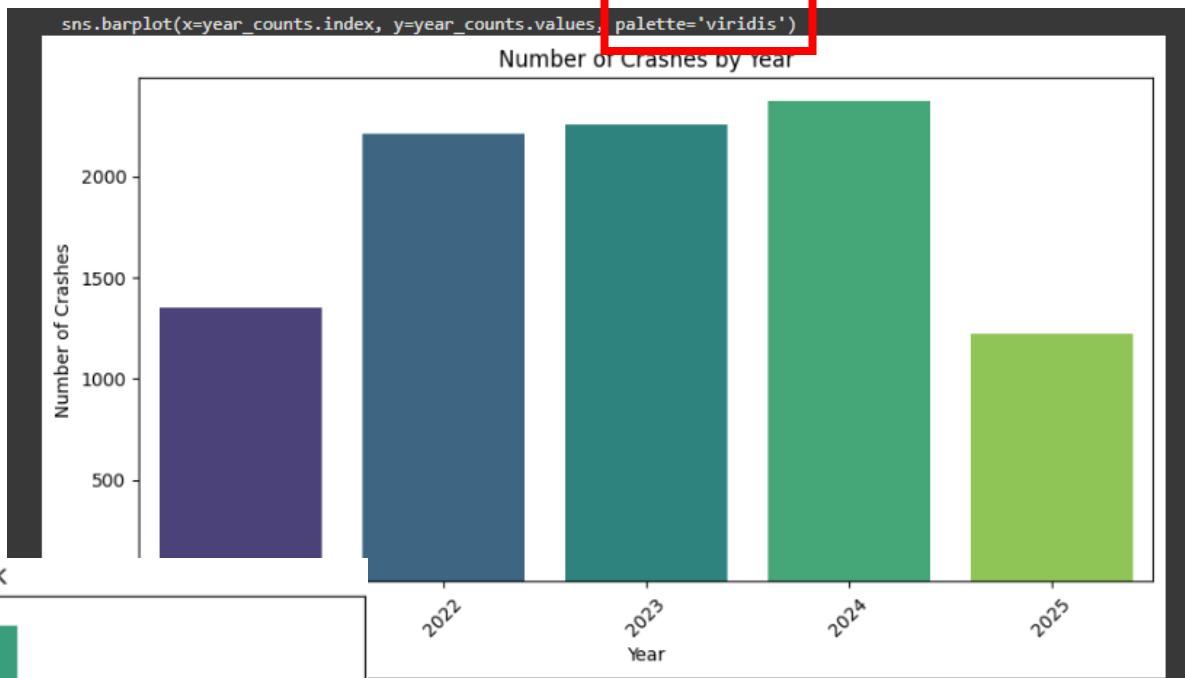
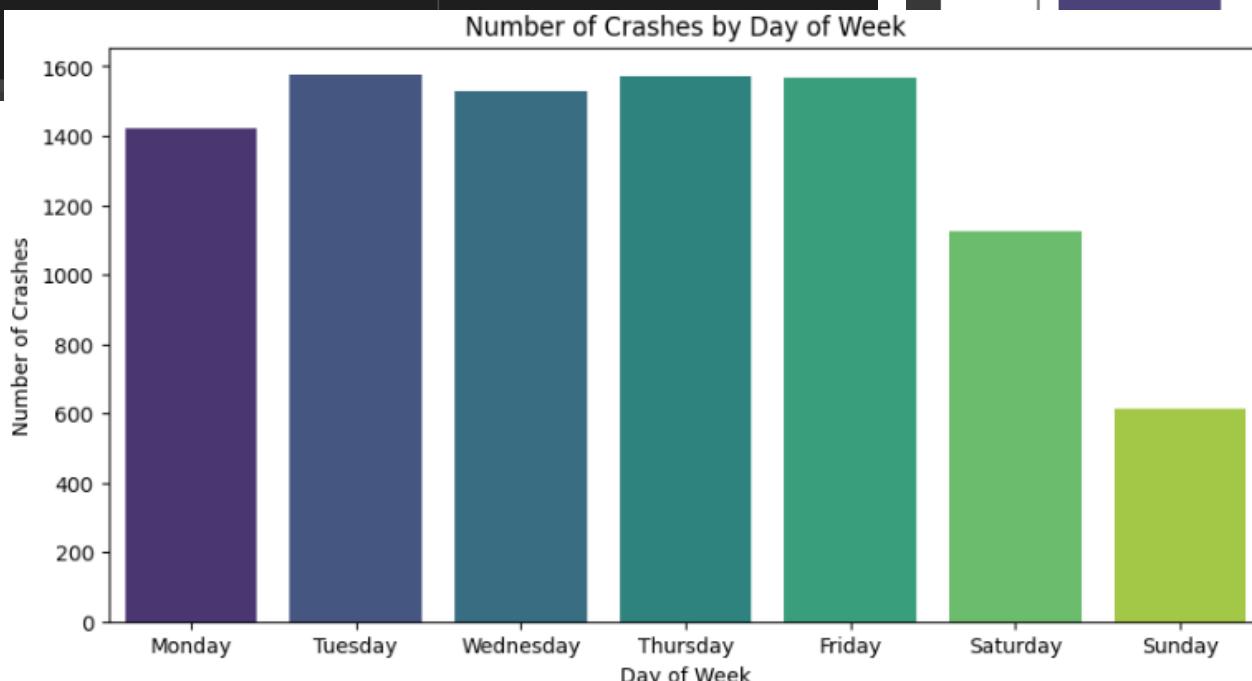
Graphs

```
# Group by Day of Week and count the occurrences
dow_counts = df['DOW'].value_counts().reindex(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])

# Create a bar plot for Day of Week
plt.figure(figsize=(10, 5))
sns.barplot(x=dow_counts.index, y=dow_counts.values, palette='viridis')
plt.title('Number of Crashes by Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Number of Crashes')
plt.show()

# Group by Year and count the occurrences
year_counts = df['Year'].value_counts().sort_index()

# Create a bar plot for Year
plt.figure(figsize=(10, 5))
sns.barplot(x=year_counts.index, y=year_counts.values, palette='viridis')
plt.title('Number of Crashes by Year')
plt.xlabel('Year')
plt.ylabel('Number of Crashes')
plt.xticks(rotation=45)
plt.show()
```



GEMINI:
graphs for day of week and year

Graphs

From this updated graph, we can see the distribution of crashes across the years 2021 to 2025.

What would you like to analyze next? Perhaps you'd like to investigate the trends in crash counts over these years, or look at the distribution of crash types within a specific year?



Can we look at the distribution of crash categories for 2025? + ▶

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

```
# Filter the DataFrame for the year 2025
df_2025 = df[df['Year'] == 2025]

# Group by Score Category and count the occurrences
category_counts_2025 = df_2025['Score Category'].value_counts()

# Create a bar plot for Score Category distribution in 2025
plt.figure(figsize=(8, 5))
ax = sns.barplot(x=category_counts_2025.index, y=category_counts_2025.values, palette='viridis')
plt.title('Distribution of Crash Categories in 2025')
plt.xlabel('Score Category')
plt.ylabel('Number of Crashes')

# Add count labels to the bars
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p.get_height() / 2., ha='center', va='center', xytext=(0, 5), textcoords='offset points')

plt.show()
```

/tmp/ipython-input-382815477.py:9: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

```
ax = sns.barplot(x=category_counts_2025.index, y=category_counts_2025.values, palette='viridis')
```

The bar chart displays the distribution of crash categories in 2025. The y-axis represents the 'Number of Crashes' from 0 to 800. The x-axis represents the 'Score Category' with three categories: 'Non-Injury', 'Injury', and 'Fatality'. The 'Non-Injury' category has 950 crashes, the 'Injury' category has 267 crashes, and the 'Fatality' category has 4 crashes. The bars are colored using the 'viridis' palette.

Score Category	Number of Crashes
Non-Injury	950
Injury	267
Fatality	4

S add a count label
◆ Yes, I can add count labels to the bars in the distribution plot for 2025 crash categories.
I have added the count labels to the bars in the distribution plot for 2025 crash categories. You can now clearly see the exact number of crashes in each category for that year.
From the plot, we can see:
◆ What can I help you build? + ▶

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

GEMINI:

Can we look at the distribution of crash categories for 2025?

Add a count label.

Graphs

```
[73] crashes_by_year_category_unstacked = crashes_by_year_category.set_index(['Year', 'Score Category']).unstack(level='Score Category', fill_value=0)
display(crashes_by_year_category_unstacked.head())
```

count

Score Category	Fatality	Injury	Non-Injury
Year	0	0	3
2021	6	336	1007
2022	6	504	1700
2023	3	503	1749
2024	3	531	1836

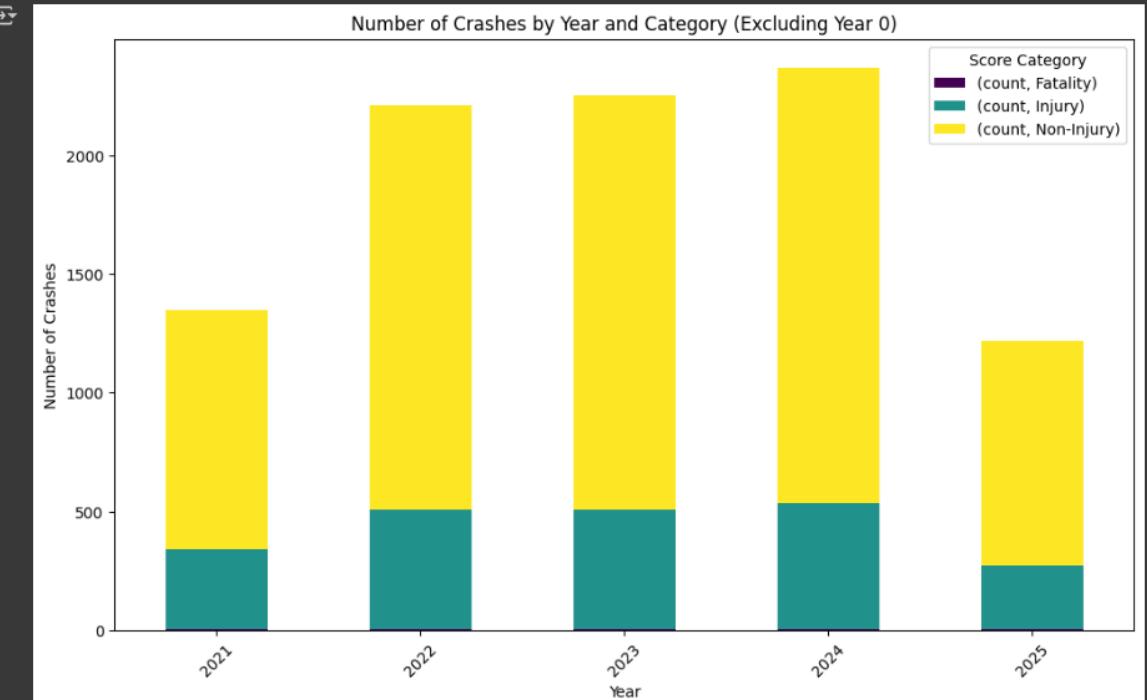
Create comparison chart

Subtask:

Generate a chart (e.g., grouped bar chart or stacked bar chart) to visualize the number of crashes in each category across the years.

Reasoning: Generate a stacked bar chart to visualize the number of crashes in each category across the years.

```
plt.figure(figsize=(12, 7))
crashes_by_year_category_unstacked[crashes_by_year_category_unstacked.index != 0].plot(kind='bar', stacked=True, colormap='viridis')
plt.title('Number of Crashes by Year and Category (Excluding Year 0)')
plt.xlabel('Year')
plt.ylabel('Number of Crashes')
plt.xticks(rotation=45)
plt.legend(title='Score Category')
plt.show()
```



GEMINI:

Comparison charts by category across the years

Export and Download

next step for deeper analysis.

```
[76] # Export the DataFrame to an Excel file
df.to_excel('crash_data_modified.xlsx', index=False)

# Save the generated plots as JPG images

# Heatmap
heatmap_data = df.groupby(['DOW', 'Hour']).size().unstack(fill_value=0).reindex(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])
plt.figure(figsize=(12, 6))
sns.heatmap(heatmap_data, cmap='Purples', annot=True, fmt='d', linewidths=.5)
plt.title('Number of Crashes by Day of Week and Hour')
plt.xlabel('Hour')
plt.ylabel('Day of Week')
plt.savefig('crashes_heatmap.jpg')

# Day of Week bar plot
dow_counts = df['DOW'].value_counts().reindex(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])
low_counts, index, y=dow_counts.values, palette='viridis'
plt.title('Number of Crashes by Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Number of Crashes')
plt.savefig('crashes_by_dow.jpg')

# Yearly bar plot
year_counts = df['Year'].value_counts()
if df['Year'].idx == 0:
    year_counts = df['Year'].value_counts().sort_index()
low_counts, index, y=year_counts.values, palette='viridis'
plt.title('Number of Crashes by Year (Excluding Year 0)')
plt.xlabel('Year')
plt.ylabel('Number of Crashes')
plt.xticks(rotation=45)
plt.savefig('crashes_by_year.jpg')

# Distribution of Crash Categories in 2025 bar plot
df_2025 = df[df['Year'] == 2025]
category_counts_2025 = df_2025['Score Category'].value_counts()
plt.figure(figsize=(8, 5))
ax = sns.barplot(x=category_counts_2025.index, y=category_counts_2025.values, palette='viridis')
plt.title('Distribution of Crash Categories in 2025')
plt.xlabel('Score Category')
plt.ylabel('Number of Crashes')
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2, p.get_height()), ha='center', va='center', xytext=(0, 5), textcoords='offset points')
plt.savefig('crashes_2025_category_distribution.jpg')

# Comparison chart by category across the years stacked bar plot (excluding year 0)
crashes_by_year_category = df.groupby(['Year', 'Score Category']).size().reset_index(name='count')
crashes_by_year_category_unstacked = crashes_by_year_category.set_index(['Year', 'Score Category']).unstack(level='Score Category', fill_value=0)
plt.figure(figsize=(12, 6))
crashes_by_year_category_unstacked['bar', stacked=True, colormap='viridis', ax=plt.gca()]
plt.title('Number of Crashes per year and the proportion of each category within that total.')
plt.xlabel('Year')
plt.ylabel('Number of Crashes')
plt.legend(title='Score Category')
plt.legend(title='Score Category')
plt.savefig('crashes_by_year_category_unstacked.jpg')

Would you like to analyze the trends for a specific category over the years, or perhaps investigate factors that might have contributed to the observed changes?
```

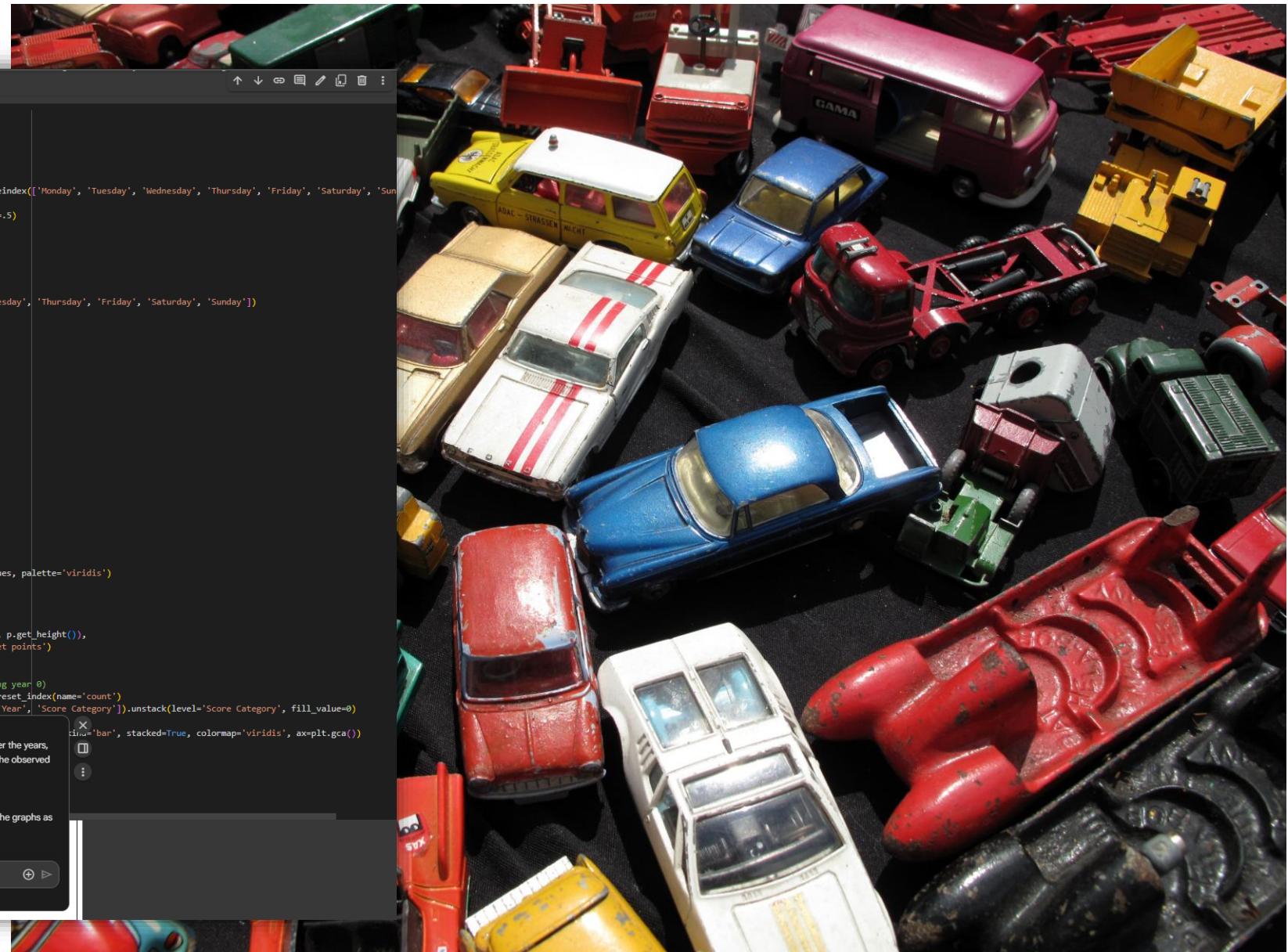
export the data with all of our changes as an excel file and all the graphs as jpg

Empty cell

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Disk 68.70 GB available



GEMINI:

Export final dataframe and all the graphs as jpgs

UNPIVOT

SPILLMAN

SPILLMAN

3.5 Years of Data
22+ text files per year

Challenges:
Export to text file
Data in Rows not Columns

A	B	C	D	E	F	G
1 03/11/25	West Jo	r dan Police Department	3265			
2 13:09	CAD	MASTER CALL TABLE	Page: 1			
3						
4 Record number	:	C10622404				
5 Call type (Law/Fire/EMS)	:	If				
6 Call nature	:	26A10 26A10 DEAF				
7 Priority of Call	:		1			
8 When Occured Earliest	:		1/31/2018 23:48			
9 When Occured Latest	:		1/31/2018 23:48			
10 When Reported	:		1/31/2018 23:49			
11 Respond-to Address	:	9658 S PAISLEY CIR; 4165 W				
12 City code	:	SJC				
13 Person to Contact	:	SPARKS, SHAWN				
14 Contacts Phone Number	:	(801)647-7127				
15 Complainant name number	:					
16 How Received	:	9 911 Line				
17 Who took Call	:	ANSBERGS, I				
18 Alarm ID number	:		158192			
19 Geobase address ID	:					
20 Directions number	:					
21 Contacts address	:	3647 W 9800 S - W Sector				
22 Call Received? (Y/N)	:	Y				
23 Hold Until	:	**;**;** */**/****				
24 Plate number of stopped veh	:					
25 Plate state abbreviation	:					
26 Determinant	:	26A10				
27 =====	=					
28						
29						
30						
31						
32						
33						
34						
35						
36						
37						
38						
39						

```
03/11/25 14:58 West Jordan Police Department 3265
CAD MASTER CALL TABLE Page: 1
Record number : C10622404
Call type (Law/Fire/EMS) : If
Call nature : 26A10 26A10 DEAF
Priority of Call : 1
When Occured Earliest : 23:48:41 01/31/2018
When Occured Latest : 23:48:44 01/31/2018
When Reported : 23:49:34 01/31/2018
Respond-to Address : 9658 S PAISLEY CIR; 4165 W
City code : SJC
Person to Contact : SPARKS, SHAWN
Contacts Phone Number : (801)647-7127
Complainant name number :
How Received : 9 911 Line
Who took Call : ANSBERGS, I
Alarm ID number :
Geobase address ID :
Directions number :
Contacts address : 3647 W 9800 S - W Sector
Call Received? (Y/N) : Y
Hold Until : **;**;** */**/****
Plate number of stopped veh :
Plate state abbreviation :
Determinant : 26A10
```

- CADApril1-15-18
- CADapril16-30-18
- CADaug1-10-18
- CADaug11-20-18
- CADaug21-31-18
- CADdec1-16-18
- CADdec17-31-18
- CADjuly1-15-18
- CADjuly16-25-18
- CADjuly26-31-18
- CADjune1-15-18
- CADjune16-30-18
- CADmarch1-15-18
- CADmarch16-31-18
- CADmay1-16-18
- CADmay17-31-18
- CADnov1-15-18
- CADnov16-30-18
- CADoct1-16-18
- CADoct17-31-18
- CADsep1-15-18
- CADsep16-30-18

ChatGPT

In Google Colab, I want to merge CAD*.txt files into one .xlsx file named CAD2018.xlsx and export it.

Here's a step-by-step guide to help you merge all CAD*.txt files into one .xlsx file named CAD2018.xlsx in Google Colab:

Steps:

1. Upload Your CAD*.txt Files to Colab

In Colab, run this cell to get a file upload widget:

```
python
```

```
from google.colab import files  
uploaded = files.upload()
```

👉 Select all your CAD*.txt files for upload.

2. Merge All CAD*.txt Files into One DataFrame

Assuming your .txt files are tab- or comma-separated (let me know if it's different):

```
python
```

```
import pandas as pd  
import glob  
  
# List all CAD*.txt files in current directory  
txt_files = glob.glob("CAD*.txt")  
  
# Read and concatenate all files  
df_list = [pd.read_csv(file, delimiter='\t') for file in txt_files] # change delimiter if needed  
merged_df = pd.concat(df_list, ignore_index=True)  
  
# Optional: Display first few rows  
merged_df.head()
```

SPILLMAN

**3.5 Years of Data
22+ text files per year
Data needed in Rows**

Drag & Drop 22 files into Google Colab.



	A	B	C	D	E
1	03/11/25	West Jordan Police Department	3265		
2	13:09	CAD	MASTER CALL TABLE	Page: 1	
3					
4	Record number	:	C10622404		
5	Call type (Law/Fire/EMS)	:	If		
6	Call nature	:	26A10 26A10 DEAF		
7	Priority of Call	:		1	
8	When Occured Earliest	:		1/31/2018 23:48	
9	When Occured Latest	:		1/31/2018 23:48	
10	When Reported	:		1/31/2018 23:49	
11	Respond-to Address	:	9658 S PAISLEY CIR; 4165 W		
12	City code	:	SJC		
13	Person to Contact	:	SPARKS, SHAWN		
14	Contacts Phone Number	:	(801)647-7127		
15	Complainant name number	:			
16	How Received	:	9911 Line		
17	Who took Call	:	ANSBERGS, I		
18	Alarm ID number	:			
19	Geobase address ID	:		158192	
20	Directions number	:			
21	Contacts address	:	3647 W 9800 S - W Sector		
22	Call Received? (Y/N)	:	Y		
23	Hold Until	:	*****:*****:*****:*****		
24	Plate number of stopped veh	:			
25	Plate state abbreviation	:			
26	Determinant	:	26A10		
27	=====	=====			
28					
29					
30					
31					
32					
33					
34					
35					
36					
37					
38					
39					

ChatGPT

in Google Colab, I need to unpivot my data set that is in rows. The first set of data starts in row 1 and ends in row 27. There are 33 blank spaces between each data set. The category names (column headers) are in column A. The only data I want to unpivot is Record number, Call nature, Priority of Call, When Reported and How Received. Return the data in column C.

Great — you've confirmed the full structure, so here's a clean and **complete solution** for Google Colab that does exactly what you need:

Goal Recap:

- Data is laid out **vertically in rows** (one record spans 27 rows).
- Field names (e.g., "Record number") are in **Column A** (index `0`).
- Desired **field values** are in **Column C** (index `2`).
- Each record is followed by **33 blank rows** before the next one.
- Fields to extract:
 - Record number
 - Call nature
 - Priority of Call
 - When Reported
 - How Received

3. Parse & Unpivot from Column C

python

```
# Constants based on your layout
record_block_size = 27
gap_between_records = 33
total_record_stride = record_block_size + gap_between_records

# Field names to extract
desired_fields = ["Record number", "Call nature", "Priority of Call", "When Reported", "How Received"]

# Prepare output list
records = []

# Iterate through all record blocks
for start_row in range(0, len(df), total_record_stride):
    record_block = df.iloc[start_row:start_row + record_block_size]

    if record_block.empty or pd.isna(record_block.iloc[0, 0]):
        continue # skip empty sections

    record_dict = {}
    for i in range(record_block.shape[0]):
        label = str(record_block.iloc[i, 0]).strip()
        value = record_block.iloc[i, 2] # Now taking values from **Column C** (index 2)

        if label in desired_fields:
            record_dict[label] = value

    if record_dict:
        records.append(record_dict)

# Build final DataFrame
flat_df = pd.DataFrame(records)

# Show the first few rows
flat_df.head()
```

Copy code

```
# Constants based on your layout
record_block_size = 27
gap_between_records = 33
total_record_stride = record_block_size + gap_between_records

# Field names to extract
desired_fields = ["Record number", "Call nature", "Priority of Call", "When Reported", "How Received"]

# Prepare output list
records = []

# Iterate through all record blocks
for start_row in range(0, len(df), total_record_stride):
    record_block = df.iloc[start_row:start_row + record_block_size]

    if record_block.empty or pd.isna(record_block.iloc[0, 0]):
        continue # skip empty sections

    record_dict = {}
    for i in range(record_block.shape[0]):
        label = str(record_block.iloc[i, 0]).strip()
        value = record_block.iloc[i, 2] # Now taking values from **Column C** (index 2)

        if label in desired_fields:
            record_dict[label] = value

    if record_dict:
        records.append(record_dict)

# Build final DataFrame
flat_df = pd.DataFrame(records)

# Show the first few rows
flat_df.head()
```

Clipboard

Font

Styles

A30

	A	B	C	D	E	F	G
1	03/11/25	West Jo	r dan Police Department	3265			
2	13:09	CAD	MASTER CALL TABLE	Page: 1			
3							
4	Record number	:	C10622404				
5	Call type (Law/Fire/EMS)	:	If				
6	Call nature	:	26A10 26A10 DEAF				
7	Priority of Call	:		1			
8	When Occured Earliest	:		1/31/2018 23:48			
9	When Occured Latest	:		1/31/2018 23:48			
10	When Reported	:		1/31/2018 23:49			
11	Respond-to Address	:	9658 S PAISLEY CIR; 4165 W				
12	City code	:	SJC				
13	Person to Contact	:	SPARKS, SHAWN				
14	Contacts Phone Number	:	(801)647-7127				
15	Complainant name number	:					
16	How Received	:	9 911 Line				
17	Who took Call	:	ANSBERGS, I				
18	Alarm ID number	:					
19	Geobase address ID	:		158192			
20	Directions number	:					
21	Contacts address	:	3647 W 9800 S - W Sector				
22	Call Received? (Y/N)	:	Y				
23	Hold Until	:	*****/*****				
24	Plate number of stopped veh	:					
25	Plate state abbreviation	:					
26	Determinant	:	26A10				
27	= =====	= =====					
28							
29							
30							
31							
32							
33							
34							
35							
36							
37							
38							
39							

Constants based on your layout
record_block_size = 27
gap_between_records = 33
total_record_stride = record_block_size + gap_between_records

Field names to extract
desired_fields = ["Record number", "Call nature", "Priority of Call", "When Reported", "How Received"]

Prepare output list
records = []

Iterate through all record blocks
for start_row in range(0, len(df), total_record_stride):
 record_block = df.iloc[start_row:start_row + record_block_size]

if record_block.empty or pd.isna(record_block.iloc[0, 0]):
 continue # skip empty sections

record_dict = {}
 for i in range(record_block.shape[0]):
 label = str(record_block.iloc[i, 0]).strip()
 value = record_block.iloc[i, 2] # Now taking values from **Column C** (index 2)

if label in desired_fields:
 record_dict[label] = value

if record_dict:
 records.append(record_dict)

Build final DataFrame
flat_df = pd.DataFrame(records)

Show the first few rows
flat_df.head()

	Record number	Call nature	Priority of Call	When Reported	How Received
0	C10622404	26A10 26A10 DEAF	1	1/31/2018 23:49	9 911 Line
1	C10622383	CELL HANGUPMIS	5	1/31/2018 23:27	9 911 Line
2	C10622371	DISTURB PEACE DISTURB PEACE REPORT	2	1/31/2018 23:17	T Telephone
3	C10622366	ASSIST PUBLIC ASSIST PUBLIC	3	1/31/2018 23:17	T Telephone
4	C10622355	DOM VERBAL IP DOMESTIC VERBAL IN PROGRESS	1	1/31/2018 23:10	9 911 Line

THANK YOU

Suzanna Howard

Suzanna.howard@westjordan.Utah.gov

West Jordan Police Department

CREATING EFFICIENCY WITH GOOGLE COLAB

HOUR TWO

**SUZANNA HOWARD
WEST JORDAN (UT) POLICE DEPARTMENT**

```
// life motto
if (sad() === true) {
  sad().stop();
  beAwesome();
}
```

Google Colaboratory, commonly known as Google Colab, is a free, cloud-based Jupyter Notebook environment provided by Google. It enables users to write and execute Python code directly in their web browser, making it particularly useful for machine learning, data analysis, and educational purposes.

Key features and aspects of Google Colab include:

Cloud-based Environment:

It runs entirely in the cloud, eliminating the need for local setup or powerful hardware.

Free Access to Resources:

Users can access computing resources, including GPUs and TPUs, which are beneficial for computationally intensive tasks like deep learning.

Jupyter Notebook Compatibility:

Colab notebooks are compatible with Jupyter notebooks, allowing for the combination of executable code, rich text, images, and other media within a single document.

Integration with Google Drive:

It seamlessly integrates with Google Drive for easy saving, sharing, and collaboration on notebooks.

AI-First Features:

Recent advancements have introduced AI-powered capabilities, such as iterative querying for code generation and explanation, a Next-Generation Data Science Agent for autonomous analysis, and effortless code transformation using natural language.

Data Import and Analysis:

Users can easily import data from various sources like Google Drive, GitHub, and spreadsheets, and leverage popular Python libraries for data analysis and visualization.

Collaboration:

Notebooks can be shared and collaborated on with multiple users, with access controlled through Identity and Access Management (IAM) for Colab Enterprise.

Paid Services:

While a free tier is available, Google also offers paid services like Colab Pro, Colab Pro+, and Colab Enterprise, which provide enhanced resources and features like faster

Use Google Colab online to build Jupyter Notebooks.

- 1. Drag and drop an Excel file into Google Colab**
- 2. Use AI-generated Python code to write the steps we would do in Excel (sort, insert columns, formulas, clean data, etc)**
- 3. Export files**
- 4. Save/download Jupyter Notebook for future use**

In Excel terms, it is similar to macros.



HOUR ONE

INTRODUCTION TO GOOGLE COLAB

CAVEATS

ACTIVITIES:

DANGEROUS INTERSECTIONS

GRAPHS / CHARTS

UNPIVOT

HOUR TWO

INTRODUCTION TO GOOGLE COLAB

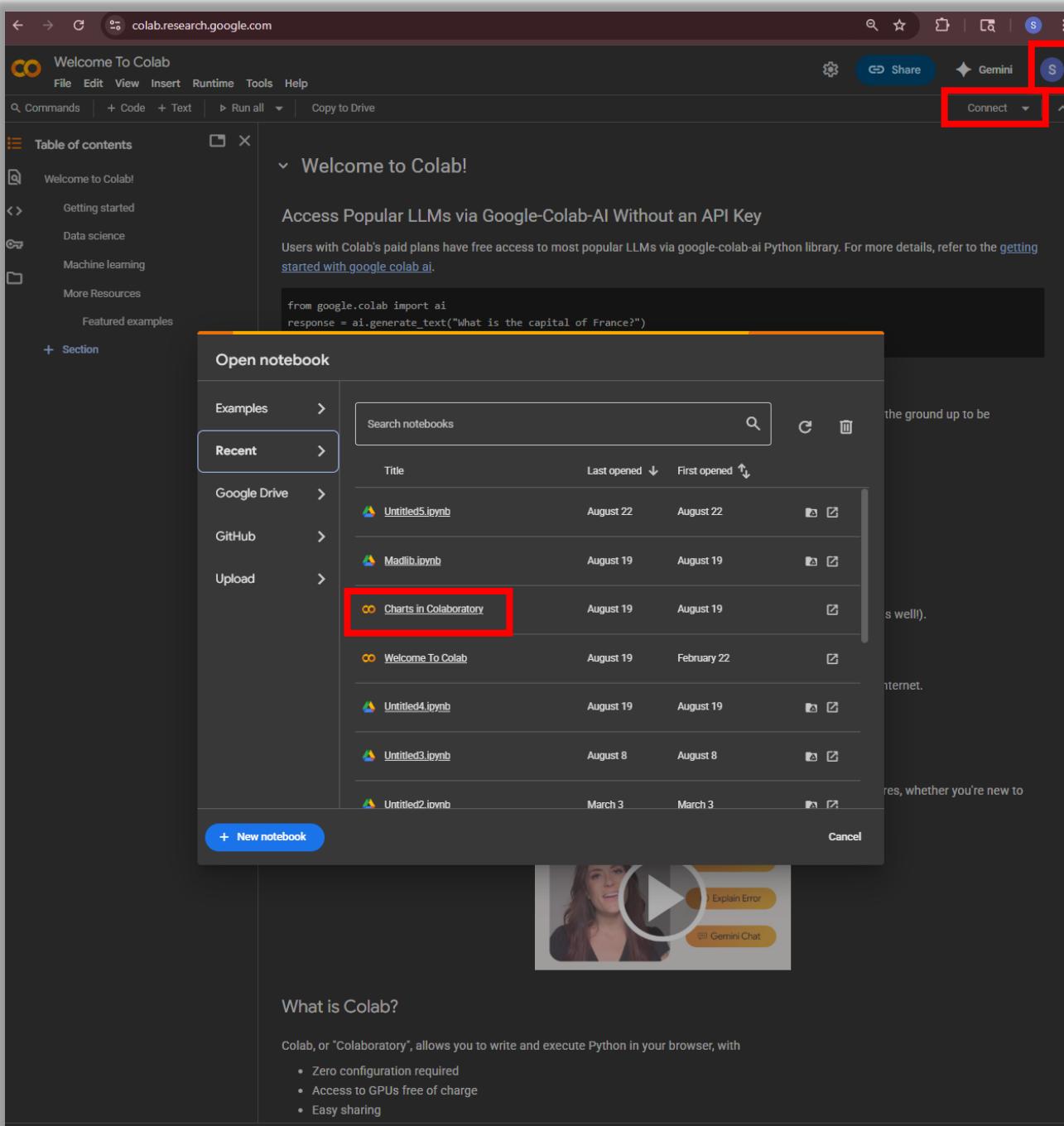
CAVEATS

ACTIVITIES:

OFFICER AVAILABILITY

FORECASTING / EDUCATED GUESSING

GRAPHS / CHARTS



Google “Google Colab”
<https://colab.research.google.com/>
Type: Colab.new

GMAIL ACCOUNT REQUIRED

FREE ONLINE TOOL

NO INSTALLATION/CONFIGURATION

JUPYTER NOTEBOOKS

**AI GENERATED PYTHON CODE
GEMINI
CHAT GPT
COMMENT**



Google what does google colab do with my data

AI Mode All Videos Images Short videos Shopping Forums Other Tools

◆ AI Overview

Google Colab handles user data in several ways, primarily related to the execution of Python code and the storage of notebooks.

Data Handling:

Notebook Storage:

Colab notebooks are, by default, stored in your Google Drive, specifically in a "Colab Notebooks" folder. This means the notebook content itself, including code and markdown cells, is saved within your Google Drive storage.

Execution Environment:

When you run code in a Colab notebook, it executes on Google's cloud servers. Any data that your code processes or generates during execution (e.g., loaded datasets, generated outputs, model weights) resides within this cloud environment.

Google Drive Integration:

You can explicitly mount your Google Drive within a Colab notebook to access files stored there. This allows your code to read from and write to your Google Drive, but this access is granted only when you explicitly authorize it.

Generative AI Features:

If you utilize generative AI features within Colab, Google collects data related to prompts, generated code, output, feature usage information, and feedback. This data is used to improve and develop Google products and services, including machine learning technologies. Google states this data is stored in a way that prevents personal identification and is retained for a limited period. ☀

Security and Privacy Considerations:

Shared Notebooks:

When sharing a Colab notebook, be aware that if the notebook contains malicious code and you execute it, it could potentially access your Google Drive data if you have mounted it and granted permissions.

Data Access:

Colab AI features do not have access to your Google Drive files or user secrets by default. Access is only granted if you explicitly request it within your code. ☀

Encryption and Transparency (Colab Enterprise):

For enterprise users, Colab Enterprise offers features like customer-managed encryption keys (CMEK) and Access Transparency, providing more control and visibility over data handling and access by Google personnel.

People also ask :

Does Google Colab use my data?

By default, Colab AI does not have access to your Google Drive files or user secrets.

However, it can generate code that accesses them at your explicit request.

Google Research

<https://research.google.com> > colaboratory > faq

Google Colab

Does Google Colab have access to my files?

Since a Colab notebook is hosted on Google's cloud servers, [there's no direct access to files on your local drive](#) (unlike a notebook hosted on your machine) or any other environment by default. However, Colab provides various options to connect to almost any data source you can imagine. Mar 14, 2025

Does Google keep sell my data?

We never sell your personal information, and give you controls over who has access. We are committed to protecting your data from third parties. That's why it's our strict policy to never sell your personal information to anyone.

Google Safety Center
<https://safety.google> > privacy > data

Your privacy is protected by responsible data practices.

Can Google see everything I do?

If You Use Google Products

If you do use Google products, they try to track even more. In addition to tracking everything you've ever searched for on Google (e.g. "weird rash"), Google also tracks every video you've ever watched on YouTube. Many people actually don't know that Google owns YouTube; now you know.

What does Google do with all your data?

Google uses the information shared by sites and apps to deliver our services, maintain and improve them, develop new services, measure the effectiveness of advertising, protect against fraud and abuse, and personalize content and ads you see on Google and on our partners' sites and apps.

Google Policies

<https://policies.google.com> > technologies > partner-sites

How Google uses information from sites or apps that use our services

Does Google access your files?

We access your private content only when we have your permission or are required to by law. With the Google Transparency Report, we share data about how the policies and actions of governments and corporations affect privacy, security, and access to information.

Does Google take your personal data?

First, we make sure that it's securely processed and stored as stated in the Google Privacy Policy. We use your personal information solely for the purpose of running your account and communicating account information to you.

Google Help
<https://support.google.com> > adsense > answer

What does Google do with my personal information?

Does Google keep your data safe?

Data is encrypted in-transit and at-rest. If you choose to access these files offline, we store this info on your device. Your Google Account comes with built-in security designed to detect and block threats like spam, phishing and malware. Your activity is stored using strong industry standards and practices.

ChatGPT

what information should i avoid uploading to google colab?

When using **Google Colab**, you should avoid uploading any sensitive, private, or regulated data. While Google Colab is a secure platform, it's still a cloud-based environment, and you should follow best practices to protect yourself and others. Specifically, **do not upload**:

🚫 Personal Identifiable Information (PII):

- Full names linked with other personal data
- Social Security Numbers (SSNs)
- Driver's license numbers
- Passport numbers
- Home addresses, email addresses, or phone numbers (if sensitive)

🚫 Financial Information:

- Credit card or bank account numbers
- Financial transaction records tied to individuals
- Tax documents

🚫 Login Credentials or Authentication Tokens:

- Passwords (even hashed ones)
- API keys or OAuth tokens
- SSH private keys
- Firebase or AWS credentials
- Google Cloud service account keys

🚫 Medical/Health Data (HIPAA-covered info):

- Patient health records
- Medical history
- Lab/test results tied to a person

🚫 Confidential or Proprietary Business Information:

- Internal strategy documents
- Trade secrets
- Non-public client or company data

🚫 Government Classified or Export-Controlled Data (e.g., ITAR, EAR):

- Data subject to national security or export restrictions



✅ Safe to Upload:

- Public datasets
- Open-source code
- Your own hobby/educational project data (that doesn't contain sensitive info)
- Anonymized or synthetic data for experiments

⚠️ Tips to Stay Safe:

- Use `.gitignore` and avoid committing secrets to notebooks
- Use environment variables for secrets (e.g., `os.environ`)
- Use Colab's "secrets" management with Google Drive or GCP, if needed
- Revoke API keys if you accidentally upload them

Would you like help scanning a Colab notebook or script for sensitive data before uploading it?

Law Enforcement Specific

- Avoid Victim/Suspect/Officer Information
- Use data that would be on a public facing website
- Get permission from Admin
- Anonymize, if possible

OFFICER AVAILABILITY

SPILLMAN / VERSATERM

The screenshot shows a Jupyter Notebook interface with the following steps:

- [1] `#import libraries`
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
- [2] # import CAD2018.xlsx
df = pd.read_excel('CAD2018.xlsx')
df.head()
- [3] df.shape
(220600, 5)
- [4] df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 220600 entries, 0 to 220599
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Date/Time   220600 non-null  datetime64[ns]
 1   Unit        220600 non-null  object  
 2   Status      220600 non-null  object  
 3   CAD         220600 non-null  object  
 4   Nature      220600 non-null  object  
dtypes: datetime64[ns](1), object(4)
memory usage: 8.4+ MB
```
- [5] df.tail()

The notebook displays five rows of data from the 'Date/Time' column:

	Date/Time	Unit	Status	CAD	Nature
0	2018-01-01 00:12:37	7P237	CMPLT	C10561458	BURGLARY RESD
1	2018-01-01 00:12:39	7P237	ENRT	C10561723	NUISANCE
2	2018-01-01 00:16:31	7P242	ENRT	C10561737	FIRE HOUSE
3	2018-01-01 00:16:55	7P231	CMPLT	C10561619	DRUG OFFENSE
4	2018-01-01 00:16:56	7P231	ENRT	C10561737	FIRE HOUSE

And the last five rows of data from the 'Date/Time' column:

	Date/Time	Unit	Status	CAD	Nature
220595	2018-12-31 23:27:26	7P242	AB	C11281855	INTOX PERSON
220596	2018-12-31 23:32:49	7P256	CMPLT	C11281518	ANIMAL BITE
220597	2018-12-31 23:58:14	7P275	CMPLT	C11281240	DOMESTIC
220598	2018-12-31 23:59:11	7P256	ENRT	C11281579	KEEP THE PEACE
220599	2018-12-31 23:59:11	7P275	ENRT	C11281579	KEEP THE PEACE

Starter Steps

Rename Untitled Notebook

Click on Files, Drag and Drop file

Import Libraries

Import Excel file

df.head() – first 5 rows

Df.tail() – last 5 rows

df.shape – rows, columns

df.info() – format categories

Object = Text

Float64 = Decimal Number

Int64 = Whole Number

Datetime64 = Date/Time

Timedelta64[ns](1) = time

Sort by Unit then Date/Time

```
✓ 0s ⏴ # sort by Unit then Date/Time
df = df.sort_values(by=['Unit', 'Date/Time'])
df.head()
```

	Date/Time	Unit	Status	CAD	Nature	grid icon	bar chart icon
474	2018-01-01 18:21:02	7P127	ENRT	C10562907	WELFARE URGENT		
475	2018-01-01 18:21:04	7P127	ARRVD	C10562907	WELFARE URGENT		
476	2018-01-01 18:22:23	7P127	ARRVD	C10562907	WELFARE URGENT		
481	2018-01-01 18:26:42	7P127		4	C10562907 WELFARE URGENT		
487	2018-01-01 18:33:15	7P127		4	C10562907 WELFARE URGENT		

```
# create a new column called Time that subtracts the current Date/Time from the previous Date/Time
```

```
[7] # create a new column called Time that subtracts the current Date/Time from the previous Date/Time
df['Time'] = df.groupby('Unit')['Date/Time'].diff()
df.head()
```

	Date/Time	Unit	Status	CAD	Nature	Time
474	2018-01-01 18:21:02	7P127	ENRT	C10562907	WELFARE URGENT	NaT
475	2018-01-01 18:21:04	7P127	ARRVD	C10562907	WELFARE URGENT	0 days 00:00:02
476	2018-01-01 18:22:23	7P127	ARRVD	C10562907	WELFARE URGENT	0 days 00:01:19
481	2018-01-01 18:26:42	7P127		4	C10562907	WELFARE URGENT
487	2018-01-01 18:33:15	7P127		4	C10562907	WELFARE URGENT

```
[8] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 220600 entries, 474 to 186386
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Date/Time   220600 non-null   datetime64[ns]
 1   Unit        220600 non-null   object  
 2   Status      220600 non-null   object  
 3   CAD         220600 non-null   object  
 4   Nature      220600 non-null   object  
 5   Time        220490 non-null   timedelta64[ns]
dtypes: datetime64[ns](1), object(4), timedelta64[ns](1)
memory usage: 11.8+ MB
```

```
df.shape
```

```
(220600, 6)
```

```
# remove any rows that have a value in the Time column over 5:00:00
```

```
[9] df.shape  
0s ➔ (220600, 6)  
  
# remove any rows that have a value in the Time column over 5:00:00  
df = df[df['Time'] <= pd.Timedelta(hours=5, minutes=0, seconds=0)]  
df.head()  
  
Date/Time    Unit  Status      CAD       Nature        Time  
475 2018-01-01 18:21:04  7P127  ARRVD C10562907 WELFARE URGENT 0 days 00:00:02  
476 2018-01-01 18:22:23  7P127  ARRVD C10562907 WELFARE URGENT 0 days 00:01:19  
481 2018-01-01 18:26:42  7P127          4 C10562907 WELFARE URGENT 0 days 00:04:19  
487 2018-01-01 18:33:15  7P127          4 C10562907 WELFARE URGENT 0 days 00:06:33  
495 2018-01-01 18:38:05  7P127          4 C10562907 WELFARE URGENT 0 days 00:04:50  
  
[11] df.shape  
0s ➔ (213913, 6)
```

```
# create a new column called Date that includes only the date from the Date/Time column
```

```
0s # create a new column called Date that includes only the date from the Date/Time column
df['Date'] = df['Date/Time'].dt.date
df.head()
```

	Date/Time	Unit	Status	CAD	Nature	Time	Date
475	2018-01-01 18:21:04	7P127	ARRVD	C10562907	WELFARE URGENT	0 days 00:00:02	2018-01-01
476	2018-01-01 18:22:23	7P127	ARRVD	C10562907	WELFARE URGENT	0 days 00:01:19	2018-01-01
481	2018-01-01 18:26:42	7P127	4	C10562907	WELFARE URGENT	0 days 00:04:19	2018-01-01
487	2018-01-01 18:33:15	7P127	4	C10562907	WELFARE URGENT	0 days 00:06:33	2018-01-01
495	2018-01-01 18:38:05	7P127	4	C10562907	WELFARE URGENT	0 days 00:04:50	2018-01-01

```
0s [13] df.shape
```

```
(213913, 7)
```

```
0s [14] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 213913 entries, 475 to 186386
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Date/Time    213913 non-null   datetime64[ns]
 1   Unit         213913 non-null   object  
 2   Status        213913 non-null   object  
 3   CAD          213913 non-null   object  
 4   Nature        213913 non-null   object  
 5   Time          213913 non-null   timedelta64[ns]
 6   Date          213913 non-null   object  
dtypes: datetime64[ns](1), object(5), timedelta64[ns](1)
memory usage: 13.1+ MB
```

convert Date column to a Date format

```
0s df.info()
→ <class 'pandas.core.frame.DataFrame'>
Index: 213913 entries, 475 to 186386
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Date/Time   213913 non-null   datetime64[ns]
 1   Unit        213913 non-null   object  
 2   Status      213913 non-null   object  
 3   CAD         213913 non-null   object  
 4   Nature      213913 non-null   object  
 5   Time        213913 non-null   timedelta64[ns]
 6   Date        213913 non-null   object  
dtypes: datetime64[ns](1), object(5), timedelta64[ns](1)
memory usage: 13.1+ MB

[15] # convert Date column to a Date format
df['Date'] = pd.to_datetime(df['Date'])
df.head()

→
          Date/Time  Unit  Status      CAD  Nature      Time      Date
475  2018-01-01 18:21:04  7P127  ARRVD  C10562907  WELFARE URGENT  0 days 00:00:02  2018-01-01
476  2018-01-01 18:22:23  7P127  ARRVD  C10562907  WELFARE URGENT  0 days 00:01:19  2018-01-01
481  2018-01-01 18:26:42  7P127       4  C10562907  WELFARE URGENT  0 days 00:04:19  2018-01-01
487  2018-01-01 18:33:15  7P127       4  C10562907  WELFARE URGENT  0 days 00:06:33  2018-01-01
495  2018-01-01 18:38:05  7P127       4  C10562907  WELFARE URGENT  0 days 00:04:50  2018-01-01

↑ ↓ ♦

0s df.info()
→ <class 'pandas.core.frame.DataFrame'>
Index: 213913 entries, 475 to 186386
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Date/Time   213913 non-null   datetime64[ns]
 1   Unit        213913 non-null   object  
 2   Status      213913 non-null   object  
 3   CAD         213913 non-null   object  
 4   Nature      213913 non-null   object  
 5   Time        213913 non-null   timedelta64[ns]
 6   Date        213913 non-null   datetime64[ns]
dtypes: datetime64[ns](2), object(4), timedelta64[ns](1)
memory usage: 13.1+ MB
```

Gemini: Filter on the Status column, only rows with ENRT if it follows a row with CMPLT

```
[1] # Create a new column to store the previous status for each unit  
df['Previous_Status'] = df.groupby('Unit')['Status'].shift(1)  
  
# Filter the DataFrame  
filtered_df = df[(df['Status'] == 'ENRT') & (df['Previous_Status'] == 'CMPLT')]  
  
# Display the filtered DataFrame  
display(filtered_df.head())
```

	Date/Time	Unit	Status	CAD	Nature	Time	Date	Previous_Status
510	2018-01-01 18:45:21	7P127	ENRT	C10562948	25B1W	0 days 00:00:01	2018-01-01	CMPLT
575	2018-01-01 20:25:30	7P127	ENRT	C10563118	ALARM BURG COMM	0 days 00:07:21	2018-01-01	CMPLT
630	2018-01-01 21:33:44	7P127	ENRT	C10563226	SHOTS FIRED	0 days 00:02:22	2018-01-01	CMPLT
692	2018-01-01 23:38:22	7P127	ENRT	C10563335	SUSP VEHICLE	0 days 02:04:16	2018-01-01	CMPLT
731	2018-01-02 00:17:37	7P127	ENRT	C10563369		90	0 days 00:24:07	2018-01-02

```
[18] df.shape  
[2] (213913, 8)
```

```
[19] filtered_df.shape  
[2] (28054, 8)
```

```
[20] # rename filtered_df to dff  
dff = filtered_df
```

```
[21] dff.shape  
[2] (28054, 8)
```

Gemini:

Create a new column called SUM, which sums the Time column, group by Unit and Date

Create a new column called AVG, which averages the Time column, group by Unit and Date

Create a new column called SumAVG, which averages the SUM column, group by Unit

Create a new column called AVGAvail, which averages the AVG column, group by Unit

```
✓ 0s ⏪ # Create a new column called SUM, which sums the Time column, group by Unit and Date
dff['SUM'] = dff.groupby(['Unit', 'Date'])['Time'].transform('sum')

# Create a new column called AVG, which averages the Time column, group by Unit and Date
dff['AVG'] = dff.groupby(['Unit', 'Date'])['Time'].transform('mean')

# Create a new column called SumAVG, which averages the SUM column, group by Unit
dff['SumAVG'] = dff.groupby('Unit')['SUM'].transform('mean')

# Create a new column called AVGAvail, which averages the AVG column, group by Unit
dff['AVGAvail'] = dff.groupby('Unit')['AVG'].transform('mean')

display(dff.head())
```

✓ 0s ⏪ dff.shape
→ (28054, 16)

Date/Time	Unit	Status	CAD	Nature	Time	Date	Previous_Status	SUM	AVG	SumAVG	AVGAvail
2018-01-01 18:45:21	7P127	ENRT	C10562948	25B1W	0 days 00:00:01	2018-01-01		CMPLT	0 days 02:14:00	0 days 00:33:30	02:01:52.702127659 00:25:47.155319148
2018-01-01 20:25:30	7P127	ENRT	C10563118	ALARM BURG COMM	0 days 00:07:21	2018-01-01		CMPLT	0 days 02:14:00	0 days 00:33:30	02:01:52.702127659 00:25:47.155319148
2018-01-01 21:33:44	7P127	ENRT	C10563226	SHOTS FIRED	0 days 00:02:22	2018-01-01		CMPLT	0 days 02:14:00	0 days 00:33:30	02:01:52.702127659 00:25:47.155319148
2018-01-01 23:38:22	7P127	ENRT	C10563335	SUSP VEHICLE	0 days 02:04:16	2018-01-01		CMPLT	0 days 02:14:00	0 days 00:33:30	02:01:52.702127659 00:25:47.155319148
2018-01-02 00:17:37	7P127	ENRT	C10563369	90	0 days 00:24:07	2018-01-02		CMPLT	0 days 00:24:07	0 days 00:24:07	02:01:52.702127659 00:25:47.155319148

remove duplicate Units

```
[28] # remove duplicate Units
dff = dff.drop_duplicates(subset=['Unit'])
dff.shape
(68, 16)

display(dff.head(16))
```

	Date/Time	Unit	Status	CAD	Nature	Time	Date	Previous_Status	SUM	AVG	SumAVG	AVGAvail
510	2018-01-01 18:45:21	7P127	ENRT	C10562948	25B1W	0 days 00:00:01	2018-01-01		CMPLT 0 days 02:14:00	0 days 00:33:30	02:01:52.702127659	00:25:47.155319148
991	2018-01-02 07:49:34	7P131	ENRT	C10563663	SUSP VEHICLE	0 days 00:00:01	2018-01-02		CMPLT 0 days 00:00:07	0 days 00:00:02.333333333	01:27:15.563282336	00:17:46.194714881
159413	2018-09-18 10:11:38	7P140	ENRT	C11082848	SUSP VEHICLE	0 days 00:47:54	2018-09-18		CMPLT 0 days 01:20:47	0 days 00:26:55.666666666	01:42:52.868000	00:19:37.519999999
1039	2018-01-02 10:12:32	7P141	ENRT	C10563727	ASSIST PUBLIC	0 days 00:01:12	2018-01-02		CMPLT 0 days 01:09:01	0 days 00:11:30.166666666	01:42:31.142028985	00:23:54.220289854
5506	2018-01-10 15:15:16	7P143	ENRT	C10579719	THEFT VEH	0 days 00:04:11	2018-01-10		CMPLT 0 days 01:02:25	0 days 00:20:48.333333333	01:24:47.610632183	00:17:11.418103448
1081	2018-01-02 14:58:12	7P145	ENRT	C10564521	ASSIST AGENCY	0 days 01:09:47	2018-01-02		CMPLT 0 days 01:43:15	0 days 00:20:39	02:04:01.953642384	00:23:01.662251655
3556	2018-01-07 08:33:30	7P155	ENRT	C10573617	WELFARE URGENT	0 days 02:06:25	2018-01-07		CMPLT 0 days 03:30:42	0 days 00:35:07	02:35:54.679525222	00:21:57.921364984
1949	2018-01-04 07:38:28	7P174	ENRT	C10567719	DISTURB PHYS IP	0 days 01:42:42	2018-01-04		CMPLT 0 days 01:43:11	0 days 00:51:35.500000	02:26:43.705035971	00:22:29.848920863
5543	2018-01-10 16:16:33	7P176	ENRT	C10580066	ASSIST PUBLIC	0 days 00:18:18	2018-01-10		CMPLT 0 days 01:18:12	0 days 00:15:38.400000	01:33:58.242990654	00:19:56.708411214
331	2018-01-01 08:41:10	7P177	ENRT	C10562165	JAIL RELEASE	0 days 00:19:27	2018-01-01		CMPLT 0 days 03:58:12	0 days 00:59:33	01:38:17.014577259	00:19:30.511661807
2974	2018-01-06 06:26:27	7P179	ENRT	C10571731	ACCIDENT	0 days 00:00:01	2018-01-06		CMPLT 0 days 04:16:10	0 days 00:42:41.666666666	02:11:53.170984455	00:23:23.290155440
7534	2018-01-13 18:32:45	7P185	ENRT	C10586256	SUSP VEHICLE	0 days 02:36:33	2018-01-13		CMPLT 0 days 03:20:36	0 days 00:28:39.428571428	01:19:00.181692094	00:12:44.101248266

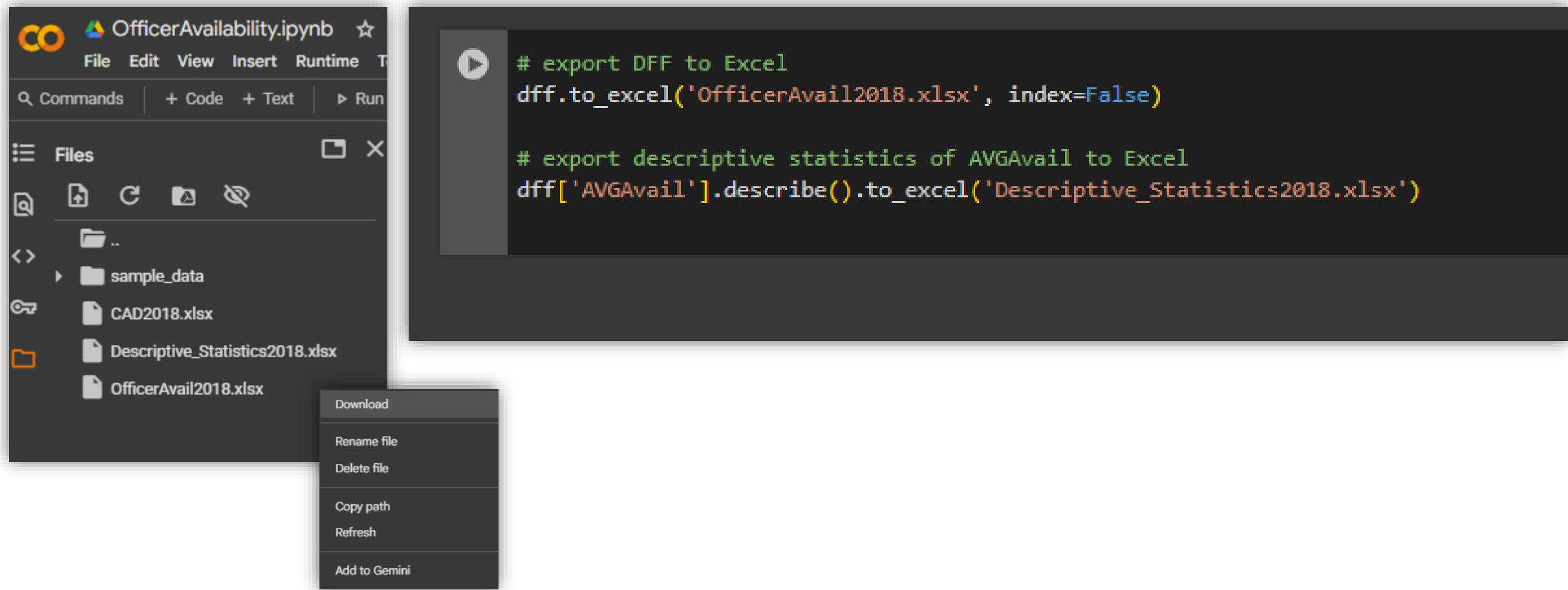
```
# run descriptive statistics on AVGAvail
```

```
✓ 0s # run descriptive statistics on AVGAvail
dff['AVGAvail'].describe()

AVGAvail
count          68
mean   0 days 00:16:40.677217359
std    0 days 00:07:02.370017696
min    0 days 00:00:01
25%   0 days 00:12:05.642669172
50%   0 days 00:17:07.646344634
75%   0 days 00:22:14.434168991
max   0 days 00:29:37.988179668

dtype: object
```

```
# export DFF and descriptive statistics of AVGAval in Excel
```



The screenshot shows a Jupyter Notebook interface with a dark theme. On the left, there's a sidebar with icons for file operations like 'Commands', 'File', 'Edit', 'View', 'Insert', 'Runtime', and 'Terminal'. Below these are sections for 'Files' (with icons for folder, file, copy, cut, paste, and refresh), 'sample_data' (a folder icon), 'CAD2018.xlsx' (file icon), 'Descriptive_Statistics2018.xlsx' (file icon), and 'OfficerAvail2018.xlsx' (file icon). A context menu is open over the 'OfficerAvail2018.xlsx' file, listing options: 'Download' (highlighted in grey), 'Rename file', 'Delete file', 'Copy path', 'Refresh', and 'Add to Gemini'. The main workspace contains two code cells. The first cell contains:

```
# export DFF to Excel  
dff.to_excel('OfficerAvail2018.xlsx', index=False)
```

The second cell contains:

```
# export descriptive statistics of AVGAval to Excel  
dff['AVGAval'].describe().to_excel('Descriptive_Statistics2018.xlsx')
```

LET'S RACE

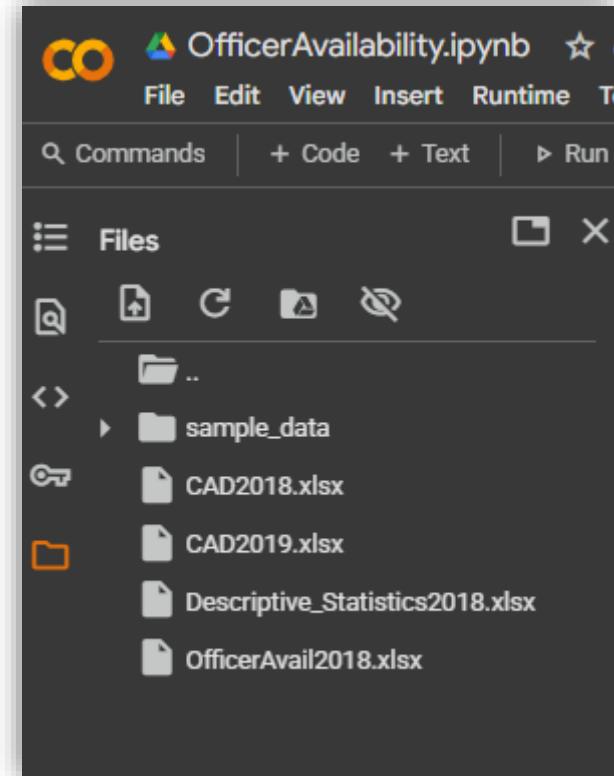
Rename import file to 2019

```
# import CAD2019.xlsx  
df = pd.read_excel('CAD2019.xlsx')  
df.head()
```

Rename export files to 2019

```
# export DFF to Excel  
dff.to_excel('OfficerAvail2019.xlsx', index=False)  
  
# export descriptive statistics of AVGAvail to Excel  
dff['AVGAvail'].describe().to_excel('Descriptive_Statistics2019.xlsx')
```

Drag & drop CAD2019.xlsx



Start at the top and push the play button on each step in order. Can you beat my time?

Raw records to Descriptive Statistics – 45.43 seconds

The image shows a desktop setup with three main windows:

- Microsoft Excel:** An open spreadsheet titled "CAD2019.xlsx" containing raw data with columns A through G. Column A is Date/Time, B is Unit, C is Status, D is CAD, E is Nature, F is, and G is. A red arrow points from the Excel window towards the Jupyter Notebook cell.
- Jupyter Notebook:** A code editor displaying Python code. The code includes:
 - # run descriptive statistics on AVGAvail
 - dff['AVGAvail'].describe()Below this, the output of the `describe()` method is shown for the "AVGAvail" column, providing statistical details like count, mean, std, min, 25%, 50%, 75%, and max. It also shows the dtype as object.
- Mobile Stopwatch:** A digital stopwatch application on an iPhone. The screen shows the time as 00:45.43. It has buttons for "Reset", "Start", and "Lap 1". At the bottom, there are icons for "World Clock", "Alarms", "Stopwatch", and "Timers".

VERSATERM

Total Daily Avg	January 2025	Fall 2024	Summer 2024	Winter 2024	Fall 2023	Summer 2023	Winter 2023	Fall 2022	Summer 2022	Winter 2022	Fall 2021	Avg by Shift
Weekday-Days	3:03:50	2:44:13	2:00:24	2:43:57	2:03:09	2:13:01	2:44:26	2:38:16	2:25:10	2:53:27	2:06:21	2:30:34
Weekday-Swings	1:41:52	1:41:34	1:14:23	1:41:06	2:00:51	1:31:19	2:11:03	1:33:03	1:28:49	1:29:58	1:47:27	1:40:08
Weekday-Graves	2:22:51	2:09:51	1:38:33	1:45:19	1:36:14	1:35:58	1:52:55	1:15:51	1:18:58	1:27:12	1:43:43	1:42:30
Weekend-Days	2:51:47	2:11:08	1:34:23	3:20:13	2:50:23	3:02:27	2:21:34	2:45:10	2:23:21	2:18:51	1:37:31	2:28:48
Weekend-Swings	1:57:38	1:40:38	1:36:47	2:35:58	2:18:38	2:12:40	1:47:11	2:02:56	1:52:35	1:42:42	1:53:52	1:58:20
Weekend-Graves	2:10:30	2:29:25	1:28:36	1:59:44	2:10:54	1:46:05	2:00:32	2:03:57	1:43:16	1:38:13	2:00:24	1:57:25
Average by Period	2:21:25	2:09:28	1:35:31	2:21:03	2:10:01	2:03:35	2:09:37	2:03:12	1:52:01	1:55:04	1:51:33	2:02:57

Average Avail	January 2025	Fall 2024	Summer 2024	Winter 2024	Fall 2023	Summer 2023	Winter 2023	Fall 2022	Summer 2022	Winter 2022	Fall 2021	Avg by Shift
Weekday-Days	0:43:12	0:15:40	0:14:52	0:25:40	0:14:25	0:15:35	0:52:53	0:20:57	0:16:43	0:23:09	0:18:46	0:23:48
Weekday-Swings	0:13:16	0:07:34	0:08:01	0:09:45	0:08:59	0:07:52	0:13:02	0:11:45	0:13:35	0:09:51	0:12:38	0:10:34
Weekday-Graves	0:37:10	0:14:20	0:15:13	0:14:35	0:12:30	0:11:35	0:14:42	0:13:17	0:12:49	0:15:35	0:15:00	0:16:04
Weekend-Days	0:25:56	0:08:21	0:12:52	0:18:11	0:14:27	0:13:49	0:14:08	0:15:18	0:11:46	0:15:33	0:12:40	0:14:49
Weekend-Swings	0:23:24	0:09:15	0:10:05	0:13:54	0:11:05	0:11:03	0:14:02	0:17:46	0:14:28	0:13:12	0:11:51	0:13:39
Weekend-Graves	0:25:12	0:13:16	0:09:00	0:12:23	0:12:00	0:08:13	0:13:24	0:18:05	0:13:10	0:14:49	0:15:03	0:14:03
Average by Period	0:28:02	0:11:24	0:11:40	0:15:45	0:12:14	0:11:21	0:20:22	0:16:11	0:13:45	0:15:22	0:14:20	0:15:30

SPILLMAN

Total Daily Availability

Year	Count	MIN	MAX	MEAN	MEDIAN
2018	67	0:00:04	3:42:39	1:42:42	1:42:30
2019	75	0:00:20	3:45:19	1:57:37	1:57:04
2020	63	0:48:48	3:26:39	2:07:36	2:05:57
2021	53	0:57:57	3:29:57	2:11:57	2:09:14

Average Availability between Calls

Year	Count	MIN	MAX	MEAN	MEDIAN
2018	67	0:00:04	0:32:18	0:18:24	0:18:24
2019	75	0:00:10	0:40:31	0:21:46	0:21:57
2020	63	0:10:44	0:49:58	0:23:27	0:21:22
2021	53	0:14:15	0:47:18	0:24:28	0:22:06

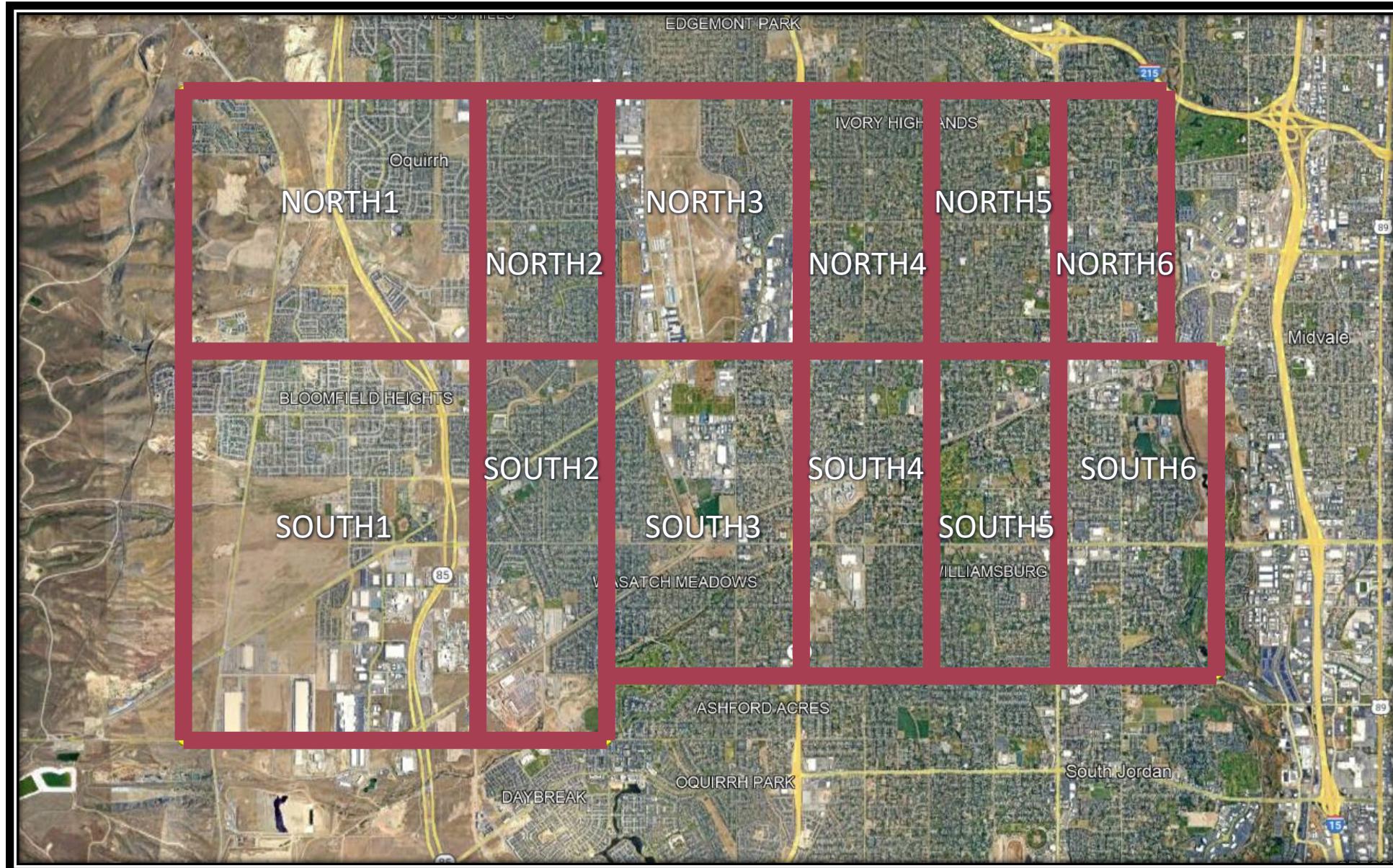
COMBINED

Year	Total Daily	Avg Btw Calls
2018	1:42:42	0:18:24
2019	1:57:37	0:21:46
2020	2:07:36	0:23:27
2021	2:01:45	0:19:24
2022	1:56:46	0:15:06
2023	2:07:44	0:14:39
2024	2:02:01	0:12:56

Versaterm – 11 Times (each shift change)
Spillman – 4 Times

FORECASTING

EXCEL



WEST JORDAN ZONES

Import Libraries – Add Excel Files – Add Column Called Zone – Merge Files

The screenshot shows a Google Colab notebook titled "Forecast.ipynb". On the left, there's a sidebar with file navigation and a list of files: "sample_data", "North1.xlsx", "North2.xlsx", "North3.xlsx", "North4.xlsx", "North5.xlsx", "North6.xlsx", "South1.xlsx", "South2.xlsx", "South3.xlsx", "South4.xlsx", "South5.xlsx", "South6.xlsx", and "merged_with_zone.xlsx". The main area contains two code cells:

```
[1] # Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

[2] import pandas as pd
import glob
import os

# If files are in your Google Drive, mount it first:
# from google.colab import drive
# drive.mount('/content/drive')

# Path to your Excel files (adjust if using Google Drive)
path = "/content/"  # folder where your 12 Excel files are stored
all_files = glob.glob(os.path.join(path, "*.xlsx"))

dfs = []

for file in all_files:
    # Read Excel file
    df = pd.read_excel(file)

    # Extract filename without path & extension
    zone_name = os.path.splitext(os.path.basename(file))[0]

    # Add Zone column
    df["Zone"] = zone_name

    dfs.append(df)

# Merge all into one DataFrame
merged_df = pd.concat(dfs, ignore_index=True)

# Save to a new Excel file
merged_df.to_excel("merged_with_zone.xlsx", index=False)
```

To the right, a ChatGPT window is open with the following message:

In google colab, I have 12 Excel files that I want to import and merge, but before I merge the files, I want to insert a column named Zone into each file and use the name of the file as the value.

ChatGPT responds:

Got it! You can do this easily in Google Colab with **pandas**. The steps will be:

1. Upload your 12 Excel files into Colab (either manually or from Google Drive).
2. Loop through each file, read it with `pd.read_excel()`.
3. Create a new column called `Zone` that stores the filename (or a cleaned version of it).
4. Append them together into one big DataFrame.

Import New Merged Excel File, df.head(), df.tail()

```
# import merged_with_zone.xlsx
df = pd.read_excel("merged_with_zone.xlsx")
df.head()
```

	Case Number	Report Date/Time	Occurred On	Status	Primary Offense	Location	Municipality	Zone
0	WJ 2025-FG31	Feb-27-2025 1710	Feb-23-2025 2000	CLOSED/COMPLETED	THVP-0 THEFT VEHICLE PARTS ...	XXXX W GOLDENPOINTE WAY	WEST JORDAN	South5
1	WJ 2025-F1F	Jan-07-2025 0800	Jan-07-2025 0800	CLOSED/COMPLETED	RVOL-0 REC VEH OUTSIDE RECOVER WJ ...	XXXX W XXXX S	WEST JORDAN	South5
2	WJ 2025-E50F	Feb-20-2025 1603	Feb-19-2025 2200	CLOSED/COMPLETED	THOT-0 THEFT OTHER ...	XXXX S XXXX W #XXX	WEST JORDAN	South5
3	WJ 2025-E4JF	Feb-20-2025 1305	Feb-19-2025 1730	CLOSED/COMPLETED	THOT-0 THEFT OTHER ...	XXXX W XXXX S	WEST JORDAN	South5
4	WJ 2025-E2J1	Feb-19-2025 1240	Feb-05-2025 0842	CLOSED/COMPLETED	BVVB-0 BURGLARY, VEHICLE BURGLARY ...	XXXX S REDWOOD RD	WEST JORDAN	South5

Next steps: [Generate code with df](#) [View recommended plots](#) [New interactive sheet](#)

```
[4]: df.tail()
```

	Case Number	Report Date/Time	Occurred On	Status	Primary Offense	Location	Municipality	Zone
12914	WJ 2021-10F4F	Jun-25-2021 1441	Jun-25-2021 0630	CLOSED/COMPLETED	POCM-0 PROPERTY CRIMINAL MISCHIEF ...	XXXX W XXXX S	WEST JORDAN	South6
12915	WJ 2021-10F43	Jun-25-2021 1412	Jun-25-2021 1412	CLOSED/COMPLETED	POCM-0 PROPERTY CRIMINAL MISCHIEF ...	XXXX S TEMPLE DR	WEST JORDAN	South6
12916	WJ 2021-10F1J	Jun-25-2021 1228	Jun-25-2021 1228	CLOSED/COMPLETED	POCM-0 PROPERTY CRIMINAL MISCHIEF ...	XXXX W XXXX S	WEST JORDAN	South6
12917	WJ 2021-10J1J	Jun-23-2021 1939	Jun-23-2021 1939	CLOSED/COMPLETED	POTR-0 PROPERTY TRESPASSING ...	XXXX W XXXX S #XX	WEST JORDAN	South6
12918	WJ 2021-1052J	Jun-23-2021 1252	Jun-23-2021 1252	CLOSED/COMPLETED	POPD-0 PROPERTY DAMAGE (NON-CRIMINAL) ...	XXXX W XXXX S	WEST JORDAN	South6

```
df.shape
```

(12919, 8)

```
df.info()
```

#	Column	Non-Null Count	Dtype
0	Case Number	12919 non-null	object
1	Report Date/Time	12919 non-null	object
2	Occurred On	12919 non-null	object
3	Status	12919 non-null	object
4	Primary Offense	12919 non-null	object
5	Location	12919 non-null	object
6	Municipality	12919 non-null	object
7	Zone	12919 non-null	object

dtypes: object(8)
memory usage: 807.6+ KB

Convert Date fields to Date Format

```
✓ [7] # convert Report Date/Time and Occurred On to Date/Time format
0s   df["Report Date/Time"] = pd.to_datetime(df["Report Date/Time"])
      df["Occurred On"] = pd.to_datetime(df["Occurred On"])
```

```
✓ [8] df.info()
```

```
→ <class 'pandas.core.frame.DataFrame'>
RangeIndex: 12919 entries, 0 to 12918
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Case Number      12919 non-null   object  
 1   Report Date/Time 12919 non-null   datetime64[ns]
 2   Occurred On      12919 non-null   datetime64[ns]
 3   Status           12919 non-null   object  
 4   Primary Offense  12919 non-null   object  
 5   Location          12919 non-null   object  
 6   Municipality     12919 non-null   object  
 7   Zone              12919 non-null   object  
dtypes: datetime64[ns](2), object(6)
memory usage: 807.6+ KB
```

Insert columns for Day of Week and Hour (based on Report Date/Time)

```
# insert a column called DOW for Day of the Week name of Report Date/Time
df.insert(2, "DOW", df["Report Date/Time"].dt.day_name())

# insert a column called Hour for the hour of day of Report Date/Time
df.insert(3, "Hour", df["Report Date/Time"].dt.hour)

df.head()
```

	Case Number	Report Date/Time	DOW	Hour	Occurred On	Status	Primary Offense	Location	Municipality	Zone
0	WJ 2025-FG31	2025-02-27 17:10:00	Thursday	17	2025-02-23 20:00:00	CLOSED/COMPLETED	THVP-0 THEFT VEHICLE PARTS ...	XXXX W GOLDENPOINTE WAY	WEST JORDAN	South5
1	WJ 2025-F1F	2025-01-07 08:00:00	Tuesday	8	2025-01-07 08:00:00	CLOSED/COMPLETED	RVOL-0 REC VEH OUTSIDE RECOVER WJ ...	XXXX W XXXX S	WEST JORDAN	South5
2	WJ 2025-E50F	2025-02-20 16:03:00	Thursday	16	2025-02-19 22:00:00	CLOSED/COMPLETED	THOT-0 THEFT OTHER ...	XXXX S XXXX W #XXX	WEST JORDAN	South5
3	WJ 2025-E4JF	2025-02-20 13:05:00	Thursday	13	2025-02-19 17:30:00	CLOSED/COMPLETED	THOT-0 THEFT OTHER ...	XXXX W XXXX S	WEST JORDAN	South5
4	WJ 2025-E2J1	2025-02-19 12:40:00	Wednesday	12	2025-02-05 08:42:00	CLOSED/COMPLETED	BVVB-0 BURGLARY, VEHICLE BURGLARY ...	XXXX S REDWOOD RD	WEST JORDAN	South5

Next steps: [Generate code with df](#) [View recommended plots](#) [New interactive sheet](#)

```
df.shape  
(12919, 10)
```

Sort by Zone, Report Date/Time (Ascending)

[11] # sort by Zone then Report Date/Time ascending
df = df.sort_values(by=["Zone", "Report Date/Time"])
df.head()

	Case Number	Report Date/Time	DOW	Hour	Occurred On	Status	Primary Offense	Location	Municipality	Zone
4217	WJ 2021-F40E	2021-06-17 07:49:00	Thursday	7	2021-06-17 07:49:00	CLOSED/COMPLETED	BVVB-0 BURGLARY, VEHICLE BURGLARY ...	XXXX S JACKLING WAY	WEST JORDAN	North1
4288	WJ 2021-1052E	2021-06-23 13:02:00	Wednesday	13	2021-06-23 13:02:00	CLOSED/COMPLETED	THVP-0 THEFT VEHICLE PARTS ...	XXXX S XXXX W	WEST JORDAN	North1
4287	WJ 2021-1054	2021-06-24 02:10:00	Thursday	2	2021-06-24 02:10:00	CLOSED/COMPLETED	RVOL-0 REC VEH OUTSIDE RECOVER WJ ...	XXXX W XXXX S	WEST JORDAN	North1
4286	WJ 2021-11542	2021-06-28 14:09:00	Monday	14	2021-06-28 14:09:00	CLOSED/COMPLETED	THVP-0 THEFT VEHICLE PARTS ...	XXXX W HAVEN MAPLE DR	WEST JORDAN	North1
4284	WJ 2021-12131	2021-07-01 12:07:00	Thursday	12	2021-07-01 12:07:00	CLOSED/COMPLETED	POTR-0 PROPERTY TRESPASSING ...	XXXX S ADVENTURE WAY	WEST JORDAN	North1

Next steps: [Generate code with df](#) [View recommended plots](#) [New interactive sheet](#)

df.tail()

	Case Number	Report Date/Time	DOW	Hour	Occurred On	Status	Primary Offense	Location	Municipality	Zone
12662	WJ 2025-3E323	2025-08-04 10:29:00	Monday	10	2025-08-04 07:00:00	CLOSED/EVIDENCE ONLY	PODP-0 PROPERTY DAMAGE PROPERTY (CRIMINAL) ...	XXXX W XXXX S	WEST JORDAN	South6
12651	WJ 2025-G003F0	2025-08-04 21:01:00	Monday	21	2025-08-04 04:30:00	CLOSED/COMPLETED	BVVB-0 BURGLARY, VEHICLE BURGLARY ...	XXXX S MILLRACE VIEW CIR	WEST JORDAN	South6
12661	WJ 2025-3E514	2025-08-05 12:39:00	Tuesday	12	2025-08-05 12:39:00	CLOSED/COMPLETED	PODP-0 PROPERTY DAMAGE PROPERTY (CRIMINAL) ...	XXXX W XXXX S	WEST JORDAN	South6
12650	WJ 2025-G0040F	2025-08-12 07:40:00	Tuesday	7	2025-08-04 08:00:00	OPEN/ACTIVE	THOT-0 THEFT OTHER ...	XXXX W XXXX S	WEST JORDAN	South6
12658	WJ 2025-425FE	2025-08-26 23:34:00	Tuesday	23	2025-08-26 23:34:00	CLOSED/COURT SUMMONS ISSUED	CSEN-0 CONT SUBST POSS PARAPHERNALIA ...	XXXX W XXXX S	WEST JORDAN	South6

Insert Days Between Hits (DBH) Calculation Column, group by Zone

Insert a new column called DBH which calculates the date difference from the line above it, group by Zone
df["DBH"] = df.groupby("Zone")["Report Date/Time"].diff().dt.days
df.head()

		Case Number	Report Date/Time	DOW	Hour	Occurred On	Status	Primary Offense	Location	Municipality	Zone	DBH
4217	WJ 2021-F40E	2021-06-17 07:49:00	Thursday	7	2021-06-17 07:49:00	CLOSED/COMPLETED	BVVB-0 BURGLARY, VEHICLE BURGLARY ...	XXXX S JACKLING WAY	WEST JORDAN	North1	Nan	
4288	WJ 2021-1052E	2021-06-23 13:02:00	Wednesday	13	2021-06-23 13:02:00	CLOSED/COMPLETED	THWP-0 THEFT VEHICLE PARTS	XXXX S XXXX W	WEST JORDAN	North1	6.0	
4287	WJ 2021-10J54	2021-06-24 02:10:00	Thursday	2	2021-06-24 02:10:00	CLOSED/COMPLETED	RVOL-0 REC VEH OUTSIDE RECOVER WJ ...	XXXX W XXXX S	WEST JORDAN	North1	0.0	
4286	WJ 2021-11542	2021-06-28 14:09:00	Monday	14	2021-06-28 14:09:00	CLOSED/COMPLETED	THWP-0 THEFT VEHICLE PARTS	XXXX W HAVEN MAPLE DR	WEST JORDAN	North1	4.0	
4284	WJ 2021-12131	2021-07-01 12:07:00	Thursday	12	2021-07-01 12:07:00	CLOSED/COMPLETED	POTR-0 PROPERTY TRESPASSING ...	XXXX S ADVENTURE WAY	WEST JORDAN	North1	2.0	

Next steps: [Generate code with df](#) [View recommended plots](#) [New interactive sheet](#)

df.shape
(12919, 11)

First I asked Gemini to round to 4 decimal places, then I asked ChatGPT

```
# Insert a new column called DBH which calculates the  
date difference from the line above it - round to two  
decimal places, group by Zone  
df["DBH"] = df.groupby("Zone")["Report  
Date/Time"].diff().dt.days.round(4)  
df.head()
```

The result of the code above is only giving me one decimal place, for example 4.0 on the days. I am getting 0.0 on some that are one day apart so I want to see that it is likely rounding down.



Ah, I see what's happening.

In your code:

```
python
```

Copy code

```
df["DBH"] = df.groupby("Zone")["Report Date/Time"].diff().dt.days.round(4)
```

👉 The issue is that `.dt.days` gives you the **integer number of days** only, throwing away any fractional part (hours/minutes/seconds). That's why you only see `4.0` or `0.0`, even when the difference was, say, 0.75 days (18 hours).

If you want decimals, you need to calculate the difference in `timedelta total seconds` and then convert to days. Example:

python

Copy code

```
# Calculate difference in days (with fractions) and round to 2 decimals  
df["DBH"] = (  
    df.groupby("Zone")["Report Date/Time"]  
    .diff()  
    .dt.total_seconds()  
    .div(86400) # 86400 seconds in a day  
    .round(2) # round to 2 decimal places  
)
```

Now, instead of just `4.0` or `0.0`, you'll see values like:

- `1.25` (1 day 6 hours)
- `0.75` (18 hours)
- `4.00` (4 full days)

Do you want me to also show you how to **keep both full days + fractional days as two separate columns** (like `DBH_days` and `DBH_hours`), or just stick with the rounded decimal days?

Rounding Issues – Updated Code

```
# Calculate difference in days (with fractions) and round to 2 decimals
df["DBH"] = (
    df.groupby("Zone")["Report Date/Time"]
    .diff()
    .dt.total_seconds()
    .div(86400) # 86400 seconds in a day
    .round(2) # round to 2 decimal places
)
df.head()
```

	Case Number	Report Date/Time	DOW	Hour	Occurred On	Status	Primary Offense	Location	Municipality	Zone	DBH
4217	WJ 2021-F40E	2021-06-17 07:49:00	Thursday	7	2021-06-17 07:49:00	CLOSED/COMPLETED	BVVB-0 BURGLARY, VEHICLE BURGLARY ...	XXXX S JACKLING WAY	WEST JORDAN	North1	NaN
4288	WJ 2021-1052E	2021-06-23 13:02:00	Wednesday	13	2021-06-23 13:02:00	CLOSED/COMPLETED	THVP-0 THEFT VEHICLE PARTS ...	XXXX S XXXX W	WEST JORDAN	North1	6.22
4287	WJ 2021-10J54	2021-06-24 02:10:00	Thursday	2	2021-06-24 02:10:00	CLOSED/COMPLETED	RVOL-0 REC VEH OUTSIDE RECOVER WJ ...	XXXX W XXXX S	WEST JORDAN	North1	0.55
4286	WJ 2021-11542	2021-06-28 14:09:00	Monday	14	2021-06-28 14:09:00	CLOSED/COMPLETED	THVP-0 THEFT VEHICLE PARTS ...	XXXX W HAVEN MAPLE DR	WEST JORDAN	North1	4.50
4284	WJ 2021-12131	2021-07-01 12:07:00	Thursday	12	2021-07-01 12:07:00	CLOSED/COMPLETED	POTR-0 PROPERTY TRESPASSING ...	XXXX S ADVENTURE WAY	WEST JORDAN	North1	2.92

Next steps: [Generate code with df](#) [View recommended plots](#) [New interactive sheet](#)

Run Descriptive Statistics by Zone

✓ 0s

▶ # prompt: descriptive statistics on DBH by Zone
Group data by 'Zone' and get descriptive statistics for 'DBH'
zone_dbh_stats = df.groupby('Zone')['DBH'].describe()
zone_dbh_stats

	count	mean	std	min	25%	50%	75%	max
Zone								
North1	407.0	3.757862	4.577731	0.0	0.8200	2.420	5.1050	43.39
North2	619.0	2.470129	2.725035	0.0	0.5100	1.540	3.4800	16.21
North3	3230.0	0.474932	0.553400	0.0	0.0800	0.240	0.7600	4.82
North4	490.0	3.102939	3.949263	0.0	0.6725	1.760	3.9850	31.69
North5	1134.0	1.364956	1.495834	0.0	0.3000	0.900	1.9000	10.32
North6	1386.0	1.115606	1.212514	0.0	0.2400	0.790	1.5375	10.36
South1	1208.0	1.269445	1.425658	0.0	0.2175	0.820	1.8100	10.11
South2	765.0	2.002353	2.220269	0.0	0.4800	1.130	2.8900	17.79
South3	912.0	1.679485	1.781650	0.0	0.3600	1.085	2.3250	13.70
South4	1379.0	1.113089	1.202710	0.0	0.2200	0.780	1.6300	8.80
South5	1107.0	1.396658	1.575061	0.0	0.2700	0.900	1.8800	13.51
South6	270.0	5.679481	5.752710	0.0	1.5800	3.915	7.8675	27.02

Next steps: [Generate code with zone_dbh_stats](#) [View recommended plots](#) [New interactive sheet](#)

Descriptive Statistics by Zone, Last Date, Mean+StDev, Forecast Date Columns, Today

```
0s
import pandas as pd
from datetime import datetime, timedelta

# --- Example: you already have zone_dbh_stats ---
# zone_dbh_stats = df.groupby("Zone")["DBH"].describe()

# Calculate last Report Date/Time for each Zone
last_dates = df.groupby("Zone")["Report Date/Time"].max().reset_index()
last_dates.rename(columns={"Report Date/Time": "Last Crime"}, inplace=True)

# Merge into your stats
zone_dbh_stats = zone_dbh_stats.merge(last_dates, on="Zone")

# Add Mean+STD
zone_dbh_stats["Mean+STD"] = zone_dbh_stats["mean"] + zone_dbh_stats["std"]

# Add Forecast = Last Crime + (Mean+STD days)
zone_dbh_stats["Forecast"] = zone_dbh_stats["Last Crime"] + pd.to_timedelta(
    zone_dbh_stats["Mean+STD"], unit="D"
)

# Add Today's date (when report is run)
zone_dbh_stats["Today"] = pd.to_datetime(datetime.today().date())

# Add Due = (Forecast - Today) in days
zone_dbh_stats["Due"] = (
    zone_dbh_stats["Forecast"] - zone_dbh_stats["Today"]
).dt.days

zone_dbh_stats.head()
```

	Zone	count	mean	std	min	25%	50%	75%	max	Last Crime	Mean+STD	Forecast	Today	Due
0	North1	407.0	3.757862	4.577731	0.0	0.8200	2.42	5.105	43.39	2025-08-24 19:08:00	8.335594	2025-09-02 03:11:15.304614701	2025-08-29	4
1	North2	619.0	2.470129	2.725035	0.0	0.5100	1.54	3.480	16.21	2025-08-24 09:28:00	5.195164	2025-08-29 14:09:02.176427371	2025-08-29	0
2	North3	3230.0	0.474932	0.553400	0.0	0.0800	0.24	0.760	4.82	2025-08-28 08:50:00	1.028332	2025-08-29 09:30:47.842479482	2025-08-29	0
3	North4	490.0	3.102939	3.949263	0.0	0.6725	1.76	3.985	31.69	2025-08-22 21:01:00	7.052202	2025-08-29 22:16:10.240277235	2025-08-29	0
4	North5	1134.0	1.364956	1.495834	0.0	0.3000	0.90	1.900	10.32	2025-08-29 08:05:00	2.860790	2025-09-01 04:44:32.271474775	2025-08-29	3

ChatGPT

In google colab, I have descriptive statistics on column 'DBH' called zone_dbh_stats that are grouped by column 'Zone', I want to add the following columns: the last Report Date/Time for each zone (called Last Crime), a calculation of the mean+std (called Mean+STD), a date column which adds the last two columns added (last date of each zone + (mean+std) (called Forecast), and todays date (day report is run) (called Today), and date diff between Today and Forecast (called Due).

Next steps: [Generate code with zone_dbh_stats](#) [View recommended plots](#) [New interactive sheet](#)

Educated Guessing

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Zone	count	mean	std	min	25%	50%	75%	max	Last Crime	Mean+STD	Forecast	Today	Due	
2	North1	407	3.757862408	4.577731396	0	0.82	2.42	5.105	43.39	2025-08-24 19:08:00	8.335593803	2025-09-02 03:11:15	2025-08-29 00:00:00	4	
3	North2	619	2.470129241	2.725034838	0	0.51	1.54	3.48	16.21	2025-08-24 09:28:00	5.195164079	2025-08-29 14:09:02	2025-08-29 00:00:00	0	
4	North3	3230	0.474931889	0.553399622	0	0.08	0.24	0.76	4.82	2025-08-28 08:50:00	1.02833151	2025-08-29 09:30:48	2025-08-29 00:00:00	0	
5	North4	490	3.102938776	3.94926308	0	0.6725	1.76	3.985	31.69	2025-08-22 21:01:00	7.052201855	2025-08-29 22:16:10	2025-08-29 00:00:00	0	
6	North5	1134	1.364955908	1.495834271	0	0.3	0.9	1.9	10.32	2025-08-29 08:05:00	2.860790179	2025-09-01 04:44:32	2025-08-29 00:00:00	3	
7	North6	1386	1.115606061	1.212514153	0	0.24	0.79	1.5375	10.36	2025-08-27 19:27:00	2.328120214	2025-08-30 03:19:30	2025-08-29 00:00:00	1	
8	South1	1208	1.269445364	1.425657751	0	0.2175	0.82	1.81	10.11	2025-08-26 17:47:00	2.695103115	2025-08-29 10:27:57	2025-08-29 00:00:00	0	
9	South2	765	2.002352941	2.220269182	0	0.48	1.13	2.89	17.79	2025-08-26 14:17:00	4.222622123	2025-08-30 19:37:35	2025-08-29 00:00:00	1	
10	South3	912	1.679484649	1.781650297	0	0.36	1.085	2.325	13.7	2025-08-25 09:23:00	3.461134946	2025-08-28 20:27:02	2025-08-29 00:00:00	-1	
11	South4	1379	1.113089195	1.202709882	0	0.22	0.78	1.63	8.8	2025-08-28 17:40:00	2.315799077	2025-08-31 01:14:45	2025-08-29 00:00:00	2	
12	South5	1107	1.396657633	1.575060903	0	0.27	0.9	1.88	13.51	2025-08-27 21:25:00	2.971718536	2025-08-30 20:44:16	2025-08-29 00:00:00	1	
13	South6	270	5.679481481	5.752709634	0	1.58	3.915	7.8675	27.02	2025-08-26 23:34:00	11.43219112	2025-09-07 09:56:21	2025-08-29 00:00:00	9	
14															

Narrowed down 32 square miles to five 2.5 mile zones in 37.89 seconds.



GRAPHS AND CHARTS

EXCEL

Temporal Matrix

✓ 2s # ChatGPT

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Create pivot table
matrix = df.pivot_table(index="DOW", columns="Hour", values="Case Number", aggfunc="count")

# Reorder days of the week for proper sorting
ordered_days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
matrix = matrix.reindex(ordered_days)

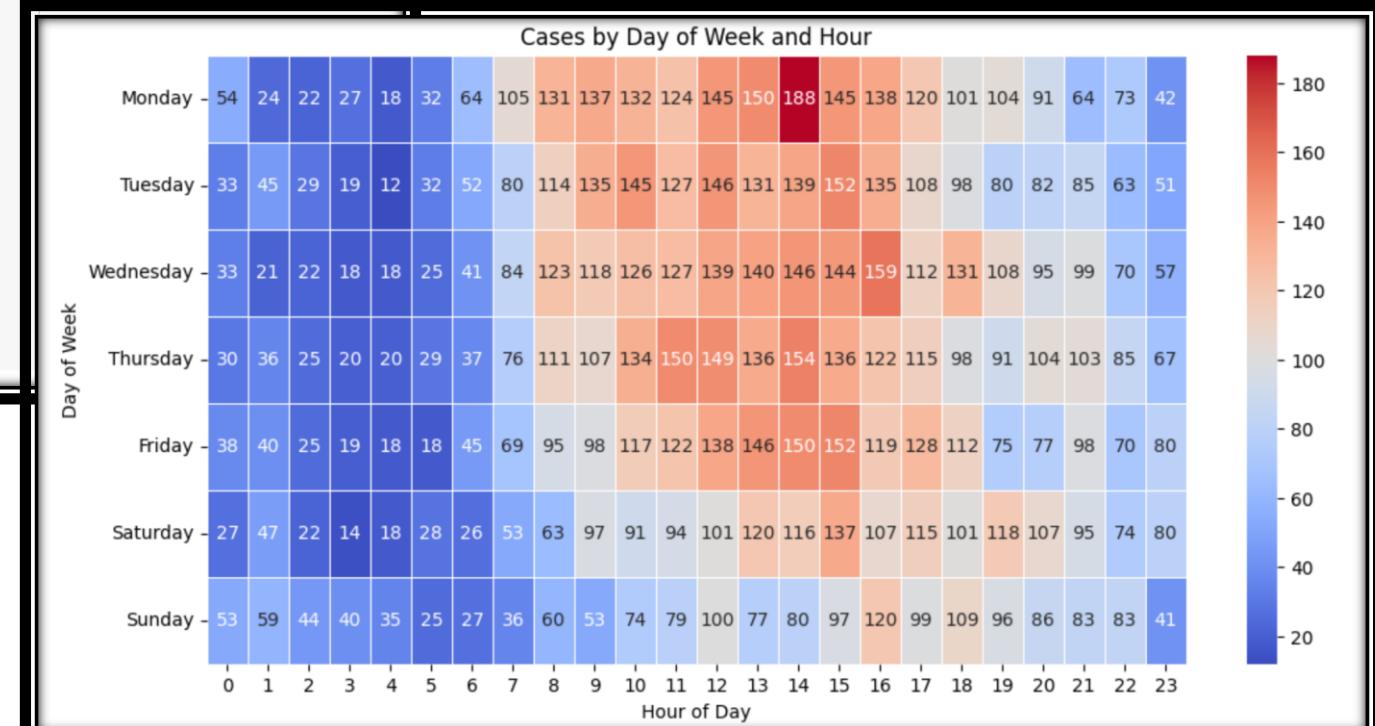
# Plot heatmap with conditional formatting
plt.figure(figsize=(12, 6))
sns.heatmap(matrix, annot=True, cmap="coolwarm", fmt="g", linewidths=0.5)

# Labels and title
plt.title("Cases by Day of Week and Hour")
plt.xlabel("Hour of Day")
plt.ylabel("Day of Week")

# Show plot
plt.show()
```

using df, create a matrix with DOW in the rows, Hour in the columns and count of case number as values, conditional formatting

ChatGPT



Temporal Matrix

✓ 1s # ChatGPT

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Create pivot table
matrix = df.pivot_table(index="DOW", columns="Hour", values="Case Number", aggfunc="count")

# Reorder days of the week for proper sorting
ordered_days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
matrix = matrix.reindex(ordered_days)

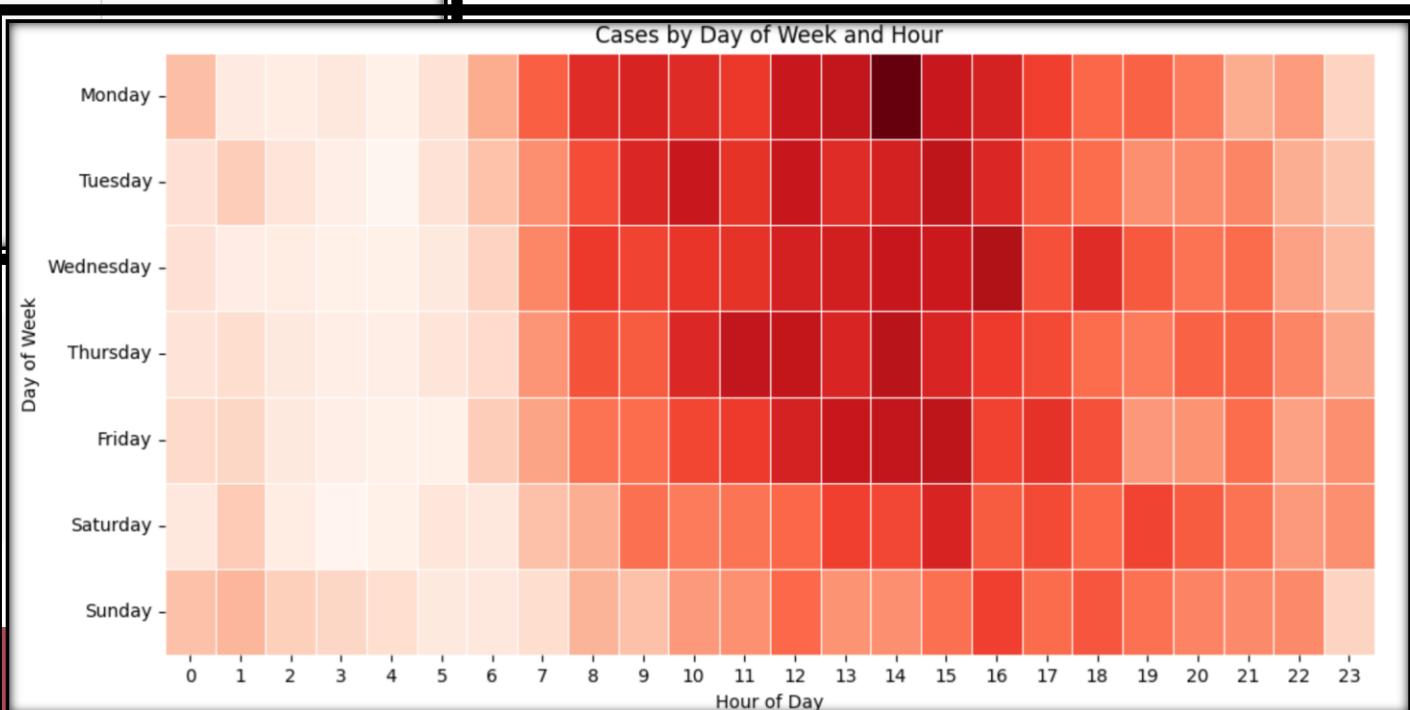
# Plot heatmap with conditional formatting
plt.figure(figsize=(12, 6))
sns.heatmap(matrix, annot=False, cmap="Reds", fmt="g", linewidths=0.5, cbar=False)

# Labels and title
plt.title("Cases by Day of Week and Hour")
plt.xlabel("Hour of Day")
plt.ylabel("Day of Week")

# Show plot
plt.show()
```

using df, create a matrix with DOW in the rows, Hour in the columns and count of case number as values, conditional formating

ChatGPT



1 Built-in Seaborn Palettes

These palettes can be used with `palette="name"` in Seaborn plots.

Palette Name	Description
"deep"	Default Seaborn colors, vibrant and distinct
"muted"	Softer colors, easier on the eyes
"bright"	Highly saturated, bold colors
"pastel"	Light, soft pastel shades
"dark"	Darker shades for contrast
"colorblind"	Optimized for color vision deficiency

2 Perceptually Uniform Sequential Palettes (Good for heatmaps & gradients)

These are single-hue palettes that transition from light to dark.

Palette Name	Description
"Blues"	Shades of blue (light to dark)
"Greens"	Shades of green
"Oranges"	Shades of orange
"Purples"	Shades of purple
"Reds"	Shades of red

3 Diverging Palettes (Good for highlighting positive/negative values)

These palettes balance two opposing colors.

Palette Name	Color Transition
"coolwarm"	Blue → Red
"RdBu"	Red → Blue
"PuOr"	Purple → Orange
"RdYlGn"	Red → Yellow → Green

4 Qualitative Palettes (For categorical data)

These palettes provide **distinct colors** for different categories.

Palette Name	Style
"Set1"	Bright and bold colors
"Set2"	Softer pastel shades
"Set3"	A mix of 12 varied colors
"Paired"	Matched color pairs
"Accent"	Soft, accentuated colors
"tab10"	10 categorical colors (default in Matplotlib)

5 Custom Palettes

You can also **create your own palette** using:

- Manually defined colors (hex codes, RGB)
- Continuous color gradients
- Using `sns.color_palette()`

Example: Custom Hex Colors

python Copy Edit

```
custom_colors = ["#FF5733", "#33FF57", "#3357FF"] # Red, Green, Blue  
sns.barplot(x="DOW", y="Case Number", data=df, palette=custom_colors)
```

Example: Continuous Gradient

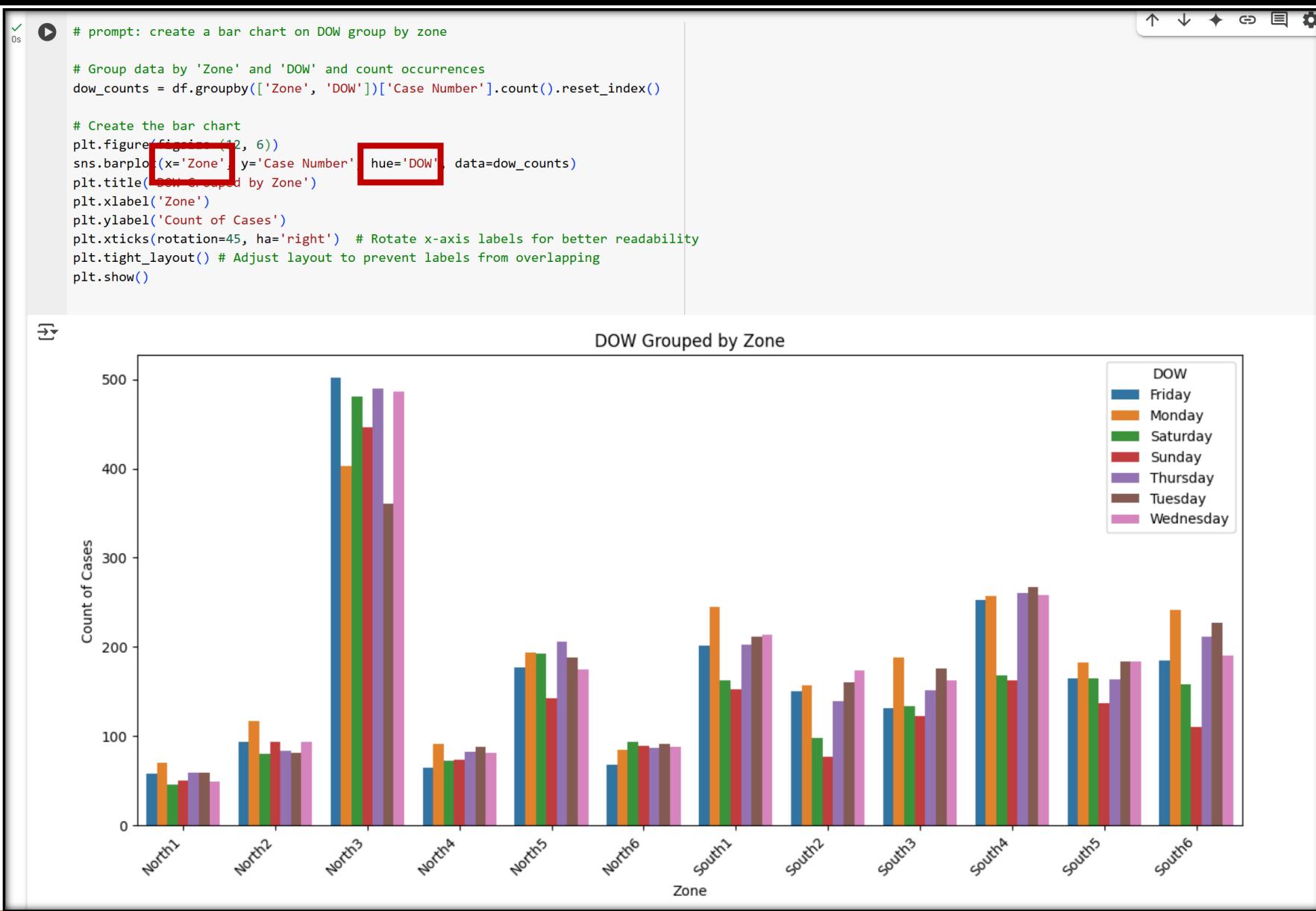
python Copy Edit

```
sns.barplot(x="DOW", y="Case Number", data=df, palette=sns.light_palette("blue", as_cmap=True)
```

Bar Chart



Bar Chart



Bar Chart – Separated by Zone

✓ 3s # ChatGPT

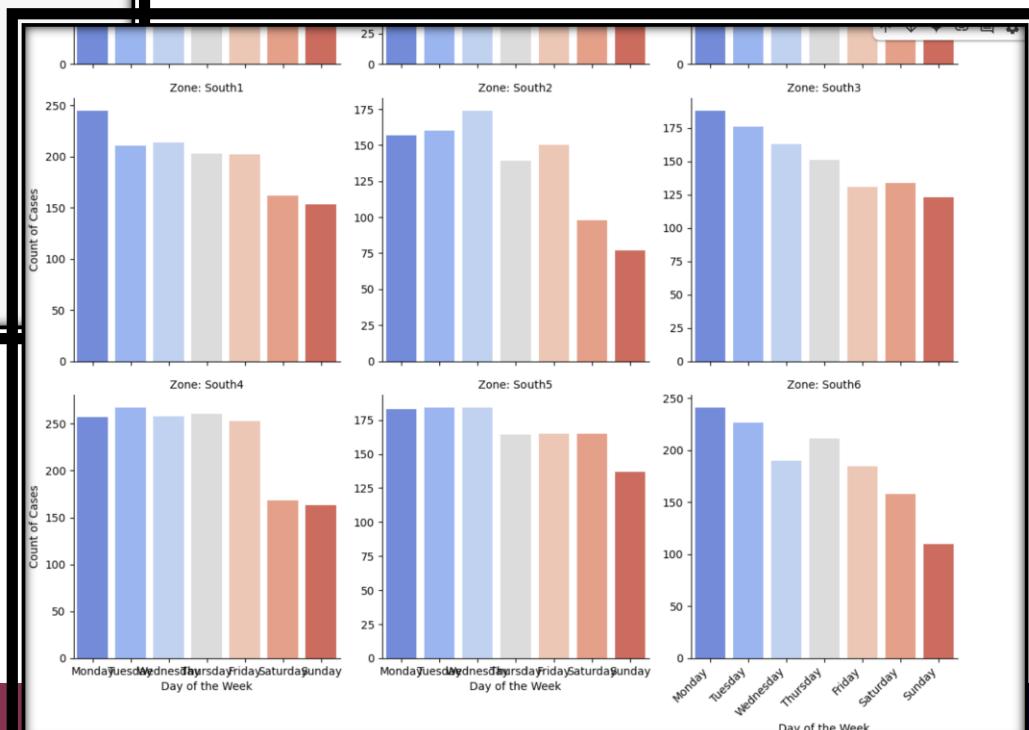
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Group data by 'Zone' and 'DOW' and count occurrences of 'Case Number'
dow_counts = df.groupby(['Zone', 'DOW'])['Case Number'].count().reset_index()

# Order days of the week for consistency
ordered_days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
dow_counts["DOW"] = pd.Categorical(dow_counts["DOW"], categories=ordered_days, ordered=True)

# Create FacetGrid to generate a bar chart for each Zone
g = sns.FacetGrid(dow_counts, col="Zone", col_wrap=3, sharey=False, height=4)
g.map_dataframe(sns.barplot, x="DOW", y="Case Number", order=ordered_days, palette="coolwarm")

# Adjust labels
g.set_axis_labels("Day of the Week", "Count of Cases")
g.set_titles(col_template="Zone: {col_name}")
plt.xticks(rotation=45, ha='right') # Rotate x-axis labels
plt.tight_layout() # Prevent label overlap
plt.show()
```



Bar Chart – Separated by Zone, Export to JPG

Forecasting.ipynb

File Edit View Insert Runtime Tools Help

Files

Analyze your files with code written by Gemini

Upload

sample_data

zone_charts

- North1.jpg
- North2.jpg
- North3.jpg
- North4.jpg
- North5.jpg
- North6.jpg
- South1.jpg
- South2.jpg
- South3.jpg
- South4.jpg
- South5.jpg
- South6.jpg

North1.xlsx

North2.xlsx

North3.xlsx

North4.xlsx

North5.xlsx

North6.xlsx

South1.xlsx

South2.xlsx

South3.xlsx

4s # chatGPT

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import os

# Group data by 'Zone' and 'DOW' and count occurrences of 'Case Number'
dow_counts = df.groupby(['Zone', 'DOW'])['Case Number'].count().reset_index()

# Order days of the week for consistency
ordered_days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
dow_counts["DOW"] = pd.Categorical(dow_counts["DOW"], categories=ordered_days, ordered=True)

# Create output directory for images (if not exists)
output_dir = "zone_charts"
os.makedirs(output_dir, exist_ok=True)

# Loop through each unique Zone and create individual bar charts
for zone in dow_counts["Zone"].unique():
    plt.figure(figsize=(8, 5))

    # Filter data for the current Zone
    zone_data = dow_counts[dow_counts["Zone"] == zone]

    # Create barplot for the Zone
    sns.barplot(x="DOW", y="Case Number", hue="DOW", data=zone_data, order=ordered_days, palette="coolwarm", legend=False)

    # Formatting
    plt.title(f"Cases by Day of Week - {zone}")
    plt.xlabel("Day of the Week")
    plt.ylabel("Count of Cases")
    plt.xticks(rotation=45, ha='right') # Rotate x-axis labels

    # Save figure as JPG
    filename = os.path.join(output_dir, f"{zone}.jpg")
    plt.savefig(filename, format="jpg", dpi=300, bbox_inches="tight")

    # Close plot to free memory
    plt.close()

print(f"Charts saved in '{output_dir}' folder.")

Charts saved in 'zone_charts' folder.
```

Cases by Day of Week - North1

A bar chart titled "Cases by Day of Week - North1". The x-axis is labeled "Day of the Week" and shows the days from Monday to Sunday. The y-axis is labeled "Count of Cases" and ranges from 0 to 70. The bars are colored using a coolwarm palette. The data shows a general downward trend from Monday to Saturday, with Sunday being the highest day of the week.

Day of the Week	Count of Cases
Monday	70
Tuesday	59
Wednesday	49
Thursday	59
Friday	58
Saturday	46
Sunday	50

THANK YOU

Suzanna Howard

Suzanna.howard@westjordan.Utah.gov

West Jordan Police Department