Final Project: Code

Duoshu Xu & Jae Hu & Regina Hou

```python
import pandas as pd
import numpy as np
import altair as alt
import pandas as pd
import matplotlib.pyplot as plt
```

```python
file_path = '/Users/kevinxu/Desktop/Final Project Raw Data/mapdataall.csv'
wildfire_data = pd.read_csv(file_path)
```

```python
file_path = '/Users/kevinxu/Desktop/Final Project Data Cleaning/Census
 ↪  data.xlsx'
census_data = pd.read_excel(file_path)
```

Data cleaning and reshaping (done by Jae Hu)

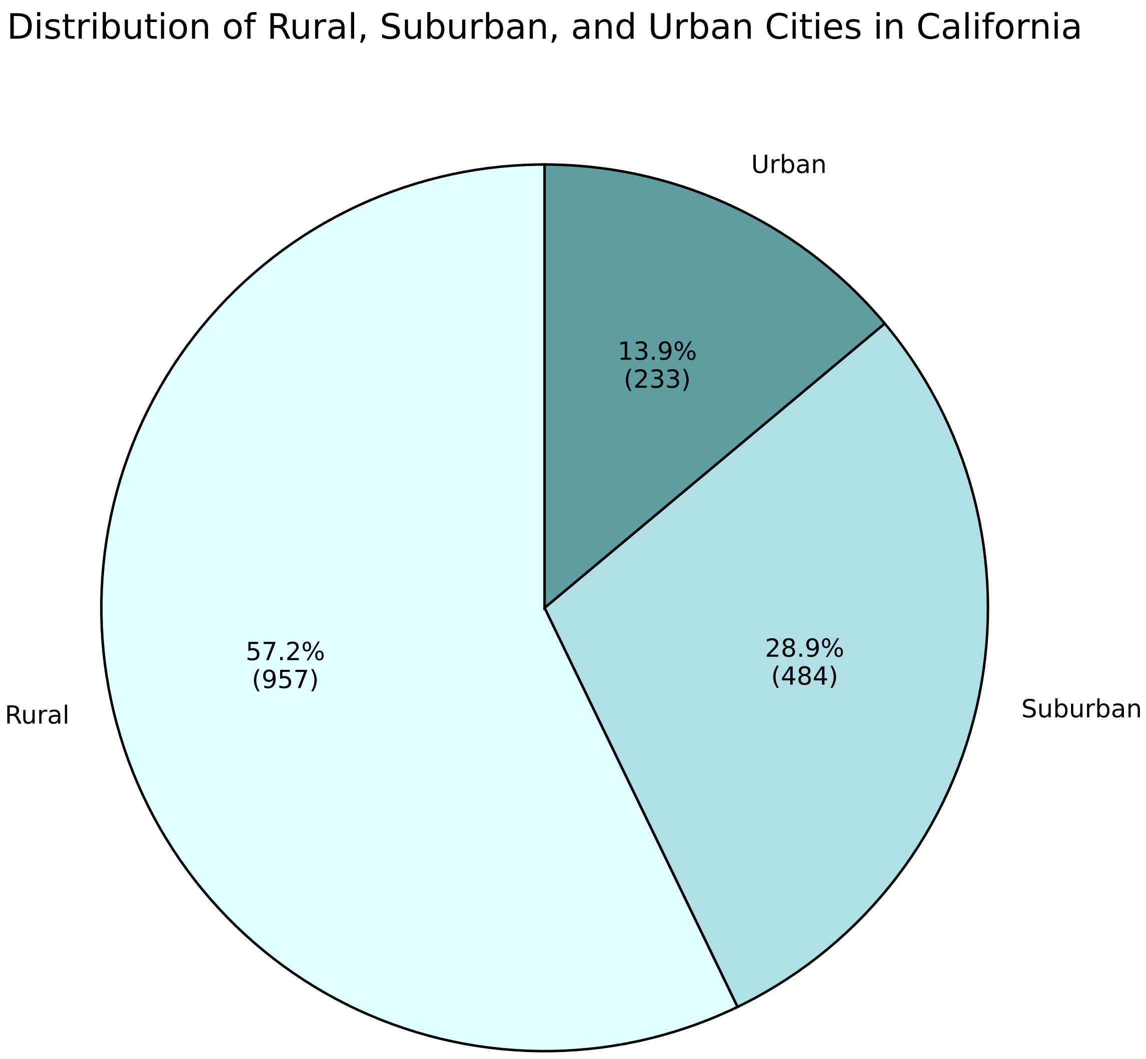
```python
census_data['Total Population'] = pd.to_numeric(
    census_data['Total Population'], errors='coerce')


def classify_geographic_type(population):
    if population > 50000:
        return "Urban"
    elif 5000 <= population <= 50000:
        return "Suburban"
    else:
        return "Rural"
```

```python
census_data['geographic type'] = census_data['Total Population'].apply(
    classify_geographic_type)

output_file_path = '/Users/kevinxu/Desktop/Final Project Raw
↪  Data/Census_with_types.csv'

census_data.to_csv(output_file_path, index=False)
```

pie chart (done by Jae Hu)

```python
# Load the modified Census data
file_path = '/Users/kevinxu/Desktop/Final Project Raw
↪  Data/Census_with_types.csv'
census_data = pd.read_csv(file_path)

# Count the number of cities in each geographic type
geo_counts = census_data['geographic type'].value_counts()

# Define a custom autopct function to add both percentages and counts
def autopct_with_counts(pct):
    count = int(round(pct * geo_counts.sum() / 100.0))
    return f'{pct:.1f}%\n({count})'

# Plot the pie chart
plt.figure(figsize=(8, 8))
colors = ["lightcyan", "powderblue", "cadetblue"]
geo_counts.plot.pie(
    autopct=autopct_with_counts,
    colors=colors,
    startangle=90,
    labels=geo_counts.index,
    wedgeprops={"edgecolor": "black"}
)

# Add title
plt.title("Distribution of Rural, Suburban, and Urban Cities in California",
↪  fontsize=14)

# Show the chart
```

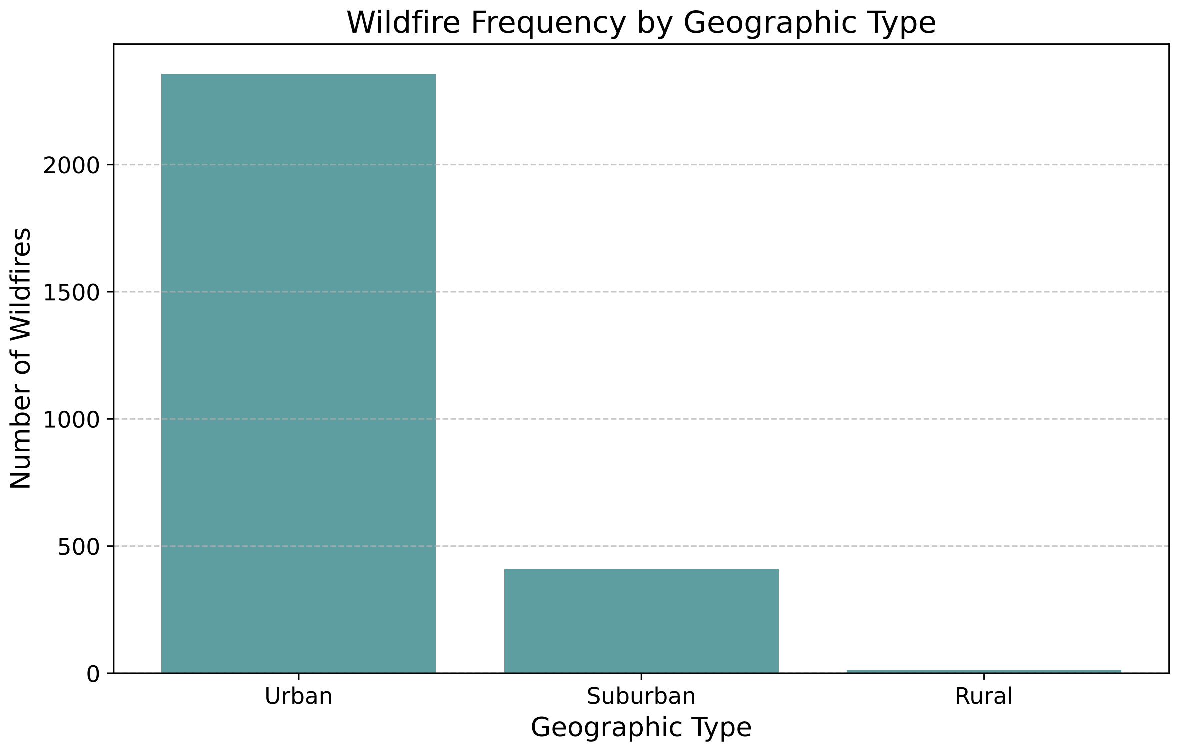
```
plt.ylabel("")
plt.show()
```



Wildfire Frequency bar chart (done by Duoshu Xu)

```
wildfire_data_path = '/Users/kevinxu/Desktop/Final Project Raw
 ↪  Data/mapdataall.csv'
census_data_path = '/Users/kevinxu/Desktop/Final Project Raw
 ↪  Data/Census_with_types.csv'
wildfire_data = pd.read_csv(wildfire_data_path)
census_data = pd.read_csv(census_data_path)


def standardize_county_name(name):
    if pd.isna(name):
        return None
    return name.split(",")[0].strip()


wildfire_data['incident_county_cleaned'] =
 ↪  wildfire_data['incident_county'].apply(
    standardize_county_name)
census_data['Geography_cleaned'] = census_data['Geography'].str.replace(
    " County", "", regex=False)

wildfire_with_geo_type = wildfire_data.merge(
    census_data[['Geography_cleaned', 'geographic type']],
    left_on='incident_county_cleaned',
    right_on='Geography_cleaned',
    how='left'
)

wildfire_counts = wildfire_with_geo_type['geographic type'].value_counts(
).reset_index()
wildfire_counts.columns = ['Geographic Type', 'Number of Wildfires']

print(wildfire_counts)
plt.figure(figsize=(10, 6))
plt.bar(
    wildfire_counts['Geographic Type'],
    wildfire_counts['Number of Wildfires'],
    color='cadetblue'
)
plt.title("Wildfire Frequency by Geographic Type", fontsize=16)
plt.xlabel("Geographic Type", fontsize=14)
plt.ylabel("Number of Wildfires", fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
  Geographic Type  Number of Wildfires
0           Urban                 2356
1        Suburban                  407
2           Rural                   10
```



```
population_data_path = '/Users/kevinxu/Desktop/Final Project Raw
↪  Data/Census_with_types.csv'
firedamage_data_path = '/Users/kevinxu/Desktop/Final Project Raw
↪  Data/POSTFIRE_MASTER_DATA_SHARE_2064760709534146017.csv'
population_data = pd.read_csv(population_data_path)
firedamage_data = pd.read_csv(firedamage_data_path)

population_data.head(), firedamage_data.head()
```

```python
population_data["Geography"] = (
    population_data["Geography"]
    .str.replace(r"\s*City\s*$", "", regex=True, case=False)
    .str.strip()
)

print(population_data["Geography"].unique()[:10])
population_data.head()
```

```
['California' 'Alameda County' 'Alameda' 'Albany' 'Ashland CDP' 'Berkeley'
 'Castro Valley CDP' 'Cherryland CDP' 'Dublin' 'Emeryville']
```

```
/var/folders/r4/x5b99tvj66zcn_88m3jn4r6w0000gn/T/ipykernel_22943/3288246651.py:4:
DtypeWarning:

Columns (12,36) have mixed types. Specify dtype option on import or set
low_memory=False.
```

	Geography	Total Population	Land Area in Square Miles	Population Per Square Mile (Land Are
0	California	39538223	155858.326771	253.680530
1	Alameda County	1682353	737.461854	2281.274605
2	Alameda	78280	10.448679	7491.856487
3	Albany	20271	1.789982	11324.694641
4	Ashland CDP	23823	1.842571	12929.213531

```python
firedamage_data["* City"] = (
    firedamage_data["* City"]
    # Removes "City" regardless of case
    .str.replace(r"\s*City\s*$", "", regex=True, case=False)
    .str.strip()  # Removes leading/trailing whitespace
)

# Verify the changes
# Display unique values to confirm "City" is removed
firedamage_data["* City"].unique()[:10]
firedamage_data.head()
```

	OBJECTID	* Damage	* Street Number	* Street Name	* Street Type (e.g. road, drive, lane, e
0	1	No Damage	8376.0	Quail Canyon	Road

	OBJECTID	* Damage	* Street Number	* Street Name	* Street Type (e.g. road, drive, lane, e
1	2	Affected (1-9%)	8402.0	Quail Canyon	Road
2	3	No Damage	8430.0	Quail Canyon	Road
3	4	No Damage	3838.0	Putah Creek	Road
4	5	No Damage	3830.0	Putah Creek	Road

```
# Remove rows where Geography or * City contains "County" in either dataset
population_data_cleaned =
 ↪ population_data[~population_data["Geography"].str.contains(
    "County", case=False, na=False)]
firedamage_data_cleaned = firedamage_data[~firedamage_data["*
 ↪ City"].str.contains(
    "County", case=False, na=False)]

# Standardize column names for merging
population_data_cleaned = population_data_cleaned.rename(
    columns={"Geography": "City"})
firedamage_data_cleaned = firedamage_data_cleaned.rename(
    columns={"* City": "City"})

# Merge the datasets based on the City column
merged_data = pd.merge(
    firedamage_data_cleaned,
    population_data_cleaned,
    on="City",
    how="inner"
)

# Display the first few rows of the merged dataset to confirm
merged_data.head()
```

	OBJECTID	* Damage	* Street Number	* Street Name	* Street Type (e.g. road, drive, lane, e
0	1	No Damage	8376.0	Quail Canyon	Road
1	2	Affected (1-9%)	8402.0	Quail Canyon	Road
2	3	No Damage	8430.0	Quail Canyon	Road
3	4	No Damage	3838.0	Putah Creek	Road
4	5	No Damage	3830.0	Putah Creek	Road

bar plot of 2020 (done by Regina Hou)

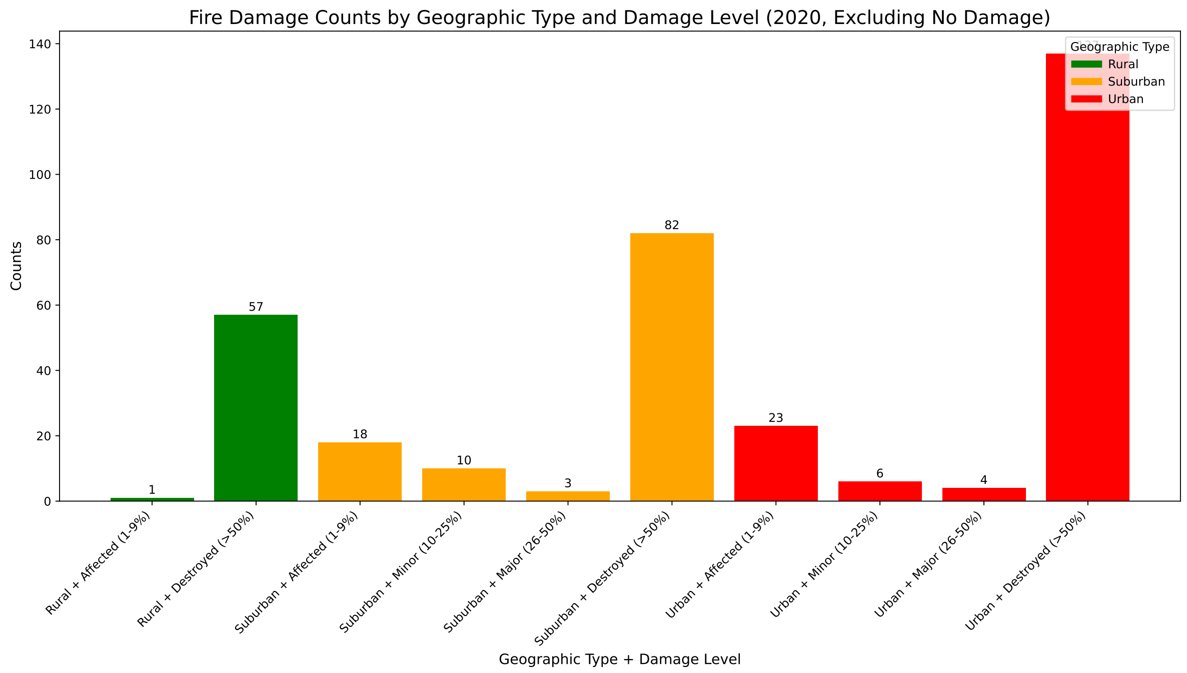
```python
# Filter for 2020 data only after merging
merged_data["Year"] = pd.to_datetime(
    merged_data["Incident Start Date"], errors="coerce").dt.year
merged_data_2020 = merged_data[merged_data["Year"] == 2020]
```

```
/var/folders/r4/x5b99tvj66zcn_88m3jn4r6w0000gn/T/ipykernel_22943/454140121.py:2:
UserWarning:

Could not infer format, so each element will be parsed individually, falling
back to `dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
```

```python
# Filter for 2020 data only after merging and exclude "No Damage"
from matplotlib.lines import Line2D
merged_data["Year"] = pd.to_datetime(
    merged_data["Incident Start Date"], errors="coerce").dt.year
merged_data_2020 = merged_data[
    (merged_data["Year"] == 2020) &
    (merged_data["* Damage"] != "No Damage")
]

# Group data by geographic type and damage level, and count occurrences
damage_counts_2020 = (
    merged_data_2020.groupby(["geographic type", "* Damage"])
    .size()
    .reset_index(name="Count")
)

# Define the order of damage levels
damage_order = [
    "Affected (1-9%)",
    "Minor (10-25%)",
    "Major (26-50%)",
    "Destroyed (>50%)"
]

# Update damage levels for sorting and visualization
damage_counts_2020["* Damage"] = pd.Categorical(
    damage_counts_2020["* Damage"], categories=damage_order, ordered=True
```

```
)

# Sort the data by Geographic Type and Damage Level
damage_counts_2020 = damage_counts_2020.sort_values(
    by=["geographic type", "* Damage"])

# Create a new x-axis label with rearranged categories
damage_counts_2020["Label"] = (
    damage_counts_2020["geographic type"] + " + " +
    damage_counts_2020["* Damage"].astype(str)
)

# Assign colors based on geographic type
colors = {"Rural": "green", "Suburban": "orange", "Urban": "red"}
damage_counts_2020["Bar Color"] = damage_counts_2020["geographic type"].map(
    colors)

# Plot the bar chart
plt.figure(figsize=(14, 8))
bars = plt.bar(
    damage_counts_2020["Label"],
    damage_counts_2020["Count"],
    color=damage_counts_2020["Bar Color"],
)

# Add counts on top of each bar
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, yval + 0.5,
             int(yval), ha="center", va="bottom", fontsize=10)

# Customize the chart
plt.title("Fire Damage Counts by Geographic Type and Damage Level (2020,
 ↪  Excluding No Damage)", fontsize=16)
plt.xlabel("Geographic Type + Damage Level", fontsize=12)
plt.ylabel("Counts", fontsize=12)
plt.xticks(rotation=45, ha="right", fontsize=10)

# Add a legend for the bar colors

legend_elements = [
    Line2D([0], [0], color="green", lw=6, label="Rural"),
    Line2D([0], [0], color="orange", lw=6, label="Suburban"),
    Line2D([0], [0], color="red", lw=6, label="Urban"),
]

plt.legend(handles=legend_elements, title="Geographic Type",
           loc="upper right", fontsize=10)

plt.tight_layout()

# Show the plot
plt.show()
```

```
/var/folders/r4/x5b99tvj66zcn_88m3jn4r6w0000gn/T/ipykernel_22943/1844741617.py:3:
UserWarning:

Could not infer format, so each element will be parsed individually, falling
back to `dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
```



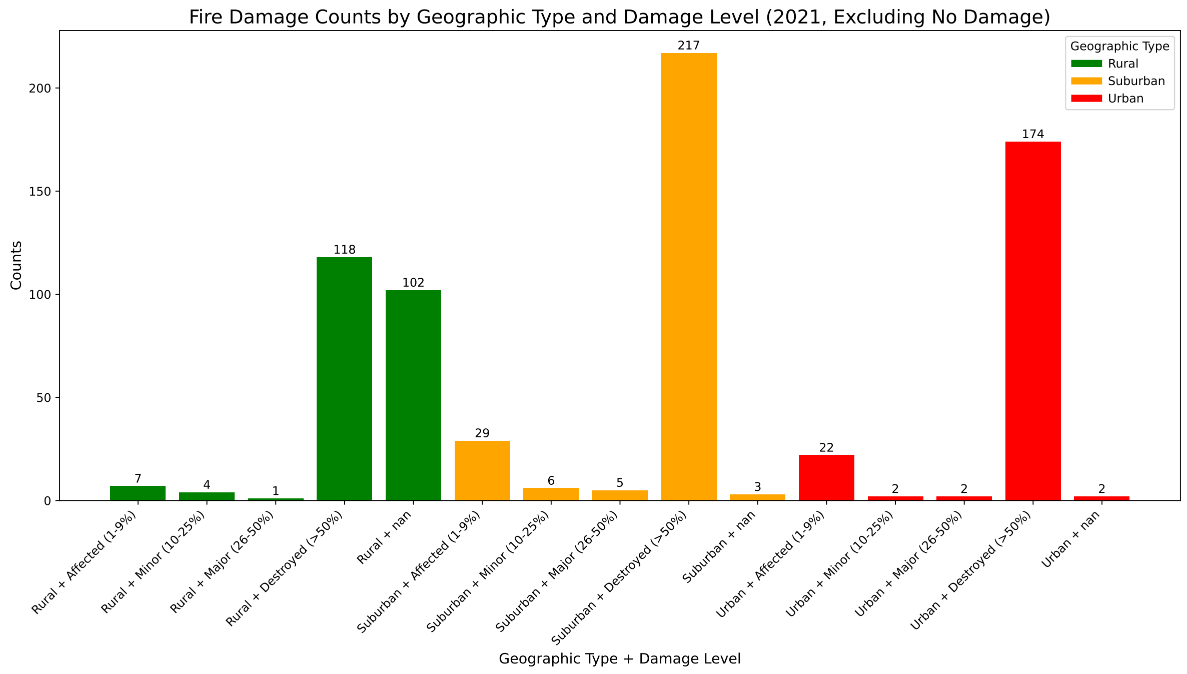
plot of 2021 (done by Regina Hou)

```
# Filter for 2021 data only after merging
merged_data_2021 = merged_data[
    (merged_data["Year"] == 2021) &
    (~merged_data["* Damage"].isna()) &
    (~merged_data["geographic type"].isna())
].copy()

# Verify that there are no NaN values remaining
print("NaN values in geographic type:", merged_data_2021["geographic
 ↪  type"].isna().sum())
print("NaN values in * Damage:", merged_data_2021["* Damage"].isna().sum())
```

```
NaN values in geographic type: 0
NaN values in * Damage: 0
```

```
# Filter for 2021 data only after merging and exclude "No Damage"
merged_data_2021 = merged_data[
    (merged_data["Year"] == 2021) &
    (merged_data["* Damage"] != "No Damage")
]

# Group data by geographic type and damage level, and count occurrences
damage_counts_2021 = (
    merged_data_2021.groupby(["geographic type", "* Damage"])
    .size()
    .reset_index(name="Count")
)

# Define the order of damage levels
damage_order = [
    "Affected (1-9%)",
    "Minor (10-25%)",
    "Major (26-50%)",
    "Destroyed (>50%)"
]

# Update damage levels for sorting and visualization
damage_counts_2021["* Damage"] = pd.Categorical(
    damage_counts_2021["* Damage"], categories=damage_order, ordered=True
```

```
)

# Sort the data by Geographic Type and Damage Level
damage_counts_2021 = damage_counts_2021.sort_values(by=["geographic type", "*
↪  Damage"])

# Create a new x-axis label with rearranged categories
damage_counts_2021["Label"] = (
    damage_counts_2021["geographic type"] + " + " + damage_counts_2021["*
↪  Damage"].astype(str)
)

# Assign colors based on geographic type
colors = {"Rural": "green", "Suburban": "orange", "Urban": "red"}
damage_counts_2021["Bar Color"] = damage_counts_2021["geographic
↪  type"].map(colors)

# Plot the bar chart
plt.figure(figsize=(14, 8))
bars = plt.bar(
    damage_counts_2021["Label"],
    damage_counts_2021["Count"],
    color=damage_counts_2021["Bar Color"],
)

# Add counts on top of each bar
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, yval + 0.5, int(yval),
↪  ha="center", va="bottom", fontsize=10)

# Customize the chart
plt.title("Fire Damage Counts by Geographic Type and Damage Level (2021,
↪  Excluding No Damage)", fontsize=16)
plt.xlabel("Geographic Type + Damage Level", fontsize=12)
plt.ylabel("Counts", fontsize=12)
plt.xticks(rotation=45, ha="right", fontsize=10)

# Add a legend for the bar colors
from matplotlib.lines import Line2D

legend_elements = [
    Line2D([0], [0], color="green", lw=6, label="Rural"),
    Line2D([0], [0], color="orange", lw=6, label="Suburban"),
    Line2D([0], [0], color="red", lw=6, label="Urban"),
]

plt.legend(handles=legend_elements, title="Geographic Type", loc="upper
 ↪  right", fontsize=10)

plt.tight_layout()

# Show the plot
plt.show()
```



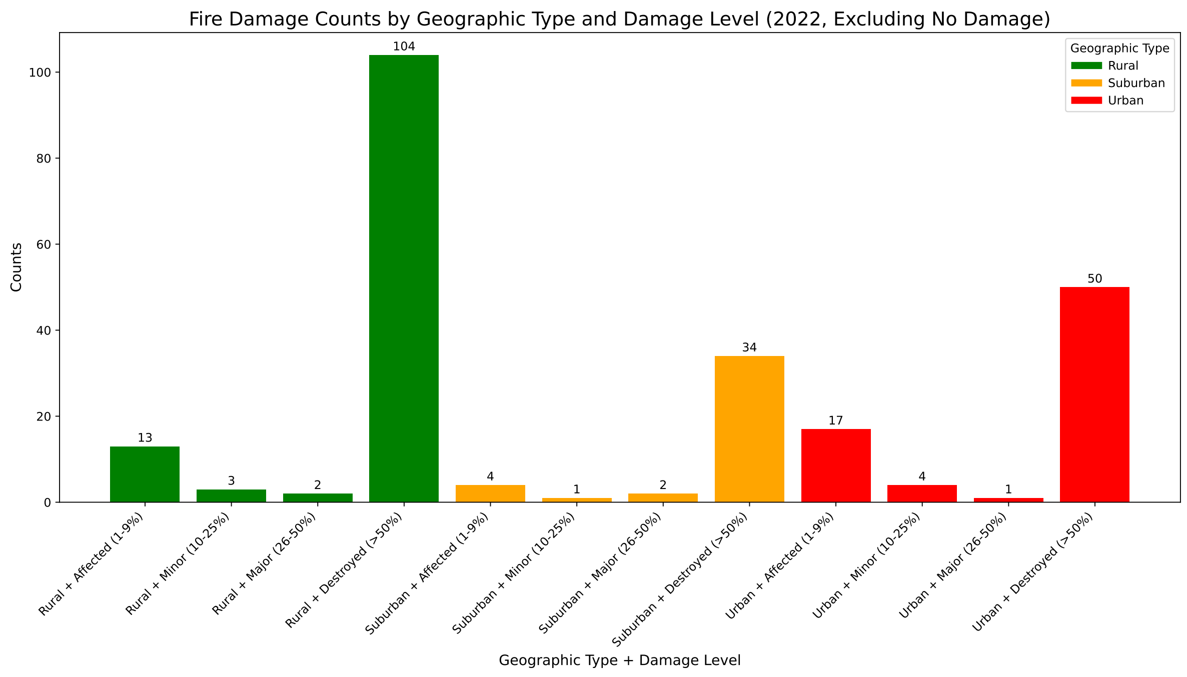
plot of 2022 (done by Regina Hou)

```
# Filter for 2022 data only after merging
merged_data["Year"] = pd.to_datetime(merged_data["Incident Start Date"],
 ↪  errors="coerce").dt.year
merged_data_2022 = merged_data[merged_data["Year"] == 2022]
```

```
/var/folders/r4/x5b99tvj66zcn_88m3jn4r6w0000gn/T/ipykernel_22943/3282159924.py:2:
UserWarning:

Could not infer format, so each element will be parsed individually, falling
back to `dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
```

```
# Filter for 2022 data only after merging and exclude "No Damage"
merged_data_2022 = merged_data[
    (merged_data["Year"] == 2022) &
    (merged_data["* Damage"] != "No Damage")
]

# Group data by geographic type and damage level, and count occurrences
damage_counts_2022 = (
    merged_data_2022.groupby(["geographic type", "* Damage"])
    .size()
    .reset_index(name="Count")
)

# Define the order of damage levels
damage_order = [
    "Affected (1-9%)",
    "Minor (10-25%)",
    "Major (26-50%)",
    "Destroyed (>50%)"
]

# Update damage levels for sorting and visualization
damage_counts_2022["* Damage"] = pd.Categorical(
    damage_counts_2022["* Damage"], categories=damage_order, ordered=True
)

# Sort the data by Geographic Type and Damage Level
damage_counts_2022 = damage_counts_2022.sort_values(by=["geographic type", "*
↪  Damage"])

# Create a new x-axis label with rearranged categories
damage_counts_2022["Label"] = (
    damage_counts_2022["geographic type"] + " + " + damage_counts_2022["*
↪  Damage"].astype(str)
)
```

```python
# Assign colors based on geographic type
colors = {"Rural": "green", "Suburban": "orange", "Urban": "red"}
damage_counts_2022["Bar Color"] = damage_counts_2022["geographic
 ↪  type"].map(colors)

# Plot the bar chart
plt.figure(figsize=(14, 8))
bars = plt.bar(
    damage_counts_2022["Label"],
    damage_counts_2022["Count"],
    color=damage_counts_2022["Bar Color"],
)

# Add counts on top of each bar
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, yval + 0.5, int(yval),
 ↪  ha="center", va="bottom", fontsize=10)

# Customize the chart
plt.title("Fire Damage Counts by Geographic Type and Damage Level (2022,
 ↪  Excluding No Damage)", fontsize=16)
plt.xlabel("Geographic Type + Damage Level", fontsize=12)
plt.ylabel("Counts", fontsize=12)
plt.xticks(rotation=45, ha="right", fontsize=10)

# Add a legend for the bar colors
from matplotlib.lines import Line2D

legend_elements = [
    Line2D([0], [0], color="green", lw=6, label="Rural"),
    Line2D([0], [0], color="orange", lw=6, label="Suburban"),
    Line2D([0], [0], color="red", lw=6, label="Urban"),
]

plt.legend(handles=legend_elements, title="Geographic Type", loc="upper
 ↪  right", fontsize=10)

plt.tight_layout()

# Show the plot
plt.show()
```



plot of 2023 (done by Regina Hou)

```
# Filter for 2023 data only after merging
merged_data["Year"] = pd.to_datetime(merged_data["Incident Start Date"],
 ↪  errors="coerce").dt.year
merged_data_2023 = merged_data[merged_data["Year"] == 2023]
```

```
/var/folders/r4/x5b99tvj66zcn_88m3jn4r6w0000gn/T/ipykernel_22943/897568859.py:2:
UserWarning:

Could not infer format, so each element will be parsed individually, falling
back to `dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
```

plot for 2020-2023 (done by Regina Hou)

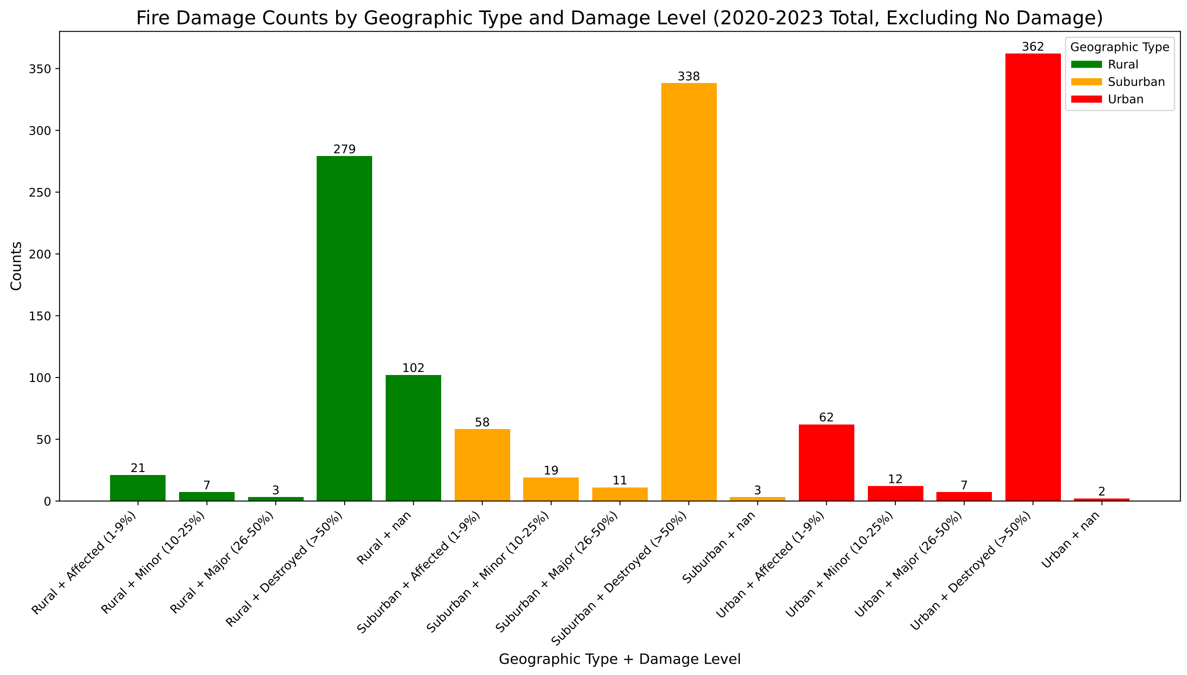
```
merged_data["Year"] = pd.to_datetime(merged_data["Incident Start Date"],
 ↪  errors="coerce").dt.year
merged_data_total = merged_data[(merged_data["Year"] >= 2020) &
 ↪  (merged_data["Year"] <= 2023)]

# Group data by geographic type and damage level for 2020-2023 combined and
 ↪  sum counts
damage_counts_total = (
    merged_data_total.groupby(["geographic type", "* Damage"])
    .size()
    .reset_index(name="Count")
)
```

```
/var/folders/r4/x5b99tvj66zcn_88m3jn4r6w0000gn/T/ipykernel_22943/857330891.py:1:
UserWarning:

Could not infer format, so each element will be parsed individually, falling
back to `dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
```

```
# Filter for 2020-2023 data and exclude "No Damage"
merged_data_total = merged_data[
    (merged_data["Year"] >= 2020) &
    (merged_data["Year"] <= 2023) &
    (merged_data["* Damage"] != "No Damage")
]

# Group data by geographic type and damage level, and count occurrences
damage_counts_total = (
    merged_data_total.groupby(["geographic type", "* Damage"])
    .size()
    .reset_index(name="Count")
)

# Define the order of damage levels
damage_order = [
    "Affected (1-9%)",
    "Minor (10-25%)",
    "Major (26-50%)",
    "Destroyed (>50%)"
]
```

```
# Update damage levels for sorting and visualization
damage_counts_total["* Damage"] = pd.Categorical(
    damage_counts_total["* Damage"], categories=damage_order, ordered=True
)

# Sort the data by Geographic Type and Damage Level
damage_counts_total = damage_counts_total.sort_values(by=["geographic type",
 ↪  "* Damage"])

# Create a new x-axis label with rearranged categories
damage_counts_total["Label"] = (
    damage_counts_total["geographic type"] + " + " + damage_counts_total["*
 ↪  Damage"].astype(str)
)

# Assign colors based on geographic type
colors = {"Rural": "green", "Suburban": "orange", "Urban": "red"}
damage_counts_total["Bar Color"] = damage_counts_total["geographic
 ↪  type"].map(colors)

# Plot the bar chart
plt.figure(figsize=(14, 8))
bars = plt.bar(
    damage_counts_total["Label"],
    damage_counts_total["Count"],
    color=damage_counts_total["Bar Color"],
)

# Add counts on top of each bar
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, yval + 0.5, int(yval),
 ↪  ha="center", va="bottom", fontsize=10)

# Customize the chart
plt.title("Fire Damage Counts by Geographic Type and Damage Level (2020-2023
 ↪  Total, Excluding No Damage)", fontsize=16)
plt.xlabel("Geographic Type + Damage Level", fontsize=12)
plt.ylabel("Counts", fontsize=12)
plt.xticks(rotation=45, ha="right", fontsize=10)

# Add a legend for the bar colors
```

```python
from matplotlib.lines import Line2D

legend_elements = [
    Line2D([0], [0], color="green", lw=6, label="Rural"),
    Line2D([0], [0], color="orange", lw=6, label="Suburban"),
    Line2D([0], [0], color="red", lw=6, label="Urban"),
]

plt.legend(handles=legend_elements, title="Geographic Type", loc="upper
 ↪  right", fontsize=10)

plt.tight_layout()

# Show the plot
plt.show()
```


Fire Damage Counts by Geographic Type and Damage Level (2020-2023 Total, Excluding No Damage)
Counts
Geographic Type + Damage Level
Geographic Type
Rural
Suburban
Urban
Rural + Affected (1-9%)
Rural + Minor (10-25%)
Rural + Major (26-50%)
Rural + Destroyed (>50%)
Rural + nan
Suburban + Affected (1-9%)
Suburban + Minor (10-25%)
Suburban + Major (26-50%)
Suburban + Destroyed (>50%)
Suburban + nan
Urban + Affected (1-9%)
Urban + Minor (10-25%)
Urban + Major (26-50%)
Urban + Destroyed (>50%)
Urban + nan