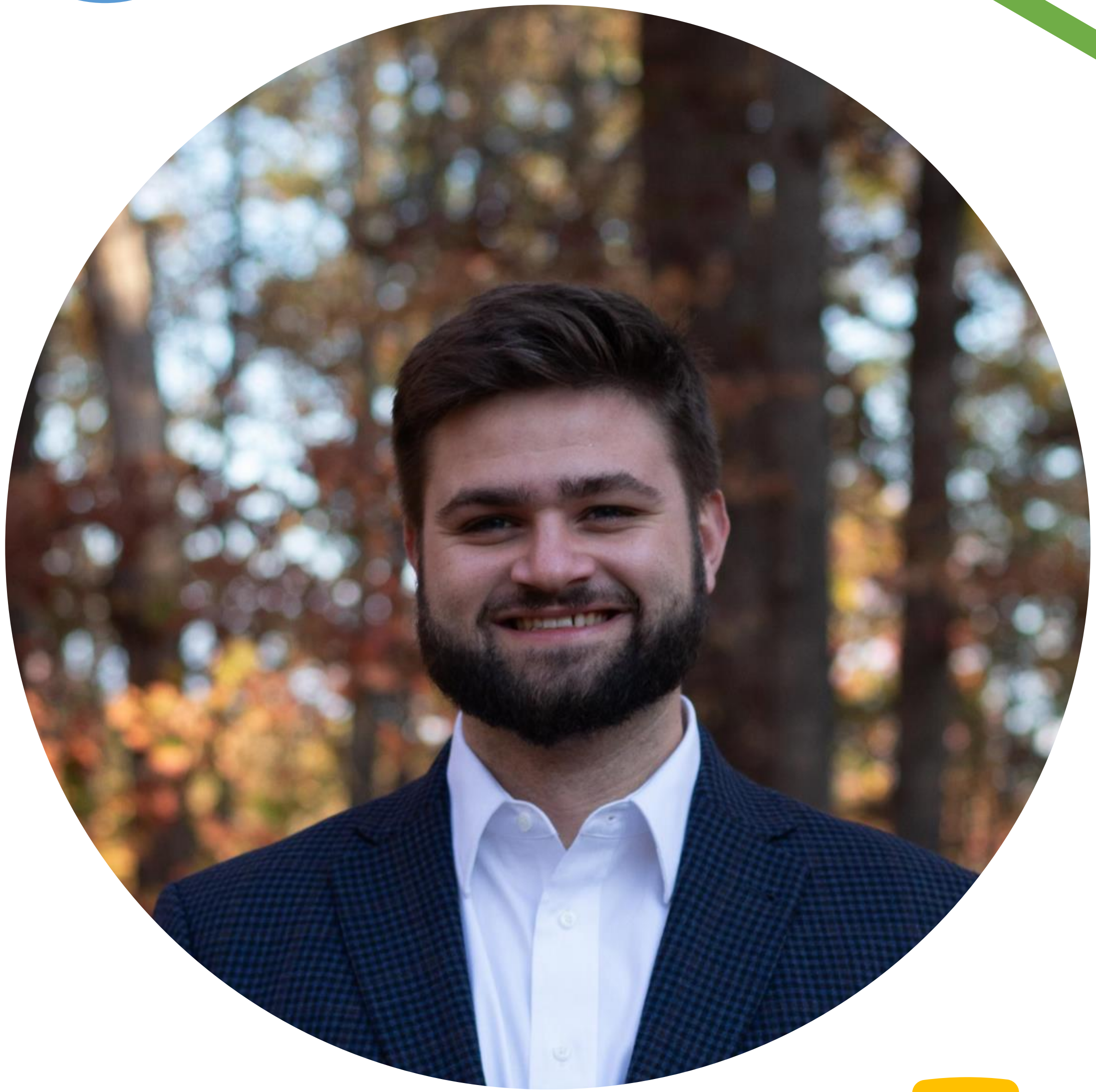


Python for Data Analysts

Brandon Akard

Senior Fire Rescue Analyst

Albemarle County Fire Rescue

Albemarle County, VA

My Background

- Originally from Sanford, Florida
- Went to The University of South Florida for Geography on an Army ROTC scholarship – studies focused on GIS
- Served 6 years in the Army Reserves as a Transportation Officer and deployed to Kuwait, Iraq, and Syria in 2018. This is where I found my love of Excel.
- Worked as a Transportation Manager for US Xpress for about 5 years while taking courses on SQL and Python. Founded manager project team that automated about a dozen processes saving hours of office time per week.
- Hired as the first Data Analyst for Albemarle County Fire Rescue in 2023!

Agenda

1.The Basics

2. Data Exploration

3. Data Transformation

4. Date Time

5. Matplotlib

6. Seaborn

7. Plotly

8. AI Coding Demo

The Basics

Foundational Terms

Programming Languages

Integrated Development Environment (IDE)

Virtual Environment

Library/ Module

Languages

Feature	Python	R
General Purpose	Full-featured programming language, ideal for data pipelines, dashboards, automation, APIs	Primarily built for statistical computing and visualization
Ease of Learning	Syntax is intuitive and readable—great for beginners from any field	More specialized; steeper learning curve for non-statisticians
Libraries & Tools	Powerful libraries like pandas, geopandas, numpy, matplotlib, plotly, scikit-learn, and folium for fire service analytics, GIS, modeling, and reporting	Excellent for statistical models and custom data visualizations using ggplot2, dplyr, tidyr, caret
GIS & Spatial Analysis	Strong integration with GeoPandas, Shapely, Folium, and web-based mapping like Leaflet and Kepler.gl	R also has packages like sf, sp, and tmap, but less support in web/GIS application development
Automation & Integration	Easy to build scripts that talk to databases, APIs, Excel, ArcGIS Pro, CAD/NFIRS exports, and automate reporting	Possible in R, but not as well-supported or versatile
Web & App Development	Seamless tools like Flask, Dash, and Streamlit for building interactive dashboards	Limited to Shiny (powerful but more complex setup)
Community & Support	Massive global user base with endless tutorials, especially for beginners and applied data scientists	Strong in academic/statistical circles, but smaller user base in emergency services context

IDEs

Feature	Jupyter Lab	VS Code
Workflow Style	Notebook-based: code, notes, tables, and plots side-by-side. Doesn't freeze up like MS Excel with thousands of rows!	Script-based: code runs top-to-bottom, outputs go to the terminal
Best For	Exploratory analysis, data cleaning, visual storytelling	Larger scripts, apps, and software development
Ease of Use	Great for beginners: run one cell at a time, tweak on the fly	Requires more structure and familiarity with files and environments
Output Display	Inline charts, tables, and maps	Plots pop out in separate windows or terminals
Documentation	Mix Markdown + Code for report-style analysis	Notes live in separate files or as comments
File Management	GUI for browsing folders and previewing datasets like Excel	Full file system view with more customization
Reproducibility	Each notebook becomes a live, explainable report	Requires extra effort to build that level of documentation

Essential Python Libraries

Tabular Data Analysis

Pandas

Datetime

Numpy

Data Visualization

Seaborn

Plotly

Matplotlib

Geospatial Libraries

ArcPy

Rasterio

Geopandas

Folium

Shapely

Apps and Web Development

Django

Flask

Dash

Streamlit

Data types

Category	Type	Example	Common Fire Service Use	Notes / Tips
Numeric	int	42	Count of calls, stations, personnel	Whole numbers
	float	3.75	Turnout time, response time, percentages	Use .round(2) for cleaner reporting
	bool	True, False	Flag calls on weekends, identify delays	Used in filtering (df[df['IsWeekend'] == True])
Text	str	"Engine 5"	Unit IDs, station names, incident types	.str.upper(), .str.contains() for filters
Date & Time	datetime	2024-01-01 08:00	Dispatch, arrival, clear time	Use pd.to_datetime() to convert
	timedelta	0 days 00:07:15	Time between dispatch and arrival	Subtract datetimes directly to get this
Collections	list	[1, 2, 3]	List of units on scene, response times per shift	Can grow/shrink; order matters
	tuple	("E5", "ST01")	Station-unit pairs, lat/lon points	Fixed-length and immutable
	dict	{"Station": "E5", "Calls": 22}	Lookups by key (e.g., staffing, zone config)	Access via keys (mydict["Station"])
Pandas Types	object	"Albemarle County"	Text column (fallback for mixed types)	Convert to category for faster operations
	int64, float64	12, 5.8	Numeric columns in DataFrames	Result of importing CSVs with numbers
	datetime64[ns]	2024-01-01 08:00:00	Time-based indexing, resampling	Required for .resample() and .rolling()
	category	"Fire", "EMS", "MVC"	Call types, districts, stations	Saves memory, improves groupby/filter performance

Let’s Get into it!

Data Exploration

```
[3]: import pandas as pd
data = pd.read_csv(r"C:\Users\bakard\Python\Scripts\presentation_dataset.csv")
```

```
•[5]: data = data[['PSAPDateTime', ## the Public Safety Answering Point
                 'CallID', ## The ID Of the Call
                 'FireRescueDistrict', ## The First Due Area/ Planning Zone
                 'CADType', ## The CAD Type - how the call went out
                 'CADCategory',##The Category of the Call Type
                 'hexID', ## An arbitrary Geographical grid for analysis purposes
                 'Longitude', ##The X Axis Coordinate
                 'Latitude', ## The Y Axis Coordinate
                 'AppOwner', ## The Apparatus Owner
                 'UnitNumber', ## The Unit Number
                 'DispatchDateTime', ## The Time the Unit Received Dispatch
                 'EnrouteDateTime', ## The Time the Unit Marked En Route
                 'ArriveDateTime', ## The Time the Unit Arrived on Scene
                 'ClearDateTime', ## The Time the Unit Cleared the Incident
                 'TransportDateTime', ## The Time the Unit marked en route to the hospital
                 'AtHospitalDateTime' ## The Time the Unit arrived at the hospital
                ]]

# Convert PSAPDateTime to actual datetime format
data['PSAPDateTime'] = pd.to_datetime(data['PSAPDateTime']).copy()
```

- Load the data
- Select Columns
- Set PSAP to a datetime

Explore the Data

```
data.dtypes
```

```
PSAPDateTime          datetime64[ns]
CallID                         int64
FireRescueDistrict            object
CADType                       object
CADCategory                   object
hexID                          int64
Longitude                    float64
Latitude                     float64
AppOwner                      object
UnitNumber                    object
DispatchDateTime              object
EnrouteDateTime               object
ArriveDateTime                object
ClearDateTime                 object
TransportDateTime             object
AtHospitalDateTime            object
dtype: object
```

```
[19]: ## explore the shape of the dataframe

      data.shape
```

```
[19]: (33189, 16)
```

Explore the Data

```
## explore the descriptive statistics of the quantatative data in the dataframe

data.describe()
```

	PSAPDateTime	CallID	hexID	Longitude	Latitude	DispatchDateTime	EnrouteDateTime	ArriveDateTime
count	33189	3.318900e+04	33189.000000	33189.000000	33189.000000	33189	29524	24444
mean	2023-07-06 18:39:20.823597312	4.257255e+06	7015.192714	-78.524458	38.043766	2023-07-06 18:44:29.746476288	2023-07-07 01:50:33.577664	2023-07-07 00:46:37.057028352
min	2023-01-01 00:31:42.523000	4.141915e+06	7.000000	-78.831409	37.734385	2023-01-01 00:33:35.380000	2023-01-01 00:34:33.697000	2023-01-01 00:42:21.053000
25%	2023-04-05 12:53:27.852999936	4.200975e+06	5580.000000	-78.581706	38.013389	2023-04-05 13:21:11.660000	2023-04-05 20:43:49.751500032	2023-04-06 12:32:33.802249984
50%	2023-07-09 05:06:10.863000064	4.260135e+06	8114.000000	-78.494559	38.059419	2023-07-09 05:08:33.516999936	2023-07-09 09:36:12.797000192	2023-07-08 22:35:48.576499968
75%	2023-10-07 01:05:11.416999936	4.314118e+06	8949.000000	-78.456382	38.079205	2023-10-07 01:11:46.569999872	2023-10-07 07:21:42.974249984	2023-10-07 02:30:32.389750016
max	2023-12-31 23:30:08.223000	4.361690e+06	11909.000000	-78.210995	38.247679	2023-12-31 23:31:58.860000	2023-12-31 23:33:36.513000	2023-12-31 23:43:49.380000
std	NaN	6.434231e+04	2884.559715	0.107547	0.074406	NaN	NaN	NaN

Explore the Data

```
[23]: # explore the distributation of the categorical and other qualatative data

      data.CADType.value_counts()
```

```
[23]: CADType
      Motor Vehicle Crash - Injuries      3819
      General Illness                     3393
      Fall                                2602
      Cardiac Related                     2411
      Respiratory                         2360
      Unconscious                         1602
      Fire Alarm                          1548
      Neurological                        1348
      Outdoor Fire                        1178
```

Let’s Move On!

Data Transformation

[29]:
```python
## Sorting

data.sort_values(by='PSAPDateTime',ascending=False)
```

	PSAPDateTime	CallID	FireRescueDistrict	CADType	CADCategory	hexID	Longitude
29756	2023-12-31 23:30:08.223	4361690	Earlysville	Unconscious	EMS	9297	-78.441633
29763	2023-12-31 23:22:43.997	4361685	Crozet	Fall	EMS	1746	-78.708843
29831	2023-12-31 23:22:43.997	4361685	Crozet	Fall	EMS	1746	-78.708843
29524	2023-12-31 22:23:33.750	4361654	Ivy	Fire Alarm	Fires	4531	-78.615538
29499	2023-12-31 22:12:47.240	4361647	Pantops	Unconscious	EMS	9436	-78.430798

```
Incidents = data.drop_duplicates(subset='CallID')

Incidents.shape
```

```
(17297, 16)
```

Handling Duplicates

```python
Incidents = data.drop_duplicates(subset='CallID')

Incidents.shape
```

```
(17297, 16)
```

```python
Cancelled_enroute = data.sort_values(by=['PSAPDateTime','ArriveDateTime'],ascending=True)
Cancelled_enroute = Cancelled_enroute.drop_duplicates(subset='CallID',keep='first')
Cancelled_enroute = Cancelled_enroute[Cancelled_enroute.ArriveDateTime.isna()]
Cancelled_enroute.shape
```

```
(1211, 16)
```

```python
arrived = data.sort_values(by=['PSAPDateTime','ArriveDateTime'],ascending=True)
arrived = data.dropna(subset='ArriveDateTime')
arrived = arrived.drop_duplicates(subset='CallID')

print(f"Total: {len(arrived) + len(Cancelled_enroute)}")
```

```
Total: 17297
```

Mask Filtering

```python
## filtering

WaterRescues = Incidents[Incidents['CADType'] == 'Water Rescue']


Scottsville_WaterRescues = Incidents[
                                    (Incidents['FireRescueDistrict'] == 'Scottsville') &
                                    (Incidents['CADType'] == 'Water Rescue')
                                    ]



## Print Statements and F Strings
print(
    f"""
    There were {len(WaterRescues)} Water Rescues in 2023
    {len(Scottsville_WaterRescues)} of {len(WaterRescues)} were in Scottsville's First Due
    """
     )
```

```

    There were 6 Water Rescues in 2023
    1 of 6 were in Scottsville's First Due
```

Query Filtering

```
## Filtering Using .query

Incidents.query("CADType == 'Water Rescue' & FireRescueDistrict == 'Scottsville'")
```

	PSAPDateTime	CallID	FireRescueDistrict	CADType	CADCategory	hexID	Longitude	Latitude
3248	2023-07-04 16:33:31.947	4257204	Scottsville	Water Rescue	Rescue Incidents	5876.0	-78.570463	37.761031

Next: Handling Date Times

```python
# To convert multiple at once:
datetime_cols = [
    'DispatchDateTime',
    'EnrouteDateTime',
    'ArriveDateTime',
    'ClearDateTime',
    'TransportDateTime',
    'AtHospitalDateTime'
]
# 'For' loop example:
for col in datetime_cols:
    data[col] = pd.to_datetime(data[col], errors='coerce')  # errors='coerce' handles bad/missing values safely


# Example: Nowe let's Filter for incidents on or after Jan 1, 2023
data = data[data['PSAPDateTime'].between('2023-01-01','2024-01-01')]
```

- Convert multiple date columns
- 'For' Loops
- Filter by a datetime column

Data types have changed!

```
#explore the data types of the dataframe
data.dtypes
```

```
PSAPDateTime             datetime64[ns]
CallID                            int64
FireRescueDistrict               object
CADType                          object
CADCategory                      object
hexID                             int64
Longitude                       float64
Latitude                        float64
AppOwner                         object
UnitNumber                       object
DispatchDateTime         datetime64[ns]
EnrouteDateTime          datetime64[ns]
ArriveDateTime           datetime64[ns]
ClearDateTime            datetime64[ns]
TransportDateTime        datetime64[ns]
AtHospitalDateTime       datetime64[ns]
dtype: object
```

Time Features Cheat Sheet

Feature	Code	Description
Date	data['DispatchDateTime'].dt.date	Removes the time portion
Time	data['DispatchDateTime'].dt.time	Removes the date portion
Hour	data['DispatchHour'] = data['DispatchDateTime'].dt.hour	Useful for hourly trends
Minute	data['DispatchMinute'] = data['DispatchDateTime'].dt.minute	Good for fine-grained time slices
Day of Week	data['DispatchDOW'] = data['DispatchDateTime'].dt.dayofweek	Monday = 0, Sunday = 6
Day Name	data['DispatchDayName'] = data['DispatchDateTime'].dt.day_name()	More readable version of DOW
Month	data['DispatchMonth'] = data['DispatchDateTime'].dt.month	1 to 12
Month Name	data['DispatchMonthName'] = data['DispatchDateTime'].dt.month_name()	Full name of month
Quarter	data['DispatchQuarter'] = data['DispatchDateTime'].dt.quarter	Q1 to Q4
Year	data['DispatchYear'] = data['DispatchDateTime'].dt.year	2023, 2024, etc.
Week Number	data['DispatchWeek'] = data['DispatchDateTime'].dt.isocalendar().week	ISO week of year
Is Weekend?	data['IsWeekend'] = data['DispatchDateTime'].dt.dayofweek >= 5	True for Saturday/Sunday
AM/PM	data['AM_PM'] = data['DispatchDateTime'].dt.strftime('%p')	Shows 'AM' or 'PM'

```python
## Time Series features

## Time of Day

data['HourOfDay'] = data['DispatchDateTime'].dt.hour
data['AM_PM'] = data['DispatchDateTime'].dt.strftime('%p')

data[['DispatchDateTime','HourOfDay','AM_PM']]
```

	DispatchDateTime	HourOfDay	AM_PM
0	2023-10-09 14:48:23.150	14	PM
1	2023-10-09 12:30:39.007	12	PM
2	2023-10-09 07:18:59.963	7	AM
3	2023-10-09 14:48:22.900	14	PM
4	2023-10-09 05:28:54.570	5	AM
...	...	...	...

```python
## Day of Week

data['DayName'] = data['DispatchDateTime'].dt.day_name()
data['IsWeekend'] = data['DispatchDateTime'].dt.dayofweek >= 5

data[['DispatchDateTime','DayName','IsWeekend']]
```

	DispatchDateTime	DayName	IsWeekend
0	2023-10-09 14:48:23.150	Monday	False
1	2023-10-09 12:30:39.007	Monday	False

```python
## Months, Quarters, Years
data['MonthName'] = data['DispatchDateTime'].dt.month_name()
data['Quarter'] = data['DispatchDateTime'].dt.quarter
data['Year'] = data['DispatchDateTime'].dt.year

data[['DispatchDateTime','MonthName','Quarter','Year']]
```

	DispatchDateTime	MonthName	Quarter	Year
0	2023-10-09 14:48:23.150	October	4	2023
1	2023-10-09 12:30:39.007	October	4	2023
2	2023-10-09 07:18:59.963	October	4	2023
3	2023-10-09 14:48:22.900	October	4	2023
4	2023-10-09 05:28:54.570	October	4	2023
...	...	...	...	...
34052	2023-09-25 11:44:58.027	September	3	2023
34053	2023-09-25 21:59:01.303	September	3	2023

```python
## Application: Create a column for Nights and Weekends/ or Daytime

# Extract hour and day of week
data['HourOfDay'] = data['DispatchDateTime'].dt.hour
data['DayOfWeek'] = data['DispatchDateTime'].dt.dayofweek  # Monday = 0, Sunday = 6

# Create the column with default value
data['TimeCategory'] = 'Nights & Weekends'

# Overwrite with 'Daytime Weekday' where condition matches
weekday_daytime_mask = (
    (data['DayOfWeek'] < 5) &  # Monday to Friday
    (data['HourOfDay'] >= 6) &
    (data['HourOfDay'] < 18)
)

data.loc[weekday_daytime_mask, 'TimeCategory'] = 'Daytime Weekday'

data[['DispatchDateTime','HourOfDay','DayOfWeek','DayName','TimeCategory']]
```

- Custom time features
- .loc

e	HourOfDay	DayOfWeek	DayName	TimeCategory
0	14	0	Monday	Daytime Weekday
7	12	0	Monday	Daytime Weekday
2023-10-09 07:18:59.963	7	0	Monday	Daytime Weekday
2023-10-09 14:48:22.900	14	0	Monday	Daytime Weekday
2023-10-09 05:28:54.570	5	0	Monday	Nights & Weekends
...	...	...	...	...
2023-09-25 11:44:58.027	11	0	Monday	Daytime Weekday
2023-09-25 21:59:01.303	21	0	Monday	Nights & Weekends

```
## The All-Important Time Deltas

data['ResponseTime'] = data['ArriveDateTime'] - data['DispatchDateTime']

data.dtypes
```

```
AtHospitalDateTime       datetime64[ns]
HourOfDay                         int32
AM_PM                            object
DayName                          object
IsWeekend                          bool
MonthName                        object
Quarter                           int32
Year                              int32
Hour                              int32
DayOfWeek                         int32
TimeCategory                     object
ResponseTime            timedelta64[ns]
dtype: object
```

RESPONSE TIME & Other Time Deltas

```python
data['ResponseTimeSeconds'] = (data['ResponseTime'].dt.total_seconds()).round(1)
data['ResponseTimeMinutes'] = (data['ResponseTime'].dt.total_seconds() / 60).round(1)

response_times = data[['DispatchDateTime','ArriveDateTime','ResponseTime','ResponseTimeSeconds','ResponseTimeMinutes']]
response_times
```

DispatchDateTime	ArriveDateTime	ResponseTime	ResponseTimeSeconds	ResponseTimeMinutes
2023-10-09 14:48:23.150	2023-10-09 14:56:05.310	0 days 00:07:42.160000	462.2	7.7
2023-10-09 12:30:39.007	2023-10-09 12:33:03.623	0 days 00:02:24.616000	144.6	2.4
2023-10-09 07:18:59.963	2023-10-09 07:27:45.213	0 days 00:08:45.250000	525.2	8.8
2023-10-09 14:48:22.900	2023-10-09 14:57:31.887	0 days 00:09:08.987000	549.0	9.1
2023-10-09 05:28:54.570	2023-10-09 05:34:33.683	0 days 00:05:39.113000	339.1	5.7
...	...	...	...	...
2023-09-25 11:44:58.027	2023-09-25 11:49:57.860	0 days 00:04:59.833000	299.8	5.0
2023-09-25 21:59:01.303	2023-09-25 22:05:29.437	0 days 00:06:28.134000	388.1	6.5
2023-09-25 19:44:40.390	2023-09-25 19:50:45.613	0 days 00:06:05.223000	365.2	6.1
2023-09-25 12:20:49.767	2023-09-25 12:24:23.970	0 days 00:03:34.203000	214.2	3.6
2023-09-25 14:58:03.410	NaT	NaT	NaN	NaN

```python
development_area_response = data[['CallID','UnitNumber','DispatchDateTime','ArriveDateTime','ResponseTimeMinutes','DevRA','FireRescueDistrict']]

development_area_response = development_area_response[development_area_response['DevRA'] == 'development area']

development_area_response = development_area_response.dropna(subset='ResponseTimeMinutes')

development_area_response = development_area_response[development_area_response['ResponseTimeMinutes'] >= 0]

development_area_response = development_area_response.sort_values(by=['CallID','ResponseTimeMinutes'], ascending=True)



development_area_response = development_area_response.drop_duplicates(subset='CallID')
```

```python
development_area_response.ResponseTimeMinutes.describe(percentiles=[.25,.5,.75,.9,.99])
```

```
count    10584.000000
mean         6.184769
std          4.055309
min          0.000000
25%          4.000000
50%          5.600000
75%          7.800000
90%         10.400000
99%         18.400000
max        152.400000
Name: ResponseTimeMinutes, dtype: float64
```

```python
def response_time_90th(vals: pd.Series) -> float:
    """
    Compute the 90th percentile response time (NFPA-style).
    Uses 'higher' interpolation to reflect real-world performance thresholds.
    """
    return vals.quantile(0.90, interpolation='higher')
```

```python
development_area_response_table = development_area_response.pivot_table(
                                            index='FireRescueDistrict',
                                            values='ResponseTimeMinutes',
                                            aggfunc=response_time_90th
                                                    )
```

PIVOT TABLES!

	ResponseTimeMinutes
FireRescueDistrict	
City	9.6
Crozet	7.3
East Rivanna	12.5
Hollymead	9.3
Ivy	9.7
Monticello	11.9
Pantops	9.5
Scottsville	9.4
Seminole	10.7
UVA	13.8

```python
development_area_response_table = development_area_response.pivot_table(
                                                index='FireRescueDistrict',
                                                values='ResponseTimeMinutes',
                                                columns='MonthNumber',
                                                aggfunc=response_time_90th
                                                )
```

MonthNumber	1	2	3	4	5	6	7	8	9	10	11	12
FireRescueDistrict												
City	11.0	10.2	10.4	9.9	9.5	8.2	10.5	12.2	11.1	8.6	9.5	7.3
Crozet	7.3	7.1	6.8	6.8	7.2	8.2	7.1	6.8	8.1	8.2	7.4	7.7
East Rivanna	10.3	12.0	9.1	14.7	14.8	12.5	14.6	11.1	13.5	14.2	9.3	11.5
Hollymead	8.5	9.5	9.2	9.4	9.8	10.4	8.7	9.3	9.1	7.9	9.7	8.6
Ivy	9.6	10.4	8.1	7.9	8.7	9.4	9.8	9.4	9.4	9.6	11.2	11.8
Monticello	10.7	11.3	11.5	11.3	12.3	13.0	13.1	12.8	11.1	11.5	11.4	12.2
Pantops	8.5	10.3	8.3	8.5	11.0	10.5	11.7	9.1	9.8	10.1	9.6	10.3
Scottsville	8.8	9.4	6.6	8.2	8.9	25.7	10.0	11.7	8.7	9.3	7.7	15.2
Seminole	10.3	9.7	11.0	9.9	10.8	11.5	10.9	10.5	11.3	10.8	11.1	11.1
UVA	7.3	8.8	6.8	8.4	9.7	7.3	14.8	5.2	7.5	13.8	11.2	NaN

PIVOT TABLES!

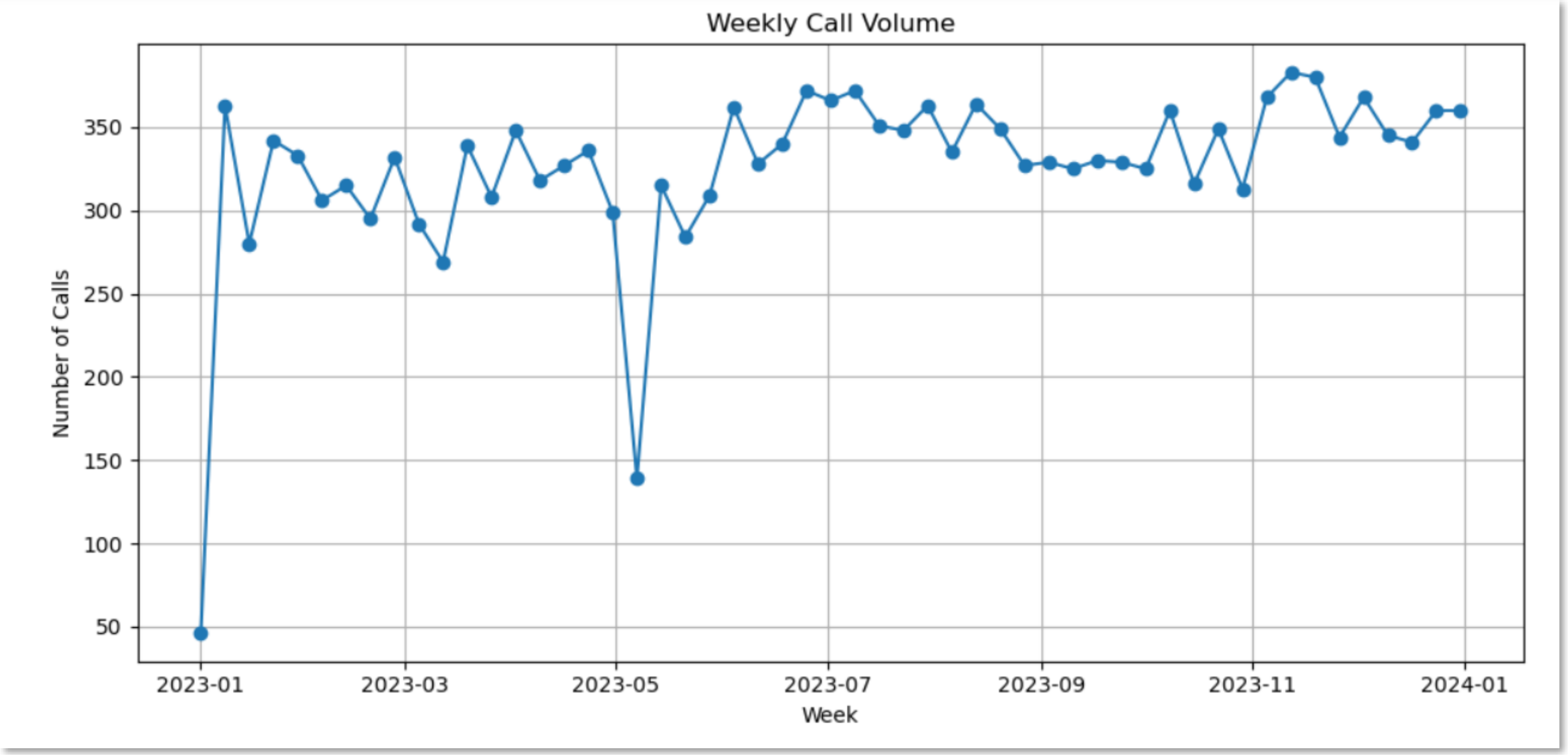
```python
data_time_index = development_area_response.set_index('DispatchDateTime')

# Total calls per day
weekly_calls = data_time_index.resample('W').size().rename('WeeklyCallCount')

# Average response time per week
dev_weekly_90th = data_time_index['ResponseTimeMinutes'].resample('W').quantile(.9).rename('ResponseTime90th')

weekly_summary = pd.concat([weekly_calls, dev_weekly_90th], axis=1,names=['Weekly_Count','Weekly_90thPercentile_Response'])
```

	WeeklyCallCount	DevelopmentArea_ResponseTime90th
DispatchDateTime		
2023-01-01	32	10.16
2023-01-08	238	10.33
2023-01-15	185	9.46
2023-01-22	212	9.59
2023-01-29	223	10.46
2023-02-05	193	9.98
2023-02-12	187	9.76
[illegible]	[illegible]	[illegible]

Let’s talk about visuals!

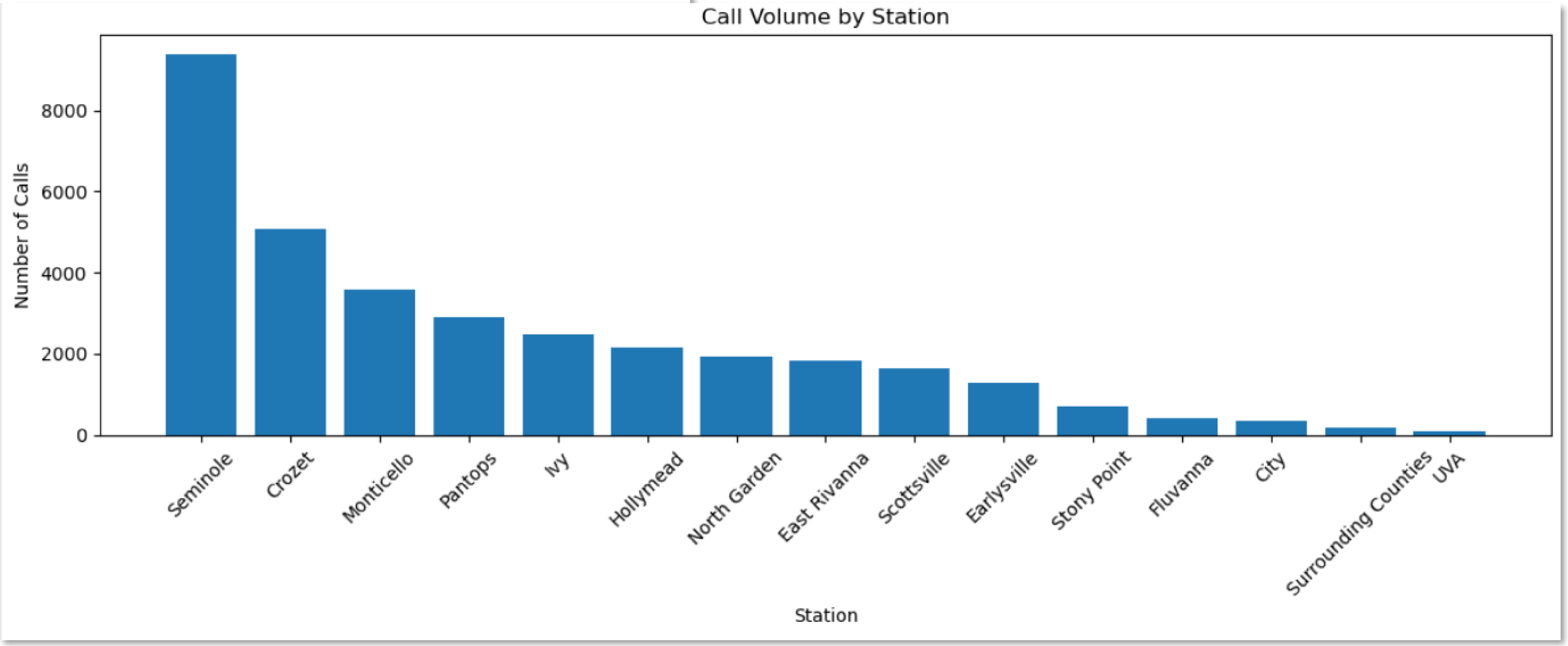
```python
import matplotlib.pyplot as plt
weekly_calls = Incidents.set_index('PSAPDateTime').resample('W').size()

plt.figure(figsize=(10, 5))
plt.plot(weekly_calls.index, weekly_calls.values, marker='o')
plt.title("Weekly Call Volume")
plt.xlabel("Week")
plt.ylabel("Number of Calls")
plt.grid(True)
plt.tight_layout()
plt.show()
```

MATPLOTLIB



```python
calls_by_station = data['FireRescueDistrict'].value_counts()

plt.figure(figsize=(12, 5))
plt.bar(calls_by_station.index, calls_by_station.values)
plt.title("Call Volume by Station")
plt.xlabel("Station")
plt.ylabel("Number of Calls")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

MATPLOTLIB



SEABORN

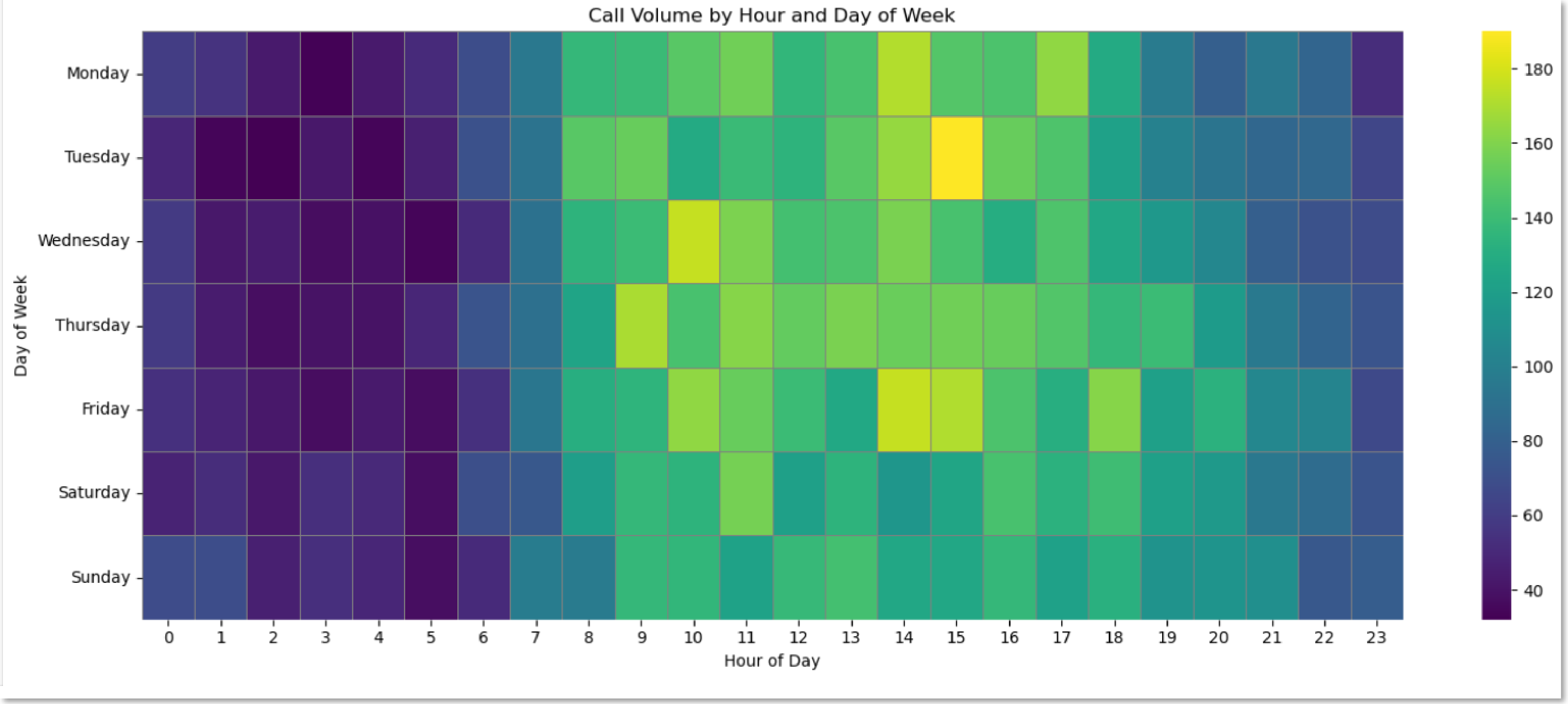
```python
import seaborn as sns
```

```python
# Create pivot table: counts of calls by Day of Week and Hour
heatmap_data = data.pivot_table(
    index='DayName',
    columns='Hour',
    values='CallID',  # or any column that exists
    aggfunc='count'
)

# Reorder days for readability
ordered_days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
heatmap_data = heatmap_data.reindex(ordered_days)
heatmap_data
```

Hour	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22	23
DayName																					
Monday	60	55	43	33	43	51	69	95	137	139	...	172	148	145	164	128	97	79	95	83	52
Tuesday	49	34	32	42	34	46	71	92	149	153	...	165	190	153	146	122	101	93	84	85	65
Wednesday	59	42	44	37	39	34	51	91	134	140	...	158	144	130	146	126	116	105	79	72	68
Thursday	59	44	38	40	40	49	73	90	124	170	...	154	156	153	147	137	140	118	96	82	73
Friday	54	48	42	37	43	37	54	94	131	135	...	176	171	145	131	161	121	133	105	103	67
Saturday	47	52	42	53	50	38	70	75	120	137	...	115	125	144	133	141	121	117	95	87	73
Sunday	68	69	46	53	49	38	51	98	97	137	...	126	126	137	122	132	113	114	110	75	78

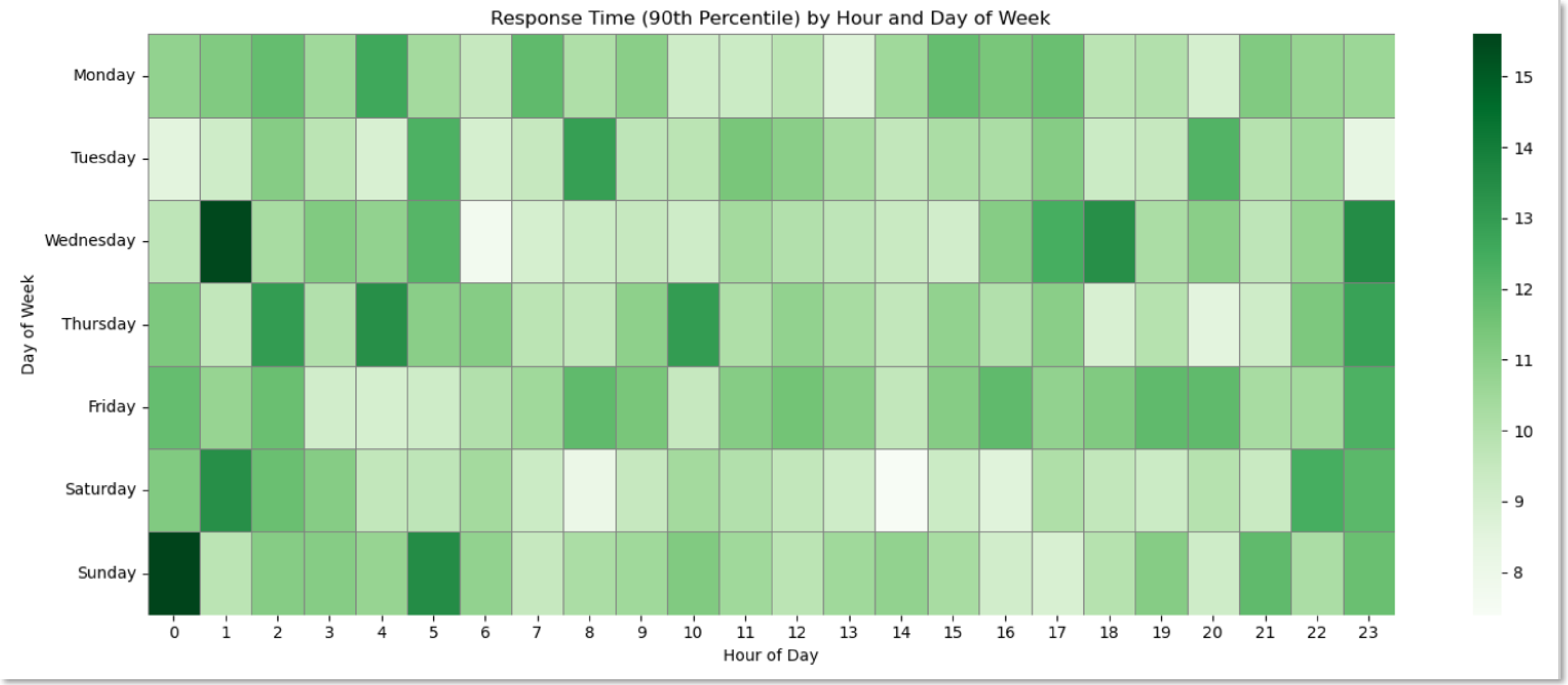
7 rows × 24 columns

```python
plt.figure(figsize=(15, 6))
sns.heatmap(heatmap_data, cmap='viridis', linewidths=0.5, linecolor='gray')

plt.title("Call Volume by Hour and Day of Week")
plt.xlabel("Hour of Day")
plt.ylabel("Day of Week")
plt.tight_layout()
plt.show()
```

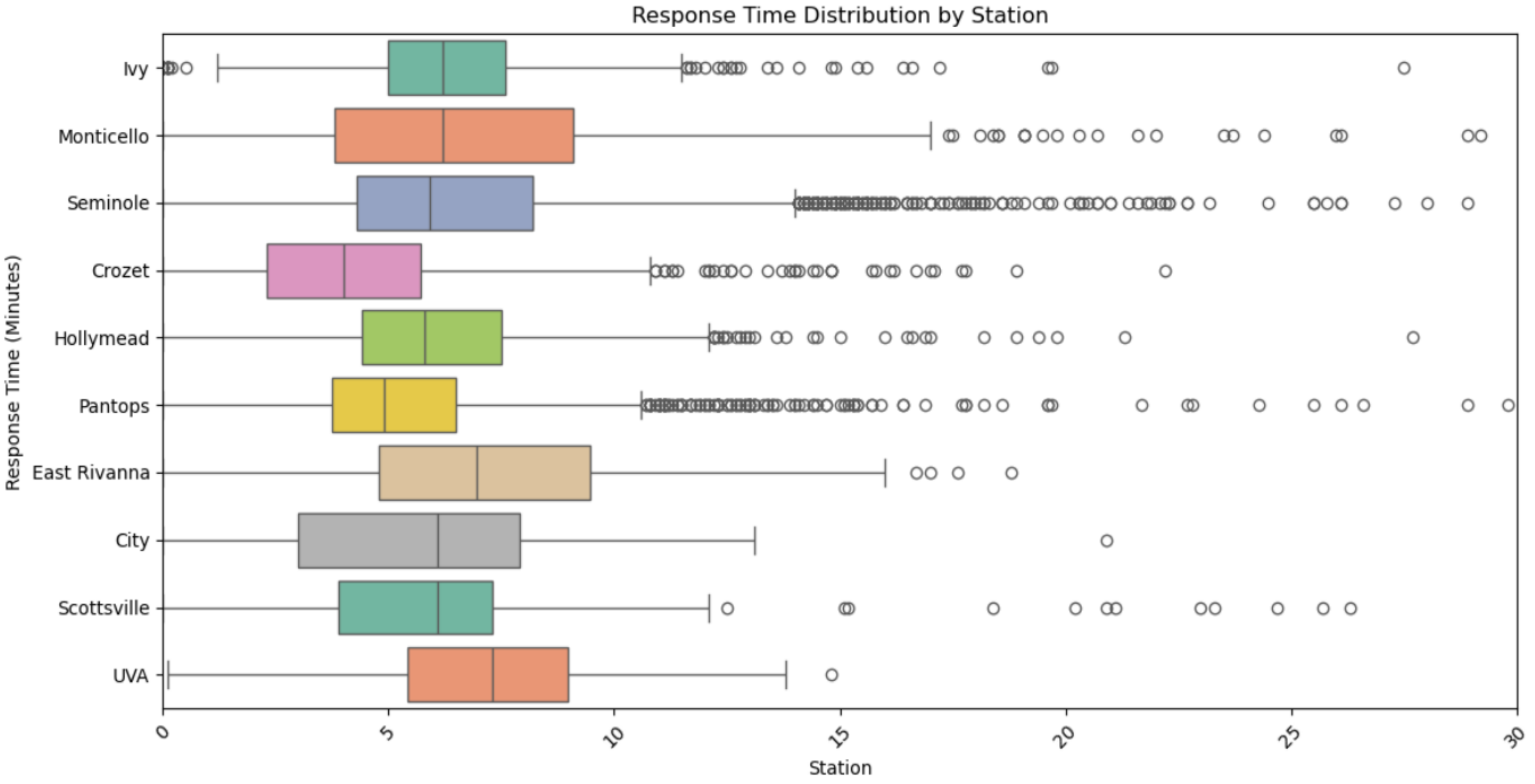


```python
import seaborn as sns


# Create pivot table: counts of calls by Day of Week and Hour
heatmap_data = development_area_response.pivot_table(
    index='DayName',
    columns='Hour',
    values='ResponseTimeMinutes',  # or any column that exists
    aggfunc= response_time_90th
)

# Reorder days for readability
ordered_days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
heatmap_data = heatmap_data.reindex(ordered_days)
heatmap_data
```

```python
plt.figure(figsize=(15, 6))
sns.heatmap(heatmap_data, cmap='viridis', linewidths=0.5, linecolor='gray')

plt.title("Call Volume by Hour and Day of Week")
plt.xlabel("Hour of Day")
plt.ylabel("Day of Week")
plt.tight_layout()
plt.show()
```

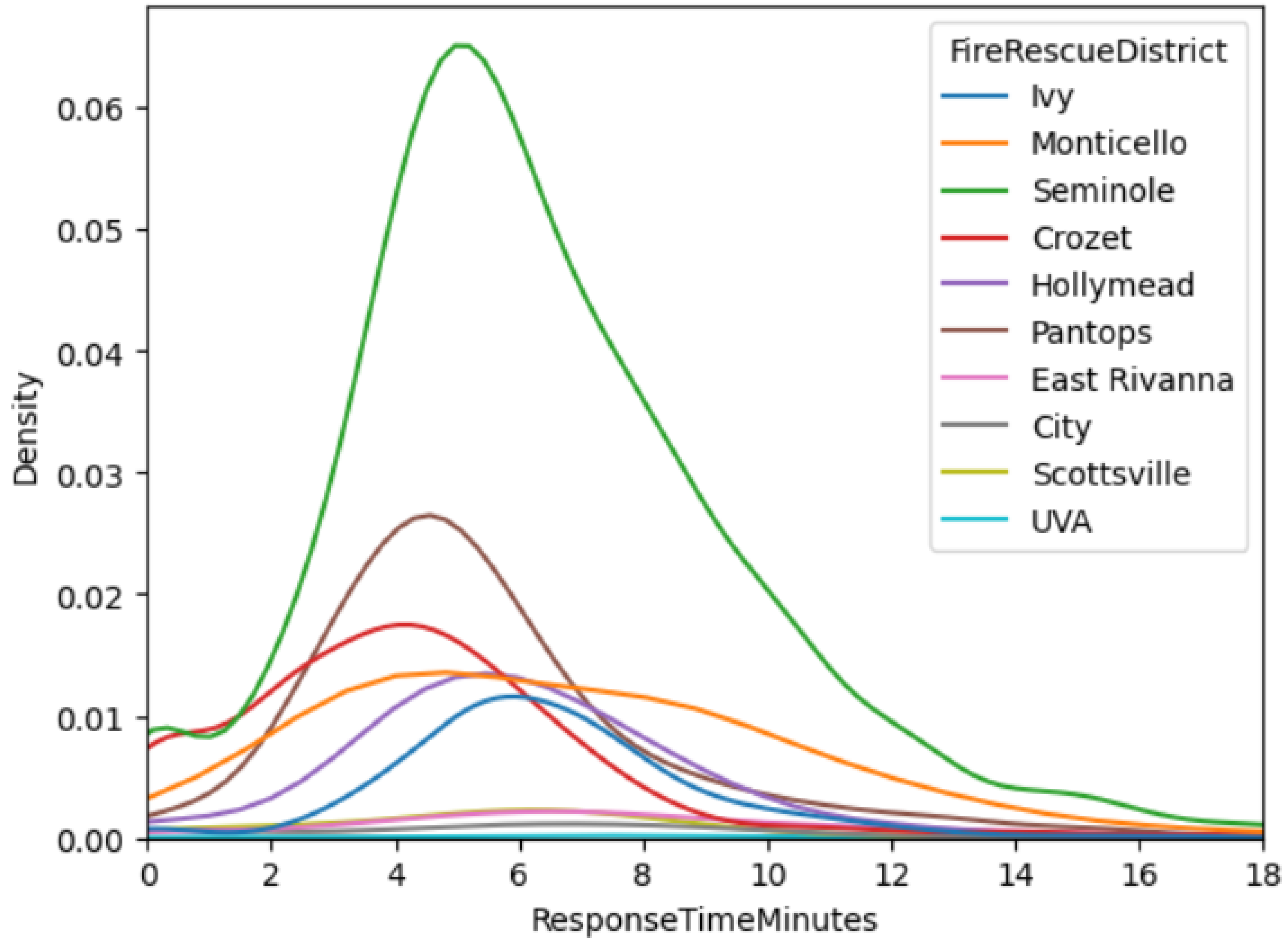


```python
plt.figure(figsize=(12, 6))

sns.boxplot(
    data=development_area_response,
    x='ResponseTimeMinutes',  # or 'UnitID', 'Shift', etc.
    y='FireRescueDistrict',
    palette='Set2',
    orient = 'h'
)


plt.title("Response Time Distribution by Station")
plt.xlabel("Station")
plt.ylabel("Response Time (Minutes)")
plt.xlim(0, 30)  # ⬅ Limit Y-axis to 30 minutes
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



```
sns.kdeplot(data=development_area_response, x='ResponseTimeMinutes', hue='FireRescueDistrict')
plt.xlim(0,18)
```

```
(0.0, 18.0)
```

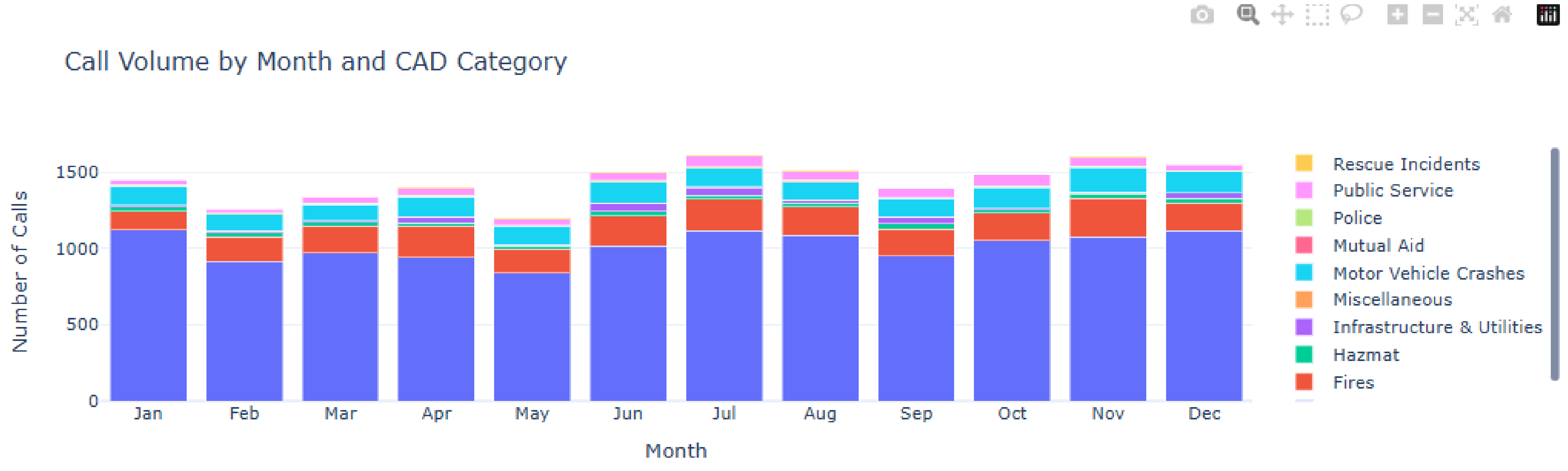


	ResponseTimeMinutes
FireRescueDistrict	
City	9.6
Crozet	7.3
East Rivanna	12.5
Hollymead	9.3
Ivy	9.7
Monticello	11.9
Pantops	9.5
Scottsville	9.4
Seminole	10.7
UVA	13.8

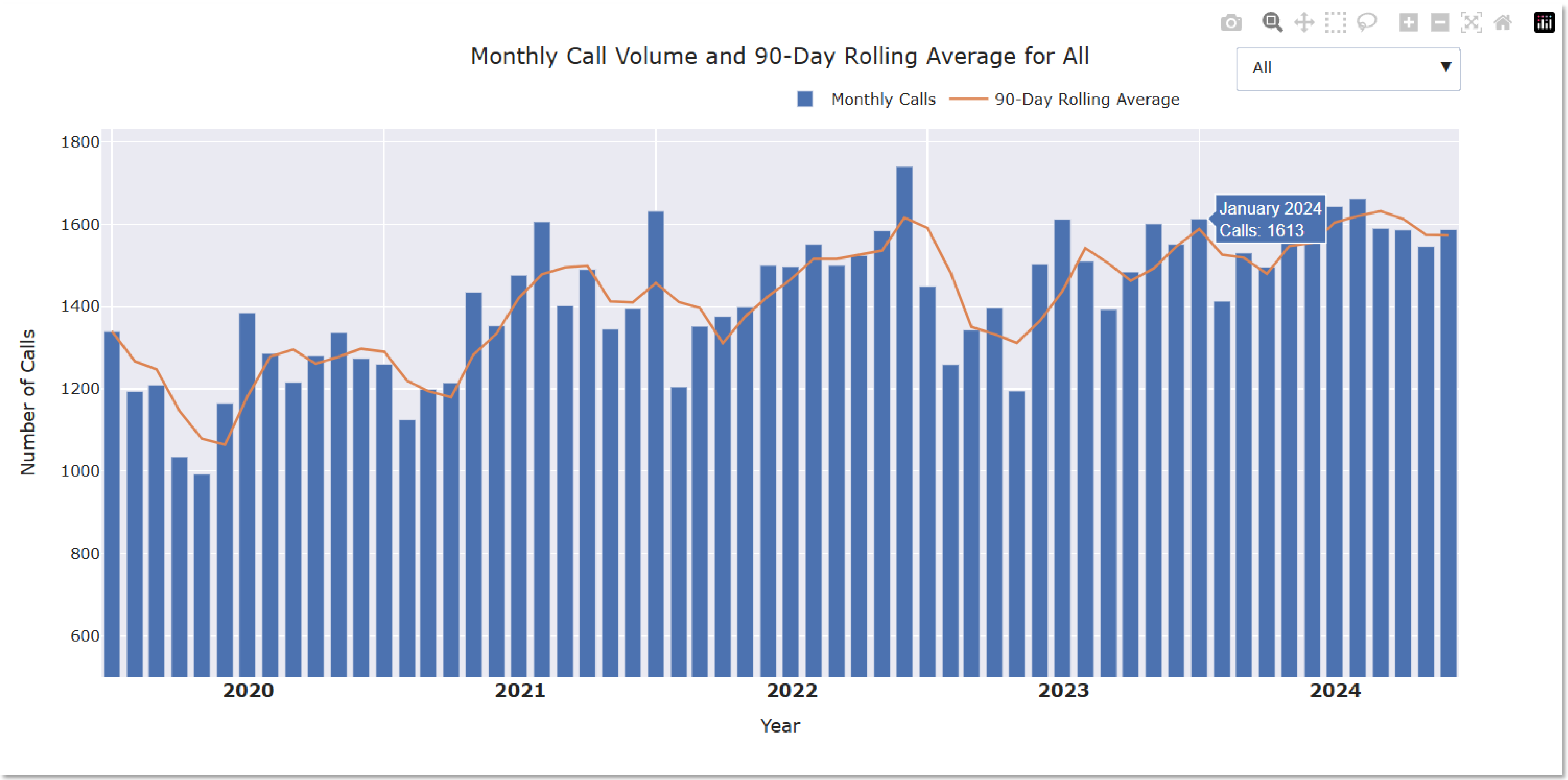
plotly

Plotly Interactive Charts!

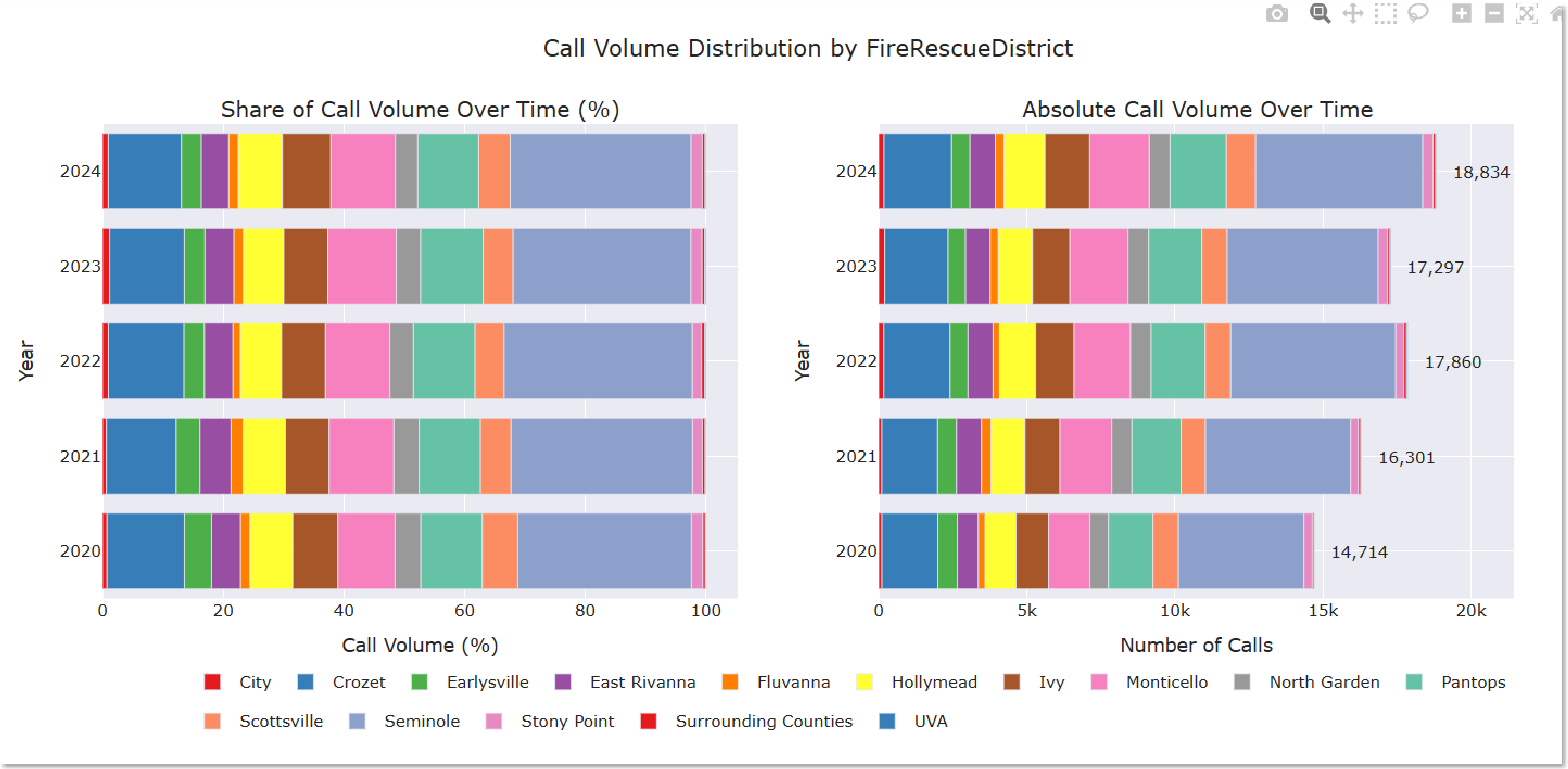
```python
# Ensure datetime column is parsed
Incidents['DispatchDateTime'] = pd.to_datetime(Incidents['DispatchDateTime'])
Incidents['Month'] = Incidents['DispatchDateTime'].dt.month

# Group by Month and CAD Category
monthly_group = Incidents.groupby(['Month', 'CADCategory']).size().unstack(fill_value=0)

# Sort by month (optional but clean)
monthly_group = monthly_group.sort_index()

# Create figure
fig = go.Figure()

# Add a stacked bar trace for each CAD Category
for category in monthly_group.columns:
    fig.add_trace(
        go.Bar(
            x=monthly_group.index,
            y=monthly_group[category],
            name=category
        )
    )

# Update layout
fig.update_layout(
    title="Call Volume by Month and CAD Category",
    xaxis_title="Month",
    yaxis_title="Number of Calls",
    barmode='stack',           # Stacked bar chart
    template="plotly_white",
    xaxis=dict(
        tickmode='array',
        tickvals=list(range(1, 13)),
        ticktext=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                  'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
```

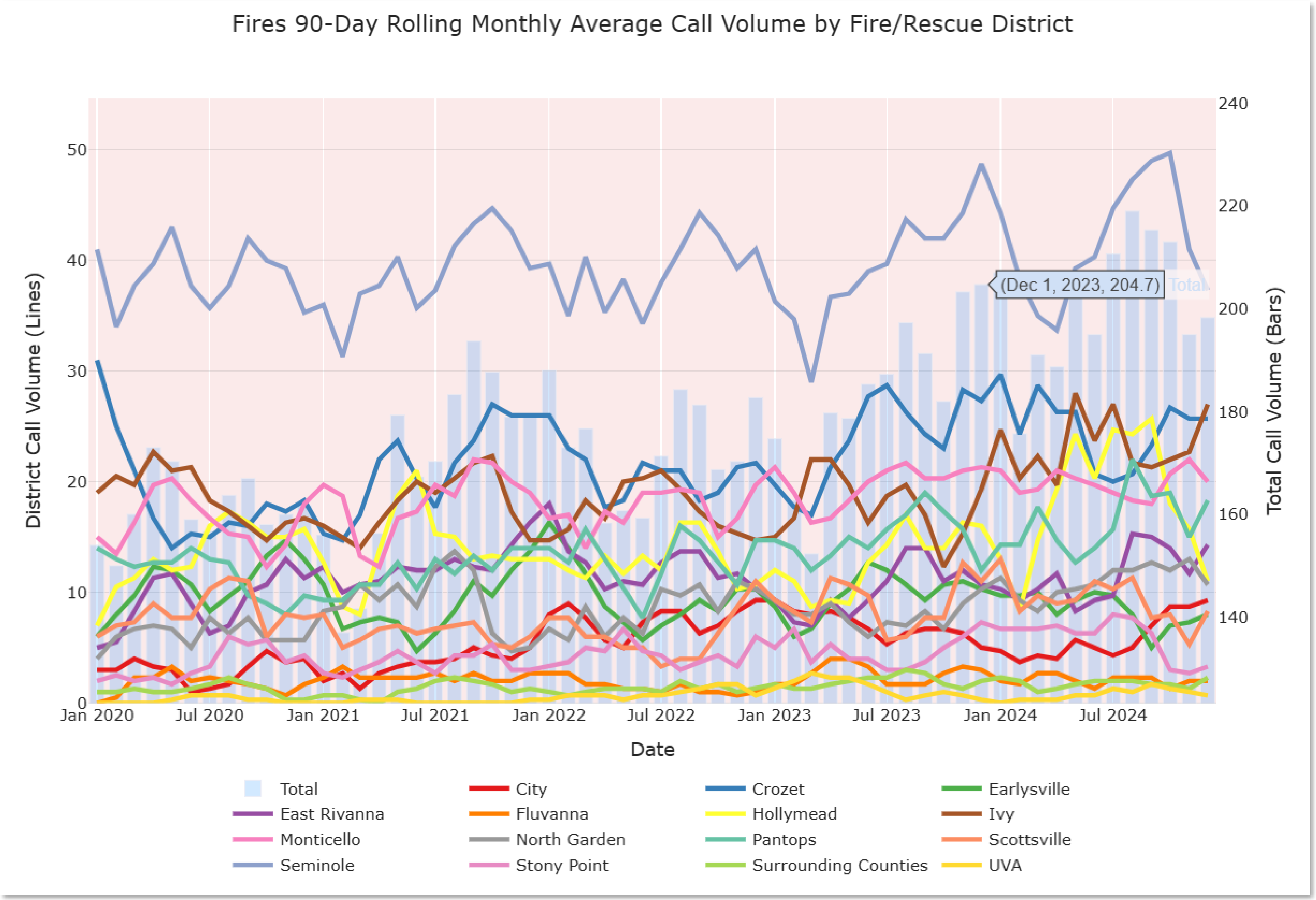


Call Volume by Month and CAD Category
Number of Calls
1500
1000
500
0
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Month
Rescue Incidents
Public Service
Police
Mutual Aid
Motor Vehicle Crashes
Miscellaneous
Infrastructure & Utilities
Hazmat
Fires



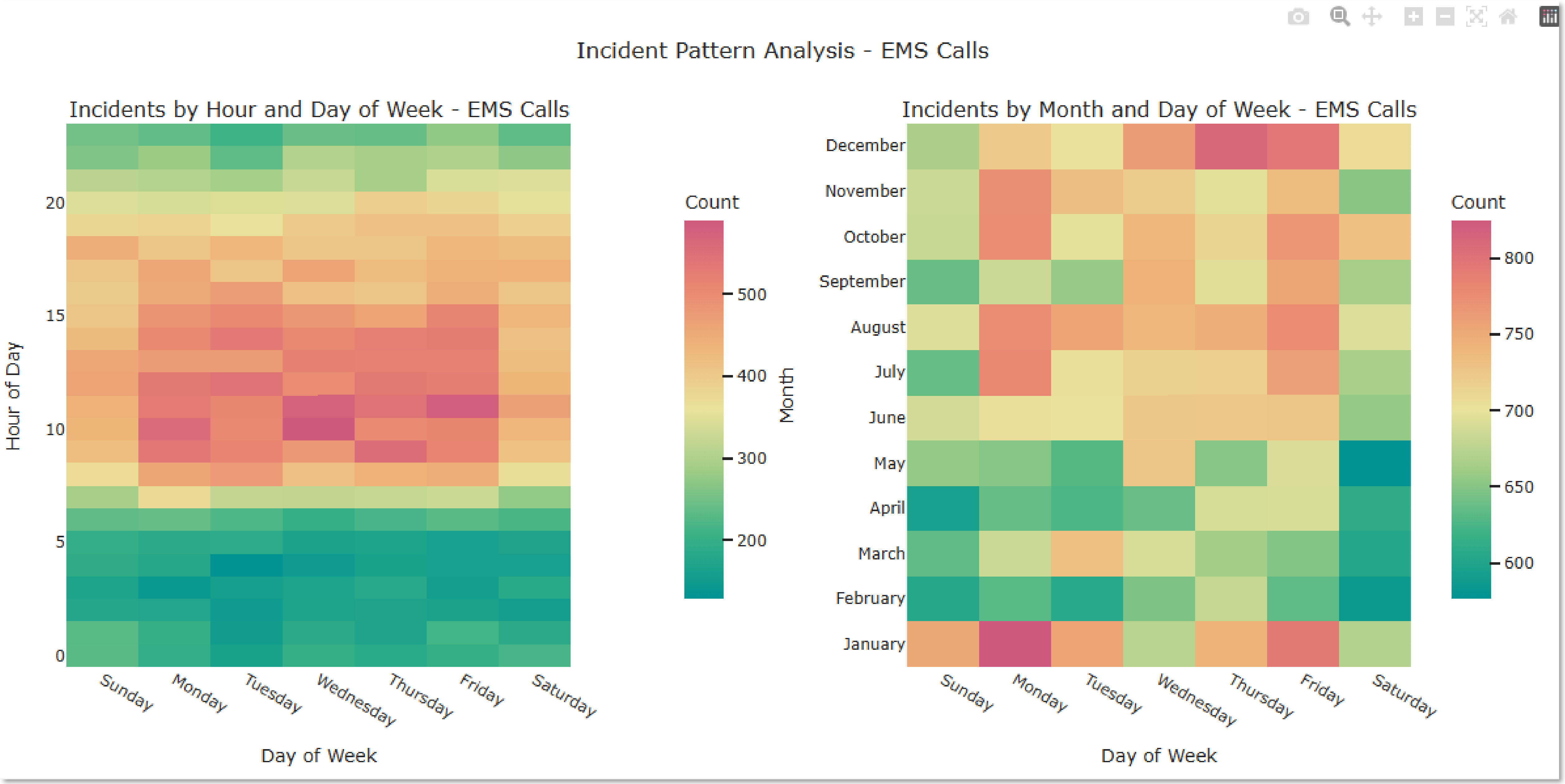
Monthly Call Volume and 90-Day Rolling Average for All
All
Monthly Calls
90-Day Rolling Average
January 2024
Calls: 1613
1800
1600
1400
1200
1000
800
600
Number of Calls
2020
2021
2022
2023
2024
Year


Call Volume Distribution by FireRescueDistrict
Share of Call Volume Over Time (%)
Absolute Call Volume Over Time
Year
2024
2023
2022
2021
2020
18,834
17,297
17,860
16,301
14,714
0
20
40
60
80
100
Call Volume (%)
0
5k
10k
15k
20k
Number of Calls
City
Crozet
Earlysville
East Rivanna
Fluvanna
Hollymead
Ivy
Monticello
North Garden
Pantops
Scottsville
Seminole
Stony Point
Surrounding Counties
UVA

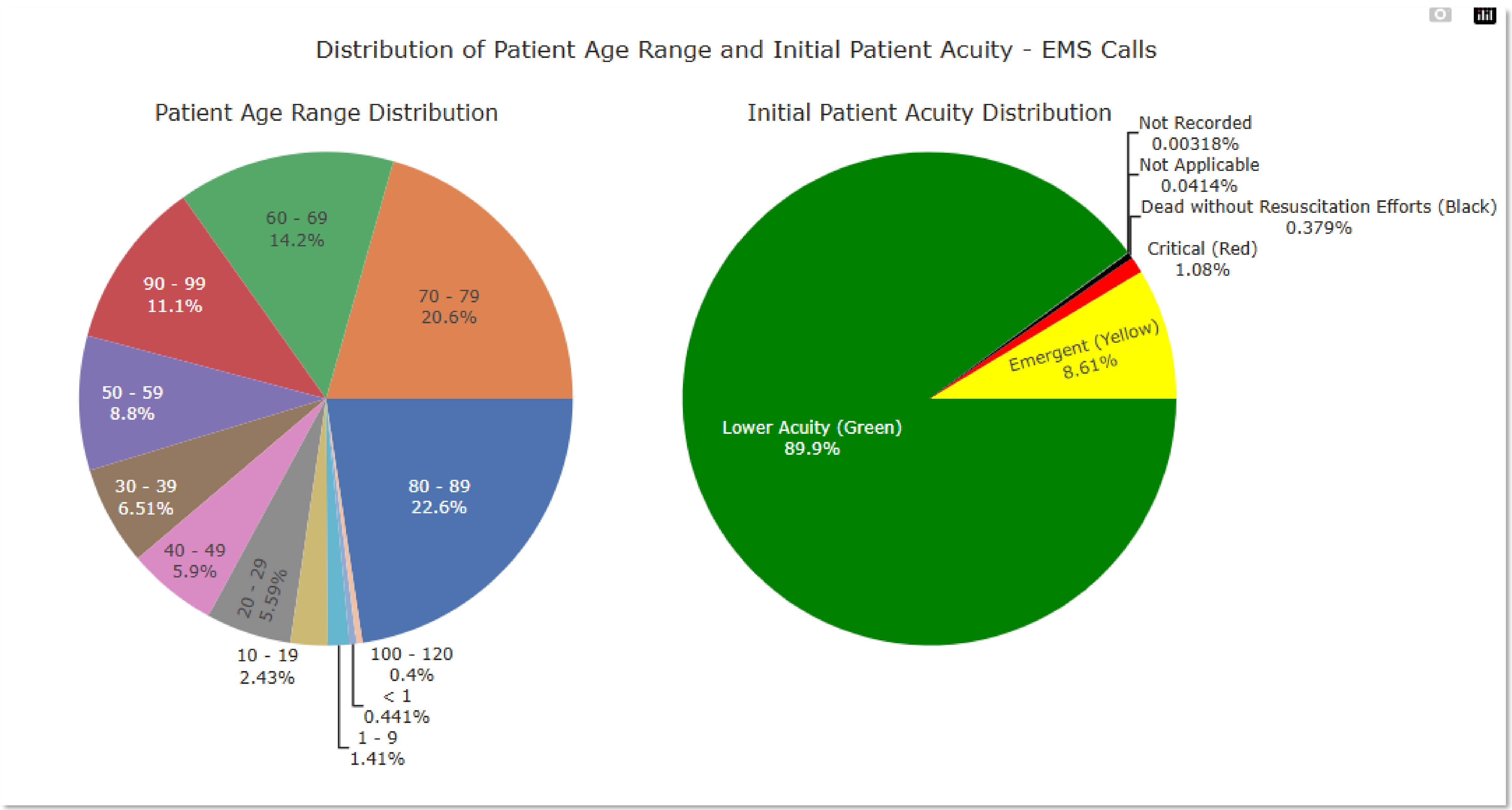


Fires 90-Day Rolling Monthly Average Call Volume by Fire/Rescue District
District Call Volume (Lines)
Total Call Volume (Bars)
0
10
20
30
40
50
140
160
180
200
220
240
Jan 2020
Jul 2020
Jan 2021
Jul 2021
Jan 2022
Jul 2022
Jan 2023
Jul 2023
Jan 2024
Jul 2024
Date
(Dec 1, 2023, 204.7)
Total
City
Crozet
Earlysville
East Rivanna
Fluvanna
Hollymead
Ivy
Monticello
North Garden
Pantops
Scottsville
Seminole
Stony Point
Surrounding Counties
UVA


Fire Risk Components by CAD Type
Trash Fire
Building Fire
Smoke Investigation
Fire False Alarm
Vehicle Fire
Fire, other
Wildland Fire
Outdoor Fire
Smoke Removal
Unauthorized Burning
Explosion
Probability
Impact
Consequence
0
0.2
0.4
0.6
0.8
1



Incident Pattern Analysis - EMS Calls
Incidents by Hour and Day of Week - EMS Calls
Hour of Day
0
5
10
15
20
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Day of Week
Count
200
300
400
500
Incidents by Month and Day of Week - EMS Calls
Month
December
November
October
September
August
July
June
May
April
March
February
January
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Day of Week
Count
600
650
700
750
800



Distribution of Patient Age Range and Initial Patient Acuity - EMS Calls
Patient Age Range Distribution
60 - 69
14.2%
70 - 79
20.6%
90 - 99
11.1%
50 - 59
8.8%
80 - 89
22.6%
30 - 39
6.51%
40 - 49
5.9%
20 - 29
5.59%
10 - 19
2.43%
100 - 120
0.4%
< 1
0.441%
1 - 9
1.41%
Initial Patient Acuity Distribution
Not Recorded
0.00318%
Not Applicable
0.0414%
Dead without Resuscitation Efforts (Black)
0.379%
Critical (Red)
1.08%
Emergent (Yellow)
8.61%
Lower Acuity (Green)
89.9%

GENERATIVE AI DEMO

Questions