Cincinnati Fire Incident Analysis

The statement of the problem that I will be exploring is to understand the patterns, trends, and relationships of fire incidents in Cincinnati and how they are related to various socio-economic and demographic factors.

This study will contribute to the current literature by providing a detailed analysis of fire incidents in Cincinnati, identifying high-risk areas, patterns and trends, and the relationship between fire incidents and socio-economic and demographic factors. This information can be used to inform emergency planning and incident response strategies, and to target fire prevention and education efforts to the areas that need it the most.

It is important to study this problem because fires are a significant public safety concern, and understanding the patterns, trends, and relationships of fire incidents can help to improve emergency planning, incident response, and fire prevention efforts in Cincinnati. This can ultimately lead to reducing the number of fire incidents and injuries and saving lives and properties.

Research questions for the project could include:

1. What are the patterns and trends of fire incidents in Cincinnati?
2. Are there any patterns or trends in the time it takes for the primary unit to arrive at the scene of the incident?
3. How can the information derived from this study be used to improve emergency planning and incident response in Cincinnati?

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
#import cincinnati fire department CSV file and converting it into a pandas dataframe
df = pd.read_csv("C:/Users/riyan/OneDrive/Documents/capstone project/Cincinnati_Fire_I
```

In [4]:
```
# Check number of rows and columns
print(f"Number of Rows:{ df.shape[0]:,}")
print(f"Number of Columns: {df.shape[1]:,}")

# Check column names
print("Column Names:", df.columns)

# Check data types of columns
print("Data Types:\n", df.dtypes)

# Check summary statistics of numerical columns
print("Summary Statistics:\n", df.describe())
```

```
Number of Rows:688,869
Number of Columns: 17
Column Names: Index(['ADDRESS_X', 'LATITUDE_X', 'LONGITUDE_X', 'AGENCY',
       'CREATE_TIME_INCIDENT', 'DISPOSITION_TEXT', 'EVENT_NUMBER',
       'INCIDENT_TYPE_ID', 'INCIDENT_TYPE_DESC', 'NEIGHBORHOOD',
       'ARRIVAL_TIME_PRIMARY_UNIT', 'BEAT', 'CLOSED_TIME_INCIDENT',
       'DISPATCH_TIME_PRIMARY_UNIT', 'CFD_INCIDENT_TYPE',
       'CFD_INCIDENT_TYPE_GROUP', 'COMMUNITY_COUNCIL_NEIGHBORHOOD'],
      dtype='object')
Data Types:
 ADDRESS_X                          object
LATITUDE_X                         float64
LONGITUDE_X                        float64
AGENCY                              object
CREATE_TIME_INCIDENT                object
DISPOSITION_TEXT                    object
EVENT_NUMBER                        object
INCIDENT_TYPE_ID                    object
INCIDENT_TYPE_DESC                  object
NEIGHBORHOOD                        object
ARRIVAL_TIME_PRIMARY_UNIT           object
BEAT                                object
CLOSED_TIME_INCIDENT                object
DISPATCH_TIME_PRIMARY_UNIT          object
CFD_INCIDENT_TYPE                   object
CFD_INCIDENT_TYPE_GROUP             object
COMMUNITY_COUNCIL_NEIGHBORHOOD      object
dtype: object
Summary Statistics:
           LATITUDE_X    LONGITUDE_X
count  665299.000000  665299.000000
mean       39.140136     -84.514690
std         0.296522       0.374391
min       -86.998771    -114.997841
25%        39.114048     -84.557141
50%        39.135938     -84.515510
75%        39.159177     -84.485088
max        90.000660     155.639117
```

Data Explanation

Cincinnati fire incident dataset contains the following columns and each column has an explanation below:

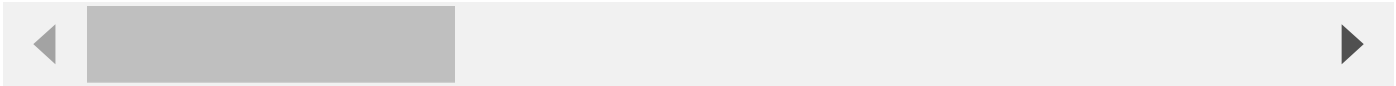
ADDRESS_X: The X coordinate of the incident address in the state plane coordinate system. LATITUDE_X: The latitude of the incident address. LONGITUDE_X: The longitude of the incident address. AGENCY: The agency responsible for responding to the incident. CREATE_TIME_INCIDENT: The date and time when the incident was created in the computer-aided dispatch (CAD) system. DISPOSITION_TEXT: The text description of the final disposition code assigned to the incident. EVENT_NUMBER: The unique identifier for the incident in the CAD system. INCIDENT_TYPE_ID: The numeric code for the type of incident. INCIDENT_TYPE_DESC: The text description of the type of incident. NEIGHBORHOOD: The name of the neighborhood where the incident occurred. ARRIVAL_TIME_PRIMARY_UNIT: The date and time when the primary responding unit arrived on the scene of the incident. BEAT: The beat or police patrol district where the incident occurred. CLOSED_TIME_INCIDENT: The date and time

when the incident was closed or resolved in the CAD system. DISPATCH_TIME_PRIMARY_UNIT: The date and time when the primary responding unit was dispatched to the incident. CFD_INCIDENT_TYPE: The code for the type of incident assigned by the Cincinnati Fire Department (CFD). CFD_INCIDENT_TYPE_GROUP: The code for the group of incident types assigned by the CFD. COMMUNITY_COUNCIL_NEIGHBORHOOD: The name of the community council in the neighborhood where the incident occurred.

In [5]:
```
df.head(10)
```

Out[5]:

	ADDRESS_X	LATITUDE_X	LONGITUDE_X	AGENCY	CREATE_TIME_INCIDENT	DISPOSITION_TEXT
0	KEMPER LN	39.126040	-84.487895	CFD	11/16/2021 01:41:53 PM	MEDD: MT DISREGARDED
1	W NORTH BEND RD	39.196987	-84.477879	CFD	11/16/2021 01:33:00 PM	GI: GOOD INTENT
2	STILLWELL RD	39.205596	-84.456781	CFD	11/16/2021 01:29:34 PM	MEDT: MEDIC TRANSPORT
3	NOTTINGHAM RD	39.155347	-84.561519	CFD	11/16/2021 01:26:42 PM	IN: INVESTIGATION
4	GEORGIA AV	39.164977	-84.549168	CFD	11/16/2021 01:23:38 PM	EMS: NO TRANSPORT
5	LOSANTIVILLE AV	39.191268	-84.457610	CFD	11/16/2021 01:12:54 PM	MED: MT RESPONSE NO TRANSPORT
6	GLENWAY AV	39.136268	-84.614533	CFD	11/16/2021 01:10:49 PM	MED: MT RESPONSE NO TRANSPORT
7	WINNESTE AV	39.184900	-84.512450	CFD	11/16/2021 01:07:30 PM	MED: MT RESPONSE NO TRANSPORT
8	HAMILTON AV	39.188275	-84.546562	CFD	11/16/2021 01:04:40 PM	CN: CANCEL,AV: ADVISED
9	PARK AV	39.128086	-84.484654	CFD	11/16/2021 01:03:08 PM	NaN



In [6]:
```
# Check missing values in columns
print("Missing Values:\n", df.info())

# Drop irrelevant columns
# columns agency, event number, beat, incident type id all have values which doesn't c
# so we will go ahead and drop them
df = df.drop(["AGENCY", "EVENT_NUMBER", "BEAT", "INCIDENT_TYPE_ID"], axis=1)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 688869 entries, 0 to 688868
Data columns (total 17 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   ADDRESS_X                       688762 non-null  object 
 1   LATITUDE_X                      665299 non-null  float64
 2   LONGITUDE_X                     665299 non-null  float64
 3   AGENCY                          688869 non-null  object 
 4   CREATE_TIME_INCIDENT            688834 non-null  object 
 5   DISPOSITION_TEXT                683826 non-null  object 
 6   EVENT_NUMBER                    688869 non-null  object 
 7   INCIDENT_TYPE_ID                685982 non-null  object 
 8   INCIDENT_TYPE_DESC              684497 non-null  object 
 9   NEIGHBORHOOD                    664081 non-null  object 
 10  ARRIVAL_TIME_PRIMARY_UNIT       589609 non-null  object 
 11  BEAT                            688294 non-null  object 
 12  CLOSED_TIME_INCIDENT            677727 non-null  object 
 13  DISPATCH_TIME_PRIMARY_UNIT      626219 non-null  object 
 14  CFD_INCIDENT_TYPE               654760 non-null  object 
 15  CFD_INCIDENT_TYPE_GROUP         685189 non-null  object 
 16  COMMUNITY_COUNCIL_NEIGHBORHOOD  661894 non-null  object 
dtypes: float64(2), object(15)
memory usage: 89.3+ MB
Missing Values:
 None
```

In [7]:
```
# percentage of missing values in each column
df.isnull().sum()/df.shape[0]*100
```

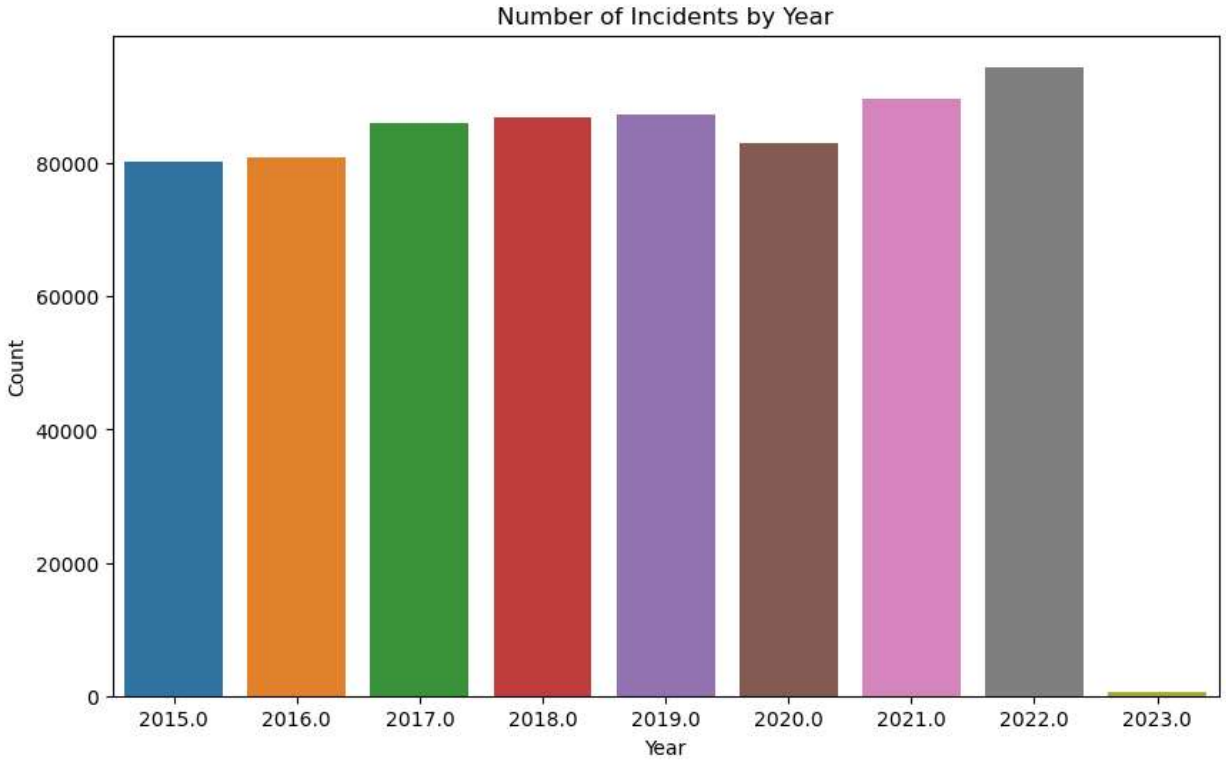
Out[7]:
```
ADDRESS_X                        0.015533
LATITUDE_X                       3.421550
LONGITUDE_X                      3.421550
CREATE_TIME_INCIDENT             0.005081
DISPOSITION_TEXT                 0.732070
INCIDENT_TYPE_DESC               0.634663
NEIGHBORHOOD                     3.598362
ARRIVAL_TIME_PRIMARY_UNIT       14.409126
CLOSED_TIME_INCIDENT             1.617434
DISPATCH_TIME_PRIMARY_UNIT       9.094617
CFD_INCIDENT_TYPE                4.951449
CFD_INCIDENT_TYPE_GROUP          0.534209
COMMUNITY_COUNCIL_NEIGHBORHOOD   3.915839
dtype: float64
```

In [8]:
```
# fill missing values in each column with mean or median as appropriate

df['LATITUDE_X'].fillna(df['LATITUDE_X'].median(), inplace=True)
df['LONGITUDE_X'].fillna(df['LONGITUDE_X'].median(), inplace=True)



# verify if there are any missing values left
print(df.isnull().sum())
```

```
ADDRESS_X                        107
LATITUDE_X                         0
LONGITUDE_X                        0
CREATE_TIME_INCIDENT              35
DISPOSITION_TEXT                5043
INCIDENT_TYPE_DESC              4372
NEIGHBORHOOD                   24788
ARRIVAL_TIME_PRIMARY_UNIT      99260
CLOSED_TIME_INCIDENT           11142
DISPATCH_TIME_PRIMARY_UNIT     62650
CFD_INCIDENT_TYPE              34109
CFD_INCIDENT_TYPE_GROUP         3680
COMMUNITY_COUNCIL_NEIGHBORHOOD 26975
dtype: int64
```

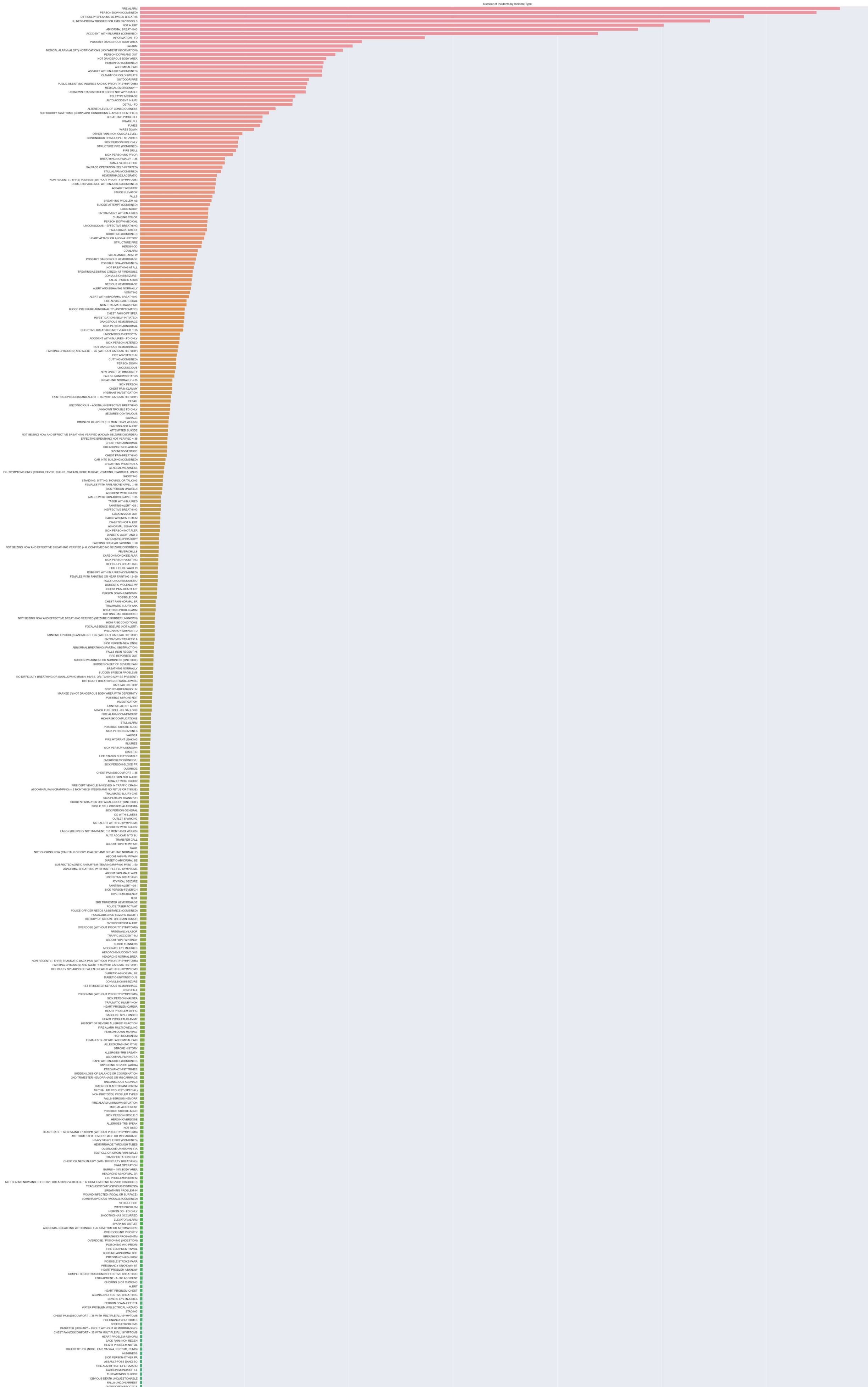
In [9]:
```python
# Convert "incident_date" column to datetime format
df["incident_date"] = pd.to_datetime(df["CREATE_TIME_INCIDENT"], format="%m/%d/%Y %I:%

# Create new columns for year, month, day, hour, and minute of incident date
df["year"] = df["incident_date"].dt.year
df["month"] = df["incident_date"].dt.month
df["day"] = df["incident_date"].dt.day
df["hour"] = df["incident_date"].dt.hour
df["minute"] = df["incident_date"].dt.minute
```

In [10]:
```python
# Visualize number of incidents by year

plt.figure(figsize=(10, 6)) # adjust width and height as needed
sns.countplot(x="year", data=df)
plt.title("Number of Incidents by Year")
plt.xlabel("Year")
plt.ylabel("Count")
plt.show()
```



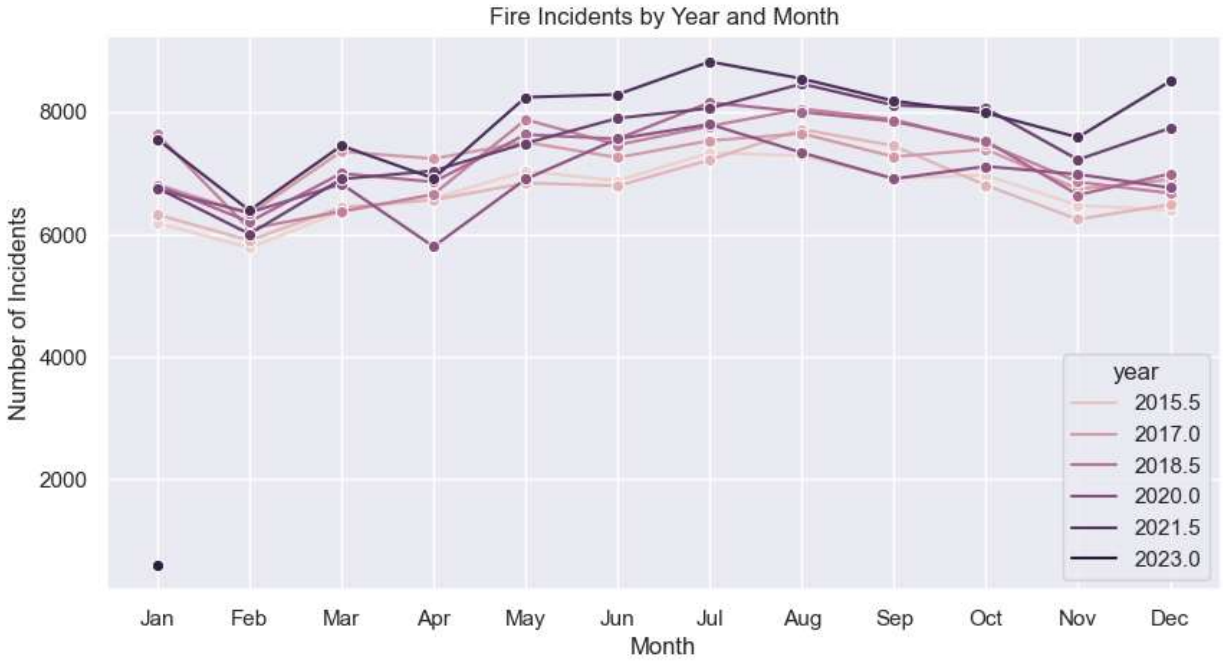
In [11]:
```
# Set figure size and font scale
sns.set(rc={"figure.figsize":(50,200)})


# Visualize number of incidents by incident type
sns.countplot(y="INCIDENT_TYPE_DESC", data=df, order=df["INCIDENT_TYPE_DESC"].value_co
plt.title("Number of Incidents by Incident Type")
plt.xlabel("Count")
plt.ylabel("Incident Type")
plt.yticks(rotation=0, ha='right') # Rotate and align labels
plt.tight_layout() # Adjust spacing between labels
plt.show()
```

```
C:\Users\riyan\AppData\Local\Temp\ipykernel_8448\4135388788.py:11: UserWarning: Glyph
26 (□) missing from current font.
  plt.tight_layout() # Adjust spacing between labels
C:\Users\riyan\anaconda3\lib\site-packages\IPython\core\pylabtools.py:151: UserWarnin
g: Glyph 26 (□) missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
```


Number of Incidents by Incident Type

In [12]:

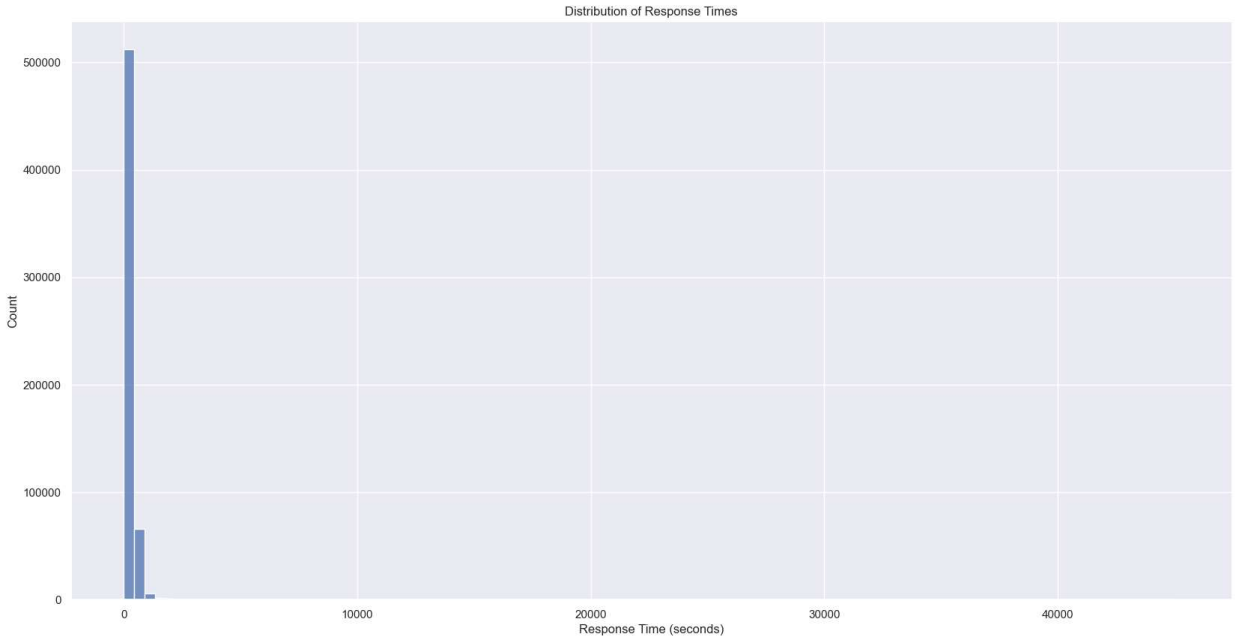
```
# group by year and month and count the number of incidents
incidents_by_month = df.groupby(['year', 'month']).size().reset_index(name='count')

# create line plot of incidents by year and month
plt.figure(figsize=(10,5))
sns.lineplot(data=incidents_by_month, x='month', y='count', hue='year', marker='o')
plt.title('Fire Incidents by Year and Month')
plt.xlabel('Month')
plt.xticks(range(1,13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep
plt.ylabel('Number of Incidents')
plt.show()
```



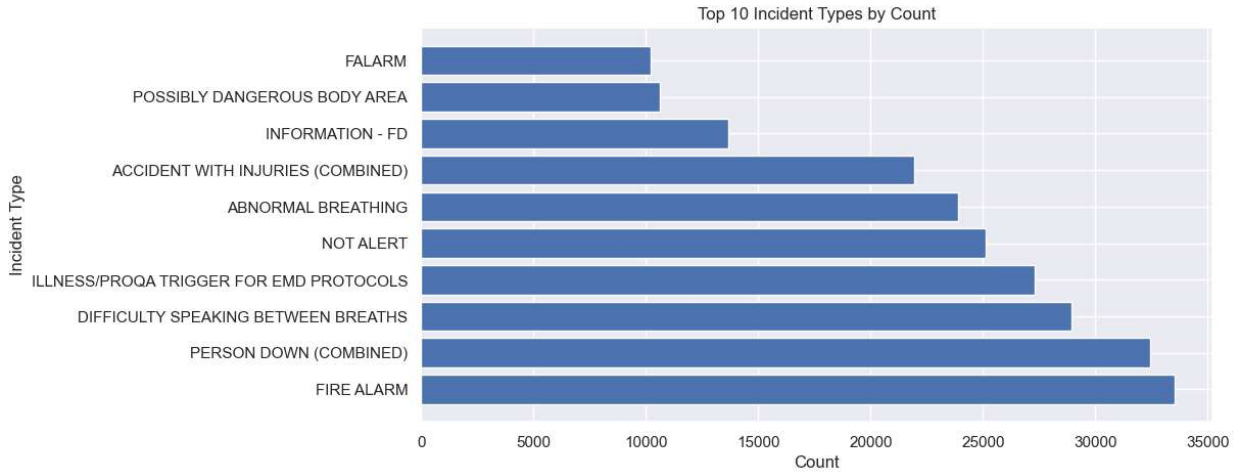
In [17]:
```
# Convert CREATE_TIME_INCIDENT and ARRIVAL_TIME_PRIMARY_UNIT columns to datetime formc
df["CREATE_TIME_INCIDENT"] = pd.to_datetime(df["CREATE_TIME_INCIDENT"], format="%m/%d/
df["ARRIVAL_TIME_PRIMARY_UNIT"] = pd.to_datetime(df["ARRIVAL_TIME_PRIMARY_UNIT"], forr

# Calculate response time in seconds
df["response_time"] = (df["ARRIVAL_TIME_PRIMARY_UNIT"] - df["CREATE_TIME_INCIDENT"]).c

# Verify if there are any missing values in response_time column
print("Missing Values in Response Time:\n", df["response_time"].isnull().sum())
```

```
Missing Values in Response Time:
 99289
```

In [35]:
```
# Filter out negative response times
response_times = df[df["response_time"] >= 0]["response_time"]

plt.figure(figsize=(20,10))

# Plot distribution of response times
sns.histplot(response_times, kde=False, bins=100)
plt.title("Distribution of Response Times")
plt.xlabel("Response Time (seconds)")
plt.ylabel("Count")
plt.show()
```



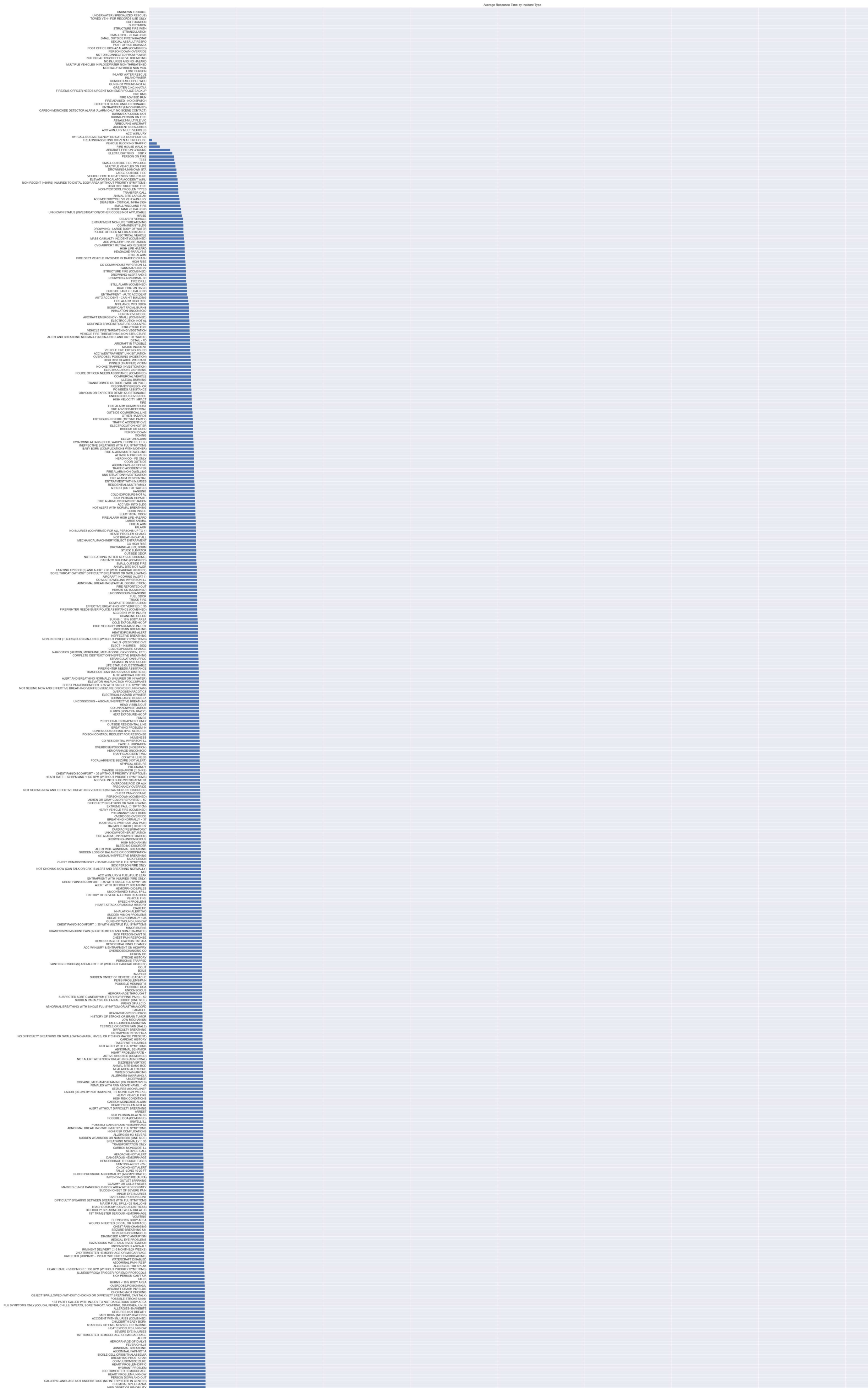
In [14]:
```
plt.figure(figsize=(10,5))
top10 = df["INCIDENT_TYPE_DESC"].value_counts().nlargest(10)
plt.barh(top10.index, top10.values)
plt.title("Top 10 Incident Types by Count")
plt.xlabel("Count")
plt.ylabel("Incident Type")
plt.show()
```



In [47]:
```
df["response_times"]=response_times
```

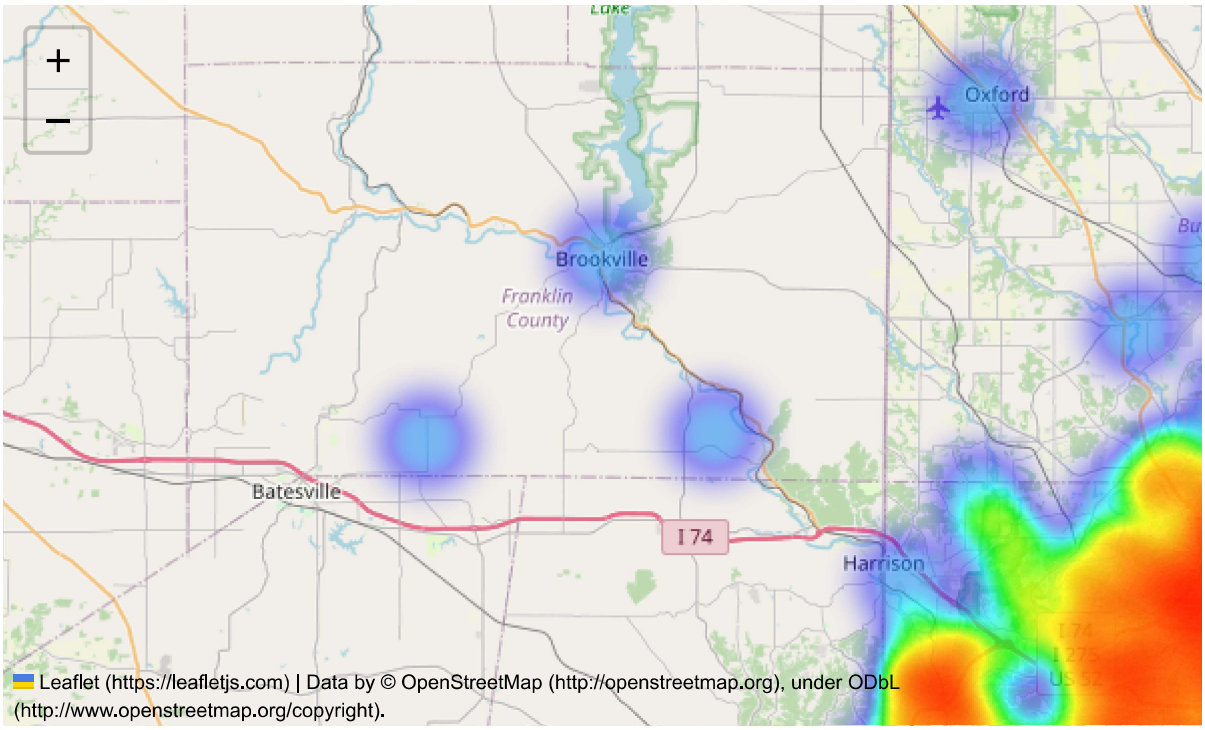
In [50]:
```
avg_response_time = df.groupby("INCIDENT_TYPE_DESC")["response_times"].mean().sort_val
plt.barh(avg_response_time.index, avg_response_time.values)
plt.title("Average Response Time by Incident Type")
plt.xlabel("Response Time (seconds)")
plt.ylabel("Incident Type")
plt.show()
```

```
C:\Users\riyan\anaconda3\lib\site-packages\IPython\core\pylabtools.py:151: UserWarnin
g: Glyph 26 (␚) missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
```

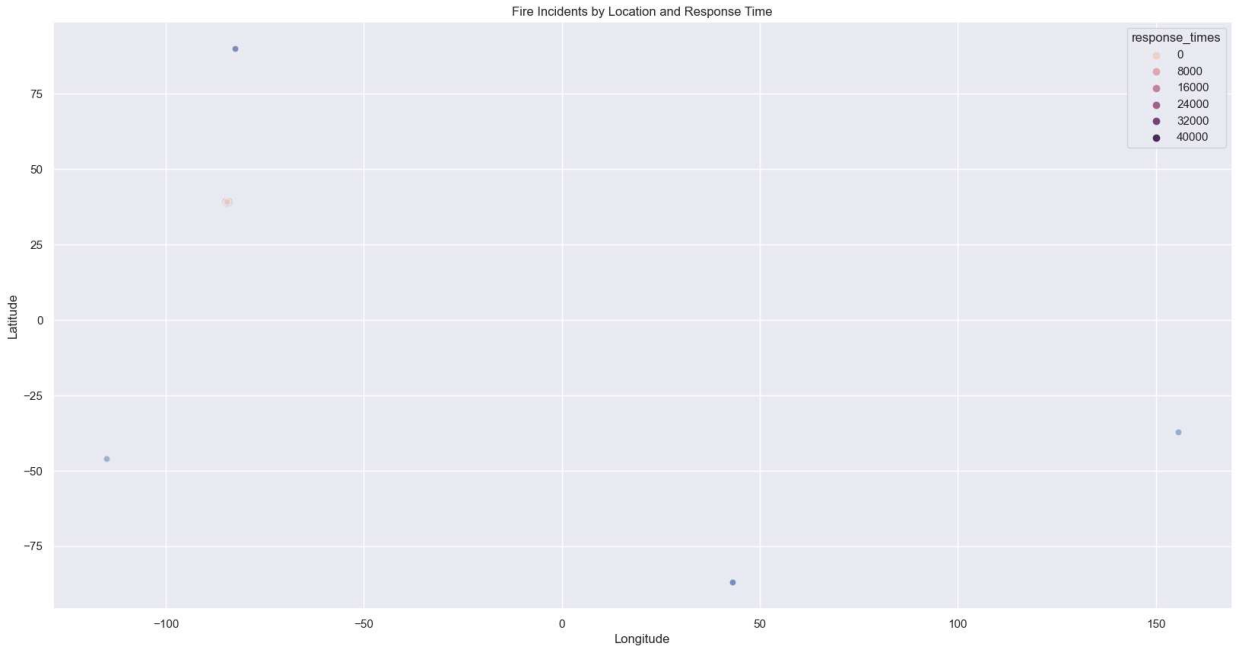

Average Response Time by Incident Type

In [26]:
```
import folium
from folium.plugins import HeatMap

# Create a map centered on the median latitude and longitude
m = folium.Map(location=[df['LATITUDE_X'].median(), df['LONGITUDE_X'].median()], zoom_

# Create a heatmap layer using the incident location data
heat_data = df[['LATITUDE_X', 'LONGITUDE_X']].values.tolist()
HeatMap(heat_data).add_to(m)

# Display the map
m
```

Out[26]:

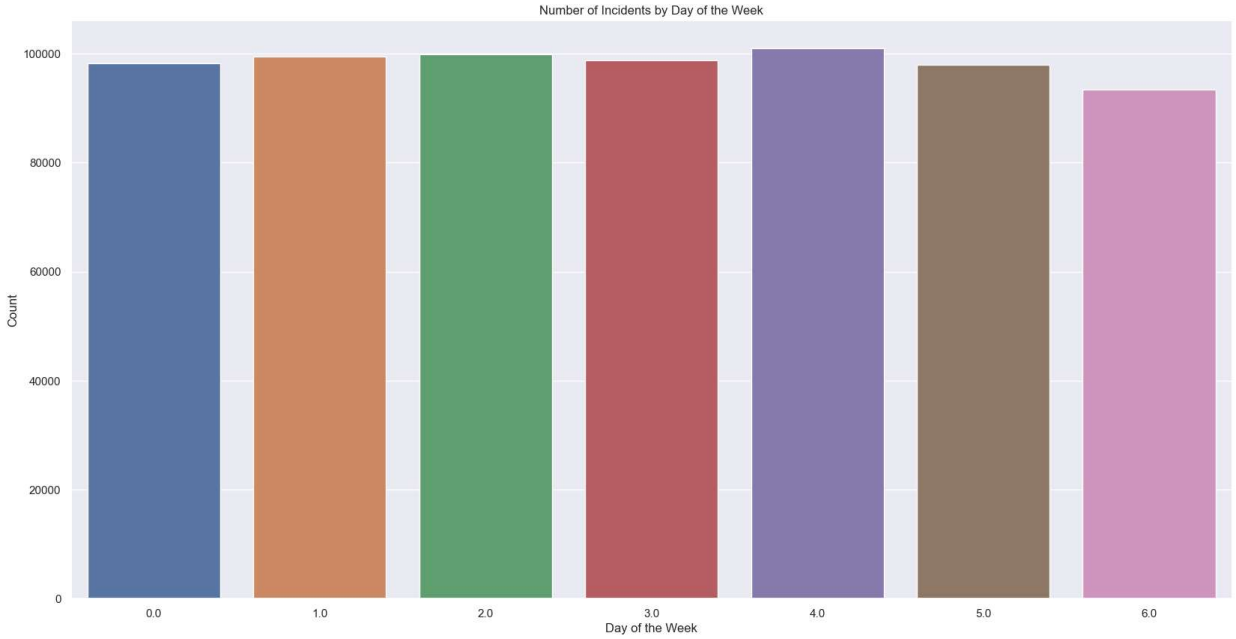


In [54]:
```
# Create a scatter plot of incidents based on their latitude and longitude
plt.figure(figsize=(20, 10))
sns.scatterplot(x='LONGITUDE_X', y='LATITUDE_X', data=df, alpha=0.5)

# Add the response time to the plot using the color of the points
sns.scatterplot(x='LONGITUDE_X', y='LATITUDE_X', hue='response_times', data=df, alpha=

# Set the title and axis labels
plt.title('Fire Incidents by Location and Response Time')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```

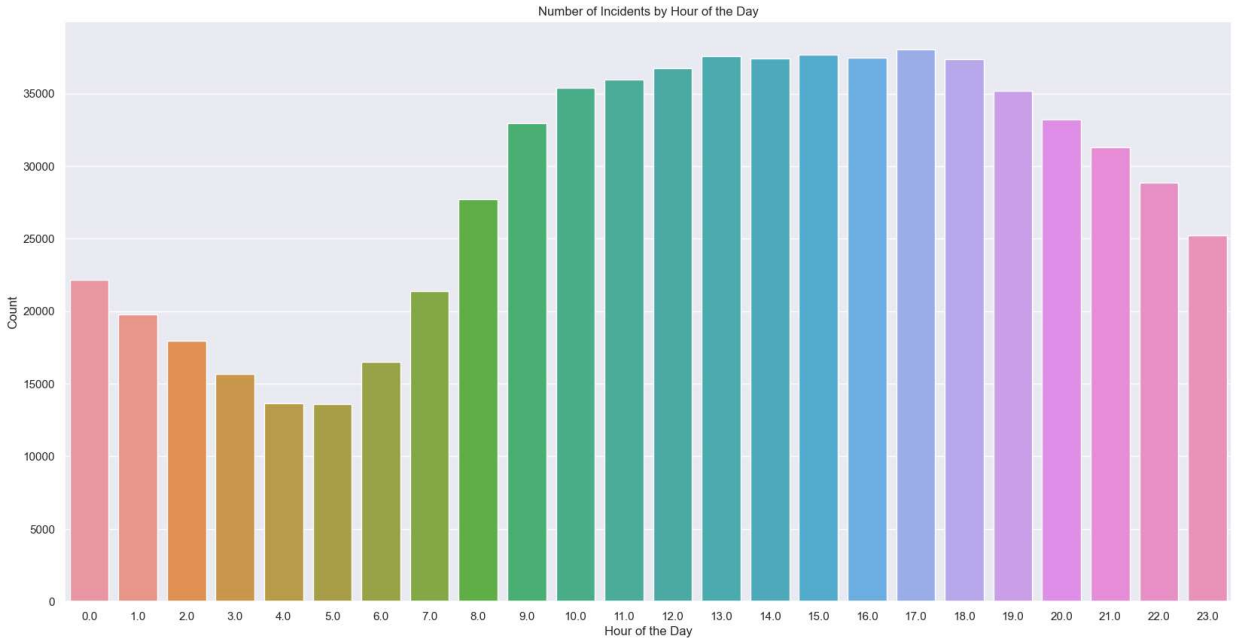


In [19]:
```
df["day_of_week"] = df["incident_date"].dt.weekday
df["hour_of_day"] = df["incident_date"].dt.hour
```

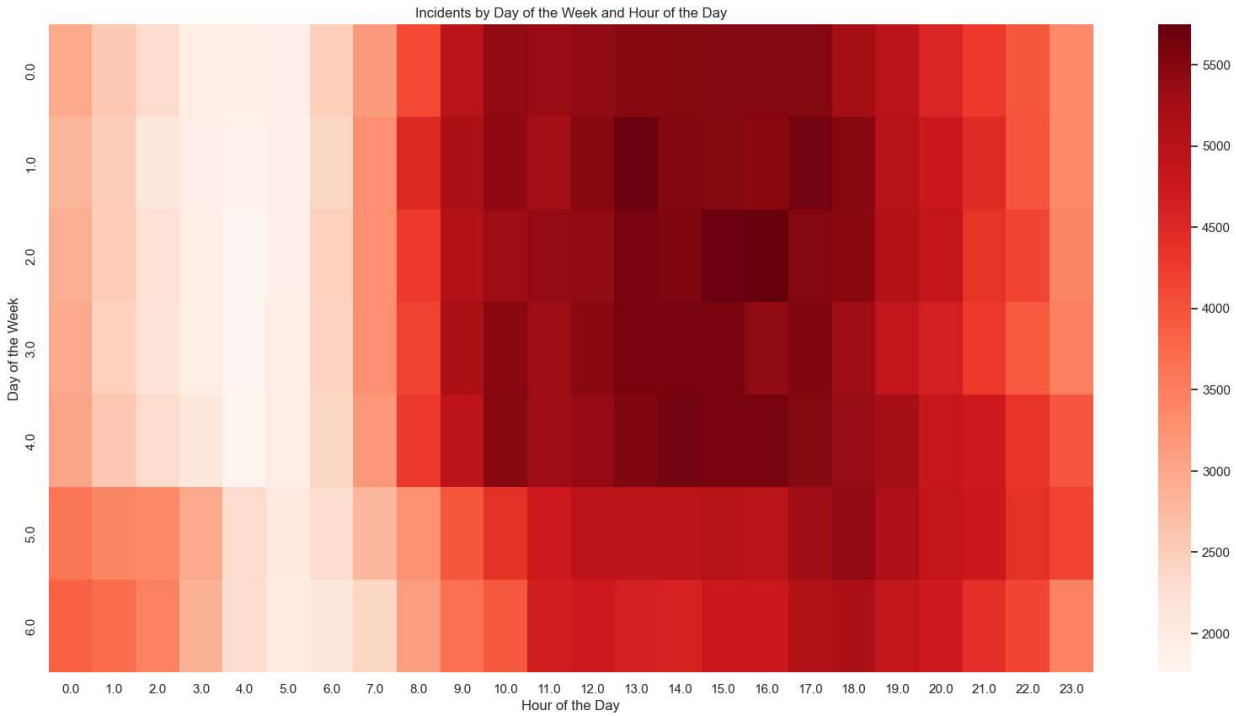
In [20]:
```
plt.figure(figsize=(20,10))
sns.countplot(x="day_of_week", data=df)
plt.title("Number of Incidents by Day of the Week")
plt.xlabel("Day of the Week")
plt.ylabel("Count")
plt.show()
```



In [21]:
```
plt.figure(figsize=(20,10))
sns.countplot(x="hour", data=df)
plt.title("Number of Incidents by Hour of the Day")
plt.xlabel("Hour of the Day")
plt.ylabel("Count")
plt.show()
```



In [22]:
```
incidents_by_day_hour = df.groupby(["day_of_week", "hour_of_day"]).size().reset_index(
incidents_by_day_hour = incidents_by_day_hour.pivot("day_of_week", "hour_of_day", "cou

plt.figure(figsize=(20,10))
sns.heatmap(incidents_by_day_hour, cmap="Reds")
plt.title("Incidents by Day of the Week and Hour of the Day")
plt.xlabel("Hour of the Day")
plt.ylabel("Day of the Week")
plt.show()
```



In [58]:
```
from prophet import Prophet
import warnings

warnings.filterwarnings('ignore')

# Create a new dataframe with the count of incidents by date
```

```
incidents_by_date = df.groupby(pd.Grouper(key="incident_date", freq="D")).size().reset
incidents_by_date = incidents_by_date.rename(columns={"incident_date": "ds", "count":

# Create and fit the Prophet model
model = Prophet()
model.fit(incidents_by_date)

# Make predictions for the next 30 days
future = model.make_future_dataframe(periods=30)
forecast = model.predict(future)

# Plot the forecast
model.plot(forecast)
plt.title("Forecast of Incidents")
plt.xlabel("Date")
plt.ylabel("Number of Incidents")
plt.show()
```

```
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to
override this.
```